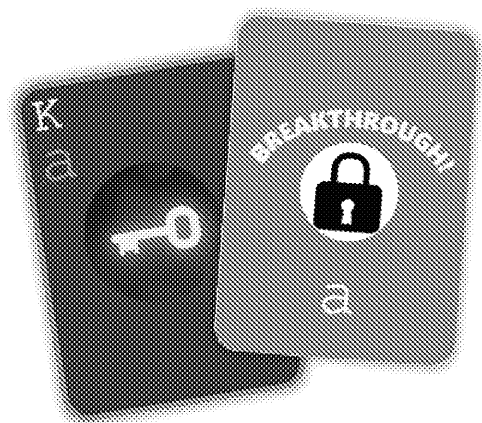


2015 specification
for the 2022 exam



PAPER 1 EXAM RESOURCE PACK 2022

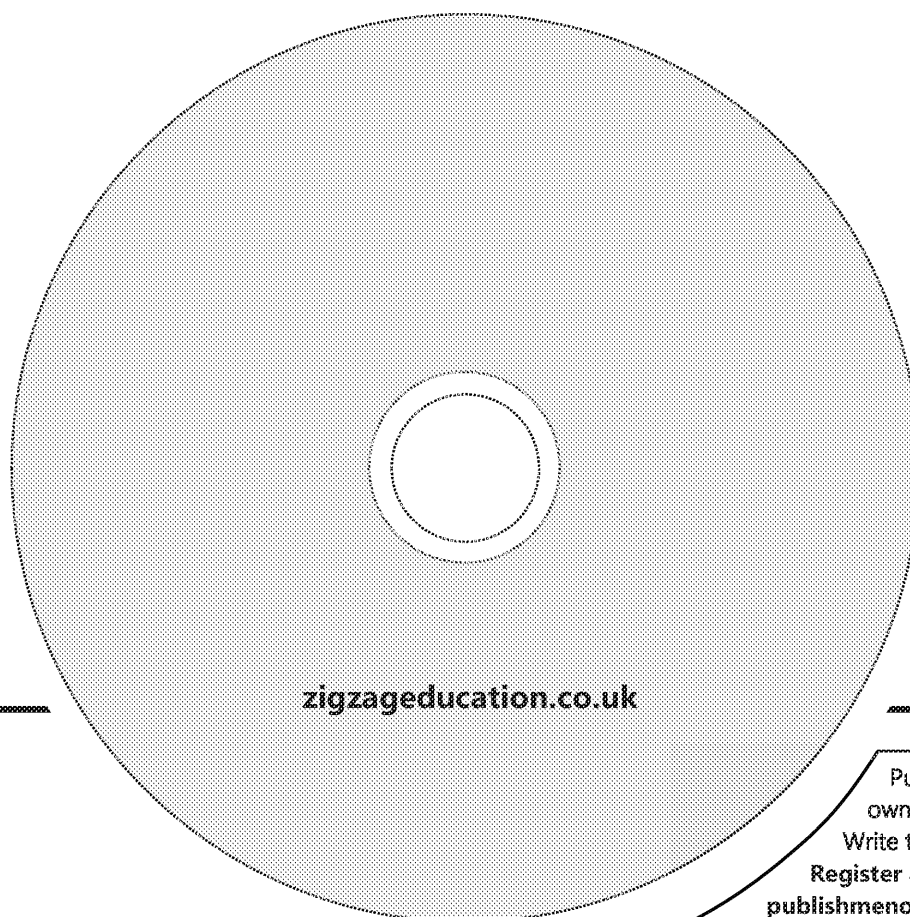
BREAKTHROUGH!

for A Level AQA Computer Science

C# EDITION

DH6/
11147

POD
11147



zigzageducation.co.uk

Publish your
own work...
Write to a brief...
Register at
publishmenow.co.uk

Contents

Product Support from ZigZag Education	ii
Terms and Conditions of Use	iii
Teacher's Introduction	iv

Printouts of CD resources (for reference)

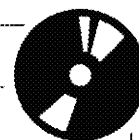
- Code Breakdown (10 pages)
- UML Class Diagram – Complete (1 page)*
- Theory Questions: Write-on version (9 pages)
- Theory Questions: Non-write-on version (4 pages)
- Coding Tasks (16 pages)
- Additional Tasks (Extension) (1 page)
- Theory Questions: Mark Scheme (6 pages)
- Programming Tasks: Mark Scheme (45 pages)
- Electronic Answer Document (4 pages)

** Note there are also electronic copies of the UML Diagrams ('Complete' & 'Activity' versions) on the CD – which can be printed in A3, making them much more usable (especially when used as activities)*

Teacher's Introduction

This resource pack is designed to help you support your students taking the A Level Computer Science Paper 1 exam. It is based on the *Breakthrough!* preliminary material (C#) – for examination summer 2022.

On the CD, you will find the following:



- | | |
|---------------|--|
| Breakthrough | this folder contains all of the content (PDF/DOCX) accessible via a HTML interface |
| Passwords.txt | for teacher use – this file contains all of the passwords for the protected PDFs (also listed below) |

* PRINTED COPIES OF ALL THE MATERIALS IN THIS DIGITAL RESOURCE PACK ARE INCLUDED FOR REFERENCE.

Installation: Copy the entire Breakthrough folder onto a network location that is accessible for students, and provide them with a shortcut to the index.html file. All content can be accessed from this page.

Passwords: All of the PDFs accessible via the *Solutions* web page are password-protected, so that students can only access them with your permission. Each password is a four-digit code, as follows:

- | | |
|-------------------------------|--------|
| c02a-UML-Diagram-Complete.pdf | ██████ |
| c06-TheoryQuestions-MS.pdf | ██████ |
| c07-CodingTasks-MS.pdf | ██████ |

Should you wish to give students access to ALL protected-PDFs, the master password for all files is: ██████

The resource pack consists of the following:

① Code Breakdown

This document gives a detailed technical overview of the skeleton program, describing in detail each class and method in turn – including their purpose/function, parameters and return values.

Note: although this section is intended to give extra support to teachers and students, it should in no way be seen as a substitute to a student exploring the code for themselves.

② Class Diagrams

Three UML Class Diagrams help students explore the skeleton program; there is a completed version, a partially-complete version (gap-fill), as well as a mostly blank template. The completed version is password-protected and accessible via the *Solutions* web page.

③ Video

Quick video going over the *Breakthrough!* card game mechanics – intended as a visual aid to accompany the notes in the official AQA preliminary material.

④ Written Questions

Theory questions testing students' understanding of the skeleton program. These questions require access to the program, but no modifications need to be made to the program. Write-on (with answer lines) and non-write-on versions are available. Suggested answers are provided via the *Solutions* web page as a password-protected PDF.

⑤ Coding Tasks

Fifteen modification exercises put students' programming skills to the test. Example solutions with suggested mark schemes are provided via the *Solutions* web page as a password-protected PDF. Note that these are example solutions and you must use your discretion to award marks accordingly where there are valid alternative solutions.

An Electronic Answer Document (EAD) is provided should you wish students to use it for ③ and/or ④ above.

This resource is intended to supplement your teaching only. Please read full disclaimer (p. iii) before using it.

BREAKTHROUGH

Skeleton Code Breakdown

Class: Breakthrough

Identifier / Data		Option
<<constructor>>		
Parameters	n/a	Initialises several private attributes including: <ul style="list-style-type: none">• Deck to a new CardCollection• Hand to a new CardCollection• Sequence to a new CardCollection• Discard to a new CardCollection• Score to 0• GameOver to False• Locks to an empty list• CurrentLock to an empty Lock• LockSolved to False Invokes the LoadLocks() method to load 'locks.txt'.
Return values	n/a	
AddDifficultyCardsToDeck (private)		
Parameters	n/a	Uses a counter-controlled loop to add 50 CardCollection objects to the Deck.
Return values	n/a	
CheckIfLockIsSolved (private)		
Parameters	n/a	Iterates through the Sequence CardCollection together the string SequenceAsString and the separator between each card description. As a new element from Sequence is compared to SequenceAsString, the string is compared to the lock using the CheckIfCondition method. If the lock is not met, the lock is incremented. If the lock is met, True is returned.
Return values	Boolean	
CheckIfPlayerHasLost (private)		
Parameters	n/a	Checks to see if there are any cards left in the Deck. If none, an appropriate message is displayed and the game is over. If there are cards still left in the Deck, False is returned, allowing the player to continue.
Return values	Boolean	
CreateSetupGameDeck (private)		
Parameters	n/a	Used by the SetupGame() method to create the correct File, Pick and Keys for each player. 5 Picks from toolkits a, b and c are added to the Deck. 3 Files and 3 Keys from toolkits a, b and c are added to the Deck.
Return values	n/a	

INSPECTION COPY

**COPYRIGHT
PROTECTED**



Identifier / Data		Description
GetCardChoice (private)		
Parameters	n/a	Used by the PlayGame() method in their Hand they would like to use.
Return values	Value : Integer	
		Contains error handling to catch not caught out of range.
GetCardFromDeck (private)		
Parameters	cardChoice : Integer	Used to get the next card from the Deck and add it to the Hand.
Return values	Integer	<p>If the Deck CardCollection has a card, the system will then check if the card is a DifficultyCard. If a DifficultyCard is found, they would like to lose a 'Key' card from the Deck. The DifficultyCard is moved to the Discard CardCollection and the card is added to the Hand on the DifficultyCard passing the parameters.</p> <p>The system then performs a check to see if the Hand is full. If not, it repopulates the Hand with cards from the Deck. If another Difficulty card is found, it is moved to the Discard CardCollection. If a Difficulty card (or cards if there is more than one) is found in the Deck, it is moved automatically to the Discard CardCollection rather than into the Hand.</p> <p>If the Deck is empty, out of cards, the game ends.</p>
GetChoice (private)		
Parameters	n/a	Used by the PlayGame() method in their Hand they would like to use a card from their Hand CardCollection on the screen.
Return values	String	
GetDiscardChoice (private)		
Parameters	n/a	Used by the PlayGame() method in their Hand they would like to play the selected card from the Discard the selected card from the CardCollection.
Return values	Choice : String	
GetRandomLock (private)		
Parameters	n/a	Returns a randomly selected lock from the Locks.
Return values	Lock	
LoadGame (private)		
Parameters	fileName : String	Uses the fileName parameter to load the game using a StreamReader. Imports the Hand and CardCollections for the Hand and Deck.
Return values	Boolean	
		True is returned if the file is loaded successfully. If an error occurs, an error message is returned.

COPYRIGHT
PROTECTED



Identifier / Data		Description
LoadLocks (private)		
Parameters	n/a	<p>Uses a hard-coded 'locks.txt' file to load the locks available for the game. Each line in the text file represents a single lock. Each line from the file is split – Challenges, using the first part as a delimiter into the Conditions and the second as the variable – LockFromFile. The Conditions are then loaded into the private attribute Locks.</p> <p>If an error occurs, an error message is displayed to advise that the locks.txt file is not found.</p>
Return values	n/a	
MoveCard (private)		
Parameters	fromCollection : CardCollection toCollection : CardCollection cardNumber : Integer	<p>Moves a card at the position specified in the CardCollection from the fromCollection to the toCollection.</p> <p>If the fromCollection is the same as the toCollection, the card is not moved. If the card has been chosen (i.e. not on the table), the card is updated appropriately. If the card is not found, all other moves from one collection to another are updated.</p> <p>Score is returned.</p>
Return values	Score : Integer	
PlayCardToSequence (private)		
Parameters	cardNumber : Integer	<p>This method is used to play a card to the Sequence to test if it is a valid move.</p> <p>The system tests to see if the card is in the CardCollection. If it is, then checks to see if the card is the same as the user is a different Tool. If the Tool is the same as the played card, the card is not played. If the Tool is different, the card can be played and the card is added to the Sequence and the card is removed from the CardCollection appropriately for that card. The user gets a new card from the CardCollection.</p> <p>If the Sequence does not have a card, the system moves the card to the Sequence and the Score is incremented.</p> <p>If the system then uses the CheckIfLockChallenge method to see if the new card added to the Sequence meets the conditions to be met and if so displays a message on the screen and increments the Score.</p>
Return values	n/a	

COPYRIGHT
PROTECTED



Identifier / Data		Description
PlayGame (public)		
Parameters	n/a	This contains the main game loop.
Return values	n/a	Checks to confirm if the private list attribute Lock by the LoadLocks() method. If none have been on the screen and the game then quits. If the list does contain locks, it initialises the following attribute Over to a false CurrentLock to a new Lock object Invokes the SetupGame() method to set up the game. The main game loop runs while the private attribute Lock is not solved. There is then an inner loop which runs while GameOver is false and the private attribute LockSolved is also False. The inner game loop displays the current user score, the current lock and the contents of the Hand, and then uses the GetChoice() method to display a choice. The inner loop then uses selection to either display the Discard card in the game. If the user selects to use a card, the system uses the GetCard() method to select a card. It then uses the GetDiscard() method to confirm if the user wants to play or discard the card. If the user selects discard, the system moves the selected card to the Discard CardCollection and gets a new card from the Deck using the GetCardFromDeck() method. If the user selects play, the system uses the PlayCard() method to move the card to the Play CardCollection. Once a card has been played or discarded, the system uses the GetLockSolved() method on the CurrentLock to check if the challenges have been met. If they have, the Lock is solved and a new lock is generated. If a lock has been solved, the inner loop returns to the main loop which checks if the game is over by invoking the GameOver() method. If this returns True the game ends.
ProcessLockSolved (private)		
Parameters	n/a	Increments the Score by 10 and displays the user score.
Return values	n/a	Uses an indefinite loop to iterate through the Discard cards, returning all of the cards back to the Deck. Reshuffles the Deck using the Shuffle() method and then uses the GetRandomLock() method to generate a new lock with the private attribute Lock.

**COPYRIGHT
PROTECTED**



Identifier / Data		Description
SetupCardCollectionFromGameFile (private)		
Parameters	lineFromFile : String cardCol : CardCollection	Used for processing lines 4 of game file which are for processing CardCollections (namely the SetupCards).
Return values	n/a	Receives a single line of text (parameter) from the external file and processes it into a CardCollection. If lineFromFile contains text, it is split into a SplitLine, using the comma as a delimiter. The SplitLine list is then processed to extract the card number and card type and added to a CardCollection. If a Difficulty is added instead of a normal TextCard, it is added instead of a normal TextCard.
SetupGame (private)		
Parameters	n/a	Called from the PlayGame() method as a message of the game on the system. It would like to load in an external game file. If the player chooses to play a new game, the system attempts to load the game file. If the game file cannot be loaded the game is not played.
Return values	n/a	If the player chooses to play a new game, a new Deck is generated using the GenerateDeck() method and then shuffled using the ShuffleDeck() method. It then moves 5 cards to start the player off. The system then adds 5 cards to the Deck using the AddDifficultyCardsToDeck() method. The cards are in random locations and a new lock is added at random to the Deck using the GetRandomLock() method.
SetupLock (private)		
Parameters	line1 : String line2 : String	Used for processing lines 2 of game file which contain the lock information.
Return values	n/a	The parameter line1 contains the lock number and the parameter line2 contains the lock name. Each line is split into a string using the comma as a delimiter. The line1 parameter is then used to add a new challenge to the Deck. A single line may contain multiple challenges. The line2 parameter is split using a semicolon to populate the Met status for each challenge using the SetChallengesMet() method.

COPYRIGHT
PROTECTED



Class: Challenge

Identifier / Data		Description
<<constructor>>		
Parameters	n/a	Initialises the following properties: • Met to False • Conditions to an empty list
Return values	n/a	
GetCondition (public)		
Parameters	n/a	Returns a list of strings of conditions in the lock.
Return values	Condition : List (String)	
GetMet (public)		
Parameters	n/a	Returns the value of the Met property.
Return values	Met : Boolean	
SetCondition (public)		
Parameters	newCondition : List (String)	Sets the value of the property Condition from the parameter newCondition.
Return values	n/a	
SetMet (public)		
Parameters	newValue : Boolean	Sets the value of the property Met to the parameter newValue.
Return values	n/a	

Class: Lock

This class does not have a specific constructor and therefore has no constructor.

Identifier / Data		Description
<<constructor>>		
Parameters	n/a	Initialises the Challenges empty list.
Return values	n/a	
AddChallenge (public) <<virtual>>		
Parameters	condition : List (String)	Initialises a new challenge condition from the parameter condition and adds it to the Challenges list.
Return values	n/a	
		Adds the new challenge to the Challenges list.
CheckIfConditionMet (public) <<virtual>>		
Parameters	sequence : String	Returns True and sets the Met property to True if the Sequence matches a challenge, otherwise it returns False.
Return values	Boolean	
ConvertConditionToString (private)		
Parameters	c : List (String)	Converts list of conditions to a string displaying on the screen. Uses the parameter c, concatenates each condition using ConditionAsString() using the delimiter.
Return values	ConditionAsString : String	
GetChallengeMet (public) <<virtual>>		
Parameters	pos : Integer	Returns the Met status of the challenge at pos in the Challenges list.
Return values	Boolean	


INSPECTION COPY

COPYRIGHT
PROTECTED



Identifier / Data		Description
GetLockDetails (public) <<virtual>>		
Parameters	n/a	Used for displaying a challenge's details through the Challenges protected together the output string LockDetails version of all the challenges for the been met or not.
Return values	LockDetails: String	
GetLockSolved (public) <<virtual>>		
Parameters	n/a	Returns the status showing if a lock through the Challenges protected there are any unmet ones, otherwise
Return values	Boolean	
GetNumberOfChallenges (public) <<virtual>>		
Parameters	n/a	Returns the number of Challenges number of challenges in this lock).
Return values	Integer	
SetChallengeMet (public) <<virtual>>		
Parameters	pos : Integer value : Boolean	Uses the SetMet() method in the attribute of a challenge at the position list to Met or not Met using the value
Return values	n/a	

Class: Card

Identifier / Data		Description
<<constructor>>		
Parameters	n/a	initialises the CardNumber static attribute (class variable) increments the static attribute NextCardNumber which is and updated for all objects of Initialises the Score protected
Return values	n/a	
		
GetCardNumber (public) <<virtual>>		
Parameters	n/a	Returns the value of the protected
Return values	CardNumber : Integer	
GetDescription (public) <<virtual>>		
Parameters	n/a	Returns the protected attribute string.
Return values	CardNumber: String	
GetScore (public) <<virtual>>		
Parameters	n/a	Returns the protected attribute
Return values	Score : Integer	
Process (public) <<virtual>>		
Parameters	deck : CardCollection hand : CardCollection sequence : CardCollection currentLock : Lock choice : String cardChoice : Integer	Base class method for the subclasses to override.
Return values	n/a	

COPYRIGHT
PROTECTED



Class: ToolCard (inherits from Card)

Identifier / Data		Description
<<constructor>>		
Parameters	t : String k : String	Initialises the following protected attributes <ul style="list-style-type: none">• ToolType from parameter t• Kit from parameter k
Return values	n/a	
		Invokes the setScore() method to assign the correct Score to the base class for the ToolType
<<constructor>>		
Parameters	t : String k : String cardNo : Integer	Initialises the following protected attributes <ul style="list-style-type: none">• ToolType from parameter t• Kit from parameter k• CardNumber from parameter cardNo
Return values	n/a	
		Invokes the setScore() method to assign the correct Score to the base class for the ToolType
SetScore (private)		
Parameters	n/a	Assigns the correct Score from the parameter
Return values	n/a	
GetDescription (public) <<override>>		
Parameters	n/a	Overrides the GetDescription() method to return the concatenated string of the ToolType and Kit of this ToolCard
Return values	String	

Class: DifficultyCard (inherits from Card)

Identifier / Data		Description
<<constructor>>		
Parameters	n/a	Initialises the following protected attributes <ul style="list-style-type: none">• CardType to "Dif"
Return values	n/a	
<<constructor>>		
Parameters	cardNo : Integer	Initialises the following protected attributes <ul style="list-style-type: none">• CardType to "Dif"• CardNumber from parameter cardNo
Return values	n/a	
GetDescription (public) <<override>>		
Parameters	n/a	Overrides the GetDescription() method to return the protected attribute CardType
Return values	String	

COPYRIGHT
PROTECTED




Identifier / Data		Description
Process (public) <<override>>		
Parameters	deck : CardCollection discard : CardCollection hand : CardCollection sequence : CardCollection currentLock : Lock choice : String cardChoice : String	Overrides the Process() method to process the user choices from the user. The user receives a difficulty card like to draw a card a key or 5 cards. Choosing the option to draw a card or select a key. This method the parameter is valid. Although errors in this check, AQA code is written as it was in
Return values	n/a	If the choice parameter contains a valid index by subtracting 1 from the player's hand, the card is removed from the hand and placed in the discard CardCollection. If the choice parameter does not contain a valid index, the card is removed from the deck and placed in the discard CardCollection.

Class: CardCollection

Identifier / Data		Description
<<constructor>>		
Parameters	n : String	Initialises the following protected attributes:
Return values	n/a	<ul style="list-style-type: none"> Name from parameter n Cards to an empty list
GetCardFromList (public)		
Parameter	x : Integer	Returns a string containing the card at index x in the Cards list by invoking the method in Card.
Return values	String	
GetCardNumberAt (public)		
Parameters	x : Integer	Returns the CardNumber attribute of the card at index x in the Cards list.
Return values	Integer	
GetName (public)		
Parameters	n/a	Returns the value of the protected attribute name.
Return values	Name : String	
AddCard (public)		
Parameters	c : Card	Adds the value of parameter c to the Cards list.
Return values	n/a	
CreateLineOfDashes (private)		
Parameter	size : Integer	Used in formatting a CardCollection.
Return values	LineOfDashes : String	Returns an appropriately sized LineOfDashes of elements in a CardCollection. If the size of the CardCollection is greater than the size of the LineOfDashes, the LineOfDashes is padded with spaces.

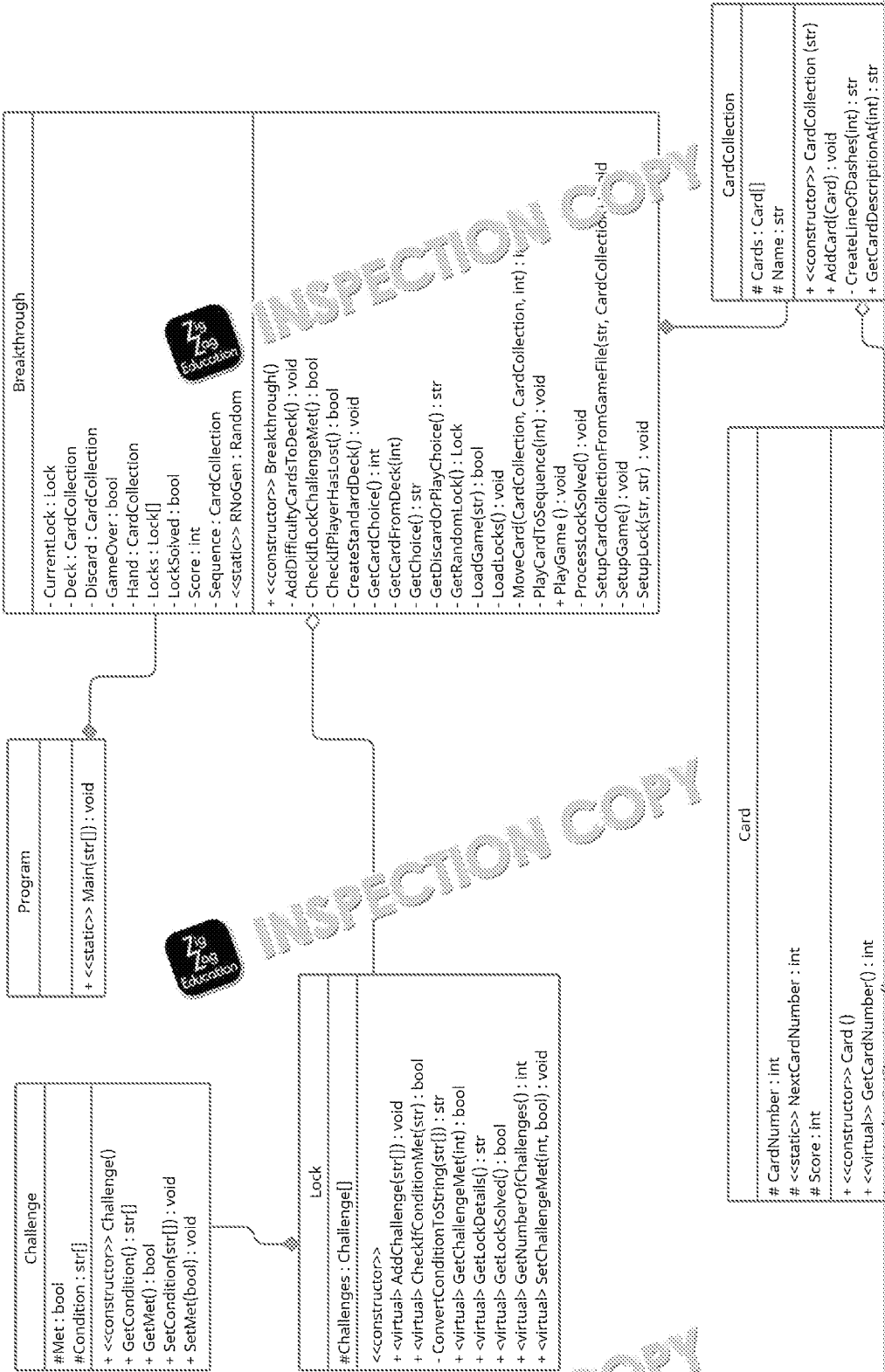
COPYRIGHT
PROTECTED



Identifier / Data		Description
GetCardDisplay (public)		
Parameters	n/a	<p>Used in formatting a CardCollection display output of a CardCollection. It takes the collection Name and card description list attributes. If there are no cards in the collection, 'empty' is returned. If there are cards in the collection, it returns a string which is either appropriately sized for the collection or is fixed at 10 if the collection is greater than 10. This fits correctly in the terminal window.</p> <p>It then uses indefinite iteration to iterate over the cards using the GetDescription() method of the card at each element and concatenates the (pipe) symbol to create a 'line of cards'. It then creates a second line of description underneath the 'line of cards' and returns the final string.</p>
Return values	CardDisplay : String	
		
GetNumberOfCards (public)		
Parameters	n/a	Returns the number of cards in the Cards.
Return values	Integer	
RemoveCard (public)		
Parameters	cardNumber : Integer	<p>Returns the card from Cards list at the index cardNumber. If cardNumber is not a valid index, an exception is thrown. If cardNumber is a valid index, the card is removed from Cards and the variable CardToGet is returned.</p>
Return values	CardtoGet : Card	
Shuffle (private)		
Parameters	n/a	<p>Uses definite iteration to perform a shuffle of the Cards from one random position to another. It iterates through the Cards attribute Cards in order to generate a random position and swaps the card at that position with the card at the end of the list.</p>
Return values	n/a	

COPYRIGHT
PROTECTED





COPYRIGHT
PROTECTED



INSPECTION COPY

BREAKTHROUGH

Theory Questions

These questions refer to the **Preliminary Material** and the **Skills** but **do not** require any additional programming.

TOTAL MARKS: 80

- 1 Exam Zig Zag Education private method `MoveCard`. Currently this method returns

(a) State a more appropriate name for this local variable.

.....

(b) Currently the `MoveCard` method returns an integer which represents the index of the card that was moved. Sometimes this return value is ignored.

Evaluate the choice (of the programmer) to ignore the return value and return 0 in some cases, and suggest an alternative implementation.

.....

.....

.....

.....

.....

.....

- 2 The class `CardCollection` currently contains an interface that exposes the structure of a list. For the sequence and the discard pile, a more appropriate data structure would be either a queue or a stack.

(a) Justify whether you would use a queue or a stack. When giving your answer, refer to the functionality of the data structure to the behaviour of the game.

.....

.....

.....

.....

.....

.....

.....

INSPECTION COPY

COPYRIGHT
PROTECTED



- (b) In order to implement a stack or a queue for the sequence, justify (and if possible, implement) that you would make to the inheritance structure.

.....

.....

.....

.....

- (c) How would you be going a new class to handle a `CardCollection` that implements encapsulation?

.....

.....

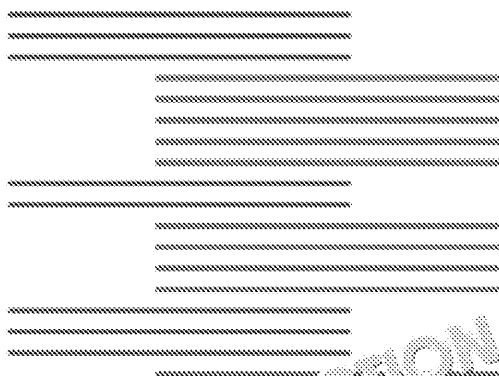
.....

.....

- 3 The `Shuffle` method of the `CardCollection` class currently swaps 10,000 cards in order to shuffle the deck.

Another way of shuffling the deck is to use a method that humans would normally use called a 'riffle shuffle'. This involves splitting the deck into two approximately equal piles and then flicking through each pile from the bottom while combining the halves together into a single deck. Another way of thinking of this would be to imagine pushing the two halves together and having a random number of cards between each card from each half as they recombine.

For example, a deck combined from a blue half and an orange half might look something like this:



Note that in the perfect case a riffle shuffle would use one card from each half at every point, but this is not desired, and in reality, between 0 and 5 cards will normally interleave from one half at any time.

COPYRIGHT
PROTECTED



- a) Write a detailed algorithm for riffle shuffle in any format you choose (pseudocode, flow chart).



INSPECTION COPY

- b) Explain the space complexity of your algorithm.



INSPECTION COPY

- 4 Examine the `CheckIfLockChallengeMet` method of the `Breakthrough` class and the `CheckIfConditionMet` method of the `Lock` class.


Lock:

Challenge Met: P c, F c, K c

Not met: P a, F a, P a

Sequence: P c, F c, K c, P a, F a, P a

- (a) For the above sequence and lock, complete the trace table below for the `CheckIfLockChallengeMet` method of the `Breakthrough` class.



Count	SequenceAsString	Return value
	""	
5		

INSPECTION COPY

**COPYRIGHT
PROTECTED**



- (b) If the above lock had a third challenge as below, then how would it work? (please fill it in below)?

Not met: F a, P a

Count	SequenceAsString	Return value
	""	
5		

- 5 Examine the `ProcessLockSolved` method in the `Breakthrough` class. What methods are called by that method.

- (a) When a new lock is set, if that lock has been solved before, it will be automatically replaced with a new lock the following turn (and treated as if it just solved the first new lock) but reward the player for solving the

.....

.....

.....

.....

- (b) Describe the logical change you would make to the code (no need to write code, just explain) to ensure that this no longer happens.

.....

.....

.....

.....

- 6 Examine the `Shuffle` method in `CardCollection`. This method will make a new set of cards in the deck.

- (a) Explain how the effectiveness and efficiency of this algorithm decreases as the number of cards in the deck reduces.

.....

.....

.....

.....

INSPECTION COPY

COPYRIGHT
PROTECTED



- (b) Other than introducing a riffle shuffle, justify how you could improve efficiency of the algorithm by describing any changes below.

.....

.....

.....

.....

- 7 ToolCard can be instantiated with either two or three arguments.

- (a) Explain what happens in the case where a third argument is supplied where only two arguments are supplied.

.....

.....

.....

.....

- (b) State the purpose of a constructor.

.....

.....

- 8 Examine the classes Card, ToolCard and DifficultyCard.

- (a) Using evidence from these classes in the program, explain the difference between an abstract and a concrete class.

.....

.....

.....

.....

.....

.....

**COPYRIGHT
PROTECTED**



- (b) Using evidence from the Card method, explain the difference between a static attribute and an attribute.

.....

.....

.....

.....

- 9 Find an example of the code for each of the following. Only write out the line/s of code.

- (a) Inheritance

.....

.....

- (b) Aggregation association

.....

.....

- (c) A dynamic data structure

.....

.....

- 10 This question refers to the concept of polymorphism and how it is used.

- (a) Choose and then write out one or more lines of the skeleton program for polymorphism and justify why this is an example of polymorphism.

.....

.....

.....

.....

.....

.....

.....

- (b) Define the term 'polymorphism'.

.....

.....

.....

.....

.....

INSPECTION COPY

COPYRIGHT
PROTECTED



- 11 A suggestion has been made to introduce a new `AdvancedLock` that challenge which is only revealed once the basic challenges have been

Explain the steps that you would take in order to do this, i.e. the logical change/addition and the reason for each step.

You are not required to implement this or to write any actual code.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

- 12 Examine the `Process` method in the `DifficultyCard` class and the `PlayGetCardFromDeck` method in the `Breakthrough` class.

Using the sequence below:

Not in sequence: P a, F a, K a

Sequence: P a, F a

Hand: P b, K a, F b, K c, P a

The player plays the 'K a' card to the sequence and then draws a difficulty card which requires them to either discard a key or five cards from the deck. The player discards the 'K c' from their hand, which is currently in position 4.

Explain what will happen when the `Process` method is called under the circumstances, including specific references to the lines of code executed and in which order, and the values of variables, especially `ChoiceAsInteger`.

You will need to ensure that you look at the `PlayCardToSequence` and `PlayGetCardFromDeck` methods in `Breakthrough` to be certain of the state of the Hand and Sequence when the `DifficultyCard` is drawn.

**COPYRIGHT
PROTECTED**





- 13 The terms 'HAND', 'SEQUENCE', 'DECK' and 'DISCARD' all appear at least once in the code and in some cases more than once. This is an example of hard-coded values. Hard-coded values are difficult to maintain and understand and also make it more prone to errors.
- (a) Describe one method of avoiding hard-coding values that makes the code more maintainable and easier to understand.



- (b) Explain why using hard-coded values makes the code more prone to errors and harder to understand.

**COPYRIGHT
PROTECTED**



14 Exception handling is used in several places in the skeleton code; two of them are the use of file handling.

(a) Describe why it is important to always use exception handling when using file handling.

.....

.....

.....

.....

(b) Give an example of another situation (not file handling) where exception handling is used (it doesn't have to be from the skeleton code) and explain why.

.....

.....

.....

.....

15 This question refers to the `PlayGame` method of the `Breakthrough` class.

Explain the use of the private attribute `GameOver` in this method, specifically how it is set and why it is used as the condition for terminating the loop.

.....

.....

.....

.....

END OF QUESTIONS

COPYRIGHT
PROTECTED



BREAKTHROUGH

Theory Questions

These questions refer to the **Preliminary Material** and the **Skills** but **do not** require any additional programming.

TOTAL MARKS: 80

- 1 Examine the private method `MoveCard`. Currently this method returns
- (a) State a more appropriate name for this local variable.
 - (b) Currently the `MoveCard` method returns an integer which represents the card that was moved. Sometimes this return value is ignored.

Evaluate the choice (of the programmer) to ignore the return value of `MoveCard` when it returns 0 in some cases, and suggest an alternative implementation.

- 2 The class `CardCollection` currently contains an interface that exposes the structure of a list. For the sequence and the discard pile, a more appropriate data structure would be either a queue or a stack.
- (a) Justify whether you would use a queue or a stack. When giving your justification, refer to the functionality of the data structure to the behaviour of the game.
 - (b) In order to implement a queue or a stack for the sequence, justify (in your answer) that you would make the inheritance structure.
 - (c) How would you be creating a new class to handle a `CardCollection` that implements the `CardCollection` interface?

- 3 The `Shuffle` method of the `CardCollection` class currently swaps 10,000 cards in order to shuffle the deck.

Another way of shuffling the deck is to use a method that humans would normally use called a 'riffle shuffle'. This involves splitting the deck into two approximately even piles and then flicking through each pile from the bottom while combining the halves together into a single deck. Another way of thinking of this would be to imagine pushing the two halves together and having a random number of cards between each card from each half as they recombine.

For example, a deck combined from a blue half and an orange half might look something like this:

Note that in the picture above a riffle shuffle would use one card from each half. In reality, between 0 and 5 cards will normally intersperse from either half any time.

- a) Write a detailed algorithm for riffle shuffle in any format you choose (pseudocode, flow chart).
- b) Explain the space complexity of your algorithm.

INSPECTION COPY

**COPYRIGHT
PROTECTED**



- 4 Examine the `CheckIfLockChallengeMet` method of the `Breakthrough` class and the `CheckIfConditionMet` method of the `Lock` class.

Lock:

Challenge Met: P c, F c, K c

Not met: P a, F a, P a

Sequence: P c, F c, K c, P a, F a, P a

- (a) For the above sequence and lock, complete a trace table like the one below for the `CheckIfLockChallengeMet` method of the `Breakthrough` class.

Count	SequenceAsString	Return value
1	"P c"	
2	"P c, F c"	
3	"P c, F c, K c"	
4	"P c, F c, K c, P a"	
5	"P c, F c, K c, P a, F a"	
6	"P c, F c, K c, P a, F a, P a"	

- (b) If the above lock had a third challenge as below, then how would the trace table be updated? (Complete an updated trace table)

Not met: F a, P a

Count	SequenceAsString	Return value
1	"P c"	
2	"P c, F c"	
3	"P c, F c, K c"	
4	"P c, F c, K c, P a"	
5	"P c, F c, K c, P a, F a"	
6	"P c, F c, K c, P a, F a, P a"	
7	"P c, F c, K c, P a, F a, P a, F a"	
8	"P c, F c, K c, P a, F a, P a, F a, P a"	

- 5 Examine the `ProcessLockSolved` method in the `Breakthrough` class and the `Lock` class. Describe the methods called by that method.

- (a) When a new lock is set, if that lock has been solved before, it will be automatically replaced with a new lock the following turn (and treated as if the player just solved the first new lock) but reward the player for solving the first lock.
- (b) Describe the logical change you would make to the code (no need to write code, although you can) to ensure that this no longer happens.

- 6 Examine the `Shuffle` method of the `CardCollection` class. This method will make a new set of cards in the deck.

- (a) Explain how the effectiveness and efficiency of this algorithm decrease as the number of cards in the deck reduces.
- (b) Other than introducing a riffle shuffle, justify how you could improve the efficiency of the algorithm by describing any changes below.

COPYRIGHT
PROTECTED



- 7 ToolCards can be instantiated with either two or three arguments.
 - (a) Explain what happens in the case where a third argument is supplied where only two arguments are supplied.
 - (b) State the purpose of a constructor.
- 8 Examine the classes Card, ToolCard and DifficultyCard.
 - (a) Using evidence from these classes in the program, explain the difference between an abstract and a concrete class.
 - (b) Using evidence from the Card method, explain the difference between a static method and an attribute.
- 9 Find an example in the code for each of the following. Only write out the line/s of code.
 - (a) Inheritance
 - (b) Aggregation association
 - (c) A dynamic data structure
- 10 This question refers to the concept of polymorphism and how it is used.
 - (a) Choose and then write out one or more lines of the skeleton program that demonstrate polymorphism and justify why this is an example of polymorphism.
 - (b) Define the term 'polymorphism'.
- 11 A suggestion has been made to introduce a new AdvancedLock that challenges which is only revealed once the basic challenges have been completed. Explain the steps that you would take in order to do this, i.e. the logical change/addition and the reason for each step.
You are not required to implement this or to write any actual code.
- 12 Examine the Process method in the DifficultyCard class and the PlayCard and GetCardFromDeck methods of the Breakthrough class.

Using the scenario below:

Not met: P a, F a, K a
 Sequence: P a, F a
 Hand: P b, K a, F b, K c, P a

The player plays the 'K a' card to the sequence and then draws a difficulty card which requires them to either discard a card or five cards from the deck. The player discards the 'K c' from their hand, which is currently in position 4.

Explain what happens when the Process method is called under the above scenario, including specific references to the lines of code executed and in which order, and the values of variables, especially ChoiceAsInteger.

You will need to ensure that you look at the PlayCardToSequence and GetCardFromDeck methods in Breakthrough to be certain of the state of the Hand and Sequence when the DifficultyCard is drawn.

COPYRIGHT
PROTECTED



- 13 The terms 'HAND', 'SEQUENCE', 'DECK' and 'DISCARD' all appear at least once in the code and in some cases more than once. This is an example of hard-coded values which are difficult to maintain and understand and also make it more prone to errors.
- (a) Describe one method of avoiding hard-coding values that makes the code easier to maintain and understand.
 - (b) Explain why using hard-coded values makes the code more prone to errors and difficult to understand.
- 14 Exception handling is used in several places in the skeleton code; two of these are the use of file handling and the use of the `try` and `catch` statements.
- (a) Describe why it is important to always use exception handling when dealing with file handling.
 - (b) Give an example of another situation (not file handling) where exception handling is used (it doesn't have to be from the skeleton code) and explain why.
- 15 This question refers to the `PlayGame` method of the `Breakthrough` class. Explain the use of the private attribute `GameOver` in this method, specifying when it is set and why it is used as the condition for two iterative statements.

END OF QUESTIONS

COPYRIGHT
PROTECTED

BREAKTHROUGH

Programming Tasks

These questions require you to load the **Skeleton Program** and to make

Note that any alternative or additional code changes that you deemed appropriate – ensuring that it appears where in the Skeleton Program those changes



Task 1

DI

This question refers to the **PlayGame** method of the **Breakthrough** class.

The number of cards left in the deck should be printed out after the current

Test the changes you have made:

Run the game and play two turns, showing the number of cards in the deck

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the **PlayGame**
- SCREEN CAPTURE(S) showing the required test



INSPECTION COPY

COPYRIGHT
PROTECTED



Task 2

DI

This question refers to the `PlayGame`, `ProcessLockSolved` and `GetChoice` Breakthrough class and the creation of a new attribute (with accessor and mutator) `PeekUsed` in the `Lock` class.

Introduce a **(P)peek** option. This can be used once per lock, and allows a player to see the next three upcoming cards. There should be a new command in `PlayGame` the 'deck peek' is still available.

Create a new attribute in the `Lock` class called `PeekUsed`. Create accessor and mutator methods in the `Lock` class to update and read the `PeekUsed` attribute (get/set).

Update the `GetChoice()` method in the `Breakthrough` class to give the user a menu option. The menu option should only appear if the `PeekUsed` attribute is `False`.

Introduce an option to the menu in the `PlayGame()` method to accept 'P' as a command. This menu option should only appear if the `PeekUsed` attribute is `False`. Display the next three cards in the deck using the `GetCardDescriptionAt()` method. Set the `PeekUsed` attribute to `True` if the peek option has been chosen by the user.

When the player is given a new lock, set the `PeekUsed` attribute appropriately. When the player chooses the peek option again by invoking the `setPeekUsed()` method on the current lock, use the `ProcessLockSolved` method to set `PeekUsed` back to `False`.

Test the changes you have made:

Run the game and peek (new option), it works and then it's no longer displayed. Make sure it doesn't reappear even though the option isn't displayed. Solve a lock and now an option is available.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the `PlayGame` class
- PROGRAM SOURCE CODE showing changes made to the `ProcessLockSolved` class
- PROGRAM SOURCE CODE showing changes made to the `GetChoice` class
- PROGRAM SOURCE CODE for the new `PeekUsed` attribute in the `Lock` class
- SCREEN CAPTURE(S) showing the required functionality

INSPECTION COPY

**COPYRIGHT
PROTECTED**



Task 3

DI

This question refers to the `PlayCardToSequence` method of the `Breakthrough` class.

Under the rules of the game, a player cannot play two cards of the same type sequentially. Currently, there is no error message warning the player when they attempt to do this.

Modify the `PlayCardToSequence` method in the `Breakthrough` class to include an error message which tells the user that they cannot play two cards of the same type sequentially.

Use the `GetCardDescription` method to highlight to the user which card is being played and explain that it is the same as the type just played.



Test the changes you have made:

Run the game and show at least one turn played where the error does not occur. Then, show the new error message under the correct conditions of playing a duplicate card. Show that (1) the error message is displayed and (2) the card is not played.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the `PlayCardToSequence` method.
- SCREEN CAPTURE(S) showing the required functionality.



INSPECTION COPY

COPYRIGHT
PROTECTED



Task 4

DI

This question refers to the `PlayGame` and `GetChoice` methods and the `card` attribute, `mulliganUsed` of the `Breakthrough` class and new methods in the

Each player gets 1 'mulligan' per game where they can take all the cards in the discard pile and the sequence, put them together and shuffle up and deal a new card drawn (when repopulating the player's hand!) should be sent to the discard pile. The current lock including any new challenges will remain unchanged.

Create a new attribute in the `Breakthrough` class called `mulliganUsed` with the value `False`. If `mulliganUsed` is `False` then display an additional **(M)ulligan** option each time the `GetChoice` method is called. Once the mulligan has been used, set the `mulliganUsed` attribute to `True`. The **(M)ulligan** option is no longer displayed or usable.

Create two new methods in the `CardCollection` class, `getAllCards()` and `moveCards()`. `getAllCards()` should return a list of all cards in the collection. `moveCards()` should easily move all the cards from one hand to another without the need for iteration.

Test the changes you have made:

Run the game, solve one challenge, use mulligan, play one card to the sequence (then attempt to mulligan again despite no menu option).

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the `PlayGame` method
- PROGRAM SOURCE CODE showing changes made to the `Breakthrough` class
- PROGRAM SOURCE CODE showing changes made to the `GetChoice` method
- PROGRAM SOURCE CODE showing changes made to the `CardCollection` class
- SCREEN CAPTURE(S) showing the required test

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 5

D

This question refers to the `PlayGame` and `GetChoice` methods of the `Breakthrough` class.

The player will have a new option in `PlayGame` to **(Q)uit**, and for this they will calculate the score for each card remaining in the deck. Print out their final score as they choose to quit.

Note that the code should exit cleanly/nice, without using any `environment.Exit` or `goto` statements, although `break` and `return` are allowed of course.

Test the code you have made:

Play one turn of a game, choose quit.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the `PlayGame` method
- PROGRAM SOURCE CODE showing changes made to the `GetChoice` method
- SCREEN CAPTURE(S) showing the required test

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 6

Diffic

This question refers to the `GetCardFromDeck` method of the `Breakthrough` class and a new method, `getCardStats` in the `CardCollection` class.

Create a new method in the `CardCollection` class called `getCardStats` which will return the number of each type are left in the deck and calculates the percentage chance that the next card drawn will be of XYZ types).

When the player receives a difficulty card, use the `getCardStats` method to calculate the percentages and the `getNumberCards` method in the `CardCollection` class to display the following information: 'Currently in the Deck there is a X% chance of getting a File', 'Currently in the Deck there is a Y% chance of getting a Key' and 'Currently in the Deck there is a Z% chance of getting a Pick'. The player can then choose to 'draw a key or discard 5 cards from the deck'.

Currently in the Deck there is a X% chance of getting a File

Currently in the Deck there is a Y% chance of getting a Key

Currently in the Deck there is a Z% chance of getting a Pick

The percentages should be displayed to one decimal place.

Replace X, Y and Z with the appropriate values. Note that they will not necessarily add up to 100% because there are also difficulty cards in the deck.

Test the changes you have made:

Run the game until a difficulty card is drawn and show the printout of the statistics (after the hand and before asking which card to draw).

Evidence you need to provide:

- PROGRAM SOURCE CODE showing changes made to the `GetCardFromDeck` method
- PROGRAM SOURCE CODE showing changes made to the `CardCollection` class
- SCREEN CAPTURE(S) showing the required test

INSPECTION COPY

COPYRIGHT
PROTECTED



When a lock has been solved, create new multi-tool cards (one of each type available for the next lock) and the deck is reshuffled (as normal) by calling `AddMultiToolCardsToDeck` method again from the `ProcessLockSolved`

Play the game and show the use of at least one multi-tool card. The screen and sequence both before and after the multi-tool is played.

- PROGRAM SOURCE CODE showing changes made to the `SetupGame` and `PlayCardToSequence` methods of the `Breakthrough` class
- PROGRAM SOURCE CODE showing changes made in the `ToolCard` class to add an optional parameter to support a "multitool" card type and overridden methods
- PROGRAM SOURCE CODE showing changes made in the `CardCollection` class to assign a multi-tool card to the right toolkit
- PROGRAM SOURCE CODE showing changes made in the `Card` class to add a multi-tool card to the requested toolkit
- PROGRAM SOURCE CODE for the new `checkMultiCard` and `AddMultiCard` methods in the `Breakthrough` class.
- SCREEN CAPTURE(S) showing the required test.

INSPECTION COPY

**COPYRIGHT
PROTECTED**



Task 8

Diffic

This question refers to the `GetLockDetails` method of the `Lock` class and the `Breakthrough` class.

Challenges should be marked as “partially met” (rather than just ‘met’ or ‘not met’). Should the player then play a card which is not in the correct sequence, the challenge should be marked back down to ‘not met’ if the sequence no longer matches the challenge.

Modify the `GetLockDetails` method from `PlayGame` to pass in the sequence of cards.

Modify `GetLockDetails` so that if the challenge is not met then it checks to see if the sequence matches the first card of the challenge or the second last card of the challenge.

For challenges of three cards, only check the last two cards and it becomes a partial match if the sequence matches the first card of the challenge or the second last card of the challenge and the last card of the sequence matches the third card of the challenge.

In general, check N-1 cards where N is the number of cards in the challenge. Challenges of one card cannot be partially met. You only need to solve the challenges exactly.

Test the changes you have made:

Run the game and play one card to the sequence that doesn't match any of the three cards. Then play a card that matches the first card of the challenge. The screen showing this entire sequence is a partial match.

Then play another card to the sequence that matches the second card of the challenge. The screen showing this entire sequence is a partial match.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the `PlayGame` class
- PROGRAM SOURCE CODE showing changes made to the `GetLockDetails` method
- SCREEN CAPTURE(S) showing the required test

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 9

Diffic

This question refers to the `PlayGame` method and a new attribute of bonus to 20 in the `Breakthrough` class.

Introduce a bonus for solving locks using fewer cards.

Once the first card is played towards the sequence for a new lock, a counter every time a player makes a move (scanning or playing to the sequence) player to solve all the challenges in as few plays as possible.

Once a lock is solved (all the challenges), a player receives the bonus score additional points for solving a lock. If the player takes more than 20 moves set to 0. Print out a message confirming the bonus points that were awarded case). When the player is assigned a new lock, reset the bonus back to 20.

Test the changes you have made:

Run the game and play two locks, one solved in under 20 cards to show a bonus in over 20 cards to show a bonus score of 0.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the `PlayGame`
- PROGRAM SOURCE CODE showing the new `bonusCounter` attribute
- SCREEN CAPTURE(S) showing the required test

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 10

Diffic

This question refers to the `ProcessLockSolved`, `SetupGame` and `GetCard` as the creation of a new method, `AddGeniusCardToDeck` of the `Breakthrough` new virtual method, `Process` in the `Card` class and the creation of a new class.

Introduce a new 'Genius Card' which is added to the deck at the start of a lock challenge. There is a 25% chance of having a 'Genius Card' in a deck.

A player can choose to use a 'Genius Card' when they draw it to solve a challenge. The card will be discarded and then reshuffled into the deck from the discard pile.

Note that if a `GeniusCard` is drawn when filling up the hand it should be discarded and a message should be printed to this effect.

Create a method called `AddGeniusCardToDeck` which has a 25% chance of adding a `GeniusCard` to the deck. This should be called from `ProcessLockSolved` and `SetupGame`.

Create a new class for the `GeniusCard` which inherits `Card` with `CardType` set to `Genius`. Implement the `GetCardFromDeck` method of `Breakthrough` to ensure that the card is drawn.

Include a new virtual method `Process` in the `Card` class which is overridden by the `GeniusCard` class. The `Process` method in the `GeniusCard` should allow the user which challenges a lock with a 'Genius Card' on (including appropriate error handling) and then set the solved status of the lock to `true`.

Test the changes you have made:

Run the game and play until a 'Genius Card' is drawn, then choose yes and challenge in the current lock.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the `ProcessLockSolved` method
- PROGRAM SOURCE CODE showing changes made to the `SetupGame` method
- PROGRAM SOURCE CODE showing changes made to the `GetCardFromDeck` method
- PROGRAM SOURCE CODE for the new `GeniusCard` class
- PROGRAM SOURCE CODE for the new `AddGeniusCardToDeck` method
- PROGRAM SOURCE CODE for the new `Process` virtual method in the `Card` class
- SCREENSHOT(S) showing the required test

INSPECTION COPY

**COPYRIGHT
PROTECTED**



Task 11

Diff

This question refers to the addition of a new attribute in the Breakthrough PlayGame, GetChoice, GetCardFromDeck and GetCardChoice methods as well as the creation of a new method, getCardPurchasePosition and new getCardTypeCount and getCardPosition in the CardCollection class and a new method in the Difficulty class.

Introduce the concept of 'Buying a tool' from the deck. Add a new attribute to the Breakthrough class, credits (contains the number of credits the player currently has in game, the player starts with 10 credits). When a player has played a card to the sequence, if they have at least 2 credits remaining, they should be given the option to 'buy' a card from the Deck rather than have it selected for them. Players can 'buy' a 'Key' card or a 'Pick' cards at the cost of 2 credits each. When the player chooses to buy a card, they are prompted with the following menu (items which have 0 availability should not be shown):

1. F a (1 available)
2. F b (1 available)
3. F c (1 available)
4. P a (1 available)
5. P b (1 available)
6. P c (1 available)
7. K a (1 available)
8. K b (1 available)
9. K c (1 available)
10. No Tool (buy nothing)

Note: the actual number available is 1

Note: keys (items 7-9) should only be shown if the player has at least 3 credits left. All menu items should have numbers given above even if the availability is 0 e.g. item 10 should always be shown even if the player changed their mind

The new getCardPurchasePosition method should display the above menu and return the position of the card to buy. Take into account the requirements for card types where no cards are available or not enough funds are available to purchase a card. The new method should use getCardTypeCount and getCardPosition into the CardCollection class. The parameter for getCardTypeCount is the card type you want to know the count for and returns the count of each of the different card types. The parameter for getCardPosition is the position of a card in the Cards list, also returning an integer. This can be implemented using LINQ. Ensure that the method has appropriate error handling to validate if a card is available before purchasing the card.

Purchasing multiple cards can create a Hand which is larger than 5 cards. Modify the GetChoice method and the Process method for a Difficulty card to take into account a player's hand size and the positions which the player can choose from.

Modify the GetChoice method to give the user an additional option to (B)uy a card. The correct menu options are displayed depending on if the user has enough credits to buy a card. If the user has enough credits, the menu option (B)uy a card is shown. If the user does not have enough credits, the menu option is not shown. Again, ensure the method has appropriate error handling to validate if a card is available before purchasing the card. The player from choosing to (B)uy a card even when the menu option is not shown.

Test the changes you have made

1. Run the game and play any card to the sequence, then choose 'b' to buy a tool. Select any tool listed as available, play it to the sequence and discard it when asked. Show all the output produced including the tool card being added to the player's hand each time.
2. Continue playing the game and buying tools until you have spent a total of 10 credits (5 pick/file and 2 keys) and then show the printed list of tools available to buy.

INSPECTION COPY

COPYRIGHT
PROTECTED



Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the PlayGame.getCardPurchasePosition methods
- PROGRAM SOURCE CODE showing changes made to the GetCard.GetCardFromDeck methods
- PROGRAM SOURCE CODE showing changes made to the Process class.
- PROGRAM SOURCE CODE for the new Credits attribute
- PROGRAM SOURCE CODE for the new getCardTypeCount and
- SCREEN CAPTURE(S) showing the required test

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 12

DI

This question refers to `PlayCardToSequence` and `CheckIfLockChallenge`. In the `Breakthrough` class, a new private attribute called `advancedMet` in the `Card` class (with associated accessor and mutator methods), new `getAdvancedMetCondition` and `removeAdvancedMet` methods in the `Lock` class, modification of the `CheckIfLockChallenge` method in the `Card` class and a new `getCollectionAsString` method in the `CardCollection` class.

Create an 'Advanced' mode where the challenge requires three or more cards. If the challenge is solved more than one card is used to solve it from the sequence of cards. The previous card on the sequence, which could then possibly be used in solving the next challenge.

For example, if the sequence contains: Fa, Kc, Pb and the current challenge is:

```

CURRENT LOCK:
-----
Not met:      P b, K b, F b

SEQUENCE:
-----
| Fa | Kc | Pb |
-----

HAND:
-----
| Fb | Pc | Pa | Kc | Kb |
-----
    
```

Suppose you play Kb and Fb to the sequence. This will solve the current challenge by extending the sequence to: Fa, Kc, Pb, Kb, Fb.

It will be confirmed that Fa, Kc and the Pb, Kb and Fb cards from the challenge that was just solved will be added back to the deck and it is shuffled ready for them to be used again.

```

HAND:
-----
| Fb | Pc | Pa | Kc | Kb |
-----

SEQUENCE:
-----
| Fa | Kc | Pb | Kb |
-----

(Discard inspect, (U)se cards)
Enter a number between 1 and 5
(Discard or (P)lay?:) p

A challenge on the lock has been met.
Cards from Advanced Search have been added.
Lock has been solved. Your score is: 26

CURRENT LOCK:
-----
Challenge met: P b, K b, F b

HAND:
-----
| Fb | Pc | Pa | Kc | Kb |
-----

SEQUENCE:
-----
| Fa | Kc |
-----

(Discard inspect, (U)se cards)
    
```

Test the changes you have made:

Run the game and restart until you get a lock with at least one challenge of three cards. Play until you solve the single card challenge and then play the three card challenge. The screen capture(s) should show the Lock, Sequence and Hand card to solve the three card challenge and the Lock and Sequence after you solve it.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the `PlayCardToSequence` method
- PROGRAM SOURCE CODE showing changes made to the `CheckIfLockChallenge` method
- PROGRAM SOURCE CODE showing changes made to the `CheckIfCardChallenge` method
- PROGRAM SOURCE CODE showing the new `advancedMet` attribute
- PROGRAM SOURCE CODE showing the new `getAdvancedMetCondition` method
- PROGRAM SOURCE CODE showing the new `removeAdvancedMet` method
- PROGRAM SOURCE CODE showing the new `getCollectionAsString` method
- SCREEN CAPTURE(S) showing the required test

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 13

DI

This question refers to `PlayGame`, `SetupGame` and `GetChoice` methods and the `SaveGame` method in the `Breakthrough` class. It also requires the creation of `getLocksSolvedForSaving` methods in the `Lock` class and `getCollectionForSaving` methods in the `CardCollection` class.

The `PlayGame` menu should have a `(S)` option which will save the game and allow it to be reloaded (from the main menu when you first start the game).

In order to write the format of the save game file, you will have to include the `LoadGame` method of the `Breakthrough` class.

Print out a suitable message stating whether the game was saved successfully.

When the player chooses to **(L)oad** a game when the program is first run, prompt them for a file name in the `SetupGame` method and then include appropriate checks to ensure the file name is valid.

Test the changes you have made:

1. Take a copy of the `game1.txt` file and rename it `backup.txt`.
2. Run the game until you get a lock with at least two challenges. Solve the challenges and save the game as `'game1.txt'` (it shouldn't prompt you). Load the game and ensure that it has been correctly restored.
3. Restore the original `game1.txt` from `backup.txt`.

Evidence you need to provide:

- PROGRAM SOURCE CODE showing changes made to the `PlayGame` method
- PROGRAM SOURCE CODE showing changes made to the `SetupGame` method
- PROGRAM SOURCE CODE showing changes made to the `GetChoice` method
- PROGRAM SOURCE CODE for the new `SaveGame` method
- PROGRAM SOURCE CODE for the new `getLocksForSaving` method
- PROGRAM SOURCE CODE for the new `getLocksSolvedForSaving` method
- PROGRAM SOURCE CODE for the new `getCollectionForSaving` method
- SCREEN CAPTURE(S) showing the required test

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 14

DI

This question refers to `PlayGame` and `PlayCardToSequence` methods and attribute, `BonusPool`, in the `Breakthrough` class. It also requires the creation of `partComplete`, in the `Lock` class, which takes `Sequence` as a parameter.

Introduce a bonus for playing consecutive cards towards a challenge that solves a challenge. A card played in a row that goes towards solving a challenge will add 5 to the bonus pool. If a card is not part of the challenge, the bonus pool will be added to the score for that card.

For example, if the bonus pool is 0 and a player plays a card towards a challenge, the score for that card will be added to the score along with their normal score and the bonus pool is increased by 5. If a player does anything except play another correct card towards challenge 1, then the bonus pool will be reset to 0; otherwise they will get the score for the card played as normal, plus the bonus pool will be increased to 10 and so on.

Test the changes you have made:

Run the game and keep discarding until you have all three cards required to solve it one card after another; continue playing and play a card to a challenge sequence that is not part of the challenge.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the `PlayGame` method
- PROGRAM SOURCE CODE showing changes made to the `PlayCardToSequence` method
- PROGRAM SOURCE CODE for the new `BonusPool` attribute
- PROGRAM SOURCE CODE for the new `partComplete` method
- SCREEN CAPTURE(S) showing the required test

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 15

Difficulty

This question refers to the `ProcessLockSolved`, `GetCardFromDeck` and `CheckIfPlayerHasLost` methods, and to the creation of new private `GenerateSolvableLock` method and `FinalLock`, in the `Breakthrough` class. It also requires the creation of a new `IsSolvable` method in the `CardCollection` class as well as a new public `IsSolvable` method in the `Deck` and `Hand` as parameters.

EXTRA FILE NEEDED: `game2.txt`

Every lock generated must be solvable based on the cards left in the deck, and must not exhaust the deck. If the lock cannot be solved, then choose a new random lock in a row (without a suitable lock being found) then display a message 'Final Lock' and generate a lock with two challenges that can be solved.

Once those challenges are solved, there should be a message from `CheckIfPlayerHasLost` instead of saying the player lost, prints out 'You have solved the final lock.'

When approaching this task you should ignore the effect of `Difficulty` cards. You should check that the `Deck` and `Hand` combined contain the requisite number of cards for a lock.

The attribute `FinalLock` should be set to 0 at the start and then set to 1 in the final lock is set. When `CheckIfPlayerHasLost` runs, it should set `FinalLock` to 1 (indicating that the final turn is played). If `FinalLock = 1` and there are no cards left in the deck, the player doesn't lose until all the cards from the deck and hand are gone.

Test the changes you have made:

1. Change the game to load the file `game2.txt` instead of `game1.txt` and run the load game.
2. Play the game until the message 'Final Lock' is displayed, then solve the final turn.

Evidence that you need to provide:

- PROGRAM SOURCE CODE showing changes made to the `ProcessLockSolved` method
- PROGRAM SOURCE CODE showing changes made to the `CheckIfPlayerHasLost` method
- PROGRAM SOURCE CODE showing changes made to the `GetCardFromDeck` method
- PROGRAM SOURCE CODE for the new `GenerateSolvableLock` method
- PROGRAM SOURCE CODE for the new `IsSolvable` method
- PROGRAM SOURCE CODE for the new `getAllCards` method
- PROGRAM SOURCE CODE for the new `FinalLock` attribute
- SCREEN CAPTURE(S) showing the required test

INSPECTION COPY

COPYRIGHT
PROTECTED



BREAKTHROUGH

Possible Additional Programming

1. Create an extra toolkit (e.g. 'd') and add a lock involving this to the lock.
2. Introduce a *Swiss Army Knife* card which can be used as any single toolkit.
3. Add a further ability to allow a player, at the start of a turn, to (U)ndo everything – including the score, sequence, hand and discard pile to the start of the previous turn. There should be one undo available per turn possible to use it on the first turn of a new lock.
4. Add a *High Scores* file and ability to view this from a main menu.
5. Add levels so that different locks have different challenges which will depend on the current level. This could be linked to (11 – complexity number of toolkits used, e.g. 2, 3 or 4).
6. Add a *Mighty Hammer* card that can smash (solve) the current lock, discard your hand and play it later.
7. Introduce a user-defined locks option. This generates a rough pseudo-lock where one player can choose a lock sequence and another has to try to unlock it (original game of *Mastermind*). A user-defined lock must follow the original rules: at least two must be files, and at least one must be a pick.
8. Introduce a second type of lock: 'Maths Lock', whereby the player chooses the way they are now used to solve new maths locks. This will involve each card called 'number'. The value of 'number' is displayed in addition to the pick (e.g. 7) currently displayed. Cards can be used for their mathematical attribute. For example, if a lock contains four files – each with a value of 5, that gives a total lock value of 20. The player needs to play a sequence of cards that gives a total value of 20. For example, if the player plays two picks, each with a value of 10, then the lock will open. These new 'Maths Locks' are solved only using the cards and are independent of the tool type and tool kit.
9. Receive a bonus of 50 if you quit and the current challenge could not be solved and deck as they are currently).
10. Add an *Autoplay* mode which shows a computer simulation of the game.
11. Design a formula to compute a complexity value for a lock.
12. Validation of card to play (with exception handling) for choosing which card to play in response to a difficulty card.
13. Validation on entry of choice (or an entry) so that the player can only enter a valid choice.
14. Be able to sacrifice a card (removed from the game) in order to challenge a lock.
15. Examine the game1.txt file (or game) closely and draw a flow diagram of the game. The player will make to move from the 'saved state' to 'end of the game' by looking at the data in the game1.txt file rather than playing the game.

INSPECTION COPY

COPYRIGHT
PROTECTED



BREAKTHROUGH!

Theory Questions (Mark Scheme)



Question	Suggested Solution	Total Marks	Marking Guidance
1			
(a)	e.g. <code>AccumalScore // CardScore</code>	1 mark	A: Similar names with meaning to explain the score. R: Spaces in names. I: case.
(b)	<p>4 marks: ignoring a return value is not good practice [1]... One alternative would be to create a new method called <code>MoveCardWithScore [1]...</code> which takes in current player Score as a parameter and returns the updated Score [1]... and to remove the return value from the current method [1].</p> <p>Examples of answers worth less than full marks:</p> <p>3 marks: make the return type void [1]... and move the logic to the place where the card is played to the sequence [1]... which is the only time the score is needed [1].</p> <p>2 marks: remove the scoring [1]... and create a separate <code>getScore()</code> method [1].</p> <p>1 mark: always check the return value as ignoring it is bad practice [1].</p>	4 marks	<p>A: any reasonable suggestion.</p> <p>A: answers without passing Score as a parameter and dealing with the extra score as per now.</p> <p>A: answers where there is a scoring method in <code>CardCollection</code> which 'knows' whether to score or not.</p> <p>A: passing score in by reference and having a new attribute on <code>CardCollection</code> to indicate if a card added/played should affect the score.</p>
2	(a) The sequence only allows cards to be added to the end and taken from the same end which is a LIFO structure [1]... and a stack is a LIFO structure that would be appropriate [1]... For the discard pile, sometimes you need to peek at the whole stack but generally just play cards to the top and then the whole pile is shuffled back in [1]... a stack could be suitable for this.	4 marks	<p>1 mark for each point (MAX. 4)</p> <p>A: stack for discard pile.</p> <p>R: queue for either.</p>

COPYRIGHT
PROTECTED



INSPECTION COPY

Question	Suggested Solution	Total Marks	Marking Guidance																																	
3	<p>(a) Splitting the deck into two halves (or using pointers to do the same thing) [1]... Choosing a number of cards from one half and add them to the combined deck (A: circular deck with counting solutions) [1]... Choosing a number of cards from the other half and add them to the combined deck [1]... Using a random number of cards (5, A, 1 to 5) [1]... Taking cards from the bottom of the split decks rather than the top. Repeating until the deck is fully combined [1].</p> <p>(b) Most algorithms will have a space complexity twice that of the random shuffle that existed before, for storing the deck split and a new merged/combined deck.</p>	6 marks	1 mark for each point																																	
4	<p>(a)</p> <table><thead><tr><th>Count</th><th>SequenceAsString</th><th>Return value</th></tr></thead><tbody><tr><td></td><td>" "</td><td></td></tr><tr><td>5</td><td>"P a"</td><td></td></tr><tr><td>4</td><td>" , P a"</td><td></td></tr><tr><td></td><td>"F a, F a"</td><td></td></tr><tr><td>3</td><td>" , F a, F a"</td><td></td></tr><tr><td></td><td>"P a, F a, F a"</td><td>True</td></tr></tbody></table> <p>(b)</p> <table><thead><tr><th>Count</th><th>SequenceAsString</th><th>Return value</th></tr></thead><tbody><tr><td></td><td>" "</td><td></td></tr><tr><td>5</td><td>"P a"</td><td></td></tr><tr><td>1</td><td>" , P a"</td><td></td></tr></tbody></table>	Count	SequenceAsString	Return value		" "		5	"P a"		4	" , P a"			"F a, F a"		3	" , F a, F a"			"P a, F a, F a"	True	Count	SequenceAsString	Return value		" "		5	"P a"		1	" , P a"		5 marks	<p>A: any version of the idea for 1 mark.</p> <p>A: circular deck solutions with space complexity equal to same as the storage for the deck as long as they are explained properly.</p> <p>1 mark for the Count column (I: spaces)</p> <p>1 mark for a final return value of True</p> <p>1 mark for the first value in the SequenceAsString column</p> <p>1 mark for the last value in the SequenceAsString column</p> <p>1 mark for the correct middle values in the SequenceAsString column (between the first and last value)</p> <p>DPT: -1 only for a missing space</p>
Count	SequenceAsString	Return value																																		
	" "																																			
5	"P a"																																			
4	" , P a"																																			
	"F a, F a"																																			
3	" , F a, F a"																																			
	"P a, F a, F a"	True																																		
Count	SequenceAsString	Return value																																		
	" "																																			
5	"P a"																																			
1	" , P a"																																			
	<p>(b)</p>	3 marks	1 mark for each column																																	
			DPT: -1 only for a missing space (note that this is across parts (a) and (b) combined, total of -1 for a missing space across the two parts)																																	

COPYRIGHT
PROTECTED



INSPECTION COPY

Question	Suggested Solution	Total Marks	Marking Guidance
5	<p>(b)</p> <p>Either: Once a lock is solved, remove it from the list of locks so that it cannot be selected again when a new random lock is chosen.</p> <p>Or: Once a lock is solved, iterate through all the challenges and set the attribute <code>Met</code> for each to <code>False</code>.</p>	2 marks	<p>2 marks available for either solution as long as the details are clearly explained, otherwise award 1 mark</p> <p>A: any reasonable solution including the idea of moving LockSolved into the Lock class and checking it when choosing a new lock.</p>
6	<p>(a)</p> <p>As the number of cards in the deck gets smaller then there will be more swaps than possible arrangements of the cards, which makes the additional swaps redundant and inefficient [1]... For example, with 2 cards there are only two combinations of cards with 6 only 720 but there are still 10000 swaps [1] ... There is also the chance of the algorithm causing an exception with 0 or 1 cards but there is no check for this [1].</p>	3 marks	<p>Award 1 mark for each point</p> <p>A: any expression of the idea that 10000 is more swaps than you need, which makes the extra ones unnecessary for the first mark.</p> <p>A: any reference to example to decreasing combinations for the second mark.</p>
	<p>(b)</p> <p>The number of swaps could be a measure of deck size or could be set to a lower threshold when the deck is small [1].</p> <p>This would be done by using a selection statement and setting a swaps variable to a lower value if the deck is short or to 10000 as now if the deck is large [1].</p>	2 marks	<p>Award 1 mark for each point</p> <p>A: any expression of each concept for each mark.</p> <p>A: any other reasonable suggestions that would give 10000 for large decks and a much lower number for small decks.</p>
7	<p>(a)</p> <p>When two arguments are supplied they are used to set the <code>ToolType</code> and <code>Kit</code> respectively [1]... The <code>CardNumber</code> is set using the next available <code>CardNumber</code> from the class/static variable <code>NextCardNumber</code> by the parent constructor when it is called [1]... In the case of a third argument being supplied, the parent constructor is not called and the <code>CardNumber</code> is set by the value of the parameter [1].</p>	3 marks	<p>1 mark for each point (MAX. 3)</p>

**COPYRIGHT
PROTECTED**



INSPECTION COPY

Question	Suggested Solution	Total Marks	Marking Guidance
8	(b) A class or static variable has the same value for every object and is changed in them all when it changes in one [1]... An attribute may start off the same but has a different value in each object and if changed in one will not affect the others [1].	2 marks	1 mark for each point
9	(a) In Inheritance is where a child gains the attributes and behaviour of its parent. Possible Example: <pre> class ToolCard : Card { protected string ToolType; protected string Kit; public ToolCard(string t, string k) : base() { ToolType = t; Kit = k; Score(); } } </pre>	1 mark	A: one of the words with similar meanings.
(b)	Aggregation association is where one class contains another class but their lifespans are not linked, they can function independently. Possible Example: <pre> private void CreateStandardDeck() { Card NewCard; for (int Count = 1; Count <= 5; Count++) { NewCard = new ToolCard("P", "3"); } } </pre>	1 mark	R: answers that do not refer to lifespan in some way.

**COPYRIGHT
PROTECTED**



INSPECTION COPY

Question	Suggested Solution	Total Marks	Marking Guidance
9	<p>(c)</p> <p>A data structure for which the memory usage will shrink and grow over time according to the storage needs.</p> <p>Example: <pre> Lock protected List<Challenge> Challenges = new List<>() { <Challenge>() }; public virtual void AddChallenge(List<string> condition) { Challenge C = new Challenge(); C.SetCondition(condition); Challenges.Add(C); } </pre> </p>	1 mark	<p>For any one possible answer: 1 mark for each point and 1 mark for the code</p> <p>A: any example of code related to inheritance for 1 mark provided the explanation gains at least 1 mark.</p> <p>R: code only with no explanation.</p>
10	<p>(a)</p> <p>Solution #1</p> <pre> MoveCard(Deck discard, Deck.GetCardNumberAt(0)); ... because CardCollection treats DifficultyCards and ToolCards [1]. All cards [1]... which is an example of polymorphism but when the statement resolves all methods will actually execute on the child's object class [1]. </pre> <p>Solution #2</p> <pre> public override void Process(CardCollection deck, CardCollection discard, CardCollection hand, CardCollection sequence, Lock currentLock, string choice, int cardChoice) ... because this overrides the Process method [1]... in the parent Card class [1]... which will mean that DifficultyCards can be treated as cards but behave as themselves [1]. </pre>	4 marks	<p>For any one possible answer: 1 mark for each point and 1 mark for the code</p> <p>A: any example of code related to inheritance for 1 mark provided the explanation gains at least 1 mark.</p> <p>R: code only with no explanation.</p>

COPYRIGHT
PROTECTED



INSPECTION COPY

Question	Suggested Solution	Total Marks	Marking Guidance
11	Create a new <code>AdvancedLock</code> class [1]... that inherits from <code>Lock</code> [1]... and override the <code>GetLockSolved</code> method [1]... so that when the basic challenges are solved it unlocks the final challenge and returns <code>False</code> instead of <code>True</code> [1]... this will then mean that it can refer to a [1] attribute such as <code>SecretUnlocked</code> [1]... when calling <code>GetLockSolved</code> the next time.	6 marks	1 mark for each point
12	The 'K' card is in position 2 meaning that when <code>GetCardFromDeck</code> is called, <code>CardChoice</code> will be [1]... The card in position 2 will have already been moved from the hand to the deck as <code>MoveCard</code> is called by <code>PlayCardToSequence</code> before <code>GetCardFromDeck</code> [1]... The hand is displayed the player enters 3 to choose 'K' as the key to discard because it is now in position 3 [1]... When <code>Process</code> is called <code>CardChoice</code> is 2 and <code>Choice</code> is 3 [1]... <code>Choice</code> is converted from a string to an integer, range checked (it is successfully and stored in <code>ChoiceAsInteger</code> [1]... as $3 \geq 2$, <code>ChoiceAsInteger</code> is decremented by 1 [1]... and as <code>ChoiceAsInteger</code> is still > 0 it is decremented by 1 again to give 1 [1]... The selection statement checks against the card in the hand at index <code>ChoiceAsInteger</code> (which is 1) and that card is now 'K' so the condition is <code>False</code> [1]... and 5 cards are ordered from the deck to the discard pile [1].	8 marks	1 mark for each point (MAX. 8)
13	(a) Using a constant [1]... which would be declared once at the top of program and could be changed in that single place [1].	2 marks	1 mark for each point
	(b) It is possible to misplace values [1]... update the wrong values [1]... or make values [1]... constants have no identifier, making a value easier to understand [1].	2 marks	1 mark for each point (MAX. 2) A: opposite points.
14	(a) File handling can always generate exceptions [1]... because files could be locked [1]... removed/unavailable/inaccessible [1]	2 marks	1 mark for each point (MAX. 2)
	(b) Converting an inputted string to an integer [1]... because if it fails you want to catch the error and ask the user to input again [1].	2 marks	1 mark for each point A: any example of validation.

COPYRIGHT
PROTECTED



INSPECTION COPY

BREAKTHROUGH

Programming Tasks (Mark Sch)

Task 1

Coding

- Printing out the number of cards left in the deck correctly each turn [1 mark]

Example Solution

```
Console.WriteLine(Hand.GetCardDisplay());  
//CHANGE  
Console.WriteLine("There are currently: " + Convert.ToString(Deck.GetCardsLeft()));  
//END CHANGE  
MenuChoice = GetChoice();
```

Testing:

- Printing out the number of cards left correctly after SEQUENCE and HAND [1 mark]

```
HAND:  
-----  
| P b | K c | P c | P c | P b |  
-----  
  
SEQUENCE: empty  
  
There are currently: 32 cards left in the deck  
(D)iscard or (U)se card:> u  
Enter a number between 1 and 5 to specify card to use:> 1  
(D)iscard or (P)lay?:> p  
  
Current score: 1  
  
CURRENT LOCK  
-----  
Not met:      K a  
Not met:      K b  
Not met:      K c  
  
SEQUENCE:  
-----  
| P b |  
-----  
  
HAND:  
-----  
| K c | P c | P b | P c |  
-----  
  
There are currently: 32 cards left in the deck  
(D)iscard inspect, (U)se card:>
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 2

Coding:

- Changing GetChoice to show Peek (even if it doesn't check GetPeekUsed) [1 mark]
- Changing PlayGame to accept 'P' and printing out the next three cards in the deck [1 mark]
- Adding the peekUsed attribute with get/set methods to Lock [1 mark]
- Invoking the new methods in the Lock class from ProcessLockSolved [1 mark]

Example Solution

Changes to GetChoice

```
private string GetChoice()
{
    Console.WriteLine("");
    //CHANGE
    if (CurrentLock.getpeekUsed())
    {
        Console.WriteLine("(D)iscard inspect, (U)se card:> ");
    }
    else
    {
        Console.WriteLine("(D)iscard inspect, (U)se card, (P)EEK next time");
    }
    //END CHANGE
    string Choice = Console.ReadLine().ToUpper();
    return Choice;
}
```

Changes to PlayGame

```
else if (DiscardOrPlay == "P")
    PlayCardToSequence(CardChoice);
    break;
}
//CHANGE
case "P":
{
    if (CurrentLock.getpeekUsed())
    {
        Console.WriteLine("The next three cards in the deck are:");
        for (int i = 0; i < 19; i++)
        {
            Console.Write("-");
        }
        Console.WriteLine();
        for (int i = 0; i < 3; i++)
        {
            Console.Write("| " + Deck.GetCardDescriptionAt(i) + " ");
        }
        Console.WriteLine("|");
        for (int i = 0; i < 19; i++)
        {
            Console.Write("-");
        }
        CurrentLock.setPeekUsed(true);
    }
    else
    {
        Console.WriteLine("You have already used the peek facility");
    }
    break;
}
//END CHANGE
}
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Changes to ProcessLockSolved

```
Deck.Shuffle();
CurrentLock = GetRandomLock();
//CHANGE
CurrentLock.setPeekUsed(false);
//END CHANGE
}
```

Changes to Lock

```
class Lock
{
    protected List<Challenge> Challenges = new List<Challenge>();
    //CHANGE
    private bool peekUsed;
    //END CHANGE

    public virtual void AddChallenge(List<string> condition)
    {
        Challenge C = new Challenge();
        C.SetCondition(condition);
        Challenges.Add(C);
    }
    //CHANGE
    public bool getpeekUsed()
    {
        return peekUsed;
    }

    public void setPeekUsed(bool updatePeekUsed)
    {
        peekUsed = updatePeekUsed;
    }
    //END CHANGE
}
```

Testing:

- Peek option, works correctly and then disappears [1 mark] →

```
Current score: 1
CURRENT LOCK
-----
Not set:      P a, F a, K a

SEQUENCE:
-----
[ P a ]

HAND:
-----
[ K a | F c | K c | K b | P b ]

(D)iscard inspect, (U)se card, (P)ick the next three cards in the deck
-----
[ K a | P b | D1f ]
Current score: 1

CURRENT LOCK
-----
P a, F a, K a

SEQUENCE:
-----
[ P a ]

HAND:
-----
[ K a | F c | K c | K b | P b ]

(D)iscard inspect, (U)se card:>
```

INSPECTION COPY

COPYRIGHT
PROTECTED



- Peek reappears for the next lock and works and then disappears and doesn't work

```
(D)iscard inspect, (U)se card:> u
Enter a number between 1 and 5 to specify card to use:> 1
(D)iscard or (P)lay?:> p

A challenge on the lock has been met.

Lock has been solved. Your score is now: 21

Current score: 21

CURRENT LOCK
-----
Not met: P a, F
Not met: K a, F

SEQUENCE:
-----
| P a | F a | K a |
-----

HAND:
-----
| P b | P b | K b | P c | K a |
-----

(D)iscard inspect, (U)se card, (P)EEK next three cards in the Deck:> p
The next three cards in the deck are:
-----
| F c | K a | P c |
-----

Current score: 21

CURRENT LOCK
-----
Not met: P
Not met: K

SEQUENCE:
-----
| P a | F a | K a |
-----

HAND:
-----
| P b | P b | K b | P c | K a |
-----

(D)iscard inspect, (U)se card:> p
You have already used the peek facility for this lock.

Current score: 21

CURRENT LOCK
-----
Not met: P a, P a
Not met: b
```

```
SEQUENCE:
-----
| P a | F a | K a |
-----

HAND:
-----
| P b | P b | K b | P c | K a |
-----

(D)iscard inspect, (U)se card:> p
You have already used the peek facility for this lock.

Current score: 21

CURRENT LOCK
-----
Not met: P a, P a
Not met: b
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 3

Coding:

- Checking for correct condition to print out the error [1 mark]
- Printing out a sensible error message with the card or tool type that is in error [1 mark]

Example Solution

Changes to PlayCardToSequence

```
if (Hand.GetCardDescriptionAt(cardChoice - 1)[0] !=  
Sequence.GetCardDescriptionAt(Sequence.GetNumberOfCards() - 1)[0])  
{  
    Score = CheckScore(Hand, Sequence, Hand.GetCardNumberAt(cardChoice - 1),  
        Hand.GetCardDescriptionAt(cardChoice));  
}  
//CHANGE  
else  
{  
    Console.WriteLine("You have just played a " + Hand.GetCardDescriptionAt(cardChoice - 1) +  
        " card. You need to pick a different card");  
}  
//END CHANGE  
}
```

Testing:

- Showing the error message and the hand and sequence afterwards confirming that the card was neither played nor discarded [1 mark] →

```
| P c | P c | K b | F c |  
-----  
(D)iscard inspect, (U)se  
Enter a number between 1 and 4:  
(D)iscard or (P)lay?> 5  
You have just played a 5 of Clubs  
Current score: 1  
CURRENT LOCK  
Not met: F c, P c  
SEQUENCE:  
-----  
| P c |  
-----  
HAND:  
-----  
| P c | K b | F c | P b |  
-----  
(D)iscard inspect, (U)se  
Enter a number between 1 and 4:  
(D)iscard or (P)lay?> 5  
You have just played a 5 of Clubs  
Current score: 1  
CURRENT LOCK  
Not met: F c, P c  
SEQUENCE:  
-----  
| P c |  
-----  
HAND:  
-----  
| P c | K b | F c | P b |  
-----  
(D)iscard inspect, (U)se
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 4

Coding:

- Printing out the correct message only when a mulligan is available [1 mark]
- Adding the mulliganUsed attribute to Breakthrough and initialising it to False [1 mark]
- Implementing the mulligan to add all the cards from the player's hand, the discard deck (this can be done through iteration or through the addAllCards method) [1 mark]
- Shuffling up and dealing again (and discarding any extra cards drawn) [1 mark]

Example Solution

Changes to GetChoice

```
private static string GetChoice()
{
    Console.WriteLine();
    //CHANGE
    if (mulliganUsed)
    {
        Console.Write("(D)iscard inspect, (U)se card:> ");
    }
    else
    {
        Console.Write("(D)iscard inspect, (U)se card, (M)ulligan:> ");
    }
    //END CHANGE

    string Choice = Console.ReadLine().ToUpper();
}
```

Changes to Breakthrough

```
private Lock CurrentLock;
private bool LockSolved;

//CHANGE
private bool mulliganUsed = false;
//END CHANGE

public Breakthrough()
```

Changes to PlayGame

```
        else if (DiscardOrPlay == "P")
            PlayCardToSequence(CardChoice);
        break;
    }
    //CHANGE
    case "M":
    {
        if (!mulliganUsed)
        {
            Deck.addAllCards(Hand.getAllCards());
            Deck.addAllCards(Sequence.getAllCards());
            Deck.addAllCards(Discard.getAllCards());
            Deck.Shuffle();
            Hand = new CardCollection("HAND");
            Sequence = new CardCollection("SEQUENCE");
            Discard = new CardCollection("DISCARD");
            int position = 0;
            int count = 0;
            while (count < 5)
            {
                if (Deck.GetCardDescriptionAt(position) != "Dif")
                {

```

INSPECTION COPY

COPYRIGHT
PROTECTED




```

        MoveCard(Deck, Hand, Deck.GetCardNumberAt(position, count) - 1);
        count += 1;
    }
    else
    {
        position += 1;
    }
}
mulliganUsed = true;
}
else
{
    Console.WriteLine("Mulligan option has already been used");
}
break;
}
//END CHANGE

```

Changes to CardCollection

```

public CardCollection(string n)
{
    Name = n;
}

//CHANGE
public List<Card> getallCards()
{
    return Cards;
}

public void addAllCards(List<Card> cardsFromHand)
{
    Cards.AddRange(cardsFromHand);
}
//END CHANGE
public string GetName()

```

```

Current score: 9
CURRENT LOCK
-----
Not met:      P
Challenge met: P

SEQUENCE:
-----
| P c | F c | P
-----

HAND:
-----
| K c | P a | F c
-----

(D)iscard inspect
Current score: 9
CURRENT LOCK
-----
Not met:      P
Challenge met: P

SEQUENCE: empty
HAND:
-----
| F c | F b | P
-----

(D)iscard inspect

```

Testing:

- Showing a mulligan being used after solving a challenge [1 mark] →

**COPYRIGHT
PROTECTED**



- Showing an attempt to use the mulligan again failing [1 mark] ↓



INSPECTION COPY

COPYRIGHT
PROTECTED



Task 5

Coding:

- Printing out quit as a menu option and including it in the selection statement in PlayGame
- Cleanly exiting the main game loop in PlayGame without using environment.exit() mechanism and successfully ending the program [1 mark]

Example Solution

Changes to GetChoice

```
Console.WriteLine();  
//CHANGE  
Console.WriteLine(D, "Discard inspect, (U)se card, (Q)uit:> ");  
//END CHANGE  
string Choice = Console.ReadLine().ToUpper();
```

Changes to PlayGame

```
GameOver = false;  
//CHANGE  
bool quitOverride = false;  
CurrentLock = new Lock();  
SetupGame();  
while (!GameOver && !quitOverride)  
//END CHANGE  
{  
    ...
```

```
        else if (DiscardOrPlay == "p")  
            PlayCardToSequence(CardChoice);  
        break;  
    }  
    //CHANGE  
    case "Q":  
    {  
        Console.WriteLine("Quit Option Selected. Current score  
        Convert.ToString(Score) + " Points");  
        Console.WriteLine("Bonus for quitting early: " +  
        Convert.ToString(Deck.GetNumberOfCards()) + " Points");  
        Console.WriteLine("Final Score: " + Convert.ToString(Score));  
        GameOver = true;  
        quitOverride = true;  
        break;  
    }  
    //END CHANGE  
}  
if (CurrentLock.GetLockSolved())
```

Testing:

- Printing out a final score of 33 (if a Pick was played), 34 (if a File was played) or 35 (if a Key was played) [1 mark] →

```
(D)iscard inspect, (U)se  
Enter a number between 1  
(D)iscard or (P)lay?> p  
  
Current score: 2  
  
CURRENT LOCK  
-----  
Not met:      P b, K b,  
  
Score: 33  
-----  
[ F b ]  
  
HAND:  
-----  
[ F a | F a | K a | F a ]  
-----  
  
(D)iscard inspect, (U)se  
Quit Option Selected. Curr  
Bonus for quitting early:  
Final Score: 34
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 6

Coding:

- Adding the functionality to the CardCollection class to be able to total up the number of Keys [1 mark]
- Correctly calculating the percentage chances of a Key, Pick or File being selected iteratively or using LINQ [1 mark]
- Adjusting the GetCardFromDeck method that will print out the percentage of each correctly (even if not to 1 decimal place). Note that correctly means dividing the number of cards in the deck. [1 mark]

Example Solution

Changes to the CardCollection class

```
public CardCollection(string n)
{
    Name = n;
}

//CHANGE
public int getCardStats(string letterStartsWith)
{
    return Cards.Count(x => x.GetDescription().StartsWith(letterStartsWith));
}
//END CHANGE

public string GetName()
```

Changes to GetCardFromDeck

```
Console.WriteLine("Difficulty encountered!");
Console.WriteLine(Hand.GetCardFromDeck());
//CHANGE
Console.WriteLine("With this you need to either lose a key or a pick");
Console.WriteLine();
Console.WriteLine("Before your choose, here are the Card Count statistics");
double percentageChanceOfFile = Math.Round((((float)Deck.getCardStats('F')) / Deck.GetNumberOfCards()) * 100, 1);
double percentageChanceOfKey = Math.Round((((float)Deck.getCardStats('K')) / Deck.GetNumberOfCards()) * 100, 1);
double percentageChanceOfPick = Math.Round((((float)Deck.getCardStats('P')) / Deck.GetNumberOfCards()) * 100, 1);
Console.WriteLine("Currently in deck there is a: " + Convert.ToString(percentageChanceOfFile) + "% chance of getting a File");
Console.WriteLine("Currently in deck there is a: " + Convert.ToString(percentageChanceOfKey) + "% chance of getting a Key");
Console.WriteLine("Currently in deck there is a: " + Convert.ToString(percentageChanceOfPick) + "% chance of getting a Pick");
Console.WriteLine();
//END CHANGE
Console.Write("(enter 1-5 to specify position of key) or (D)iscard file or (P)ick a card: ");
```

Note: This task could alternatively be solved using iteration rather than LINQ

INSPECTION COPY

COPYRIGHT
PROTECTED



Testing:

- Showing the percentage of at least one tool (even if incorrect) to one decimal place which card the player would like to select or whether to discard from the deck.

Note that the percentages are unlikely to match the ones below. [1 mark] ↓

```
(D)iscard inspect, (U)se card:> u
Enter a number between 1 and 5 to specify card to use:> 1
(D)iscard or (P)lay:> d

Difficulty encountered!

HAND:
-----
| P c | F |
-----

To deal with this you need to either lose a key

Before your choose, here are the Card Count stats for the current deck:
Currently in deck there is a: 22.6% chance of getting a File
Currently in deck there is a: 25.8% chance of getting a Key
Currently in deck there is a: 38.7% chance of getting a Pick

(enter 1-5 to specify position of key) or (D)iscard five cards from the deck:>

Current score: 0

CURRENT LOCK
-----
Not met:      P b, K b, F b
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 7

Coding:

- Adding one multi-tool of each kind to the deck at creation time [1 mark]
- Adding one multi-tool of each kind to the deck whenever a lock is solved [1 mark]
- Adding the virtual method updateMultiToolkit to the Card class which is over [1 mark]
- Changing PlayCardToSequence to ask for which tool the player would like when new method checkMultiCard [1 mark]
- Calling assignToolkitAt for the card and toolkit from checkMultiCard

Example Solution

Changes to SequenceGame

```
AddDifficultyCardsToDeck();
//CHANGE
AddMultiToolCardsToDeck();
//END CHANGE
Deck.Shuffle();
CurrentLock = GetRandomLock();
```

Changes to ProcessLockSolved

```
while (Discard.GetNumberOfCards() > 0)
{
    MoveCard(Discard, Deck, Discard.GetCardNumberAt(0));
}
//CHANGE
AddMultiToolCardsToDeck();    Adds new multi-tool into the deck
//END CHANGE
Deck.Shuffle();
CurrentLock = GetRandomLock();
}
```

Changes to PlayCardToSequence

```
if (Sequence.GetNumberOfCards() > 0)
{
    //CHANGE
    checkMultiCard(cardChoice);
    //END CHANGE
    if (Hand.GetCardDescriptionAt(cardChoice - 1)[0] !=
        Sequence.GetCardDescriptionAt(Sequence.GetNumberOfCards() - 1)[0])
    {
        Score += MoveCard(Hand, Sequence, Hand.GetCardNumberAt(cardChoice - 1));
        GetCardFromDeck(cardChoice);
    }
}
else
{
    //CHANGE
    checkMultiCard(cardChoice);
    //END CHANGE
    Score += MoveCard(Hand, Sequence, Hand.GetCardNumberAt(cardChoice - 1));
    GetCardFromDeck(cardChoice);
}
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Creation of checkMultiCard in Breakthrough

```
//CHANGE
private void checkMultiCard(int cardChoice)
{
    if (Hand.GetCardDescriptionAt(cardChoice - 1)[2] == 'm')
    {
        bool validToolKitChoice = false;
        string toolKitChoice = "m";
        while (!validToolKitChoice)
        {
            switch (Hand.GetCardDescriptionAt(cardChoice - 1)[0])
            {
                case 'P':
                    Console.WriteLine("Pick Multi-tool card played. like it to be assigned to, A, B or C?");
                    break;
                case 'F':
                    Console.WriteLine("File Multi-tool card played. like it to be assigned to, A, B or C?");
                    break;
                case 'K':
                    Console.WriteLine("Key Multi-tool card played. it to be assigned to, A, B or C?");
                    break;
            }
            toolKitChoice = Console.ReadLine();
            if (toolKitChoice == "A" || toolKitChoice == "B" || toolKitChoice == "C")
            {
                validToolKitChoice = true;
            }
            else
            {
                Console.WriteLine("not a valid input - please try again");
            }
        }
        Hand.AssignToolKitAt(cardChoice - 1, toolKitChoice.ToLower());
    }
}
//END CHANGE
```

Creation of AddMultiToolCardsToDeck in Breakthrough

```
//CHANGE
private void AddMultiToolCardsToDeck()
{
    Deck.AddCard(new ToolCard("P", "m", true));
    Deck.AddCard(new ToolCard("F", "m", true));
    Deck.AddCard(new ToolCard("K", "m", true));
}
//END CHANGE
```

INSPECTION COPY

**COPYRIGHT
PROTECTED**



Changes to Card

```

class ToolCard : Card
{
    protected string ToolType;
    protected string Kit;
    //CHANGE
    private bool multiToolCard;

    public ToolCard(string t, string k, bool multi = false) : base()
    //END CHANGE
    {
        ToolType = t;
        Kit = k;
        //CHANGE
        multiToolCard = multi;
        //END CHANGE
        SetScore();
    }

    public ToolCard(string t, string k, int cardNo)
    {
        ToolType = t;
        Kit = k;
        CardNumber = cardNo;
        SetScore();
    }

    //CHANGE
    public override void updateMultiToolKit(string toolkit)
    {
        Kit = toolkit;
    }
    //END CHANGE

    private void SetScore()
    {
        switch (ToolType)
        {
            case "K":
            {
                Score = 3;
                break;
            }
            case "F":
            {
                Score = 2;
                break;
            }
            case "P":
            {
                Score = 1;
                break;
            }
        }
    }
    //CHANGE
    if (multiToolCard)
    {
        Score = 0;
    }
    //END CHANGE
}

public override string GetDescription()

```

INSPECTION COPY

**COPYRIGHT
PROTECTED**



Changes to CardCollection

```
//CHANGE
public void assignToolKitAt(int x, string toolkit)
{
    Cards[x].updateMultiToolKit(toolkit);
}
//END CHANGE
```

Creation of virtual method in Card

```
//CHANGE
public virtual void updateMultiToolKit(string toolkit)
{
}
//END CHANGE
```

Testing:

- Showing the sequence updated with the card played of the toolkit chosen [1]

```
Not met:      P a, F a, P a
Not met:      P b, F b, P b
Challenge met: K c

SEQUENCE:

-----
| K c | P a |
-----

HAND:

-----
| K a | P m | P b | K b | F m |
-----

(D)iscard inspect, (U)se card to use:
Enter a number between 0 and 1 to specify card to use:> 5
(D)iscard on collection:
File MultiToolKit.cs played. Which toolkit would you like it to be assigned to?
A
Current score: 9

CURRENT LOCK

-----
Not met:      P a, F a, P a
Not met:      P b, F b, P b
Challenge met: K c

SEQUENCE:

-----
| K c | P a | F a |
-----

HAND:

-----
| K a | P m | P b | K b | F b |
-----
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 8

- Changing PlayGame to pass in the argument for the sequence to GetLockDetails
GetLockDetails to accept the new parameter [1 mark]
- Changing GetLockDetails to match a single card on the sequence to the first
message to partially met, also to not crash when there is an empty sequence [1 mark]
- Changing GetLockDetails to generate a partially met message when only the
the first card of a challenge [1 mark]
- Changing GetLockDetails to generate a partially met message when the last
the first two cards of a challenge. [1 mark]

Example Solution

Changes to GetLockDetails

```
//CHANGE
public virtual string GetLockDetails(CardCollection sequence)
//END CHANGE
{
    string LockDetails = Environment.NewLine + "CURRENT LOCK" + Environment.NewLine;
    foreach (var C in Challenges)
    {
        if (C.GetMet())
        {
            LockDetails += "Challenge met: ";
        }
        else
        //CHANGE
        {
            int sequenceLength = sequence.GetNumberOfCards() - 1;
            List<string> condition = C.GetCondition();
            if (condition.Count() == 3)
            {
                if (sequenceLength > 0 && condition[1] ==
                    sequence.GetCardDescriptionAt(sequenceLength) && condition[2] ==
                    sequence.GetCardDescriptionAt(sequenceLength - 1))
                {
                    LockDetails += "Partially Met: ";
                }
                else if (sequenceLength >= 0 && condition[0] ==
                    sequence.GetCardDescriptionAt(sequenceLength))
                {
                    LockDetails += "Partially Met: ";
                }
                else
                {
                    LockDetails += "Not met: ";
                }
            }
            else if (condition.Count() == 2)
            {
                if (sequenceLength >= 0 && condition[0] ==
                    sequence.GetCardDescriptionAt(sequenceLength))
                {
                    LockDetails += "Partially Met: ";
                }
                else
                {
                    LockDetails += "Not met: ";
                }
            }
            else
            {
                LockDetails += "Not met: ";
            }
        }
    }
    //END CHANGE
}
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Changes to PlayGame

```
while (!LockSolved && !GameOver)
{
    Console.WriteLine();
    Console.WriteLine("Current score: " + Score);
    //CHANGE
    Console.WriteLine(CurrentLock.GetLockDetails(Sequence));
    //END CHANGE
    Console.WriteLine(Sequence.GetCardDisplay(), "\n");
}
```

Testing:

- First card played to see if it doesn't generate partially met (if it doesn't match)

```

CURRENT LOCK
Not met: P b, K b, F b

SEQUENCE:
| F b |

HAND:
| K b | P a | F a | K c | P b |

(D)iscard inspect, (U)se card:> u
Enter a number between 1 and 5 to specify card to use:> 5
(D)iscard or (P)lay?> p

Current score: 3

CURRENT LOCK
Partially Met: P b, K b, F b

SEQUENCE:
| F b | P b |

HAND:
| K b | P a | F a | K c | F b |

(D)iscard inspect, (U)se card:>

```

```

Partially Met: P b, K b, F b

SEQUENCE:
| F b | P b | F a | K c |

HAND:
| K b | P a | F b | P a |

(D)iscard inspect, (U)se card:>
Enter a number between 1 and 5 to specify card to use:> 5
(D)iscard or (P)lay?> p

Current score: 12

CURRENT LOCK
Partially Met: P b, K b, F b

SEQUENCE:
| F b | P b | F a | K c |

HAND:
| P a | F b | P a | P a |

(D)iscard inspect, (U)se card:>

```

- Last two cards on the sequence matching first two of a challenge generates partially met



**COPYRIGHT
PROTECTED**



Task 9

Coding:

- Adding a variable for bonusCounter and initialising it to 20 for each new lock [1 mark]
- Subtracting 1 to the variable each time a card is played or discarded [1 mark]
- Awarding the correct bonus once the lock is solved (including 0 if over 20 cards) and setting the variable to 20 [1 mark]

Example Solution

Changes to Breakthrough

```
private bool LockSolved;

//CHANGE
private int bonusCounter = 20;
//END CHANGE
```

Changes to PlayGame

```
while (!LockSolved && !GameOver)
{
    Console.WriteLine();
    Console.WriteLine("Current score: " + Score);
    //CHANGE
    Console.WriteLine("The current bonus score is: " + Convert.ToString(bonusCounter));
    //END CHANGE
    Console.WriteLine(CurrentLock.GetLockDetails());
    Console.WriteLine(Sequence.GetCardDisplay());
    Console.WriteLine(Hand.GetCardDisplay());

    MenuChoice = GetChoice();
    switch (MenuChoice)
    {
        case "D":
            Console.WriteLine(Discard.GetCardDisplay());
            break;
        case "U":
            {
                int CardChoice = GetCardChoice();
                string DiscardOrPlay = GetDiscardOrPlayChoice();
                //CHANGE
                bonusCounter -= 1;
                if (bonusCounter < 0)
                {
                    Console.WriteLine("You have played too many cards. Bonus is 0.");
                    bonusCounter = 0;
                }
                //END CHANGE
                if (DiscardOrPlay == "D")
                {
                    MoveCardFromDiscard(Hand.GetCardNumberAt(CardChoice), Discard);
                }
                else if (DiscardOrPlay == "P")
                {
                    PlayCardToSequence(CardChoice);
                }
                break;
            }
    }

    if (CurrentLock.GetLockSolved())
    {
        LockSolved = true;
    }
}
```

INSPECTION COPY

COPYRIGHT
PROTECTED



```

        ProcessLockSolved();
        //CHANGE
        if (bonusCounter != 0)
        {
            Console.WriteLine("You solved this lock in less than 20
            + Convert.ToString(bonusCounter) + " points");
        }
        else
        {
            Console.WriteLine("You took more than 20 moves to solve
            awarded on this occasion.");
        }

        Score = bonusCounter;
        Console.WriteLine("Your score is: " + Convert.ToString(Score));
        bonusCounter = 20;
        //END CHANGE
    }
}
GameOver = CheckIfPlayerHasLost();

```

Testing:

- Solving a lock in under 20 cards and getting the correct bonus (which doesn't have [1 mark]) ↓

SEQUENCE:

| P c | F c | P c | K b | P a | F a |

HAND:

| P a | P a | P c | F a |

(D)iscard inspect, (U)se card:> u

Enter a number between 1 and 5 to specify card to use:> 1

(D)iscard or (P)lay?> p

A challenge on the lock has been met.

Lock has been solved. Your score is now: 31

You solved this lock in less than 20 moves. Bonus awarded is: 4 points

Your score is: 35

Current score: 35

The current bonus score is: 20

CURRENT LOCK

Not met: K a

Not met: K b

Not met: K c

**COPYRIGHT
PROTECTED**



- Solving a lock in over 20 cards and getting 0 bonus [1 mark] ↓

```
(D)iscard inspect, (U)se card:> u
Enter a number between 1 and 5 to specify card to use:> 2
(D)iscard or (P)lay?:> d
You have played too many cards, setting bonus to 0

Current score: 3
The current bonus score is: 0

CURRENT LOCK
-----
Not met:      F C, P C, F C

SEQUENCE
-----
| F C | P C |
-----

HAND:
-----
| P a | F c | F a | K a | P c |
-----

(D)iscard inspect, (U)se card:> u
Enter a number between 1 and 5 to specify card to use:> 2
(D)iscard or (P)lay?:> p
You have played too many cards, setting bonus to 0

A challenge on the lock has been met.

Lock has been solved. Your score is now: 20
You took more than 20 moves to solve the lock - no bonus awarded on this
Your score is: 20

Current score: 20
The current bonus score is: 0

CURRENT LOCK
-----
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 10

Coding:

- Creating a AddGeniusCardToDeck method to have a 25% chance of adding a GeniusCard to the Deck during SetupGame [1 mark]
- Modifying ProcessLockSolved to invoke the AddGeniusCardToDeck method
- Creating a GeniusCard class that inherits from Card, has a constructor which sets the CardType to "Genius"
- Asking the user to enter a challenge number or discard a genius card is drawn
- Processing the GeniusCard correctly to solve the challenge chosen [1 mark]
- Processing the GeniusCard correctly to discard it [1 mark]
- Handling the discarding of a GeniusCard correctly if drawn while refilling the hand and print a message [1 mark]

Example Solution

Creation of AddGeniusCardToDeck

```
//CHANGE
private void AddGeniusCardToDeck()
{
    if (RNoGen.Next(4) == 1)
    {
        Deck.AddCard(new GeniusCard());
    }
    //To test this feature you may want to include a loop to put lots
    //of cards into the deck to increase the chances of a card coming
    //
    //for (int Count = 1; Count <= 30; Count++)
    //{
        Deck.AddCard(new GeniusCard());
    //}
}
//END CHANGE
```

Creation of overloaded "Process" method in Card

```
public virtual void Process(CardCollection deck, CardCollection discard,
    CardCollection hand, CardCollection sequence, Lock currentLock,
    string choice, int cardChoice)
{
}

//CHANGE
public virtual void Process(Lock currentLock) //overload
{
}
//END CHANGE
```

Creation of GeniusCard

```
//CHANGE
class GeniusCard : Card
{
    protected string CardType;

    public GeniusCard()
    {
        CardType = "Gen";
    }

    public GeniusCard(int cardNo)
    {
        CardType = "Gen";
    }
}
```

INSPECTION COPY

COPYRIGHT
PROTECTED



```

    CardNumber = cardNo;
}

public override string GetDescription()
{
    return CardType;
}

public override void Process(Lock CurrentLock)
{
    if (CurrentLock.GetNumberOfChallenges() == 1)
    {
        Console.WriteLine("Since you only have 1 lock challenge  
will automatically be used for that lock. Press any key  
to continue.");
        Console.ReadLine();
        CurrentLock.SetChallengeMet(0, true);
    }
    else
    {
        int challengeChoice = -1;
        while (challengeChoice < 1 || challengeChoice >
            CurrentLock.GetNumberOfChallenges())
        {
            Console.WriteLine("Which of the " +
                Convert.ToString(CurrentLock.GetNumberOfChallenges()) +
                " challenges would you like to use the Genius card on?");
            if (int.TryParse(Console.ReadLine(), out challengeChoice))
            {
                if (challengeChoice < 1 || challengeChoice >
                    CurrentLock.GetNumberOfChallenges())
                {
                    Console.WriteLine("That is not a valid choice");
                }
                else
                {
                    CurrentLock.SetChallengeMet(challengeChoice, true);
                    Console.WriteLine("That challenge has been resolved. Make  
sure you want to resolve it? Y/N");
                    string confirm = Console.ReadLine().Trim();
                    if (confirm == "y")
                    {
                        CurrentLock.SetChallengeMet(challengeChoice, true);
                    }
                    else
                    {
                        challengeChoice = -1;
                    }
                }
            }
            else
            {
                Console.WriteLine("That is not a valid choice");
            }
        }
    }
}

//END CHANGE

```

**COPYRIGHT
PROTECTED**



Changes to SetupGame

```
for (int Count = 1; Count <= 5; Count++)
{
    MoveCard(Deck, Hand, Deck.GetCardNumberAt(0));
}
AddDifficultyCardsToDeck();
//CHANGE
AddGeniusCardToDeck();
//END CHANGE
Deck.Shuffle();
```

Changes to ProcessLockSolved

```
while (Discard.GetNumberOfCards() > 0)
{
    MoveCard(Discard, Deck, Discard.GetCardNumberAt(0));
}
//CHANGE
AddGeniusCardToDeck();
//END CHANGE
Deck.Shuffle();
```

Changes to GetCardFromDeck

```
private void GetCardFromDeck(int cardChoice)
{
    if (Deck.GetNumberOfCards() > 0)
    {
        if (Deck.GetCardDescriptionAt(0) == "Dif")
        {
            Card CurrentCard = Deck.RemoveCard(Deck.GetCardNumberAt(0));
            Console.WriteLine();
            Console.WriteLine("Difficulty encountered!");
            Console.WriteLine(Hand.GetCardDisplay());
            Console.WriteLine("To deal with this you need to either lose a card or add a card to specify position of key) of the deck: ");
            string userChoice = Console.ReadLine();
            Console.WriteLine();
            Discard.AddCard(CurrentCard);
            CurrentCard.Process(Deck, Discard, Hand, Sequence, CurrentLock);
        }
        //CHANGE
        if (Deck.GetCardDescriptionAt(0) == "Gen")
        {
            Card CurrentCard = Deck.RemoveCard(Deck.GetCardNumberAt(0));
            Console.WriteLine();
            Console.WriteLine("Genius Card encountered!");
            string userChoice = "";
            while (userChoice.ToLower() != "d" && userChoice.ToLower() != "s")
            {
                Console.WriteLine("You can use this to (S)olve ONE or (D)iscard it so that it can come up after reshuffling");
                userChoice = Console.ReadLine();
                if (userChoice.ToLower() != "s" && userChoice.ToLower() != "d")
                {
                    Console.WriteLine("That is not a valid option");
                }
            }
            if (userChoice.ToLower() == "d")
            {
                Discard.AddCard(CurrentCard);
            }
            else
            {
                Discard.AddCard(CurrentCard);
                CurrentCard.Process(CurrentLock);
            }
        }
    }
}
```

INSPECTION COPY

**COPYRIGHT
PROTECTED**



```

    }
    Console.WriteLine();
}
//END CHANGE
}
while (Hand.GetNumberOfCards() < 5 && Deck.GetNumberOfCards() > 0)
{
    if (Deck.GetCardDescriptionAt(0) == "Dif")
    {
        MoveCard(Deck, Discard, Deck.GetCardNumberAt(0));
        Console.WriteLine("A Dumb Card was discarded from the hand.");
    }
    //CHANGE
    else if (Deck.GetCardDescriptionAt(0) == "Gen")
    {
        MoveCard(Deck, Discard, Deck.GetCardNumberAt(0));
        Console.WriteLine("A Genius Card was discarded from the deck.");
    }
    //END CHANGE
    else
    {
        MoveCard(Deck, Hand, Deck.GetCardNumberAt(0));
    }
}
if (Deck.GetNumberOfCards() == 0 && Hand.GetNumberOfCards() < 5)
{
    GameOver = true;
}
}

```

Testing:

- Using a GeniusCard successfully in a game

```

Current score: 6
CURRENT LOCK
-----
Not met: F a, P a
Not met: F b, P b
Not met: F c

SEQUENCE:
-----
| F c | P b | K b |
-----

HAND:
-----
| F a | K a | F c | K c | P c |
-----

(D)iscard inspect, (U)se card:> u
Enter a number between 1 and 5 to specify card to use:> 2
(D)iscard or (P)lay?> d

Genius Card encountered!
You can use this to (S)olve ONE of the challenges or (D)iscard it so that it can be
used again. Which of the 3 challenges do you want to use the Genius card on?
1
2
3

A Genius Card was discarded from the deck when refilling the hand.
A Genius Card was discarded from the deck when refilling the hand.

Current score: 6
CURRENT LOCK
-----
Challenge met: P a, F a, P a
Not met: P b, F b, P b
Not met: K c

```

INSPECTION COPY

COPYRIGHT
PROTECTED



- Discarding a GeniusCard successfully [1 mark] ↓

```

Current score: 7

CURRENT LOCK
-----
Challenge met: P a, F a, P a
Not met:      P b, F b, P b
Not met:      K c

SEQUENCE:
-----
| F c | P b | K b | P a |
-----

HAND:
-----
| F a | K a | P b | P a | F b |
-----

(D)iscard inspect, (U)se card:> u
Enter a number between 1 and 5 to specify card to use:> 1
(D)iscard or (P)lay?> p

Genius Card encountered!
You can use this to (S)olve ONE of the lock challenges or (D)iscard it so that it
d

Current score: 9

CURRENT LOCK
-----
Challenge met: P a, F a, P a
Not met:      P b, F b, P b
Not met:      K c

SEQUENCE:
-----
| F c | P b |
-----

HAND:
-----
| K a | P b | P a | F b | P c |
-----

(D)iscard inspect, (U)se card:>

```

INSPECTION COPY

COPYRIGHT
PROTECTED

INSPECTION COPY



Task 11

Coding:

- Adding the Credits attribute and initialising it to 10 [1 mark]
- Asking whether the player would like to buy a tool only when they have played at least 2 credits left [1 mark]
- Ensuring that keys are not listed if player has <3 credits remaining (even if they did)
- Adding a tool card of the correct type and toolkit to the player's hand [1 mark]
- Removing the tool card from the deck at the correct position [1 mark]
- Deducting the correct number of credits for buying a card [1 mark]
- Printing a list showing the number of cash tool available and not printing tools where
- Printing option 10 correctly at the end of the menu, once and once only [1 mark]
- Correctly updating the new possible size of the hand collection when playing a card [1 mark]
- Correctly getting the appropriate card type and card position for helping with the

Example Solution

Adding the Credits attribute

```
//CHANGE
private int Credits;
//CHANGE
public Breakthrough()
...
    ...
    Score = 0;
    //CHANGE
    Credits = 10;
    //END CHANGE
    LoadLocks();
}
```

Changes to PlayGame

```
Console.WriteLine(Sequence.ToString());
//CHANGE
Console.WriteLine("Current funds are: " + Convert.ToString(Credits));
//END CHANGE
MenuChoice = GetChoice();
```

```
...
else if (DiscardOrPlay == "p")
    PlayCardToSequence(CardChoice);
break;
}
//CHANGE
case "B":
{
    int positionToPurchase = getCardPurchasePosition();
    if (positionToPurchase >= 0)
    {
        Card cardToPurchase = Deck.RemoveCard(Deck.GetCardName(positionToPurchase));
        Hand.AddCard(cardToPurchase);
        if (cardToPurchase.GetDescription()[0] == 'K')
        {
            Credits -= 1;
        }
        else
        {
            Credits -= 2;
        }
    }
    break;
}
//END CHANGE
}
```

INSPECTION COPY

COPYRIGHT
PROTECTED



New getCardPurchasePosition method in Breakthrough

```
//CHANGE
private int getCardPurchasePosition()
{
    bool validChoice = false;
    string userCardChoice = "";
    int cardChosen = 0;
    List<int> menuCounter = new List<int>();
    string[] cardTypes = new string[] { "F", "Fc", "Pa", "Pb" };
    while (!validChoice)
    {
        for (int i = 0; i < cardTypes.Length; i++)
        {
            if (cardTypes[i][0] == 'K')
            {
                if (Deck.getCardTypeCount(cardTypes[i]) > 0 && Credit > 0)
                {
                    Console.WriteLine(Convert.ToString(i + 1) + ": " +
                        Convert.ToString(Deck.getCardTypeCount(cardTypes[i])) + " cards");
                    menuCounter.Add(i + 1);
                }
            }
            else if (Deck.getCardTypeCount(cardTypes[i]) > 0)
            {
                Console.WriteLine(Convert.ToString(i + 1) + ": " +
                    Convert.ToString(Deck.getCardTypeCount(cardTypes[i])) + " cards");
                menuCounter.Add(i + 1);
            }
        }
        Console.WriteLine("10: No Tool (buy nothing)");
        Console.WriteLine("Please select your choice from the menu");
        userCardChoice = Console.ReadLine();
        if (!int.TryParse(userCardChoice, out cardChosen))
        {
            Console.WriteLine("That is not a valid input, please try again");
        }
        else if (cardChosen == 10)
        {
            return -1;
        }
        else if (cardChosen < menuCounter.Min() || cardChosen > menuCounter.Max())
        {
            Console.WriteLine("That is not a valid input, please try again");
        }
        else
        {
            validChoice = true;
        }
    }
    return Deck.getCardPosition(cardTypes[cardChosen - 1]);
}
//END CHANGE
```

Change to GetCardChoice

```
int Value;
do
{
    //CHANGE
    Console.Write("Enter a number between 1 and " + Hand.GetNumberOfCards() +
        " to use:> ");
    //END CHANGE
    Choice = Console.ReadLine();
}
```

INSPECTION COPY

**COPYRIGHT
PROTECTED**



New getCardTypeCount and getCardPosition methods in CardCollection

```
//CHANGE
public int getCardTypeCount(string c)
{
    return Cards.Count(n => n.GetDescription()[0] == c[0] && n.GetDe
}

public int getCardPosition(string cardType)
{
    string cardTest = cardType[0] + " " + cardType[1];
    int position = Cards.FindIndex(n => n.GetDescription() == cardT
    return position;
}
//END CHANGE
```

Change to GetChoice

```
private string GetChoice()
{
    Console.WriteLine();
    //CHANGE
    string Choice = "";
    if (Sequence.GetNumberOfCards() > 0 || Discard.GetNumberOfCards() > 0)
    {
        if (Credits > 1)
        {
            Console.Write("(D)iscard inspect, (U)se card, (B)uy a card: ");
        }
        else
        {
            Console.Write("(D)iscard inspect, (U)se card:> ");
            Choice = Console.ReadLine().ToUpper();
            if (Choice != "B")
            {
                return Choice;
            }
        }
    }
    else
    {
        Console.Write("(D)iscard inspect, (U)se card:> ");
        Choice = Console.ReadLine().ToUpper();
        if (Choice != "B")
        {
            return Choice;
        }
        else
        {
            return "";
        }
    }
    Choice = Console.ReadLine().ToUpper();
    //END CHANGE
    return Choice;
}
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Change to Process override in Difficulty

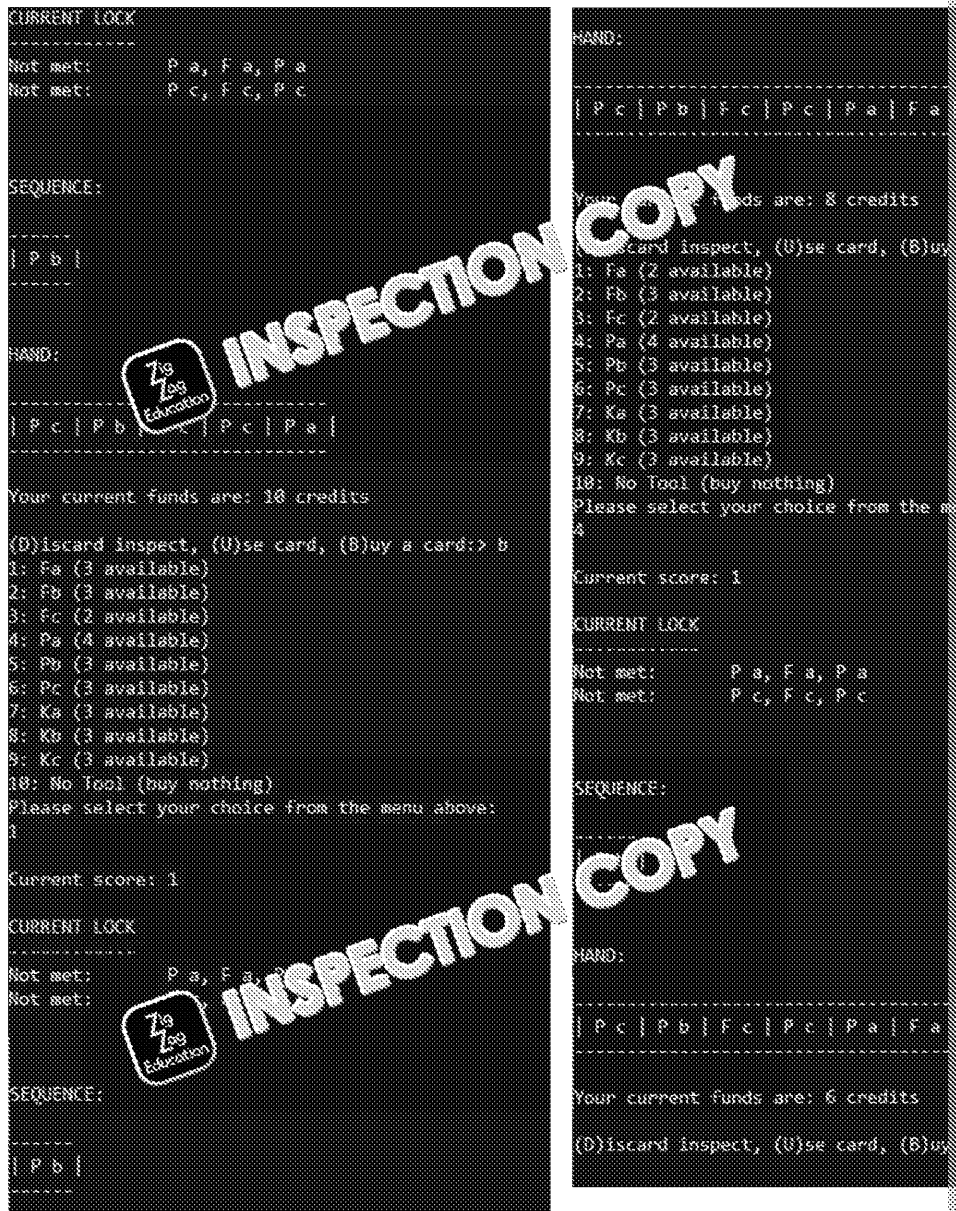
```
if (int.TryParse(choice, out ChoiceAsInteger))
{
    //CHANGE
    if (ChoiceAsInteger >= 1 && ChoiceAsInteger <= hand.GetNumberOfCards()
    {
        ChoiceAsInteger -- 1;          //Code removed from here as it
        //END CHANGE
        if (hand.GetCardDescriptionAt(ChoiceAsInteger)[0] == 'K')
```

Change to GetCardFromDeck

```
Console.WriteLine(hand.GetCardFromDeck().ToString());
Console.WriteLine("You have " + hand.Credits + " with this you need to either lose a key ");
//CHANGE
Console.WriteLine("enter 1- " + Convert.ToString(hand.GetNumberOfCards()) + " to buy a key (or) (D)iscard five cards from the deck:> ");
//END CHANGE
string Choice = Console.ReadLine();
Console.WriteLine();
```

Testing:

- Buying two tools [1 mark] ↓



Left Screenshot:

```
CURRENT LOCK
-----
Not met: P a, F a, P a
Not met: P c, F c, P c

SEQUENCE:
-----
[ P b ]

HAND:
-----
[ P c | P b | F c | P c | P a | F a ]

Your current funds are: 18 credits

(D)iscard inspect, (U)se card, (B)uy a card: b
1: Fa (3 available)
2: Fb (3 available)
3: Fc (3 available)
4: Pa (4 available)
5: Pb (3 available)
6: Pc (3 available)
7: Ka (3 available)
8: Kb (3 available)
9: Kc (3 available)
10: No Tool (buy nothing)
Please select your choice from the menu above:
1

Current score: 1

CURRENT LOCK
-----
Not met: P a, F a, P a
Not met: P c, F c, P c

SEQUENCE:
-----
[ P b ]
```

Right Screenshot:

```
HAND:
-----
[ P c | P b | F c | P c | P a | F a ]

Your current funds are: 8 credits

(D)iscard inspect, (U)se card, (B)uy a card: b
1: Fa (2 available)
2: Fb (3 available)
3: Fc (2 available)
4: Pa (4 available)
5: Pb (3 available)
6: Pc (3 available)
7: Ka (3 available)
8: Kb (3 available)
9: Kc (3 available)
10: No Tool (buy nothing)
Please select your choice from the menu above:
2

Current score: 1

CURRENT LOCK
-----
Not met: P a, F a, P a
Not met: P c, F c, P c

SEQUENCE:
-----
[ P b ]
```

INSPECTION COPY

COPYRIGHT
PROTECTED



- Trying to buy a tool with 2 credits left [1 mark] ↓

Current score: 2

CURRENT LOCK

Not met: P b, K b, F b

SEQUENCE:

| F a |

HAND:

| F a | P b | F b | K c | P c | F a | F c | F b | F b |

Your current funds are: 2 credits

(D)iscard inspect, (U)se card, (B)uy a card:> b

3: Fc (2 available)

4: Pa (5 available)

5: Pb (4 available)

6: Pc (4 available)

10: No Tool (buy nothing)

Please select your choice from the menu above:

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 12

Coding:

- Iterate through the sequence to remove solution cards and place them back into the Deck
- New list attribute in CheckIfLockChallengeMet to store the sequence for testing
- New Boolean attribute in the Challenge class (with associated accessor and mutator) to be "Advanced Met" [1 mark]
- New method in the Lock class for testing if a challenge has been met in advance
- Changes to CheckIfConditionMet to handle if a Lock has been met in advance
- New method getCollectionAsStringList in CardCollection class to export the sequence for testing in advanced mode [1 mark]

Changes to ProcessCardToSequence

```

if (CheckIfLockChallengeMet())
{
    Console.WriteLine();
    Console.WriteLine("A challenge on the lock has been met.");
    Console.WriteLine();
    Score += 5;

    //CHANGE
    List<string> advancedMetConditions = new List<string>();
    advancedMetConditions = CurrentLock.getAdvancedMetConditions();
    int removalStartPoint = 0;
    if (advancedMetConditions.Count() > 0)
    {
        for (int i = 0; i < advancedMetConditions.Count(); i++)
        {
            for (int j = removalStartPoint; j < Sequence.GetNumberOfCards(); j++)
            {
                if (Sequence.GetCardDescriptionAt(j) == advancedMetConditions[i])
                {
                    MoveCard(Sequence, Deck, Sequence.GetCardNumberAt(j));
                    removalStartPoint = j;
                    break;
                }
            }
        }
        Console.WriteLine("Cards from Advanced Search have been moved back into the Deck");
        Deck.Shuffle();
        CurrentLock.removeAdvancedMet();
    }
    ///END CHANGE
}
    
```

Changes to CheckIfLockChallengeMet

```

SequenceAsString = Sequence.GetCardDescriptionAt(0) + Sequence.GetCardDescriptionAt(1) + Sequence.GetCardDescriptionAt(2) + Sequence.GetCardDescriptionAt(3) + Sequence.GetCardDescriptionAt(4) + Sequence.GetCardDescriptionAt(5) + Sequence.GetCardDescriptionAt(6) + Sequence.GetCardDescriptionAt(7) + Sequence.GetCardDescriptionAt(8) + Sequence.GetCardDescriptionAt(9)
//CHANGE
List<string> SequenceAsStringList = Sequence.getCollectionAsStringList()
//END CHANGE
if (CurrentLock.CheckIfConditionMet(SequenceAsString, SequenceAsStringList))
    
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Changes to Challenge

```
class Challenge
{
    protected List<string> Condition;
    protected bool Met;
    //CHANGE
    private bool advancedMet;
    //END CHANGE
    public Challenge()
    {
        Met = false;
    }

    //CHANGE
    public void SetAdvancedMet()
    {
        advancedMet = true;
    }

    public bool getAdvancedMet()
    {
        return advancedMet;
    }
    //END CHANGE
}
```

Changes to Lock

```
class Lock
{
    protected List<Challenge> Challenges = new List<Challenge>();

    //CHANGE
    public List<string> getAdvancedMetConditions()
    {
        List<string> emptyList = new List<string>();
        foreach (Challenge c in Challenges)
        {
            if (c.getAdvancedMet())
            {
                return c.GetCondition();
            }
        }
        return emptyList;
    }

    public void removeAdvancedMet() //Stops a challenge from showing
    {
        foreach (Challenge c in Challenges)
        {
            if (c.getAdvancedMet())
            {
                c.SetMet(true);
            }
        }
    }
    //END CHANGE
}
```

INSPECTION COPY

**COPYRIGHT
PROTECTED**



Changes to CheckIfConditionMet in the Lock class

```
//CHANGE
public virtual bool CheckIfConditionMet(string sequence, List<string>
{
    foreach (var C in Challenges)
    {
        List<string> ConditionAsStringList = C.GetCondition();
        if (!C.GetMet() && sequence == Convert.ToString(ConditionAsStringList))
        {
            C.SetMet(true);
            return true;
        }
        List<string> AdvancedSequence = new List<string>();
        int searchStartingPoint = 0;
        for (int i = 0; i < ConditionAsStringList.Count(); i++)
        {
            for (int j = searchStartingPoint; j < sequence.Length; j++)
            {
                if (ConditionAsStringList[i] == sequence[j])
                {
                    AdvancedSequence.Add(sequence[j]);
                    searchStartingPoint = j + 1;
                    break;
                }
            }
        }
        if (ConditionAsStringList.SequenceEqual(AdvancedSequence))
        {
            C.SetMet(true);
            C.setAdvancedMet();
            return true;
        }
    }
    //END CHANGE

    return false;
}
```

New getCollectionAsStringList method in CardCollection

```
//CHANGE
public List<string> getCollectionAsStringList()
{
    List<string> stringListOfCards = new List<string>();
    foreach (Card c in Cards)
    {
        stringListOfCards.Add(c.GetDescription());
    }
    return stringListOfCards;
}
//END CHANGE
```

INSPECTION COPY

**COPYRIGHT
PROTECTED**



Testing:

- Play two cards to a sequence, then solve the challenge with three more cards. "Challenge Met" and the three solution cards should be moved back to the deck the sequence underneath [1 mark] ↓

```
Current score: 7

CURRENT LOCK
-----
Not met:    P a, F a, P a
Not met:    K b

SEQUENCE
-----
| P b | K a | P a | F a |
-----

HAND:
-----
| K c | F c | P c | F c | P a |
-----

(D)iscard inspect, (U)se card:> u
Enter a number between 1 and 5 to specify card to use:> 5
(D)iscard or (P)lay?> p
```

```
A challenge on the lock has been met.

Cards from the search have been moved from the Sequence back into the deck.

Current score: 13

CURRENT LOCK
-----
Challenge met: P a, F a, P a
Not met:      K b

SEQUENCE:
-----
| P b | K a |
-----

HAND:
-----
| K c | F c | P c | F c | P a |
-----

(D)iscard inspect, (U)se card:>
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 13

Coding:

- Changing GetChoice to correctly prompt you to (S)ave the game and PlayGame if 'S' was chosen [1 mark]
- Returning a string of the correct format for the save file from getLocksForSave [1 mark]
- Returning a string of the correct format for the save file from getLocksSolvedFor [1 mark]
- Saving the current score to the save game file [1 mark]
- Saving the current lock to the save game file [1 mark]
- Saving the hand, sequence, deck and discard pile to the save game file [1 mark]
- Having a method or a loop that creates the string for a collection in the correct format [1 mark]

Example Solution

Changes to GetChoice

```
private string GetChoice()
{
    Console.WriteLine();
    //CHANGE
    Console.WriteLine("(D)iscard inspect, (U)se card, (S)ave current game");
    //END CHANGE
    string Choice = Console.ReadLine().ToUpper();
}
```

Changes to PlayGame

```
//CHANGE
case "S":
{
    if (saveGame())
    {
        Console.WriteLine("File Saved");
    }
    break;
}
//END CHANGE
```

Code for SaveGame

```
//CHANGE
public bool saveGame()
{
    bool validInput = false;
    string fileName = "";
    while (!validInput)
    {
        Console.WriteLine("Please enter the filename in the format: 'game2.txt'");
        fileName = Console.ReadLine();
        if (fileName.Substring(fileName.Length - 4).ToLower() == ".txt")
        {
            if (File.Exists(fileName))
            {
                Console.WriteLine("That file already exists - please try again");
            }
            else
            {
                validInput = true;
            }
        }
        else
        {
            Console.WriteLine("That filename does not appear valid");
        }
    }
}
```

INSPECTION COPY

COPYRIGHT
PROTECTED



```

try
{
    using (StreamWriter sw = new StreamWriter(fileName))
    {
        sw.WriteLine(Score);
        sw.WriteLine(CurrentLock.getLocksForSaving());
        sw.WriteLine(CurrentLock.getLocksSolvedForSaving());
        sw.WriteLine(Hand.getCollectionForSaving());
        sw.WriteLine(Sequence.getCollectionForSaving());
        sw.WriteLine(Discard.getCollectionForSaving());
        sw.WriteLine(Deck.getCollectionForSaving());
    }
    return true;
}
catch
{
    Console.WriteLine("Error in writing to file");
    return false;
}
}
//END CHANGE

```

Code for getLocksForSaving and getLocksSolvedForSaving in Lock

```

class Lock
{
    protected List<Challenge> Challenges = new List<Challenge>();
    //CHANGE
    public string getLocksForSaving()
    {
        string LocksString = "";
        foreach (Challenge c in Challenges)
        {
            List<string> LockConditions = c.GetCondition();
            for (int i = 0; i < LockConditions.Count(); i++)
            {
                LocksString += LockConditions[i];
                if (i < LockConditions.Count() - 1)
                {
                    LocksString += ",";
                }
            }
            LocksString += ";";
        }
        return LocksString.Remove(LocksString.Length - 1, 1);
    }

    public string getLocksSolvedForSaving()
    {
        string LocksSolvedString = "";
        foreach (Challenge c in Challenges)
        {
            if (c.GetMet())
            {
                LocksSolvedString += "Y";
            }
            else
            {
                LocksSolvedString += "N";
            }
            LocksSolvedString += ";";
        }
        return LocksSolvedString.Remove(LocksSolvedString.Length - 1, 1);
    }
}
//END CHANGE

```

**COPYRIGHT
PROTECTED**



Code for getCollectionForSaving in CardCollection

```
//CHANGE
public string getCollectionForSaving()
{
    string collectionString = "";
    foreach (Card c in Cards)
    {
        collectionString += c.GetDescription();
        collectionString += " " + Convert.ToString(c.GetCardNumber());
        collectionString += ",";
    }
    if (collectionString.Length != 0)
    {
        new Zig Zag Education collectionString.Remove(collectionString.Length - 1, 1);
    }
    else
    {
        return "";
    }
}
//END CHANGE
```

Changes to SetupGame

```
if (Choice == "L")
{
    //CHANGE
    bool validInput = false;
    string fileName = "";
    while (!validInput)
    {
        Console.WriteLine("Please enter the filename in the format: 'game2.txt'");
        fileName = Console.ReadLine();
        if (fileName.Length > 4 && fileName.Substring(fileName.Length - 4).ToLower() == ".txt")
        {
            new Zig Zag Education if (File.Exists(fileName))
            {
                validInput = true;
            }
            else
            {
                Console.WriteLine("That file does not exist - please try again");
            }
        }
        else
        {
            Console.WriteLine("That filename does not appear valid");
        }
    }
}
//END CHANGE
```

INSPECTION COPY

**COPYRIGHT
PROTECTED**



Testing:

- Saving game then loading game [1 mark] ↓

```
Current score: 15

CURRENT LOCK
-----
Not met:      P a, F a, P a
Challenge met: K b

SEQUENCE:
-----
| K b | P a | F b | K c | P a |
-----

HAND:
-----
| K a | P b | F c | K c | P c |
-----

(D)iscard inspect, (U)se card, (S)ave current game:> s
Please enter the filename in the format: name.txt for example 'game2
game2.txt
File Saved
```

```
Enter l to load a game from a file, anything else to play a new game
Please enter the filename in the format: name.txt for example 'game2
game2.txt

Current score: 15

CURRENT LOCK
-----
Not met:      P a, F a, P a
Challenge met: K b

SEQUENCE:
-----
| K b | P b | F b | K c | P a |
-----

HAND:
-----
| K a | P b | F c | K c | P c |
-----

(D)iscard inspect, (U)se card, (S)ave current game:>
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 14

Coding:

- Adding 5 to BonusPool after adding 5 to the score when completing a challenge
- Adding 5 to BonusPool when playing a card to the sequence that is a partial solution
- Resetting BonusPool to 0 under all circumstances where a card is not played to the sequence
- Creating the new attribute BonusPool and initialising it to 0 [1 mark]
- Writing the code for partComplete such that it returns true if the card just played to the sequence is a partial solution, otherwise false if it didn't add to an existing challenge [1 mark]

Example Solution

Changes to Player and Sequence

```
if (CheckIfChallengeMet())
{
    Console.WriteLine();
    Console.WriteLine("A challenge on the lock has been met.");
    Console.WriteLine();
    //CHANGE
    Score += 5 + BonusPool;
    BonusPool += 5;
}
else
{
    if (CurrentLock.partComplete(Sequence))
    {
        BonusPool += 5;
    }
    else
    {
        BonusPool = 0;
    }
}
//END CHANGE
}
```

Changes to PlayGame

```
while (!LockSolved && !GameOver)
{
    Console.WriteLine();
    Console.WriteLine("Current score: " + Score);
    //CHANGE
    Console.WriteLine("The current Bonus Pool is: " + BonusPool);
    //END CHANGE
    Console.WriteLine(CurrentLock.GetLockDetails());
    Console.WriteLine(Sequence.GetCardDisplay());
    Console.WriteLine(Hand.GetCardDisplay());

    ...

    if (DiscardOrPlay == "D")
    {
        MoveCard(Hand, Discard, Hand.GetCardNumberAt(CardChoice));
        GetCardFromDeck(CardChoice);
        //CHANGE
        BonusPool = 0;
        //END CHANGE
    }
}
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Code for partComplete in the Lock class

```
//CHANGE
public bool partComplete(CardCollection Sequence)
{
    bool partial = false;
    foreach (Challenge C in Challenges)
    {
        List<string> Condition = C.GetCondition();
        if (Condition.Count == 3)
        {
            int sequenceLength = Sequence.GetNumberOfCards() - 1;
            if (sequenceLength > 0 && Condition[1] ==
                Sequence.GetCardDescriptionAt(sequenceLength) && Condition[2] ==
                Sequence.GetCardDescriptionAt(sequenceLength - 1))
            {
                partial = true;
            }
            else if (sequenceLength >= 0 && Condition[0] ==
                Sequence.GetCardDescriptionAt(sequenceLength))
            {
                partial = true;
            }
        }
    }
    return partial;
}
//END CHANGE
```

Code for new attribute BonusPool

```
//CHANGE
private int BonusPool = 0;
//END CHANGE
```

Testing:

- Playing three cards to solve a challenge then solve it one card after another [1 mark] →

Current score: 3
The current Bonus Pool is:

CURRENT LOCK

Not met: P a, F a, P
Not met: P b, F b, P
Not met: K c

SEQUENCE:

| P a | F a |

HAND:

| P b | P b | P a | K b |

(D)iscard inspect, (U)se card
Enter a number between 1 and
the number of cards in the hand or (P)lay?:> p

A challenge on the lock has been met.

Current score: 19
The current Bonus Pool is:

CURRENT LOCK

Challenge met: P a, F a, P
Not met: P b, F b, P
Not met: K c

INSPECTION COPY

COPYRIGHT
PROTECTED



A card is played out of sequence which resets BonusPool back to zero...

```
Current score: 19
The current Bonus Pool is: 19

CURRENT LOCK
-----
Challenge met: P a, F a, P a
Not met:      P b, F b, P b
Not met:      K c

SEQUENCE:
-----
[ P a | F a | P a ]

HAND:
-----
[ P b | P b | K b | c | P c ]

(D)iscard inspect, (U)se card: > u
Enter a number between 1 and 5 to specify card to use: > 4
(D)iscard or (P)lay?: > p

Difficulty encountered!

HAND:
-----
[ P b | P b | K b | P c ]

To deal with this you need to either lose a key (enter 1-5 to specify position of key) or (D)iscard five
from the deck: >

A difficulty card was discarded from the deck when refilling the hand.

Current score: 21
The current Bonus Pool is: 0

CURRENT LOCK
-----
Challenge met: P a, F a, P a
Not met:      P b, F b, P b
```

A new card is played in sequence towards a new challenge which starts adding to BonusPool again...

```
SEQUENCE:
-----
[ P a | F a | P a ]

HAND:
-----
[ P b | P b | K b ]

(D)iscard inspect,
Enter a number between 1 and 5 to specify card to use: > 4
(D)iscard or (P)lay?: > p

Current score: 22
The current Bonus Pool is: 0

CURRENT LOCK
-----
Challenge met: P a, F a, P a
Not met:      P b, F b, P b
Not met:      K c

SEQUENCE:
-----
[ P a | F a | P a ]

HAND:
-----
[ P b | K b | P c ]
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Task 15

Coding:

- Adding FinalLock as a private attribute and initialising it to 0 [1 mark]
- Adding the selection conditions for FinalLock == 1 and FinalLock == 2 to CheckIfPlayerHasLost (they don't have the correct contents) [1 mark]
- Changing the condition of the selection statement in GetCardFromDeck correctly [1 mark]
- Changing ProcessLockSolved to have 10 attempts to find a solvable lock [1 mark]
- Changing ProcessLockSolved to call GenerateSolvableLock once 10 attempts have failed [1 mark]
- Changing ProcessLockSolved to set FinalLock to 1 and skip the main body of the loop [1 mark]
- Writing GenerateSolvableLock such that it always generates a solvable lock (the tools of the trade type is excluded) [2 marks]
- Returning true and False correctly from IsSolvable [1 mark]

Example Solution

Addition of FinalLock attribute

```
//CHANGE
private int FinalLock = 0;
//CHANGE
```

Changes to CheckIfPlayerHasLost

```
private bool CheckIfPlayerHasLost()
{
    //CHANGE
    if (FinalLock == 1)
    {
        FinalLock = 2;
    }
    else if (FinalLock == 2)
    {
        if (Deck.GetNumberOfCards() == 0)
        {
            Console.WriteLine("You have run out of cards in your hand. Your score is: " + Score);
        }
        else
        {
            Console.WriteLine("You have solved the final lock. Your score is: " + Score);
        }
    }

    return true;
}
//END CHANGE
if (Deck.GetNumberOfCards() == 0)
```

Changes to GetCardFromDeck

```
else
{
    MoveCard(Deck, Hand, Deck.GetCardNumberAt(0));
}
}
//CHANGE
if (Deck.GetNumberOfCards() == 0 && Hand.GetNumberOfCards() < 5 && Hand.GetNumberOfCards() == 0)
//END CHANGE
{
    GameOver = true;
}
```

INSPECTION COPY

COPYRIGHT
PROTECTED



Changes to ProcessLockSolved

```
private void ProcessLockSolved()
{
    Score += 10;
    Console.WriteLine("Lock has been solved. Your score is now: "
    //CHANGE
    if (FinalLock < 2)
    {
        while (Discard.GetNumberOfCards() > 0)
        {
            MoveCard(Discard, Deck, Discard.GetCardNumberAt(0));
        }
        Deck.Shuffle();
        int attempts = 0;
        while (attempts < 10)
        {
            CurrentLock = GetRandomLock();
            if (CurrentLock.IsSolvable(Deck, Hand))
            {
                break;
            }
            else
            {
                attempts += 1;
            }
        }
        if (attempts == 10)
        {
            Console.WriteLine("Final Lock");
            CurrentLock = GenerateSolvableLock();
            FinalLock = 1;
            GameOver = true;
        }
    }
    //END CHANGE
}
```

Code for GenerateSolvableLock

```
//CHANGE
private Lock GenerateSolvableLock()
{
    List<string> cardsLeft = new List<string>();
    List<string> solvableLockChallenge = new List<string>();
    cardsLeft = Deck.GetAllCards().Select(x => x.GetDescription()).ToList();
    List<string> cardsDiffRemoved = new List<string>();
    cardsDiffRemoved = cardsLeft.Where(x => x != "Dif").ToList();
    Random rand = new Random();
    if (cardsDiffRemoved.Count == 1)
    {
        solvableLockChallenge.Add(cardsDiffRemoved[0]);
    }
    if (cardsDiffRemoved.Count > 2)
    {
        while (solvableLockChallenge.Count < 2)
        {
            int cardSelected = rand.Next(0, cardsDiffRemoved.Count);
            if (solvableLockChallenge.Count > 0 && !(solvableLockChallenge.Contains(cardsDiffRemoved[cardSelected])))
            {
                solvableLockChallenge.Add(cardsDiffRemoved[cardSelected]);
            }
            if (solvableLockChallenge.Count == 2)
            {
                break;
            }
        }
    }
}
```

INSPECTION COPY

COPYRIGHT
PROTECTED



```

        solvableLockChallenge.Add(cardsDiffRemoved[cardSelected]);
        cardsDiffRemoved.RemoveAt(cardSelected);
    }
}
}
Lock newLock = new Lock();
newLock.AddChallenge(solvableLockChallenge);
return newLock;
}
//END CHANGE

```

Code for getAllCards in CardCollection

```

//CHANGE
public List<string> getAllCards()
{
    return Cards;
}
//END CHANGE

```

Code for isSolvable in the Lock class

```

//CHANGE
public bool isSolvable(CardCollection Deck, CardCollection Hand)
{
    List<string> allAvailableCards = new List<string>();
    allAvailableCards = Hand.getAllCards().Select(x => x.GetDescription());
    allAvailableCards.AddRange(Deck.getAllCards().Select(x => x.GetDescription()));

    List<string> allLockConditions = new List<string>();
    foreach (Challenge c in Challenges)
    {
        foreach (string s in c.GetLockConditions())
        {
            allLockConditions.Add(s);
        }
    }
    List<string> solvableCombination = allLockConditions.Where(x =>
    allAvailableCards.Contains(x)).ToList();
    if (solvableCombination.Count < allLockConditions.Count)
    {
        return false;
    }
    else
    {
        return true;
    }
}
//END CHANGE

```

**COPYRIGHT
PROTECTED**



Testing:

- Printing out the final lock [1 mark] ↓

(These screen shots run sequentially)

Current score: 16

CURRENT LOCK

Not met: P c, F c, P c

SEQUENCE:

| P a | K

HAND:

| P c | K b | P a | F c | P b |

(D)iscard inspect, (U)se card:> u
Enter a number between 1 and 5 to specify card to use:> 1
(D)iscard or (P)lay?> p

A challenge on the lock has been met.

Lock has been solved. Your score is now: 32

Final lock

Current score: 32

CURRENT LOCK

Not met: P b, K a

SEQUENCE:

| P a | K c | F c | P c |

HAND:

| K b | P a | F c | P b | K b |

(D)iscard inspect, (U)se card:> u
Enter a number between 1 and 5 to specify card to use:> 1
(D)iscard or (P)lay?> p

Current score: 35

CURRENT LOCK

Not met: P b, K

SEQUENCE:

| P a | K c | P c | F

HAND:

| P a | F c | P b | K

(D)iscard inspect, (U)se card:> u
Enter a number between 1 and 5 to specify card to use:> 1
(D)iscard or (P)lay?> p

Current score: 36

CURRENT LOCK

Not met: P b, K

SEQUENCE:

| P a | K c | P c | F

HAND:

| P a | F c | P b | K

(D)iscard inspect, (U)se card:> u
Enter a number between 1 and 5 to specify card to use:> 1
(D)iscard or (P)lay?> p

A challenge on the lock has been met.

Lock has been solved. You have solved the final lock.

INSPECTION COPY

COPYRIGHT
PROTECTED



Name

ZigZag Education supporting

A Level AQA Computer Science Paper

Summer 2022

 **BREAKTHROUGH!**

Electronic Answer Document (EAD)

Instructions

- Enter your name in the box at the top of this page
- Answer **all** questions by entering your answers into this document
- Remember to **save** this document regularly
- Save and print this document and any additional pages
- Answer **all** questions
- The marks available for each question are shown in brackets
- You will need:
 - ☐ access to a computer
 - ☐ access to a printer
 - ☐ access to appropriate software
 - ☐ electronic copies of the required skeleton code
 - ☐ EAD (Electronic Answer Document)

Total marks:

INSPECTION COPY

COPYRIGHT
PROTECTED



Programming Theory Question

Answer all questions. Remember to save this document

Q	Answer																																																
1	<div>(a)</div> <div>(b)</div>																																																
2	<div>(a)</div> <div>(b)</div> <div>(c)</div>																																																
3	<div>(a)</div> <div>(b)</div>																																																
4	<div>(a)</div> <table border="1"> <thead> <tr> <th>Count</th> <th>SequenceAsString</th> <th>Return value</th> </tr> </thead> <tbody> <tr> <td></td> <td>""</td> <td></td> </tr> <tr> <td>5</td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> </tr> </tbody> </table> <div>(b)</div> <table border="1"> <thead> <tr> <th>Count</th> <th>SequenceAsString</th> <th>Return value</th> </tr> </thead> <tbody> <tr> <td></td> <td>""</td> <td></td> </tr> <tr> <td>5</td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> </tr> </tbody> </table>	Count	SequenceAsString	Return value		""		5																		Count	SequenceAsString	Return value		""		5																	
Count	SequenceAsString	Return value																																															
	""																																																
5																																																	
Count	SequenceAsString	Return value																																															
	""																																																
5																																																	
5	<div>(a)</div> <div>(b)</div>																																																
6	<div>(a)</div> <div>(b)</div>																																																

INSPECTION COPY

COPYRIGHT
PROTECTED



INSPECTION COPY

Q	Answer	
7	(a)	
	(b)	
8	(a)	
	(b)	
9	(a)	
	(b)	
	(c)	
10	(a)	
	(b)	
11		
12		
13	(a)	
	(b)	
14	(a)	
	(b)	
	(c)	
15		

COPYRIGHT
PROTECTED



Programming Tasks

Answer all questions. Remember to save this document

Q	Answer
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

INSPECTION COPY

COPYRIGHT
PROTECTED

