Zig Zag Education

Computer Science

A Level | AQA | 7517

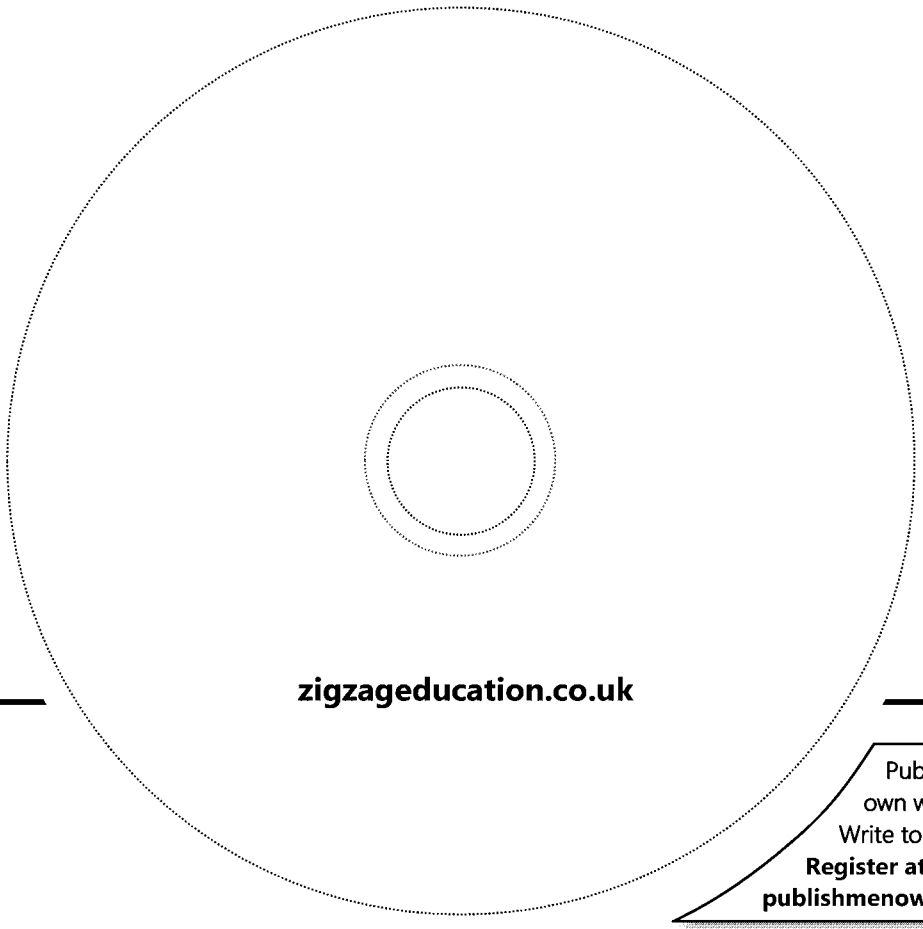
2015 specification for the 2020 exam

PAPER 1 EXAM RESOURCE PACK 2020

Food Magnate Simulation

for A Level AQA Computer Science

C# EDITION



DC7/ 9838

POD 9838

zigzageducation.co.uk

Contents

Product Support

Get Product Updates

Occasionally we make improvements to resources after you receive them. Download the updated content **free of charge**.

Download Support Files

Any support files for your purchased resources are available for download. This includes HTML links pages for resources that contain lots of hyperlinks.

Send us Your Feedback

For every completed review, **get a £10 voucher** to use on your next order! Tell us what you thought, and report any issues or ideas for improvement.

Get New Resource Notifications

Opt in to receive email alerts about new resources for your subject(s).

Contact Us

zigzageducation.co.uk
sales@zigzageducation.co.uk

0117 950 3199

ZigZag Education
Unit 3 Greenway Centre
Doncaster Road
Bristol BS10 5PY

Register today via:

ZigZagEducation.co.uk → Computer Science & IT → Product Support

Quick link: zzed.uk/PS

Terms and Conditions of Use

Terms and Conditions

Please note that the **Terms and Conditions** of this resource include point 5.3, which states:

> **"You acknowledge that you rely on your own skill and judgement in determining the suitability of the Goods for any particular purpose."**

"We do not warrant: that any of the Goods are suitable for any particular purpose (e.g. any particular qualification), or the results that may be obtained from the use of any publication, or expected exam grades, or that we are affiliated with any educational institution, or that any publication is authorised by, associated with, sponsored by or endorsed by any educational institution."

Copyright Information

Every effort is made to ensure that the information provided in this publication is accurate and up to date but no legal responsibility is accepted for any errors, omissions or misleading statements. It is ZigZag Education's policy to obtain permission for any copyright material in their publications. The publishers will be glad to make suitable arrangements with any copyright holders whom it has not been possible to contact.

Students and teachers may not use any material or content contained herein and incorporate it into a body of work without referencing/acknowledging the source of the material ("Plagiarism").

Disclaimer

This resource is intended to supplement your teaching only.

It is the teacher's responsibility to decide how to use this resource to assist themselves and their students appropriately. You may simply wish to read this material to better inform yourself and to help you prepare your lessons and to give you ideas for your teaching. You may also consider whether it is appropriate to hand out some of the sheets for reference and to use some of the activities for classwork or homework. You may also consider whether it is appropriate to hand out the booklet to be worked through by your students more independently.

As with all pre-release material, it is the teacher's responsibility to decide in what way to assist their students, and to decide how this resource in particular can be used to fit into that assistance.

The resources here are provided as an interpretation of the pre-release material.
The author does not have any special knowledge of what to expect on any particular exam.

ZigZag Education is not affiliated with AQA, Pearson, Edexcel, OCR, WJEC, CEA, International Baccalaureate Organization or DFE in any way nor is this publication authorised by, associated with, sponsored by or endorsed by these institutions unless explicitly stated on the front cover of this publication.

This publication is designed to supplement teaching only. Practice questions may be designed to follow the content of a specification and may also attempt to prepare students for the type of questions they will meet in the examination, but will not attempt to predict future examination questions. ZigZag Education do not make any warranty as to the results that may be obtained from the use of this publication, or as to the accuracy, reliability or content of the publication.

Where the teacher uses any of the material from this resource to support examinations or similar then the teacher must ensure that they are happy with the level of information and support provided pertaining to their personal point of view and to the constraints of the specification and to others involved in the delivery of the course. It is considered essential that the teacher adapt, extend and/or censor any parts of the contained material to suit their needs, the needs of the specification and the needs of the individual or group concerned. As such, the teacher must determine which parts of the material, if any, to provide to the students and which parts to use as background information for themselves. Likewise, the teacher must determine what additional material is required to cover all points on the specification and to cover each specification point to the correct depth.

Teacher's Introduction

This resource pack is designed to help you support your students taking the **A Level Computer Science Paper 1** examination. It is based on the '**Food Magnate Simulation**' preliminary material (C#) – for examination June 2020.

On the CD, you will find the following:

- `FoodMagnateSim` 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 `FoodMagnateSim` 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.

- `c01-Commentary.pdf`
- `c06a-ClassDiagramTasksMS.pdf`
- `c06b-ClassDiagramFull.pdf`
- `c07-TheoryQuestionsMS.pdf`
- `c08-ProgrammingTasksMS.pdf`

The resource pack consists of the following:

① **Pre-release Commentary** including a general explanation of how the program works (text and video), plus a more detailed, technical overview* describing each subroutine, class and attribute in turn.

> ***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. For this reason, this content has been placed on the 'Answers & Solutions' HTML page as a password-protected file, to allow you to control if/when students access it.*

② **Class Diagram Tasks**

Three partially complete UML class diagram tasks for students to complete while getting to grips with the skeleton program. Students must add any missing attributes, subroutines, access modifiers, parameters and return types. Completed versions are provided via the *Solutions* web page as a password-protected PDF.

③ **Written Questions**

Theory questions testing students' understanding of the skeleton program, like Section C in the exam. 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 version are available format. Suggested answers are provided via the *Solutions* web page as a password-protected PDF.

④ **Programming Tasks**

Fifteen modification exercises put students' programming skills to the test, like Section D in the exam. 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.

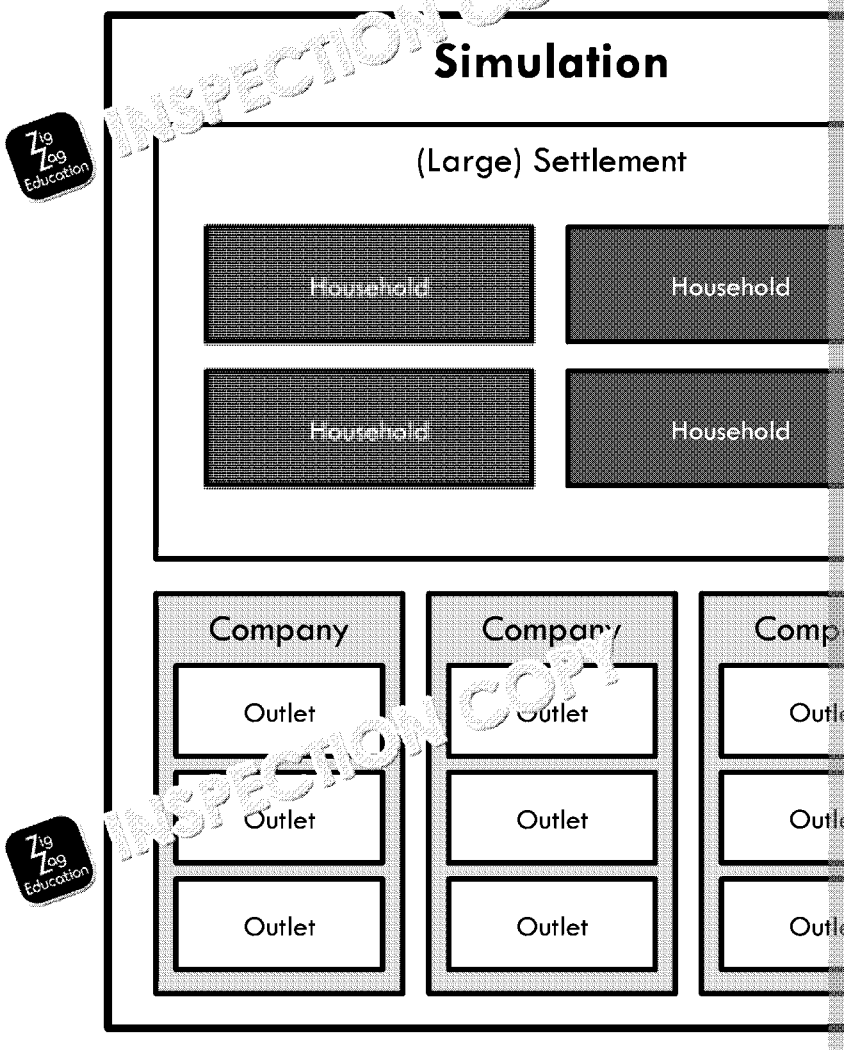
An **Electronic Answer Document (EAD)** is provided should you wish students to use it for ③ and/or ④ above.

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

Food Magnate Simulation

Introduction

The *Food Magnate Simulation* models how profitable different types of restaurant c
simulated settlement. The structure of the simulation, conceptually, is as follows:



The program contains the following objects:

- A single object of type `Simulation`, which is responsible for constructing
remaining objects
- A single object of type `Settlement`, which is stored within the `Simulat`
can be an object of type `LargeSettlement`, which behaves identically b
contain more households.
- An unlimited number of `Household` objects, each of which is contained w
settlement begins with 250 of these (or a `LargeSettlement` object mi
- An unlimited number of `Company` objects, although the default starting nu
The `Company` objects are contained directly within the `Simulation` obje
- An unlimited number of `Outlet` objects, with each outlet being stored in
A company cannot exist without at least one outlet.

Let's look at the attributes for each class in detail...

Class: Simulation

The `Main` subroutine creates the program's single `Simulation` object and calls its onwards, it is the `Simulation` object that is responsible for creating and managing are as follows:

- A single `Settlement` object, `simulationSettlement`; since the `Set` `Household` objects, the simulation cannot in[illegible]d[illegible]ectly with a household subroutine in the `Settlement` class
- An integer variable to store the number of companies, `noOfCompanies`
- A double called `fuelCostPerUnit`, which is used to calculate delivery c[illegible] to the company; it is passed to the constructor of the `Company` class
- A second double, called `baseCostForDelivery`, which is also passed t[illegible] subsequently used to calculate delivery costs
- A list called `companies`, to store objects of type `Company`; a list is a bett[illegible] number of companies can be stored
- A random number generator called `rnd`, used to generate random events

Class: Company

A company can be either a fast-food company, a family company or a named chef outlets belonging to a company will also be of that type; a company cannot have, f outlets and some family outlets. The `Company` class's attributes are as follows:

- A string, `name`, to store the name of the company
- A second string, called `category`; this st[illegible]res th[illegible]ype of company, and ca 'named chef'
- A series of double attributes:
  - `bal[illegible]e` amount of money a company owns, which can be p
  - `rep[illegible]ationScore`: a measure of how well regarded the compa[illegible] reputation score is more likely to be visited than one with a low re
  - `avgCostPerMeal`: how much the company pays for a meal
  - `avgPricePerMeal`: how much a customer pays for a meal when outlets
  - `dailyCosts`: single cost per day, per company, initially set to 10[illegible] will change, up or down, between days
  - `familyOutletCost`: the cost of opening an outlet for a 'family'
  - `fastFoodOutletCost`: the cost of opening an outlet for a 'fast
  - `namedChefOutletCost`: the cost of opening an outlet for a 'na
  - `fuelCostPerUnit`: for companies with multiple outlets, this is p their 'main' outlet (the first one to be created) and each subsequer
  - `baseCostOfDelivery`: for companies wit[illegible] any number of outl day
- A list called `outlets`, to store ob[illegible]s of type `Outlet`; again, this offers m
- A series of integer attri[illegible]es:
  - `fami[illegible]OutletCapacity`: capacity for a 'family' outlet, ini
  - `fas[illegible]FoodOutletCapacity`: capacity for a 'fast food' outlet, ini
  - `[illegible]medChefOutletCapacity`: capacity for a 'named chef' outle
- A random number generator called `rnd`, used to generate random reputat

Class: Outlet

An `Outlet` object is stored within a data structure in a `Company` object, modellin by a single company. Outlets are also associated with the settlement, since each ou settlement, and households, if they choose to visit a company's outlet, will always v

The `Outlet` class's attributes are as follows:

- A series of integer attributes:
  - `visitsToday`: the num[illegible] tim[illegible] a household has visited this worth noting th[illegible] [illegible]othing preventing this exceeding the o
  - `xCoord` [illegible] `yCoord`: the outlet's location within the settlement
  - [illegible]ap[illegible]: how many seats are in the outlet, used to calculate da
  - `xCapacity`: the maximum to which the capacity can be extend costs
- A double, `dailyCosts`, which is how much the outlet costs to run each d not fixed, and can be changed as capacity changes. It could also be chang `Outlet` class's `AlterDailyCost` subroutine; however, this is never calle
- A random number generator called `rnd`, used to generate `maxCapacity`

Class: Settlement

The simulation contains a single settlement, and each household and outlet has a l The settlement is easy to visualise in terms of a grid, but it is not stored as a two-di this is twofold. Firstly, most elements within a settlement array would be empty. T settlement with one million possible locations, but only 250 households and 12 co settlement contains objects of different types – namely h[illegible]olds and outlets. In household stores its own X and Y coordinates, whi[illegible]e within the bounds of

The `Settlement` class's attribute[illegible] f[illegible]ows:

- An integer variab[illegible] `startNoOfHouseholds`, which is how many household sim[illegible] [illegible] existing code, this can go up but not down as the simulat
- Two [illegible]r integers, `xSize` and `ySize`, which store between them the si value for each of these is 1,000, meaning there are one million possible loc
- A list, `households`, to store each `Household` object
- A random number generator called `rnd`, used to generate random locatio

The `LargeSettlement` class inherits from `Settlement`, and allows the user to the values of `startNoOfHouseholds`, `xSize` and `ySize`.

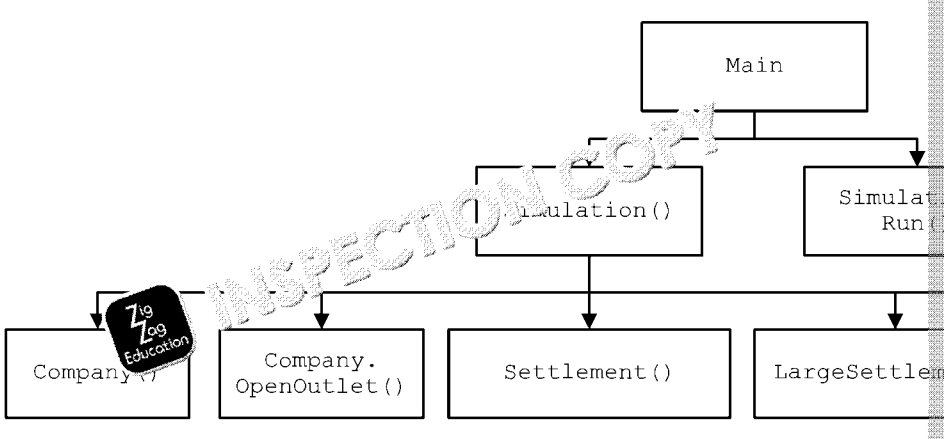
Class: Household

To all intents and purposes, this class models a consumer, since the whole househo There is scope here for modelling households of different [illegible] or households in w go out to eat, or even households where different [illegible]s eat out at different ou `Household` class's attributes are as follo[illegible]:

- A double, `chanceF[illegible]EatToday`, storing a value between 0 and 1, repr household [illegible] to eat
- Inte[illegible]iables `xCoord` and `yCoord`, representing a household's locatio
- A static integer, `nextID`, which numbers each new household incremental
- An additional integer, `ID`, which stores the value contained in `nextID` at p
- A random number generator called `rnd`, used to generate `chanceEatOu` changes

Overview

When the program begins:



1. A new `Simulation` object is constructed
2. That `Simulation` object constructs a new object of type `Settlement` c inherits from `Settlement`), depending on user input. The `Settlement` of `Household` repeatedly.
3. That `Simulation` object, as part of being constructed, also constructs ne either the three default companies hard-coded into the Skeleton Program user input
4. The `AddCompany` subroutine is called if user-defined companies are selec specific details of each company
5. The `Company` constructor will make at least o t the `Outlet` const have at least one outlet
6. The `Simulation` object' outine is called (see next hierarchy dia

```
**********************************
*******  MENU      **********
**********************************
1. Display details of households
2. Display details of companies
3. Modify company
4. Add new company
6. Advance to next day
Q. Quit

Enter your choice:
```

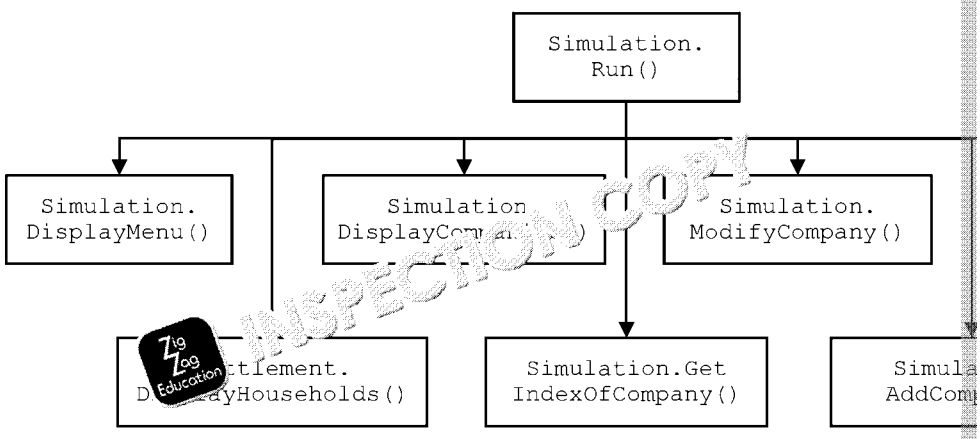
Main menu

```
**********************************
*******  MODIFY COMPANY   *******
**********************************
1. Open new outlet
2. Close outlet
3. Expand outlet

Enter your choice:
```

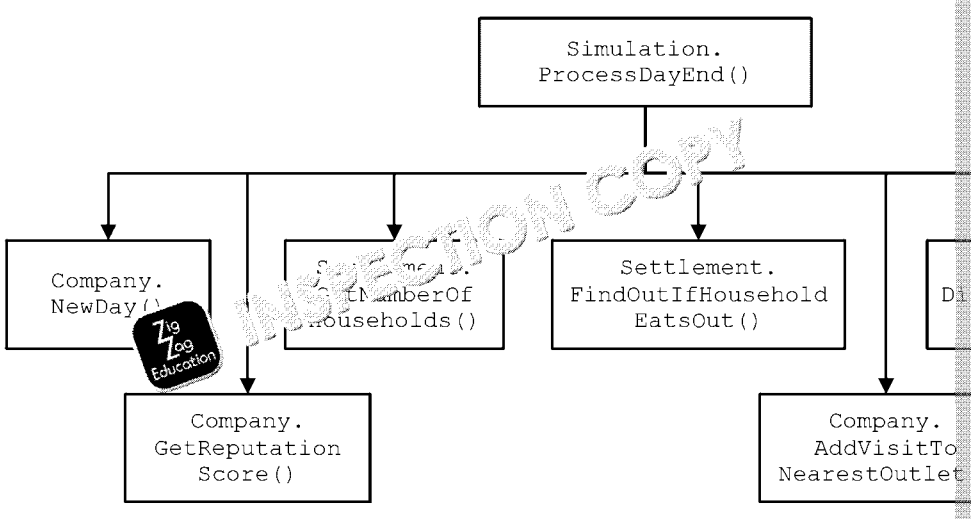
'Modif py' menu, shown wh th menu and enters a valid c

When the `Run` subroutine is called:



7. The user is presented with a menu as a result of a call to `DisplayMenu`
8. If the user selects option 1, a call is made to `DisplayHouseholds`, whic
the settlement via a call to `DisplayHouseholds` in the `Settlement` cl
`GetDetails` in each instance of the `Household` class
9. If the user selects option 2, details of all companies are displayed via a call
calls the `GetDetails` subroutine in each `Company` object. The `GetDet`
`Company` class calls a `GetDetails` subroutine for each outlet, meaning t
menu results in details of all companies, and their outlets, being displayed.
10. If the user selects option 3, to modify a company, they are prompted for a
passed to `GetIndexOfCompany` in order to find the index of that compa
`ModifyCompany` presents the user with a second menu containing three
outlet (`OpenOutlet()`), close an outlet (`CloseOutlet()`) or expand a
(`ExpandOut`
11. If the user selects option 4, a new company can be created via a call to `Ad`
user for details of the new company's name, type and starting balance. Th
outlet in a random location within the settlement.
12. There is no option 5, which makes it quite likely that adding an option 5 wi
13. If the user selects option 6, a call is made to `ProcessDayEnd` in the `Simu`
identically named subroutine exists in the `Company` class, so be sure not t
most involved subroutine in the program and is addressed in the next hier

When each day ends:



14. The call to `NewDay` (in the `Company` class) calls `NewDay` (a subroutine in
number of visits to zero
15. The reputation score of each company is accessed and added to a list, with
storing a running total of all reputation scores so far (i.e. the first value sto
second value stored is the sum of the reputation scores for the first two co
the sum of the reputation scores for the first three companies, and so on)
16. The call to `GetNumberOfHouseholds` is to
a loop through all
17. The call to `FindOutIfHouseho`
, for each `Household` obje
more likely to be true for
with a higher probability of eating ou
selected at rand
list from step 15, with companies holding a
likel
18. For
mpany that is chosen, the nearest outlet to the household eating
all outlets belonging to the chosen company are examined, and distances
calculated in the following way:
    - Distances are calculated by assuming movement is only possible north, south, east or west. This means that the distance from the house to outlet A is 4, while the distance from the house to outlet B is 3.
    - In the event that two outlets are of an equal distance from a house, the outlet examined first (i.e. the outlet appearing earlier in the outlets list) will be the one visited.
19. The call to `DisplayCompaniesAtDayEnd` calls the `ProcessDayEnd`
This subroutine calculates changes to the com
balance, which is affe
day, the price at which meals are bou
and
, and the distance betwe
for which the delivery of ingr
rs a cost based on the price of fue
above). The old bala
nd
new balance are then displayed, along wi
and its outlet
20. The
`DisplayEventsAtDayEnd` generates either a random proba
the s
rement, a change of fuel cost for a company chosen at random, a c
company chosen at random or a change of daily costs for a company chos

Program Subroutines

The program's functions, Ⓕ, and procedures, Ⓟ, are described below.

'Household' Class

Subroutine	Data	Description
GetChanceEatOut Ⓕ	Parameters: – Returns: chanceEatOutPerDay: double Called From: Settlement.FindOutIfHouseholdEatsOut Calls: –	Returns the value of chanceEatOutPerDay
GetDetails Ⓕ	Parameters: – Returns: details: String Called From: Settlement.DisplayHousehold Calls: –	1. Declares an empty string 2. Populates it with all attributes other than nextID and rnd 3. Returns the string
GetX Ⓕ	Parameters: – Returns: xCoord: int Called From: Settlement.FindOutIfHouseholdEatsOut Calls: –	Returns the value of xCoord
GetY Ⓕ	Parameters: – Returns: yCoord: int Called From: Settlement.FindOutIfHouseholdEatsOut Calls: –	Returns the value of yCoord

'Settlement' Class

Subroutine	Data	Description
`DisplayHouseholds` Ⓟ	Parameters: – Returns: – Called From: `Simulation.Run` Calls: `Household.GetDetail`	1. Outputs 'details of all households' 2. Loops through each household in the `households` list 3. Outputs the result of a call to the `GetDetails` subroutine of each household

`FindOutIf HouseholdEats` Ⓕ	Parameters: `householdNo: int, x: int, y: int`
	Returns: `Boolean`
	Called From: `Simulation.ProcessDayEn`
	Calls: `Household.GetX`
	`Household.GetY`
	`Household.GetChanceEatOut`
		1. Generate random double between 0 and 1
		2. If this number is less than the probability of the household's (that's the household passed as parameter) chance of eating out, return 'true'
		3. Otherwise, return 'false'

`GetNumberOf Households` Ⓕ	Parameters: – Returns: `households.Count: int` Called From: `Simulation.ProcessDayEnd` Calls: –	Returns the number of items in the `households` list
`GetRandom Location` Ⓟ	Parameters: – Returns: – Called From: `Simulation.AddCompany` `Settlement.AddHousehold` Calls: –	1. Accepts references to `X` and `Y` as parameters as these are passed by reference, they change in the subroutine from which `GetRandomLocation` is called, even though there is no return value 2. Sets `X` and `Y` to random values within the bounds of the `Settlement` object
`GetXSize` Ⓕ	Parameters: – Returns: `xSize: int` Called From: `Simulation.ModifyCompany` Calls: –	Returns the value of `xSize`

'Outlet' Class

Subroutine	Data	Description
`AlterCapacity` Ⓕ	Parameters: `change: int` Returns: `change: int` Called From: `Company.ExpandOutlet` Calls: –	1. Stores the current capacity in a local variable, `oldCapacity` 2. Adds the parameter to the `capacity` attribute; the parameter can be a negative number, in which case `capacity` is decreased 3. If the capacity has, as a result of step (2), risen above `maxCapacity`, set `capacity` to `maxCapacity` and return the difference between `oldCapacity` and `maxCapacity` (i.e. the amount of increase after capacity was limited) 4. If the capacity has fallen below zero, set `capacity` to zero 5. Recalculate the daily costs based on the new capacity and return the value of change **NB If there is a negative change attempted that is larger than possible, e.g. an outlet with a capacity of 5 has this subroutine called with a parameter of -5, it is -5 that is then returned, even though the capacity only goes down by three**
`AlterDailyCost` Ⓟ	Parameters: `amount: double` Returns: – Called From: – Calls: –	Accepts a value as a parameter and adds this value to the `dailyCosts` attribute **NB This subroutine is not called from anywhere.**
`CalculateDaily ProfitLoss` Ⓕ	Parameters: `avgCostPerMeal: double, avgPricePerMeal: double` Returns: `double` Called From: `Company.ProcessDayEnd` Calls: –	The calculation of daily profit or loss entails calculating the profit/loss for a single meal, multiplying by the number of meals, then subtracting the outlet's daily costs

Subroutine	Data	Description
`GetX` Ⓕ	Parameters: – Returns: `xCoord: int` Called From: `Company.AddVisitToNearestOutlet` `Company.GetDistanceBetweenTwoOutlets` Calls: –	Returns the value of `xCoord`
`GetY` Ⓕ	Parameters: – Returns: `yCoord: int` Called From: `Company.AddVisitToNearestOutlet` `Company.GetDistanceBetweenTwoOutlets` Calls: –	Returns the value of `yCoord`
`IncrementVisits` Ⓟ	Parameters: – Returns: – Called From: `Company.AddVisitToNearestOutlet` Calls: –	Adds 1 to the `visitsToday` attribute
`NewDay` Ⓟ	Parameters: – Returns: – Called From: `Company.NewDay` `Outlet` constructor Calls: –	Sets the `visitsToday` attribute to zero

'Company' Class

Subroutine	Data	Description

`AddVisitTo NearestOutlet` Ⓟ	Parameters: `x: int, y: int`
	Returns: –
	Called From: `Simulation.ProcessD`
	Calls: `Outlet.GetX`
	`Outlet.GetY`
	`Outlet.IncrementVisits`
		1. The parameters, `x` and `y`, are the coordinates of a household whose occupant(s) will e t on the current day
		2. Initialise local variable `stOutlet` to zero and declare two double variables
		3. Initialise `nearestOutlet` first outlet in the list (see step 5 below)
		4. Loops through each outlet in `outlets` list
		5. Calculates the distance from th household to the outlet. This is only along a straight line if th outlet and the household are in the same row or column of the lement; otherwise, there will be a combination of either up own with either left or right, along with a 90-degree turn.
		6. If the current outlet being examined closer than the closest found so far, store the index of the cu t outlet in `nearestOutlet`
		7. After the loop, access the outlet indexed `nearestOutlet` and call its `IncrementVisits` subroutine

`AlterReputation` Ⓟ	Parameters: `change: double` Returns: – Called From: `Simulation.ProcessReputationChangeEvent` Calls: –	Accepts a value as a parameter and adds this value to the `reputationScore` attribute
`AlterAvgCost PerMeal`	Parameters: `change: double` Returns: –	Accepts a value as a parameter and adds this value to the `avgCostPerMeal` attribute

Subroutine	Data	Description
`AlterFuelCost PerUnit` Ⓟ	Parameters: `change: double` Returns: – Called From: `Simulation.ProcessCostOfFuelChangeEvent` Calls: –	Accepts a value as a parameter and adds this value to the `fuelCostPerUnit` attribute

`Calculate DeliveryCost` Ⓕ	Parameters: –
	Returns: `totalCost: double`
	Called From: `Company.GetDetails`
	`Company.ProcessDayEnd`
	Calls: `Company.GetListOfOutlets`
	`Company.GetDistanceBetweenTwoOutlets`
		1. Calls `GetListOfOutlets` to receive an integer List, with one integer for each outlet beginning with zero
		2. Declares an integer, `totalDistance`, initialised to zero
		3. Loops once per outlet minus one (this subroutine is actually not called if a company has only one outlet)
		4. Calculates the distance between the current outlet (0 on the first iteration, 1 on the second, 2 on the third, etc.) and the outlet indexed one value higher (distance between outlets 0 and 1, distance between outlets 1 and 2, distance between outlets 2 and 3, etc.)
		5. For each run through the loop, add this distance to `totalDistance`
		6. Return the product of `totalDistance` and the `fuelCostPerUnit` attribute

`CloseOutlet` Ⓕ	Parameters: `ID: int` Returns: `closeCompany: Boolean` Called From: `Simulation.ModifyCompany` Calls: –	1. The outlet, identified by an `ID` integer passed as a parameter, is removed from the outlets list 2. If the list is now empty, return true; otherwise, return false
`ExpandOutlet` Ⓟ	Parameters: `ID: int` Returns: –	1. The parameter is the index of the outlet to be expanded 2. User is prompted for how much they want to expand capacity

Subroutine	Data	Description
GetReputationScore Ⓕ	Parameters: – Returns: reputationScore: double Called From: Simulation.ProcessDayEnd Calls: –	Returns the value of reputationScore
GetDetails Ⓕ	Parameters: – Returns: details: String Called From: Simulation.DisplayCompanies Calls: Company.CalculateDeliveryCost Outlet.GetDetails	1. Declares an empty string 2. Appends all attributes of this string, including the contents of the outlets list, which are addressed in turn via a for loop 3. Returns the string – this is essentially a toString subroutine
GetDistanceBetweenTwoOutlets Ⓕ	Parameters: outlet1: int, outlet2: int Returns: double Called From: Company.CalculateDeliveryCost Calls: –	1. Returns the distance between two outlets. Each outlet has a grid position, and the distance between outlets is the sum of the horizontal difference (between the column in which each one exists) and the vertical distance (between the row in which each one exists), i.e. not a straight line unless they share a row or a column.
GetListOfOutlets Ⓕ	Parameters: – Returns: temp: int[] Called From: Company.CalculateDeliveryCost Calls: –	1. A new integer list is created 2. A loop iterates through each outlet in the outlets list 3. The zero-based index of each outlet is added to the list 4. The list is returned
GetName Ⓕ	Parameters: – Returns: name: String Called From: Simulation.DisplayCompaniesAtDayEnd Simulation.ProcessCostOfFuelChangeEvent	Returns the value of name

Subroutine	Data	Description
`NewDay` Ⓟ	Parameters: – Returns: – Called From: `Simulation.ProcessDayEnd` Calls: `Outlet.NewDay`	1. Loops through each outlet in the `outlets` list 2. Calls the `NewDay` subroutine for each outlet, resetting the number of visits for each to zero
`OpenOutlet` Ⓟ	Parameters: `x: int, y: int` Returns: – Called From: `Simulation` constructor `Simulation.ModifyCompany` `Company` constructor Calls: `Outlet` constructor	1. Coordinates are passed as parameters 2. Depending on the type of company (fast-food, family, named chef), the company's balance is updated by subtracting the start-up costs 3. A new `Outlet` object is constructed using the coordinates which were passed in as parameters as well as the capacity (which is an integer variable, set according to company category) 4. The new outlet is added to the `outlets` list
`ProcessDayEnd` Ⓕ	Parameters: – Returns: `details: String` Called From: `Simulation.DisplayCompaniesAtDayEnd` Calls: `Company.CalculateDeliveryCost` `Outlet.CalculateDailyProfitLoss`	1. Declares an empty string, `details` 2. Declares double variable to track profit/loss overall and a further variable to track profit/loss for each outlet in turn, as well as one to track delivery costs 3. If there is only one outlet, `deliveryCosts` is set to the `baseCostOfDelivery` attribute 4. If there is more than one outlet, `deliveryCosts` is set to the `baseCostOfDelivery` attribute plus a call to `CalculateDeliveryCost` 5. `deliveryCosts` is then appended to `details` 6. Loops through each outlet in the `outlets` list, calling `CalculateDailyProfitLoss` for each outlet 7. The return from this call is appended to the `details` string and

'Simulation' Class

Subroutine	Data	Description
`AddCompany` Ⓟ	Parameters: – Returns: – Called From: `Simulation.Run` Calls: `Company.GetReputationScore`	1. Local variables declared to store details of the new company 2. User is prompted for name and starting balance for company 3. User is repeatedly prompted to enter 1, 2 or 3 (fast-food, family restaurant, named chef respectively) until either 1, 2 or 3 is entered 4. Depending on user input, the local variable `typeOfCompany` is set to either 'fast food', 'family' or 'named chef' 5. A call to `GetRandomLocation` sets the new `x` and `y` coordinates 6. New `Company` object is constructed and added to the `Companies` list
`DisplayCompaniesAtDayEnd` Ⓟ	Parameters: – Returns: – Called From: `Simulation.processDayEnd` Calls: `Company.GetName` `Company.ProcessDayEnd`	1. Outputs 'Companies:' to the console 2. Loops through each company in the `companies` list 3. Outputs the company's name 4. Outputs the return from a call to the `ProcessDayEnd` subroutine

`DisplayEventsAtDayEnd`
Ⓟ
	Parameters: –
	Returns: –
	Called From: `Simulation.ProcessDayEnd`
	Calls: `Simulation.ProcessAddHouseholdEvent`
	`Simulation.ProcessCostOfFuelChangeEvent`
	`Simulation.ProcessReputationChangeEvent`
	`Simulation.ProcessCostChangeEvent`
		1. Writes 'Events:' to the console
		2. Generates a random double, with 25% chance of entering a selection structure
		3. If the selection structure is entered, another random double is generated, giving a 25% chance of a call to `ProcessAddHouseholdEvent`
		4. Another random double is generated, still within the selection structure described in step 2; this one causes a 50% chance of a call to `ProcessCostOfFuelChangeEvent`
		5. Still within the selection structure from step 2, another random double is generated, causing a 50% chance of a call to `ProcessReputationChangeEvent`
		6. Still within the selection structure from step 2, a final random double is

Subroutine	Data	Description
`DisplayCompanies` Ⓟ	Parameters: – Returns: – Called From: `Simulation.Run` Calls: `Company.GetDetails`	1. Text 'details of all companies:' is output to the console 2. Loop initiated to run once per object in the `companies` List 3. Call to `GetDetails` for each company is made
`DisplayMenu` Ⓟ	Parameters: – Returns: – Called From: `Simulation.Run` Calls: –	Displays menu items, one on each line, and prompts the user for a choice; this subroutine does not validate input or even store the user's response; user input is handled in run **NB The menu has no option 5**
`GetIndexOfCompany` Ⓕ	Parameters: `companyName: String` Returns: `index: int` Called From: `Simulation.Run` Calls: `Company.GetName`	1. Integer variable declared and set to -1 2. Loops through all companies in turn 3. If a company name matches the passed parameter, return the zero-based index of that company 4. If the loop terminates without a match, return -1

`ModifyCompany` Ⓟ	Parameters: `index: int`
	Returns: –
	Called From: `Simulation.Run`
	Calls: `Company.GetNumberOfOutlets`
	`Company.CloseOutlet`
	`Company.ExpandOutlet`
	`Settlement.GetXSize`
	`Settlement.GetYSize`
	`Company.OpenOutlet`
		1. Submenu is displayed, with three options, and user input is accepted
		2. If input is 2 (close outlet) or 3 (expand outlet), the user is prompted for the ID of the outlet; if input is 1 (open outlet), jump to step 7
		3. Entered ID is cast as an integer
		4. If 2 (close outlet) was entered, a call to `CloseOutlet` is made, and if that was the last outlet for that company, the company is removed
		5. If 3 (expand outlet) was entered, a call to `ExpandOutlet` is made
		6. If an out-of-range ID is entered in step 3, display error message
		7. If input is 1 (open new outlet), prompt user for `X` and `Y` coordinates
		8. If the coordinates are within the grid bounds, call `OpenOutlet`

Subroutine	Data	Description
`ProcessReputationChangeEvent` (P)	Parameters: – Returns: – Called From: `Simulation.DisplayEventsAtDayEnd` Calls: `Company.GetName` `Company.AlterReputati`	1. Generates a random number, always to a single decimal place, between 0.1 and 0.9 2. Generates a random integer of either 1 or 0 called `upOrDown` 3. Generates a random company ind 4. If `upOrDown` is zero, the randomly sen company's reputation score goes up by the decimal amount; othe , it goes down by that amount 5. Step 4 is implemented by outputting th sult and by calling the company's `AlterReputation` subrou
`ProcessAddHouseholdEvent` (P)	Parameters: – Returns: – alled From: `Simulation.DisplayEventsAtD nd` lls: `Settlement.AddHousehold`	1. Generates a random integer between 1 a 2. Loops that many times, calling the `Settlem nt` lass's `AddHousehold` subroutine, effectively creating that many ne ousehold objects 3. Output to console the number of new household created
`ProcessCostOfFuelChangeEvent` (P)	meters: – Re rns: – Ca le From: `Simulation.DisplayEventsAtDayEn` Cal `Company.GetName` `Company.AlterFuelCostPerUnit`	1. Generates a random number, always to a singl cimal place, between 0.1 and 0.9 2. Generates a random integer of either 1 or 0 calle OrDown 3. Generates a random company index 4. If `upOrDown` is zero, the randomly chosen company's el cost per unit goes up by the decimal amount; otherwise, it goes down by that amount 5. Step 4 is implemented by outputting the result and by calling the company's `AlterFuelCostPerUnit` subroutine
`ProcessCostChangeEvent` (P)	Parameters: – Returns: – Called From: `Simulation.DisplayEventsAtDayEnd`	1. Integer variables `costToChange` and `upOrDown` are randomly initialised to either zero or one 2. A further integer, `companyNo`, is initialised to a randomly selected index

Subroutine	Data	Description
`ProcessDayEnd` Ⓟ	Parameters: – Returns: – Called From: `Simulation.Run` Calls: `Company.NewDay` `company.GetReputationSco` `Settlement.GetNumberOfHouseholds` `Settlement.FindOutIfHouseh` `EatsOut` `Company.AddVisitToNearestOu` `t` `Simulation.DisplayCompaniesA` `yEnd` `Simulation.DisplayEventsAtDayE` `d`	1. Loop through each company in the `companies` list 2. Call the `NewDay` subroutine for each company, resetting the number of visits for each company to zero 3. Get each company's reputation sc `urn`, adding it to a `totalReputation` local variable 4. For each company, append the cumul `reputation total to a list 5. Loop through each household in the si `tion 6. For each household, if an occupant will n `visit a restaurant (determined using a random number in `FindOutIfH` `eholdEatsOut`), set a local variable to a random integer between `ero` and the cumulative reputation score 7. Loop through the cumulative reputation list, `menting a zero-initialised variable `current` with each iterati 8. If the randomly generated number (step 6) is les `han the value in the List at index `current`, a visit is added to the compar `ndexed by `current` in the `companies` list, with the household's coor `tes passed to `AddVisitToNearestOutlet`
	Parameters: - Returns: – Called From: `Program.Main` Calls: `Simulation.DisplayMenu` `Settlement.DisplayHouseholds` `Simulation.DisplayCompanies` `Simulation.GetIndexOfCompany` `Simulation.ModifyCompany`	1. Initiates a loop that will repeat until the user enters 'Q `the prompt 2. If the user enters '1', call `DisplayHouseholds` (which is in the Settlement class) 3. If the user enters '2', call `DisplayCompanies` 4. If the user enters '3', prompt the user for a company name, which is passed to `GetIndexOfCompany`. The call to this subroutine will return either the integer index of the company, if it exists, or -1, if it doesn't. If -1

Program Classes

The classes defined in the program, and their attributes, are described below.

Class	Description
`Household`	An individual household wit[illegible]ttlement; each household has its own unique locatio[illegible]ach household has a probability of a person eating out each day
`Settlement`	A 1,000 by 1,000 grid, with some [illegible]ces within the grid (initially 250) occupied by household[illegible]ouseholds are stored within a list, meaning the empty 'cells' of t[illegible]rid are not themselves represented.
`LargeSettlement`	Inherits from settlement, but the gri[illegible]n be larger than 1,000 by 1,000, and the starting number[illegible]households can be larger than 250
`Outlet`	An individual restaurant, belonging to a[illegible]mpany. Although each outlet is a distinct object, all outl[illegible]of a company are of a particular category (i.e. all fast-food, all [illegible]ily or all named chef – never a combination).
`Company`	Each company consists of one or more o[illegible]ts, each of which is stored in a list. A company can close[illegible]outlet, expand an outlet or open a new outlet.
`Simulation`	'Manages' each day, processing events whi[illegible]re generated via random numbers. These events include[illegible]construction of new households within a settlement as well as ch[illegible]es to a company in cost or reputation.
`Program`	Contains the main subroutine to run the progra[illegible] which entails creating a new `Simulation` object

'Household' Attributes

Attribute	Type	Default Value	Description
`chanceEatOutPerDay`	Double	Random, between 0 and 1	Represents the probability that a member of the household will eat out on a given day

'Settlement' Attributes

The `LargeSettlement` class inherits from the Settlement class, but does not include any additional subroutines or attributes. Instead, it includes three additional parameters to the constructor, allowing any calling class to vary the initialisation values of `xSize`, `ySize` and `startNoOfHouseholds`.

Attribute	Type	Default Value	Description
`startNoOfHouseholds`	Integer	250	The number of households in the settlement at the beginning of the simulation
`xSize`	Integer	1,000	One dimension of the size of the settlement 'grid'. As with the 'x' dimension, it might be helpful to visualise this as the settlement's width, but the program does not store the grid in any form, such as a 2D array.
`ySize`	Integer	1,000	The other dimension for the size of the settlement
`households`	List of `Household` objects	Empty	Collection of all `household` objects within the settlement
`rnd`	Random	Static; initialised as a new `Random`	Random number generator, used to generate random locations to position households

'Outlet' Attributes

Attribute	Type	Default Value	Description
`visitsToday`	Integer	0	The number of times the outlet has been visited on the current day, which is incremented with each visit
`xCoord`	Integer	Passed as a parameter to constructor	The X coordinate of the outlet's location within the settlement
`yCoord`	Integer	Passed as a parameter to constructor	The Y coordinate of the outlet's location within the settlement
`capacity`	Integer	See →	The maximum number of visits that an outlet can receive, initialised to 60% of `maxCapacityBase`, which is a parameter passed to the Outlet class's constructor
`maxCapacity`	Integer	See →	The highest value that capacity can be increased to by way of calls to the class's

'Company' Attributes

Attribute	Type	Default Value	Description
name	String	Passed as a parameter to constructor	The name of the company
category	String	Passed as a parameter to constructor	The type of company – one of 'fast food', 'family' or 'named chef'
balance	Double	Passed as a parameter to constructor	The amount of money the company has
reputationScore	Double	See →	A measure of a company's reputation. It is initialised to 100, then altered depending on the category of the company, with named chef companies likely to have a higher score than family companies, which, in turn, are likely to have a higher score than fast-food companies. This is not guaranteed, however, as random numbers are used in these calculations
avgCostPerMeal	Double	See →	The average cost of a meal (as bought by the company), set to 5, 12 and 20 for fast-food, family and named chef companies respectively
avgPricePerMeal	Double	See →	The average price of a meal (as sold to a customer), set to 10, 14 and 40 for fast-food, family and named chef companies respectively
dailyCosts	Double	100	Part of a company's expenses, regardless of the number of outlets
familyOutletCost	Double	1,000	The cost of a family company opening a new outlet
fastFoodOutletCost	Double	2,000	The cost of a fast-food company opening a new outlet
namedChefOutletCost	Double	15,000	The cost of a named chef company opening a new outlet
fuelCostPerUnit	Double	Passed as a parameter to constructor	Used in the calculation of delivery to each outlet
baseCostOf	Double	Passed as a parameter to constructor	Used in the calculation of delivery to a company, regardless of the

Simulation Attributes

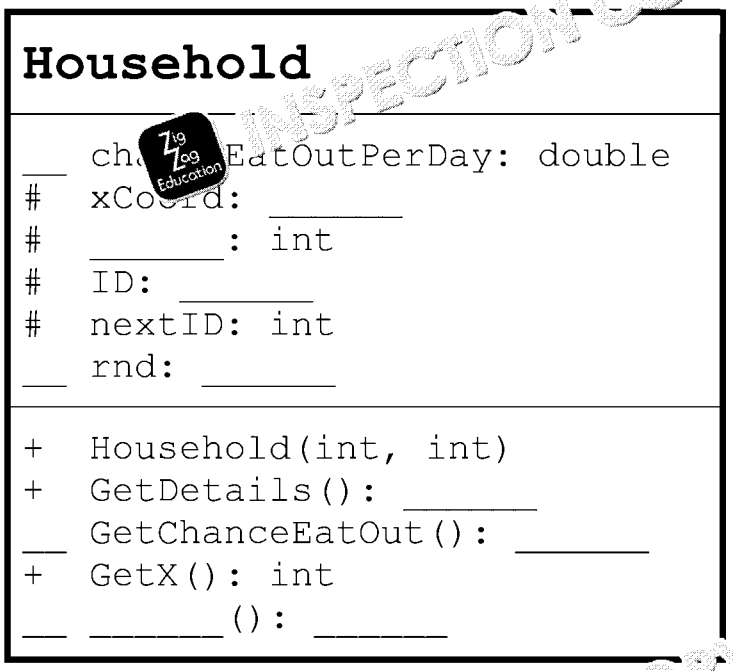
Attribute	Type	Default Value	Description
`simulationSettlement`	Settlement	See →	The single settlement for the game, which will be initialised to either a new instance of `Settlement` or a new instance of `LargeSettlement`, depending on user input
`noOfCompanies`	Integer	See →	The number of companies in the simulation, which is initialised to either 3 or a user-input value, depending on a menu selection
`fuelCostPerUnit`	Double	0.0098	Passed to the `Company` constructor to subsequently be used in calculations of delivery for each of a company's outlets
`baseCostForDelivery`	Double	100	Passed to the `Company` constructor to subsequently be used in calculations of delivery costs for a company
`companies`	List of `Company` objects	Empty	Collection of all of the simulation's companies
`rnd`	Random	Static; initialised as a new `Random`	Random number generator, used to generate random numbers throughout the Simulation class

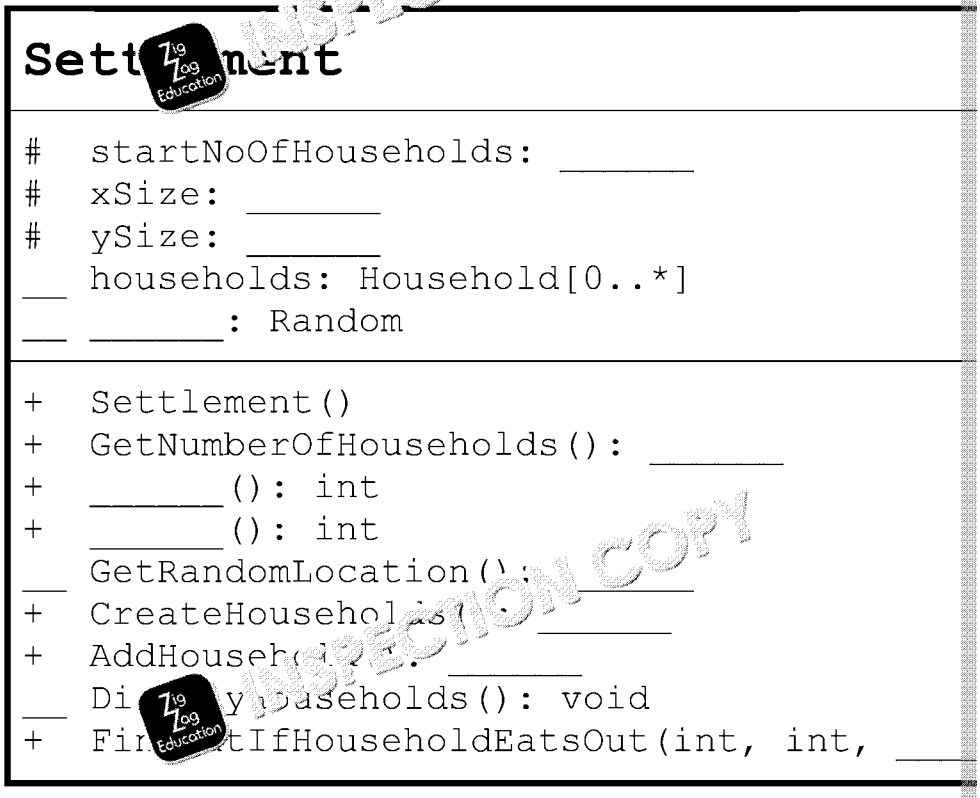
The class containing the *`Main`* subroutine has no attributes

Food Magnate Simulation

Class Diagram Tasks

Complete each of the following unfinished UML class diagrams. Each one is missing subroutines, access modifiers, parameters and return types.





Complete the UML class diagram for the `Company` class.
The first attribute and the first subroutine are provided for you.

Food Magnate Simulation

Programming Theory Questions

These questions refer to the preliminary material and require you to load the skel require any additional programming.

1. State the name of an identifier for:

   a) Two attributes i… `…ement` class that that would not be instantiat

   ……

   b) A subroutine in the `Settlement` class that returns something other tha

   ……

   c) A subclass [1]

   ……

   d) A local variable that is used to return a Boolean [1]

   ……

   e) A subroutine from the `Company` class that **cannot** be called from outsid

   ……

   f) A library string funct… …alle… …om the `GetIndexOfCompany` subroutir

   ……

   g) A collection attribute in the `Company` class [1]

   ……

   h) An instance of `Settlement` [1]

   ……

2. Showing and explaining your working, give the probability of a call to `Proces` being made from the `DisplayEventsAtDayEnd` subroutine in the `Simula`

   ……

   ……

   ……

   ……

   ……

3. Explain how validation might be added to the `OpenOutlet` subroutine of the new outlet being created beyond the bounds of the settlement. Remember, y code. [3]

.....

.....

.....

.....

.....

4. Each `Household` object is stored within a list called `Households`. Describe been used instead of a list to store `Household` objects. [3]

.....

.....

.....

.....

.....

5. Describe in full how the `GetDistanceBetweenTwoOutlets` subroutine o the distance between two outl

.....

.....

.....

.....

.....

.....

6. Explain the role of the object of type `Random` in the `Household` class. [2]

.....

.....

.....

7. Explain the role of the variable `upOrDown` in the `ProcessCostOfFuelCha` `Simulation` class. [3]

.....

.....

.....

.....

.....

8. In the `S...ration` constructor, the integer literals 100000, 200 and 203 are constructor when creating the 'AQA Burgers' company. State the role of each

.....

.....

.....

.....

.....

9. Describe in full the operation of the `GetIndexOfC...pany` subroutine in the

.....

.....

.....

.....

.....

.....

.....

.....

10. Describe the circumstances under which the `Modify...pany` subroutine of output the text 'Invalid coordinates'. [3]

.....

.....

.....

.....

.....

11. Currently, a call to the `LargeSettlement` constructor could not result in a s
1,000 by 1,000. This is true even if negative numbers are entered by the user
and y values.

Explain how a call to the `LargeSettlement` constructor never results in a si

.....

.....

.....

.....

.....

12. Describe the concept of constructor overloading, and explain how constructor
used instead of inheritance for the creation of a new large settlement. [4]

.....

.....

.....

.....

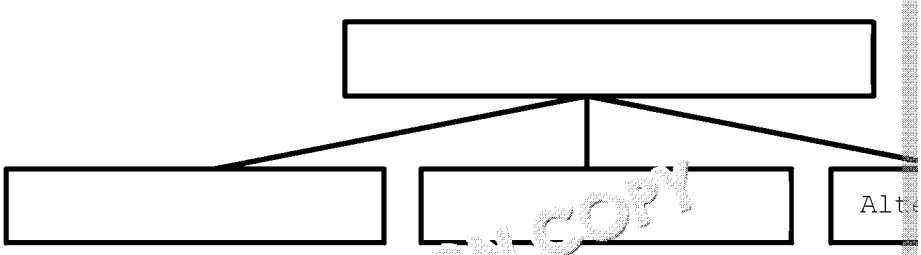
.....

.....

.....

.....

13. Complete the following hierarchy chart for part of the `Simulation` class of t
You should **not** include calls to any library subroutines. [3]



14. Describe how the pr[illegible] respond to a call to the `Company` construct
neither 'f[illegible]' nor 'named chef'. [2]

.....

.....

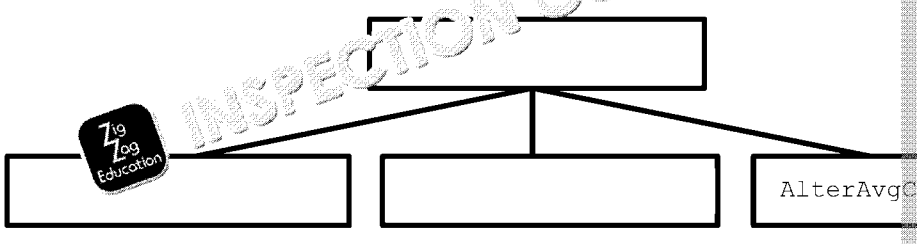
.....

Food Magnate Simulation

Programming Theory Questions

These questions refer to the preliminary material and require you to load the skel require any additional programming.

1. State the name of an identifier for:
   a) Two attributes in the Sett[illegible]n class that would not be instantiated ir
   b) A subroutine in th[illegible]ent class that returns something other tha
   c) A sub-class [illegible]
   d) A [illegible]ariable that is used to return a Boolean [1]
   e) A s[illegible]utine from the Company class that **cannot** be called from outside
   f) A library string function called from the GetIndexOfCompany subrouti
   g) A collection attribute in the Company class [1]
   h) An instance of Settlement [1]
2. Showing and explaining your working, give the probability of a call to Proces being made from the DisplayEventsAtDayEnd subroutine in the Simula
3. Explain how validation might be added to the OpenOutlet subroutine of the new outlet being created beyond the bounds of the settlement. You do **not** n
4. Each Household object is stored within a list called Households. Describe been used instead of a list to store Household objects. [3]
5. Describe in full how the GetDistanceBetweenTwoOutlets subroutine o the distance between two outlets. [4]
6. Explain the role of the object of type Ra[illegible]n[illegible] Household class. [2]
7. Explain the role of the variab[illegible]wn in the ProcessCostOfFuelCha Simulation clas[illegible]
8. In the [illegible]ation constructor, the integer literals 100000, 200 and 203 are constru[illegible]hen creating the 'AQA Burgers' company. State the role of each
9. Describe in full the operation of the GetIndexOfCompany subroutine in the
10. Describe the circumstances under which the ModifyCompany subroutine of output the text 'Invalid coordinates'. [3]
11. Currently, a call to the LargeSettlement constructor could not result in a se by 1,000. This is true even if negative numbers are entered by the user when pr values. Explain how a call to the LargeSettlement constructor never results
12. Describe the concept of constructor overloading, and explain how constructor used instead of inheritance for the creation of a new large settlement. [4]
13. Copy and complete the following hierarchy chart for pa[illegible]f the Simulation You should **not** include calls to any library sub[illegible] [3]



14. Describe how the program would respond to a call to the Company construct neither 'fast food', 'family' nor 'named chef'. [2]

Food Magnate Simulation

Programming Tasks

The following require you to open the skeleton program and make modifications

Task 1

This questi[illegible] subroutine `ModifyCompany` within the `Simulat`

Currently, th[illegible] is prompted to enter a value of 1, 2 or 3, but if nothing is en responds by outputting a blank line.

Change the subroutine `ModifyCompany` to present the user with an addition user enters anything other than 1, 2, 3 or an upper-case 'C', the menu should b either 1, 2, 3 or C is selected. If 1, 2 or 3 is entered, `ModifyProgram` shoul upper-case 'C' is entered, the program should output 'Operation Cancelled', an terminate without executing any additional code.

Test that the changes you have made work:

- run the **Skeleton Program**
- leave the first prompt blank, to indicate a normal-si[illegible]ed settlement
- enter D at the next prompt for default com[illegible]
- enter 3 for 'modify company'
- enter 'AQA Burgers' whe[illegible] for a company name
- enter 'X' at the fi[illegible] of the 'modify company' submenu
- ent[illegible]at [illegible] second prompt of the 'modify company' submenu

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine `Modi`

b. SCREEN CAPTURE(S) showing the required test

Task 2

This question refers to the subroutine `GetRandomLocation` within the `Set`

This subroutine generates a random location within the bounds of the settleme
new household. Currently, there is no mechanism for ensuring that a new hou
location of an existing household.

Change the subroutine `GetRandomL...on` to ensure that only unoccupie
to returning the location, a ch... should be made to determine whether the loc
household. If it is al... occupied, a new location should be generated, repe

Test that th... ges you have made work:

- modify the Settlement constructor in the following ways:
  - change `xSize = 1000;` to `xSize = 3;`
  - change `ySize = 1000;` to `ySize = 3;`
  - change `startNoOfHouseholds = 250;` to `startNoO`
- run the **Skeleton Program**
- leave the first prompt blank, to indicate a normal-sized settlement
- enter D at the next prompt for default companies
- enter 1 for 'display details of households'

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine `GetR`

b. S... CAPTURE(S) showing the required test

Task 3

This question refers to the subroutine `ExpandOutlet` within the `Company` c
subroutine, `ExtendCapacity`, within the `Outlet` class.

Currently, each outlet has a limit, beyond which it cannot be expanded. Any at
limit results in the capacity being set at the limit itself.

Create a new subroutine in the `Outlet` class called `ExtendCapacity` whi
parameter. When this subroutine is called, the value of the attribute `maxCapa`
the value of this parameter.

Change the `ExpandOutlet` subroutine so that, in the event of an attempt to
`maxCapacity` value, a random number is generated. Using this number, the
calling the new subroutine, `ExtendCapacity`, and passing a value of 2, a 3
passing a value of 3, and a 25% chance of calling it and passing a value of 4.
capacity expanded' should be displayed in these circumstances, with no other

Test that the changes you have made work:

- run the **Skeleton Program**
- leave the first prompt blank, to indicate a normal-sized settlement
- enter D at the next prompt for default companies
- enter 2 for 'display details of companies', ensuring that outlet 4 for 'Palt
  screenshot
- enter 3 for 'modify company'
- enter 'Paltry Poultry' when prompted for company name
- enter 3 for 'expand outlet'
- enter 4 when prompted for an ID
- enter 200 when asked for an amount
- enter 2 for 'display details of companies', ensuring that outlet 4 for 'Palt
  screenshot

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine `Expa`

b. Your PROGRAM SOURCE CODE for the new subroutine `ExtendCa`

c. SCREEN CAPTURE(S) showing the required test

Task 4

This question refers to the subroutine `ProcessDayEnd` within the `Simulat` subroutine, `ProcessLeavers`, within the `Settlement` class.

Currently, there is no mechanism for households to leave a settlement.

Create a new subroutine in the `Settlement` class called `ProcessLeaver` accept no parameters and should return an integer value. Each household in t subject to a random 2% chance of leaving the settlement. The `ProcessLea` remove households that are applicable from the households data structure, and retu that were removed.

Modify the subroutine `ProcessDayEnd`, so that the final instructions are to c output the number of households that left the settlement.

Test that the changes you have made work:

- run the **Skeleton Program**
- leave the first prompt blank, to indicate a normal-sized settlement
- enter D at the next prompt for default companies
- enter 6 for 'advance to next day'

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine `Proc`

b. Your PROGRAM SOURCE CODE for the new subroutine `ProcessL`

c. SCREEN CAPTURE(S) showing the required test

Task 5

This question refers to the subroutine `Run` within the `Simulation` class.

Currently, when modifying a company, the user needs to enter a company nam
either remember it or scroll up the console window to see it previously displaye

Change the subroutine `Run` so that when the u er ..nt.s option 3 from the ma
they are presented with a numbered l.s. o. .nes of companies. The first com
should be displayed next to a mb.r 1, even though its index in the `compani`

When the ..te. the number next to the company name, the program sho
as it would .. one had the company's name been entered.

Entering the company's name should no longer be effective, and the input mes
number next to the company you wish to modify'.

Test that the changes you have made work:

- run the **Skeleton Program**
- leave the first prompt blank, to indicate a normal-sized settlement
- enter D at the next prompt for default companies
- enter 3 for 'modify company'
- enter 3 when asked for a number
- enter 3 for 'expand outlet'
- enter 4 for the ID
- enter 1000 for the amount
- enter 2 for 'display details of c.mp..s'

Evidence you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine `Run`

b. SCREEN CAPTURE(S) showing the required test

Task 6

This question refers to an additional constructor for the `Outlet` class.

Currently, when a new outlet is constructed, this takes place by way of X and Y
the `Outlet` constructor. In this task, you will create an additional constructor
placement.

Without making any changes to the existing constructor, create an additional c
class. This constructor should take as parameters three integers – `maxX`, `max`
and a Boolean `isDefault`. The `maxX` and `maxY` integers represent the high
outlet could possibly have within its settlement.

The constructor should contain code that sets `xCoord` to a random integer val
inclusive, as well as code that sets `yCoord` to a random integer value betwee
other attributes of the `Outlet` class should be set identically to those of the e

Test that the changes you have made work:

- Modify the call to the `Outlet` class's constructor within the subroutine `Company` class; it should read as follows:
  - `Outlet newOutlet = new Outlet(5, 10, capac`
- run the **Skeleton Program**
- leave the first prompt blank, to indicate a normal-sized settlement
- enter D at the next prompt for default companies
- enter 2 for 'display details of companies'

Evidence that you need to provide:

a. You PROGRAM SOURCE CODE for the new `Outlet` constructor

b. SCREEN CAPTURE(S) showing the required test

Task 7

This question refers to the subroutines `DisplayMenu` and `Run` within the `Si`

Currently, the program allows the user to advance the simulation for multiple d selecting option 6 from the menu.

Change `DisplayMenu` to include an additional option '5. Advance'.

Change `Run` so that if option 5 is selected, the user is prompted for the numbe simulation to advance by. This number, which does not require validation, will be subroutine `DayEnd` is called.

Test that the changes you have made work:

- run the **Skeleton Program**
- leave the first prompt blank, to indicate a normal-sized settlement
- enter D at the next prompt for default companies
- enter 5 for 'advance'
- enter 3 when asked for a number

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine `Run`

b. Your PROGRAM SOURCE CODE for the amended subroutine `Disp`

c. SCREEN CAPTURE (S) showing the required test

Task 8

This question refers to the subroutine `AddCompany` within the `Simulation`

Functionality will be added to allow a company type to be assigned at random.

Change the `AddCompany` subroutine so that the user is prompted to enter 1, should be updated to indicate that '4' means a randomly chosen type of compa

If 1, 2 or 3 is entered, the program should continue as before. If '4' is entered, used to determine which of the three types of company will be created. Each t equally likely to be created.

Test that the changes you have made work:

- run the **Skeleton Program**
- leave the first prompt blank, to indicate a normal-sized settlement
- leave the second prompt blank, to indicate user-defined companies
- enter 1 when asked for a number of companies
- enter the company name 'Random Restaurant'
- enter a starting balance of 50000
- enter 4 to indicate a 'random' new company
- enter 2 for 'display details of companies'

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine `AddC`

b. SCREEN CAPTURE(S) showing the required test

Task 9

This question refers to the subroutine `ProcessDayEnd` in the `Company` clas
subroutine, `CloseAllOutlets`, also in the `Company` class.

Currently, a company can continue operating irrespective of how far below zer

Create a new subroutine called `CloseAllOutlets` in the `Company` class,
outlets belonging to the company and closing them with a call to `CloseOutl`

Modify the subroutine `ProcessDayEnd` in the `Company` class, so that imme
statement, the value of the `balance` field is checked. If it is below zero, a cal
made, and a message is output to notify the user that all outlets have closed.

Test that the changes you have made work:

- run the **Skeleton Program**
- leave the first prompt blank, to indicate a normal-sized settlement
- leave the second prompt blank, to indicate user-defined companies
- enter 1 when asked for a number of companies
- enter the company name 'Bankrupt Burgers'
- enter a starting balance of 1000
- enter 2 to indicate a family restaurant
- enter 6 in the main menu, 'advance to next day'
- enter 2 in the main menu, 'display details of companies'

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine `Modi`

b. Your PROGRAM SOURCE CODE for the new subroutine `CloseAll`

c. SCREEN CAPTURE(S) showing the required test

Task 10

This question refers to a new class, called `FoodTruck`, as well as the subrou
`Company` class.

Create a new class, called `FoodTruck`, which inherits from the class `Outle`
subroutines and attributes, `FoodTruck` should inc... a new subroutine calle
take place in a random direction, moving ... north, south, east or wes
the food truck's movement, which is ... to leave the settlement as a res
values being beyond the ... bounds.

The constr... `FoodTruck` should take `xCoord` and `yCoord` integers
to the super... constructor along with a value of 10 for capacity.

Change `ProcessDayEnd` so that the `move` subroutine is called for any outle
`FoodTruck`.

Test that the changes you have made work:

- Modify the call to the `Outlet` class's constructor within the `OpenOut`
  `Company` class; it should read as follows:

  ```
  FoodTruck newOutlet = new FoodTruck(x, y);
  ```

- run the **Skeleton Program**
- leave the first prompt blank, to indicate a ... sized settlement
- leave the second prompt blank, to ... user-defined companies
- enter 1 when asked for a ... companies
- enter the compa... 'Taco Truck'
- ent... ... ance of 30000
- ent... indicate a fast-food restaurant
- enter 2 to display details of companies
- enter 6 to advance to the next day
- enter 2 to display details of companies

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the new class `FoodTruck`

b. Your PROGRAM SOURCE CODE for the amended subroutine `Proc`

c. SCREEN CAPTURE(S) showing the ... test, ensuring that the l
truck is visible after each ... the main menu

Task 11

This question refers to the subroutine `GetIndexOfCompany` within the `Sim`

Currently, when a company is searched for using this subroutine, the whole co
order to generate a match.

Change `GetIndexOfCompany` so that if the user enters a search term that
one company, the index of that company is returned. If the text is contained wi
companies, the user should be presented with all matching company names be
of them in full in order of index.

The subrou ould continue to be non-case-sensitive, and a search for a c
an attempt to select a matching company that doesn't actually match one of the
return a value of -1.

Test that the changes you have made work:

- run the **Skeleton Program**
- leave the first prompt blank, to indicate a normal-sized settlement
- enter D at the next prompt for default companies
- enter 3 for 'modify company'
- enter a lower-case 't' for the company name
- type 'Paltry Poultry' when asked to type the name of a company

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine `GetI`

b. SCREEN CAPTURE(S) showing the required test

Task 12

This question refers to the subroutine `CloseOutlet` in the `Company` class.

Currently, closing an outlet incurs no expense on the part of the company.

Change `CloseOutlet` so that a company's balance decreases for each outlet closing the outlet depends on both the type of the company and the capacity of the closed. Taking 'capacity' as being the number of seats in an outlet, the costs are as follows:

Fast-food outlet: 75 per seat

Family outlet: 50 per seat

Named chef outlet: 150 per seat

Test that the changes you have made work:

- run the **Skeleton Program**
- leave the first prompt blank, to indicate a normal-sized settlement
- enter D at the next prompt for default companies
- enter 2 for 'display details of companies
- enter 3 for 'modify company'
- enter 'Paltry Poultry'
- enter 2 for 'close outlet'
- enter 4 when prompted for an ID
- enter 2 for 'display details of companies'

Evidence you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine `Clos`

b. SCREEN CAPTURE(S) showing the required test, ensuring all details outlets are visible after each request of 'display details of companies'

Task 13

This question refers to the subroutines `DisplayMenu` and `Run` in the `Simul`

Currently, it is possible to add a company to a simulation, but not to remove on

Change `DisplayMenu` to include an additional opti[illegible] Remove company'

Change `Run` so that if option 5 is selected, the user is prompted for the name remove. After the user has entered the name of the company, that company's `GetIndexOfCompany`. The program should repeatedly ask them for a com name has been entered, or 'cancel' (any combination of upper case and lower

If 'cancel' is entered, the user should be returned to the main menu. Otherwise matching the user entry should be removed from the simulation, and the user s menu.

Test that the changes you have made work:

- run the **Skeleton Program**
- leave the first prompt blank, to indicate a normal-sized settlement
- enter D at the next prompt for default companies
- enter 5 for 'remove company'
- type CANCEL (all in upper case) when prompted for the name of a com
- enter 5 for 'remove company'
- type 'Paltry Poultry' when prompted again for the name of a company
- type 2 for 'display details of companies'

Evidence you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine `Disp`

b. Your PROGRAM SOURCE CODE for the amended subroutine `Run`

c. SCREEN CAPTURE(S) showing the required test

Task 14

This question refers to the subroutines `DisplayMenu` and `Run` in the `Simul`
new subroutines: `RunToTarget` in the `Simulation` class, and `GetBalar`

Currently, the simulation runs day by day, regardless of the effects of any chan

Create a new subroutine in the `Company` class ca`GetBalance`, which s
and return the value of the balanc

Create a new subroutine in the `Simulation` class called `RunToTarget`. T
the user fo and lower limits, storing them in integer variables called `upp`
`lowerLimit`. No validation is required for user input.

The simulation should run, via repeated calls to `ProcessDayEnd` in the `Sim`
company has a balance either equal to or above `upperLimit` or equal to or
calls to the new subroutine `GetBalance`. At this point, there should be no ac
`ProcessDayEnd`, and the program should output the name of the company,
days that have elapsed since the beginning of the simulation.

In the event that multiple companies reach `upperLimit` and/or `lowerLimi`
program need only display the details of one of the companies.

Change `DisplayMenu` to include an additional option: '5. Run to target'.

Change `Run` so that if the user enters option 5, a call is made to `RunToTarg`

Test that the changes you have made work:

- run the **Program**
- leave the first prompt blank, to indicate a normal-sized settlement
- enter D at the next prompt for default companies
- enter 5 for 'run to target'
- enter 0 when prompted for the lower limit
- enter 100000 (one hundred thousand) when prompted for the upper lim

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended subroutine `Disp`

b. Your PROGRAM SOURCE CODE for the amended subroutine `Run`

c. Your PROGRAM SOURCE CODE for the new subroutine `RunToTar`

d. Your PROGRAM SOURCE CODE for the new subroutine `GetBalar`

e. SCREEN CAPTURE(S) showing the required test; only the final 'even
be included

Task 15

This question refers to a new class called `CitySettlement`, as well as the `Simulation` class.

Currently, a new `Settlement` object is constructed by way of calls to the cor `Settlement` or `LargeSettlement`.

Create a new class called `CitySe...e...nt`, which inherits from `LargeSe` should have the same pa... as the `LargeSettlement` constructor, a `LargeSett...me...` ...structor, passing on its own parameters.

`CitySett...ment` should include a subroutine called `AddHousehold`, whi `AddHousehold` subroutine of the `Settlement` class. This new subroutine location within the city, then select a random integer between 2 and 20 inclusiv households should then be created and added to the settlement at the single lo

This is to model families living in apartment blocks. In this class, creating a set should have the effect of creating 100 such apartment blocks, with each block households. This means that a 250-household `CitySettlement` will conta as it should contain 250 blocks of households.

Change the constructor of the `Simulation` class to present the user with a c normal-sized settlement' or 'a city settlement' (the order is unimportant). If a ci user should be prompted for additional x-size, y-size and h...useholds values. sent as parameters to a call to the new `CitySe...e...nt` constructor.

Test that the changes you have ...t...o x.

- run th... **ke... ...ogram**
- at t... ...mpt, indicate a city settlement
- ente... value of zero for each of the additional prompts
- enter D for default companies
- enter 1 for 'Display details of households'

Evidence that you need to provide:

a. Your PROGRAM SOURCE CODE for the amended `Simulation` c

b. Your PROGRAM SOURCE CODE for the new class `CitySettlem`

c. Your PROGRAM SOURCE CODE for ...e `...dHousehold` subrouti `Settlement` class

d. SCREEN C... ...E(S) showing the required test, ensuring at least th ho... ...ld... ...re visible

Food Magnate Simulation

Additional Programming Tasks (Extension)

These challenges are presented without solutions, and offer to further explore and u

1. Add validation any user input to ens[ure] that something is entered and [that] nothing is entered
2. Add validation according to type, such that inputs required to be numer
3. Create additional category of restaurant, with a new company of tha default companies
4. Generate random likelihood of an outlet being closed as a day progres
5. Create a random instance of a company or an outlet running a promoti expenses go up but its reputation score also rises
6. Close an outlet that runs at a loss for five consecutive days, adding a n happened to the events
7. Display details of the restaurant which, during a day, was either the mo or the one with the highest reputation rating
8. Prevent a new outlet being opened within a certain distance of another the same type
9. Redesign `Company` to be an abstract class, with categories of compa
10. Generate a random budget and store it as an attribute within each `Hou` that e[ach] will only eat out with a company whose prices are within t
11. Incorporate weather into the simulation; it can rain at random, in which eating out are all halved, and the presence of rain is indicated within th
12. Generate random events, such as power cuts, fuel shortages and festi an impact on the probability of each house eating out
13. Incorporate a text file from which initial companies and outlets are crea the default companies), rather than having the user manually enter the run
14. Add a feature in which meals can be delivered, rather than customers who choose not to eat out might, according to random chance, order fo this costs the company extra in fuel according to the distance of deliver
15. The named chef outlets generally make a profit, while other categories subroutine to determine, for each type of company, how much an outle meal during the previous day in order to have operated at a profit

Food Magnate Simulation

Class Diagram Tasks – Solution

Shaded areas indicate the correct answers. Any differences in with case and spaci
misspellings should not be awarded a mark. Accept 'integer' in lieu of 'int'.

Household

```
#  chanceEatOutPerDay: double
#  xCoord: int
#  yCoord: int
#  ID: int
#  nextID: int
-  rnd: Random
---
+  Household(int, int)
+  GetDetails(): String
+  GetChanceEatOut(): double
+  GetX(): int
+  GetY(): int
```

Settlement

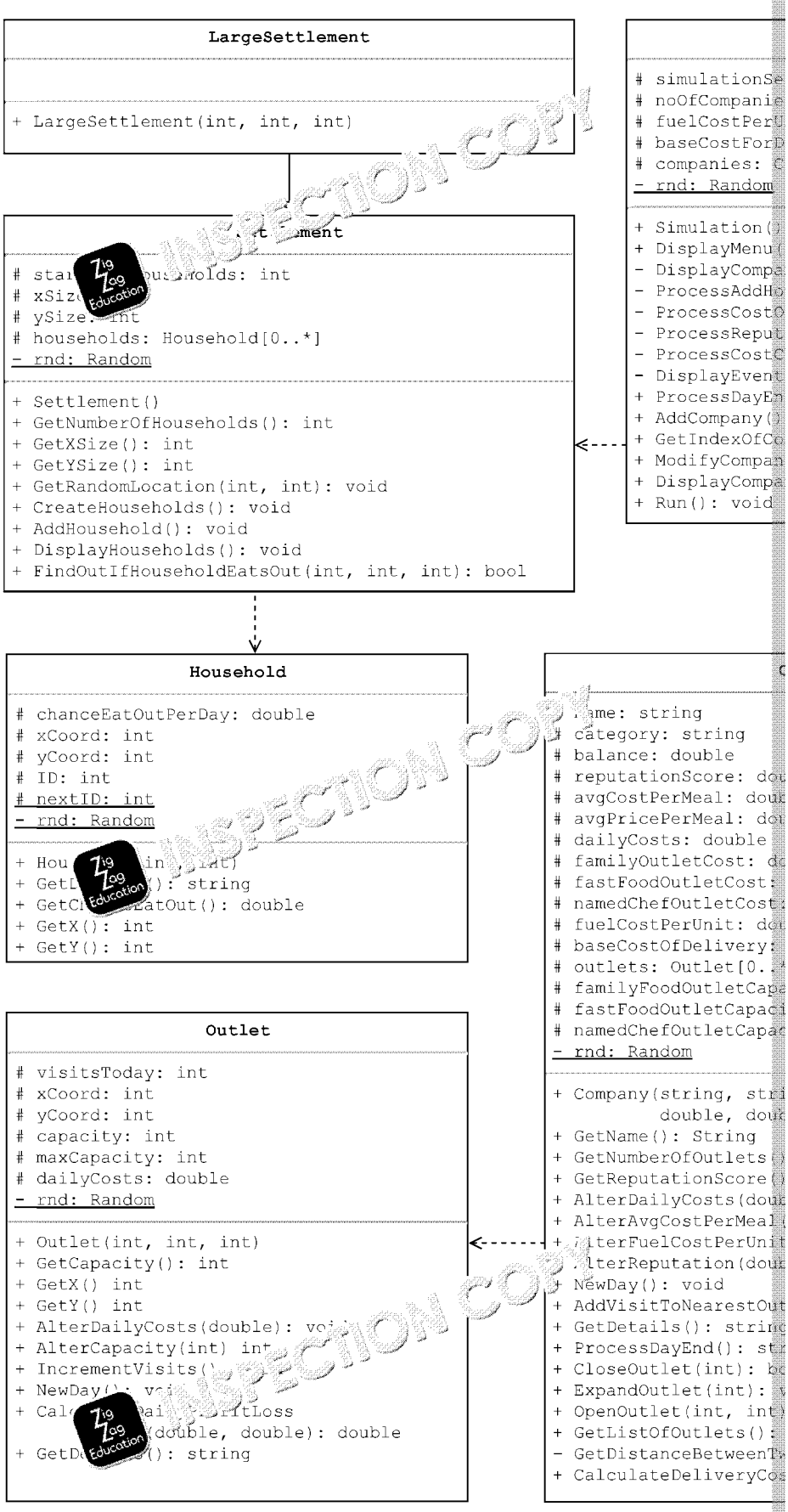
```
#  staticNumberOfHouseholds: int
#  xSize: int
#  ySize: int
#  households: Household[0..*]
-  rnd: Random
---
+  Settlement()
+  GetNumberOfHouseholds(): int
+  GetXSize(): int
+  GetYSize(): int        *fine if getXSize and getYSize are the ot
+  GetRandomLocation(): int[2]
+  CreateHouseholds(): void
+  AddHousehold(): void
+  DisplayHouseholds(): void
+  FindOutIfHouseholdEatsOut(int, int, int)
```

For the `Company` class, the first attribute and the first two subroutines are included i
marked. All other lines are worth two marks each, one for the content before the col
the colon. You can accept 'integer' in lieu of 'int'. The alternating grey rows are only
Penalise for misspellings and any extra content within a line; do not penalise for capi
attention is drawn to the private access level of one attribute and two subroutines. P

Company

```
# name: String
# categ: String
# bala double
# reput onScore: double
# avgCostPerMeal: double
# avgPricePerMeal: double
# dailyCosts: double
# familyOutletCost: double
# fastFoodOutletCost: double
# namedChefOutletCost: double
# fuelCostPerUnit: double
# baseCostOfDelivery: double
# outlets: Outlet[0..*]
# familyFoodOutletCapacity: int
# fastFoodOutletCapacity: int
# namedChefOutletCapacity: int
- rnd: Random
```

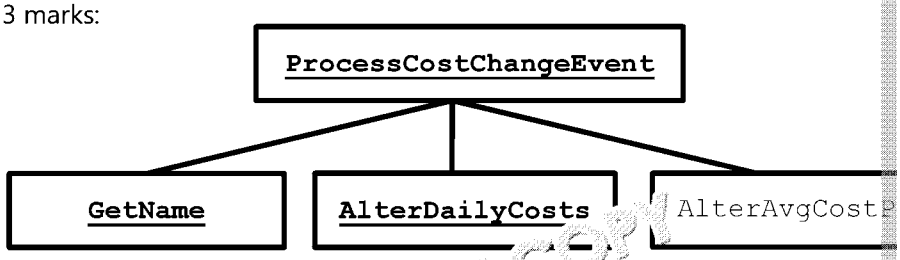
```
+ Company(String, Strin ing, int, int, doubl
+ GetName(): Strin
+ GetNumberOf e s(): int
+ GetR at onScore(): double
+ Alte yCosts(double): void
+ AlterAvgCostPerMeal(double): void
+ AlterFuelCostPerUnit(double): void
+ AlterReputation(double): void
+ NewDay(): void
+ AddVisitToNearestOutlet(int, int): void
+ GetDetails(): String
+ ProcessDayEnd(): String
+ CloseOutlet(int): boolean
+ ExpandOutlet(int): void
+ OpenOutlet(int, int): void
+ GetListOfOutlets(): int[0..*]
- GetDistanceBetweenTwoOutlets(in t): double
+ CalculateDeliveryCost(): do le
```



Food Magnate Simulation

Programming Theory Questions – Mark Scheme

Q	Answer/Guidance
1a	1 mark for each of: • `rnd` • `nextID`
1b	1 mark • `tR[illegible]Location`
1c	1 mark • `LargeSettlement`
1d	1 mark: • `closeCompany`
1e	1 mark for of: • `GetDistanceBetweenTwoOutlets`
1f	1 mark for of: • `ToLower`
1g	1 mark: • `outlets`
1h	1 mark: • `simulationSettlement`
2	3 marks: • 1 mark for showing calc[illegible] 0.25 * 0.5 • 1 mark for explaining: 0.5 probability nested in selection statement f[illegible] any explanation that makes clear that the 0.5 probability is only eval[illegible] • 1 mark for correct probability: 0.125 / 12.5% / $^1/_8$ / 1 in 8
3	3 marks: • (Dimensions of) `Settlement` must be passed (as a parameter) • Selection/if statement to compare x and y to dimensions of settleme • Creation/construction of outlet inside the selection/if structure (credit can be given for code that is described, but not code in isolation)
4	3 marks: • *HashMap* uses key and value pairs • An index would be the key; the `Household` object would be the va • Households would be unordered // no guarantee of order
5	4 marks: • One X (or Y) coordinate is subtracted from the other • Result of this is squared • Square root of this is found [illegible] mark and the previous mark clear that the distance is always positive) • Result is added to the result of the same process performed on Y (o
6	2 ma[illegible] • [illegible]erates a random double/fraction • To set `chanceEatOutPerDay` // to set the probability of eating o

Q	Answer/Guidance
7	3 marks • Randomly set to either 0 or 1 • Determines whether fuel cost goes up or down // if it is set to zero, • If it is set to 1, fuel cost goes down (last two marks can be given if reference is instead mad to variable `fuelCo`
8	3 marks: • 100,000: company's balan • 200: x position of company / first outlet • 203: y po of company / first outlet (last cannot be awarded if they simply refer to 'x' and 'y' – these must b
9	5 mark • Loop through all companies (in companies list) • Convert search term / parameter / company name to lower case • Compare this with (lower-case version of) each company's name / ca • If there is a match, the index / value of current / company's location • If loop ends without a match, -1 is returned
10	3 marks: • User enters 1 / selects 'open new outlet' at the menu • X coordinate is outside the range of the settlement • **or** Y coordinate is within the range of the settlement (for last 2 marks, there must be a clear expression that either x or y being be the settlement would trigger the message)
11	3 marks: • `LargeSettlement` constru calls `Settlement`/superclass co • `Settlement` const (x and y) values of 1,000 when placi • Reduction of `Size` would take place after this // houses a co value of 1,000, even if those values are subsequently re
12	4 ma • Constructor overloading is multiple constructors in a single class • Each constructor requires a different signature / order of parameter • `Settlement` class could have used constructor overloading • `LargeSettlement` would need (in addition to `Settlement` con parameters for (additional) size and number of households
13	3 marks:  1 mark per underlined term; ord of `Name` and `AlterDailyCosts` is must be on the bottom for the mark.
14	2 mark • ' clause would be executed • alues for a named chef restaurant would be used // variables woul 20, 40 and a random double multiplied by 50
TOTAL MARKS	

Food Magnate Simulation

Programming Tasks – Suggested Mark Scheme

Note that the following are recommended solutions, and not an exhaustive list of all
The marking guidance should be used as a guide only. Discretion should be used in
alternative solutions are given.

Task 1

1 mark loc p ModifyCompany to repeat until something other than 'C'

1 mark loop contain call to menu display and user input, and must not begin

```
string choice = "";
int outletIndex, x, y;
bool closeCompany;
while (choice != "C" && choice != "1" && choice != "2" && choice !=
{
    Console.WriteLine("\n**********************************");
    Console.WriteLine("*******  MODIFY COMPANY   *******");
```

1 mark selection statement to catch entry of an upper-case 'C'

1 mark 'operation cancelled' displayed within selection clause

1 mark all inputs, old and new, dealt with correctly

```
        else
        {
            Console.WriteLine("Invalid to  tes.");
        }
    } else if (choice ==  )
    {
        C    e.  ) eline("Operation Cancelled");
    }
}
```

1 mark screenshot showing 'X' causing a repeat of the loop and 'C' causing redispl

```
Enter your choice: X

**********************************
*******  MODIFY COMPANY   *******
**********************************
1. Open new outlet
2. Close outlet
3. Expand outlet

Enter your choice: C
Operation Cancelled


**********************************
**********                **********
**********************************
1. Displ  etails of households
2. Display details of companies
3. Modify company
4. Add new company
6. Advance to next day
Q. Quit
```

Task 2

1 mark Boolean or equivalent to store whether a collision has occurred

1 mark outer loop to continue until a location is returned, even if location is invalid

1 mark random location generated inside outer loop

1 mark inner loop to iterate through households list

1 mark selection statement to determine whether location is already occupied

1 mark flag set in inner loop to store that a location was already occupied

1 mark Boolean (placed / valid) set to reflect no collision occurring

1 mark return statement will only return a non-collision location

```
public void GetRandomLocation(ref int x, ref int y)
{
    bool placed = false;
    bool collision;

    while (placed == false)
    {
        collision = false;
        x = Convert.ToInt32(rnd.NextDouble() * xSize);
        y = Convert.ToInt32(rnd.NextDouble() * ySize);

        for (int count = 0; count < households.Count; count++)
        {
            if ((households[count].GetX() == x) &&
                    (households[count].GetY() == y))
            {
                collision = true;
            }
        }
        if (collision == false)
        {
            placed = true;
        }
    }

}
```

1 mark eight household locations are all different:

```
1     Coordinates: (1, 0)     Eat out probability: 0.889230
2     Coordinates: (3, 0)     Eat out probability: 0.131675
3     Coordinates: (2, 0)     Eat out probability: 0.301126
4     Coordinates: (0, 2)     Eat out probability: 0.983540
5     Coordinates: (2, 2)     Eat out probability: 0.698545
6     Coordinates: (2, 1)     Eat out probability: 0.301843
7     Coordinates: (3, 3)     Eat out probability: 0.915217
8     Coordinates: (1, 3)     Eat out probability: 0.066149
```

***NB** Owing to the way in which random values are generated in the skeleton code, values may be displayed in the terminal window. This could happen if the existing random number generator can still be used, provided that each location is unique.*

Task 3

1 mark new subroutine declared correctly in the `Outlet` class

1 mark `maxCapacity` multiplied by the parameter (name of parameter unimport

```
public void ExtendCapacity(int multiplier)
{
    maxCapacity *= multiplier;
}
```

1 mark call to generate a random n[illegible] that could be appropriate to the 40/35/25

1 mark probabilities ar[illegible] 40%, 35% and 25%

1 mark cor[illegible]arameter passed to `ExtendCapacity` in all circumstances

1 mark only output message in these circumstances is 'outlet max capacity expand

```
if (result == change)
{
    Console.WriteLine("Capacity adjusted.");
}
else
{
    int multiplier = rnd.Next(0, 100);
    if (multiplier < 40) {
        outlets[ID].ExtendCapacity(2);
    } else if (multiplier < 75) {
        outlets[ID].ExtendCapacity(3);
    } else {
        outlets[0].ExtendCapacity(4);
    }
    Console.WriteLine("Outlet max capacity expanded");
}
```

1 mark correct messa[illegible]ut, and Paltry Poultry outlet 4 max capacity doubled,

```
Number of outlets: 4
Outlets
1. Coordinates: (800, 390)     Capacity: 120      Maximum Ca
2. Coordinates: (400, 390)     Capacity: 120      Maximum Ca
3. Coordinates: (820, 370)     Capacity: 120      Maximum Ca
4. Coordinates: (800, 600)     Capacity: 120      Maximum Ca
```

```
Enter ID of outlet: 4
Enter amount you would like to expand the capacity by: 1000
Outlet max capacity expanded
```

```
Number of outlets: 4
Outlets
1. Coordinates: (800, 390)     Capacity: 120      Maximum Ca
2. Coordinates: (400, 390)     Capacity: 120      Maximum Ca
3. Coordinates: (820, 370)     Capacity: 120      Maximum Ca
4. Coordinates: (800, 600)     Capacity: 213      Maximum Ca
```

Task 4

1 mark new subroutine, correct name, type and parameters in the `Settlement` c

1 mark integer to store number of leavers

1 mark loop to iterate over all households

1 mark selection structure, based on 2% probability

1 mark removal of household from list, inside selection clause

1 mark integer variable incremented inside selection clause, and returned after loo

```
public int ProcessLeavers()
{
    int leavers = 0;
    for (int index = 0; index < households.Count; index++)
    {
        if (rnd.Next(0, 100) < 2)
        {
            households.RemoveAt(index);
            leavers++;
        }
    }
    return leavers;
}
```

1 mark call to `ProcessLeavers` as the final instruction in `ProcessDayEnd`

```
DisplayCompaniesAtDayEnd();
DisplayEventsAtDayEnd();
Console.WriteLine(simulationSettlement.ProcessLeavers() +
    " households left the settlement");
```

1 mark screenshot displays number of households removed, which might be the n
text 'households left the settlement') and/or zero:

```
***********************
*****   E          *****
***********************

No events.
4 households left the settlement

*********************************
**********    MENU     **********
*********************************
1. Display details of households
2. Display details of companies
3. Modify company
4. Add new company
6. Advance to next day
Q. Quit
```

Task 5

1 mark new input message within code executed when user enters '3'

1 mark loop to iterate over all companies in the list

1 mark output to contain call to company's `GetName` subroutine

1 mark output to display 1-based indices instead of 0-based indices

1 mark user input stored in the `index` variable

1 mark user input decremented

```
case "3":
    do
    {
        Console.WriteLine("Enter the number next to the company you
        for (int i = 0; i < companies.Count; i++)
        {
            Console.WriteLine((i + 1) + ": " + companies[i].GetName(
        }
        index = Convert.ToInt32(Console.ReadLine()) - 1;
    } while (index == -1);
    ModifyCompany(index);
    break;
case "4":
```

1 mark user input should now be based on a list, with outlets numbered 1, 2, 3; fin
should have a capacity equal to its max capacity:

```
**********************************
**********     MENU     **********
**********************************
1. Display details of household
2. Display details of company
3. Modify company
4. Add new company
6. Advance to next day
Q. Quit

 Enter your choice: 3
Enter the number next to the company you wish to modify:
1: AQA Burgers
2: Ben Thor Cuisine
3: Paltry Poultry
3
```

```
Number of outlets: 4
Outlets
1. Coordinates: (800, 390)     Capacity: 120      Maximum Capacity:
2. Coordinates: (400, 390)     Capacity: 120      Maximum Capacity:
3. Coordinates: (820, 370)     Capacity: 120      Maximum Capacity:
4. Coordinates: (800, 600)     Capacity: 210      Maximum Capacity:
```

Task 6

1 mark subroutine correctly declared as a constructor

1 mark correct parameters declared

1 mark `xCoord` and `yCoord` set to random integers between 0 and `maxX`/`maxY`

```
public Outlet(int maxX, int maxY, int maxCapacityBase, bool isRandom
{
    xCoord = rnd.Next(0, maxX + 1);
    yCoord = rnd.Next(0, maxY + 1);
```

1 mark the rest of the subroutine executes as normal. **A.** call to a new subroutine that contains the extra lines. **R.** if call is made to other construct

```
    capacity = Convert.ToInt32(maxCapacityBase * 0.6);
    maxCapacity = maxCapacityBase + Convert.ToInt32(rnd.NextDouble()
        Convert.ToInt32(rnd.NextDouble() * 50);
    dailyCosts = maxCapacityBase * 0.2 + capacity * 0.5 + 100;
    NewDay();
}
```

1 mark screen capture shows values for outlet coordinates that are all 10 or lower

```
Name: Paltry Poultry
Type of business: fast food
Current balance: 17000.0
Average cost per meal: 5.0
Average price per meal: 10.0
Daily costs: 100.0
Delivery costs: 0.1597236
Reputation: 97.53389

Number of outlets: 4
Outlets
1. Coordinates: (0, 0)     Capacity: 120
2. Coordinates: (1, 1)     Capacity: 120
3. Coordinates: (4, 2)     Capacity: 120
4. Coordinates: (3, 0)     Capacity: 120
```

Task 7

1 mark option 5 correctly added in `DisplayMenu`

```
Console.WriteLine("4. Add new company");
Console.WriteLine("5. Advance");
Console.WriteLine("6. Advance to next day");
Console.WriteLine("Q. Quit");
```

1 mark selection structure in `Run` captures an input of 5

1 mark any suitable prompt for the user to enter a number of days

1 mark integer variable stores user input

1 mark loop structure runs the correct number of times

1 mark call to `ProcessDayEnd` inside the loop and no other code

```
case "4":
    AddCompany();
    break;
case "5":
    Console.Write("Enter number of days: ");
    int numberOfDays = Convert.ToInt32(Console.ReadLine());
    for (int x = 0; x < numberOfDays; x++)
    {
        ProcessDayEnd();
    }
    break;
case "6":
```

1 mark three days' worth of financials and events should be displayed, although if Poultry is anything other than 17000 and should match code, credit can be

```
Paltry Poultry

Daily cost for company: 100.0
Cost for delivering produce to outlets: 110.30317
Outlet 1 profit/loss: -195.0
Outlet 2 profit/loss: -90.0
Outlet 3 profit/loss: -150.0
Outlet 4 profit/loss: -155.0
Previous balance for company: 15364.395
New balance for company: 14564.092
```

Task 8

1 mark prompt updated to include reference to '4' generating a random company

1 mark loop updated to include '4' as a terminating condition

```
do {
    Console.write("Enter 1 for a fast food company, 2 for a family c
            + "3 for a named chef company or 4 [illegible] company chosen
    typeOfCompany = Console.readLine();
} while (!typeOfCompany.equals("1") [illegible] !typeOfCompany.equals("2") &&
        !typeOfCompany.equals("3") && !typeOfCompany.equals("4"));
if (typeOfCompany.equals("3")) {
```

1 mark updated selection structure to ensure 3 results in 'named chef'

1 mark additional to selection structure to catch 4 or 'else' (i.e. not 1, 2 or 3)

1 mark random number generated, even if likelihoods are not evenly distributed

1 mark selection structure sets restaurant type according to random number

1 mark three company types are equally likely to be created

1 mark additional code does not impede code from the Skeleton Program

```
} else if (typeOfCompany.equals("3")) {
    typeOfCompany = "named chef";
} else {
    int companyType = rnd.nextInt(3);
    if (companyType == 0) { typeOfCompany = "fast food"; }
    else if (companyType == 1) { typeOfCompany = "family"; }
    else { typeOfCompany = "named chef"; }
}
int[] tempLocation = simulationSettlement.getRandomLocation();
```

1 mark input of '4', 'Random Restaurant' and '50000', resulting in a company of an

```
Enter a name for the company: Random Restaurant
Enter the starting balance for the company: 50000
Enter 1 for a fast food company, 2 for a family company,
3 for a named chef company or 4 for a company chosen at ran
```

```
Name: Random Restaurant
Type of business: fast food
Current balance: 48000.0
Average cost per meal: 5.0
Average price per meal: 10.0
Daily costs: 100.0
Delivery costs: 0.0
Reputation: 96.481125
```

Task 9

1 mark new subroutine created, with no parameters and no return

1 mark loop to iterate over all outlets in the `outlets` list

1 mark call to `CloseOutlet` for each outlet

```
public void CloseAllOutlets()
{
    for (int current = 0; current < outlets.Count; current++)
    {
        CloseOutlet(current);
    }
}
```

1 mark selection statement in `ProcessDayEnd` to check for balance of less than statement

1 mark call to `CloseAllOutlets` in selection structure

```
balance += profitLossFromOutlets - dailyCosts - deliveryCosts;
details += "New balance for company: " + balance.ToString();
if (balance < 0)
{
    CloseAllOutlets();
}
return details;
```

1 mark entering '2' in the main menu should reveal that 'Bankrupt Burgers' has no

```
Name: Bankrupt Burgers
Type of business: family
Current balance: -133.0
Average cost per meal: 12.0
Average price per meal: 1
Daily costs: 100.0
Delivery
Reputatio 16606

Number of outlets: 0
Outlets
```

Task 10

1 mark class definition, which includes inheritance

1 mark creation of a new `Random` object

1 mark constructor correctly declared, with two parameters

1 mark call within constructor to the `Outlet` constructor, passing correct values

1 mark `Move` subroutine declared

1 mark random number generated

1 mark selection statement to deal with four different possible values of random number

1 mark each of north, south, east and west correctly simulated

```
class FoodTruck : Outlet
{
    private static Random rnd = new Random();

    public FoodTruck(int xCoord, int yCoord)
        : base(xCoord, yCoord, 10)
    { }

    public void Move()
    {
        int direction = rnd.Next(4);
        if (direction == 0) { xCoord += 1; }
        else if (direction == 1) { xCoord -= 1; }
        else if (direction == 2) { yCoord += 1; }
        else { yCoord -= 1; }
    }
}
```

1 mark either creation of loop to iterate through outlets in `ProcessDayEnd` subroutine or existing loop modified to move, even if syntactically invalid

1 mark selection statement to check whether an outlet is an instance of a `FoodTruck`

1 mark casting the object as a `FoodTruck`

1 mark `move` subroutine called for all `FoodTruck` objects and only `FoodTruck` objects

```
for (int current = 0; current < outlets.Count; current++)
{
    if (outlets[current] is FoodTruck)
    {
        FoodTruck f = (FoodTruck)outlets[current];
        f.Move();
    }
}
```

(continues on next page)

1 mark screen captures show a difference of 1 in **either** X **or** Y coordinates betwee

```
Enter number of companies that exist at start of simulation: 1
Enter a name for the company: Taco Truck
Enter the starting balance for the company: 30000
Enter 1 for a fast food company, 2 for a family company or 3 fo
```

```
Name: Taco Truck
Type of business: fast food
Current balance: 28000.0
Average cost per meal: 5.0
Average price per meal: 10.0
Daily costs: 100.0
Delivery costs: 0.0
Reputation: 60.0

Number of outlets: 1
Outlets
1. Coordinates: (595, 285)
```

```
Name: Taco Truck
Type of business: fast food
Current balance: 28355.0
Average cost per meal: 5.0
Average price per meal: 10.0
Daily costs: 100.0
Delivery costs: 0.0
Reputation: 93.62324

Number of outlets: 1
Outlets
1. Coordinates: (594, 285)
```

Task 11

1 mark data structure created to store indexes

1 mark selection structure has been changed from 'equals' to 'contains' or equivale

1 mark inside selection structure, adding the index to the data structure

```
List<int> indexes = new List<int>();

for (int current = 0; current < companies.Count; current++)
{
    if (companies[current].GetName().ToLower().
        Contains(searchName.ToLower())) {
        indexes.Add(current);
    }
}
```

1 mark selection structure to check for only a single match, **A.** if 'else' by process o

1 mark correct index returned for a single match

```
if (indexes.Count == 1) { return indexes[0]; }
```

1 mark selection structure to check for multiple matches, **A.** if 'else' by process of e

1 mark prompt for user entry, either before or after attempt to display matches

1 mark loop to iterate through all matches in an attempt to display them

1 mark name of each matching outlet displayed

1 mark user input requested only if multiple matches have been found

1 mark loop to iterate through the matches in an attempt to compare with user inp

1 mark comparison ('equals', not 'contains') is made inside the loop, with case ignc

1 mark correct index returned in the event of a match

1 mark value returned if no matches are found or if second entry does not m

```
else if (indexes.Count > 1)
{
    Console.WriteLine("Please enter one from the following: ");
    for (int i = 0; i < indexes.Count; i++)
    {
        Console.WriteLine(companies[indexes[i]].GetName());
    }
    string input = Console.ReadLine();
    for (int i = 0; i < indexes.Count;i++)
    {
        if (input.ToLower() == companies[indexes[i]].GetName().ToLow
        {
            return indexes[i];
        }
    }
}
return -1;
```

(continues on next page)

1 mark screen capture displays 'Paltry Poultry' and 'Ben Thor Cuisine', with 'Paltry P

```
**********************************
**********    MENU     **********
**********************************
1. Display details of households
2. Display details of companies
3. Modify company
4. Add new company
6. Advance to next day
Q. Quit

 Enter your choi
Enter c
Please e one from the following:
Ben Thor isine
Paltry Poultry
Paltry Poultry

**********************************
*******  MODIFY COMPANY   *******
**********************************
1. Open new outlet
2. Close outlet
3. Expand outlet

Enter your choice: _
```

Task 12

1 mark outlet with a matching `ID` is accessed

1 mark number of seats is set to a call to `GetCapacity`

1 mark selection structure uses `category` attribute of the `Company` class

1 mark cost per seat is set correctly for all categories

1 mark balance is decremented by the cost per seat multiplied by the number of s

1 mark the original code, to remove the outlet and check for zero outlets, should r

```
public bool CloseOutlet(int ID)
{
    Outlet o = outlets[ID];
    int seats = o.GetCapacity();
    int costPerSeat;
    if (category == "fast food") { costPerSeat = 75; }
    else if (category == "family") { costPerSeat = 50; }
    else { costPerSeat = 150; }
    balance -= seats * costPerSeat;

    bool closeCompany = false;
    outlets.RemoveAt(ID);
    if (outlets.Count == 0)
    {
        closeCompany = true;
    }
    return closeCompany;
}
```

1 mark current balance of Paltry Poultry sho ch ge from 17000 to 8000, and th

```
Name: Paltry Poultry
Type of business:
Current
Average  meal: 5.0
Average p  per meal: 10.0
Daily costs: 100.0
Delivery costs: 10.30317
Reputation: 101.39155

Number of outlets: 4
Outlets
1. Coordinates: (800, 390)     Capacity: 120
2. Coordinates: (400, 390)     Capacity: 120
3. Coordinates: (820, 370)     Capacity: 120
4. Coordinates: (800, 600)     Capacity: 120
```

```
Name: Paltry Poultry
Type of business: fast food
Current balance: 8000.0
Average cost per meal: 5.0
Average price per meal: 10.0
Daily costs: 100.0
Delivery costs: 8.04
Reputation:

Number of ts: 3
Outlets
1. Coordinates: (800, 390)     Capacity: 120
2. Coordinates: (400, 390)     Capacity: 120
3. Coordinates: (820, 370)     Capacity: 120
```

Task 13

1 mark additional line added to `DisplayMenu` with correct number and text

```
Console.WriteLine("4. Add new company");
Console.WriteLine("5. Remove company");
Console.WriteLine("6. Advance to next day");
```

1 mark addition of '5' to the selection structure in m

1 mark prompt for company name and st[illegible]or user input in a string variable

1 mark selection structure to ch[illegible] the index either is or is not -1

1 mark removal of com[illegible] at the correct index, only if the index is not -1

1 mark loo[illegible]er[illegible]ace if the loop is not -1

1 mark loop [illegible]erminate if 'cancel', in any combination of upper case / lower cas

```
case "4":
    AddCompany();
    break;
case "5":
    string nameOfCompany;
    do
    {
        Console.Write("Enter company name: ");
        nameOfCompany = Console.ReadLine();
        index = GetIndexOfCompany(nameOfCompany);
        if (index > -1) { companies.RemoveAt(index); }
    } while (index == -1 && nameOfCompany.ToUpper() != "CANCEL");
    break;
case "6":
    ProcessDayEnd();
    break;
```

1 mark entering 'CANCEL' r[illegible]e to the main menu; entering 'Paltry Poultry' being r[illegible]n simulation – AQA Burgers and Ben Thor Cuisine:

```
 Enter [illegible]hoice: 5
Enter company name: CANCEL

************************************
**********     MENU     **********
************************************
1. Display details of households
2. Display details of companies
3. Modify company
4. Add new company
5. Remove company
6. Advance to next day
Q. Quit

 Enter your choice: _
```

```
********
*** Detai
********

Name: AQ
Type of
Current
Average
Average
Daily co
Delivery
Reputati

Number o
Outlets
1. Coord
2. Coord
3. Coord
4. Coord
5. Coord
6. Coord
7. Coord


Name: Be
Type of
Current
Average
Average
Daily co
Delivery
Reputati

Number o
Outlets
1. Coord
```

Task 14

1 mark `GetBalance` declared in the `Company` class, with no parameters and cor

1 mark correct return statement, **R.** if any additional code

```
public double GetBalance()
{
    return balance;
}
```

1 mark additional option added to Display Menu

```
Console.WriteLine("5. Add new company");
Console.WriteLine("5. Run to target");
Console.WriteLine("6. Advance to next day");
```

1 mark selection structure modified to include '5'

1 mark call to new `RunToTarget` subroutine if '5' is entered

```
case "4":
    AddCompany();
    break;
case "5":
    RunToTarget();
    break;
case "6":
    ProcessDayEnd();
    break;
```

1 mark new `RunToTarget` subroutine declared

1 mark user prompted for an upper limit and a lower limit

1 mark each user input stored as a separate integer

1 mark variable names `upperLimit` and `lowerLimit` used as instructed

1 mark variables for number of days, balance and company name, of appropriate t

```
public void RunToTarget()
{
    Console.Write("Upper limit: ");
    int upperLimit = Convert.ToInt32(Console.ReadLine());
    Console.Write("Lower limit: ");
    int lowerLimit = Convert.ToInt32(Console.ReadLine());
    int numberOfDays = 0;
    Company c;
    String companyName = "";
    double balance = 0;
```

1 mark loop to run until a balance is equal to or above the upper limit, or equal to

1 mark loop to iterate over each company in the simulation

1 mark comparison with both upper and lower limits

1 mark call to `ProcessDayEnd` once within each iteration

1 mark no call to `ProcessDayEnd` when balance has already reached termination c
termination condition already reached by the start of the first loop (i.e.
possible)

1 mark number of days incremented within each iteration

1 mark calls to `GetName` and `GetBalance` have occurred before loop terminates

```
do {
    for (int i = 0; i < companies.Count; i++) {
        c = companies[i];
        if (c.GetBalance() >= upperLimit || c.GetBalance() <= lower
            companyName = c.GetName();
            balance = c.GetBalance();
        }
        if (companyName == "") {
            numberOfDays += 1;
            ProcessDayEnd()
        }
    }
} while (companyName == "");
```

1 mark values for the company's name and balance, and the number of days, are c

1 mark output values are correct under all circumstances

```
Console.WriteLine("Company: " + companyName);
Console.WriteLine("Balance: " + balance);
Console.WriteLine("Days:    " + numberOfDays);
```

1 mark screen evidence showing that multiple days have passed, and the balance

```
**********************
*****   Events:   *****
**********************

No events.
Company: Ben Thor Cuisine
Balance: 100780.8
Days:    22
```

Task 15

1 mark class definition with correct identifier, which inherits from `LargeSettlem`

1 mark object of type `Random` included as an attribute

1 mark valid constructor with three integer parameters

1 mark valid call within constructor to constructor of superclass

1 mark subroutine `AddHousehold` declared correctly with override modifier

1 mark generation of random number within `AddHousehold`, between 2 and 20

1 mark generation of random X and Y coordinates within the settlement (easiest via class `CityRandomLocation` subroutine, but any valid approach can be

1 mark loop structure that will iterate once for each household in this location (integ

1 mark each new household, within the loop, added to the `households` list

```
class CitySettlement : LargeSettlement
{
    private static Random rnd = new Random();

    public CitySettlement(int extraXSize, int extraYSize, int extraH
        : base(extraXSize, extraYSize, extraHouseholds)
    { }

    public override void AddHousehold() {
        int numberOfHouseholds = rnd.Next(2, 21);
        int x = 0, y = 0;
        GetRandomLocation(ref x, ref y);
        for (int i = 0; i < numberOfHouseholds; ++) {
            households.Add(new Household(x, y));
        }
    }
}
```

1 mark virtual modifier added to pre-existing `AddHouseholds` subroutine

```
public virtual void AddHousehold()
{
    int x = 0, y = 0;
    GetRandomLocation(ref x, ref y);
    Household temp = new Household(x, y);
    households.Add(temp);
}
```

1 mark first prompt of the simulation amended to allow user to select a city settler

1 mark user selecting a city settlement results in prompts for additional X and Y coord

1 mark user selecting a city settlement results in call to `CitySettlement` constr

(continues on next page)

1 mark other inputs should continue to work as previously (i.e. new code does not

```
public Simulation()
{
    fuelCostPerUnit = 0.0098;
    baseCostForDelivery = 100;
    string choice;
    Console.Write("Enter L for a large settlement, C for a city sett
        + "anything else for a normal size settlement: ");
    choice = Console.ReadLine();
    if (choice == "L" || choice =
    {
        int extraX, extraY, extraHouseholds;
        Console.Write("Enter additional amount to add to X size of se
        extraX = Convert.ToInt32(Console.ReadLine());
        Console.Write("Enter additional amount to add to Y size of se
        extraY = Convert.ToInt32(Console.ReadLine());
        Console.Write("Enter additional number of households to add 
        extraHouseholds = Convert.ToInt32(Console.ReadLine());
        if (choice == "L") {
            simulationSettlement = new LargeSettlement(extraX, extraY
        } else {
            simulationSettlement = new CitySettlement(extraX, extraY,
        }
    } else {
        simulationSettlement = new Settlement();
    }
```

1 mark screen capture should show consecutive households at the same location

```
2687     Coordinates: (42, 578)     Eat out probability: 0.5362351
2688     Coordinates: (42, 578)     Eat out probability: 0.45420235
2689     Coordinates: (42, 578)     Eat out probability: 0.37879705
2690     Coordinates: (42, 578)     Eat out probability: 0.29859382
2691     Coordinates: (42, 578)     Eat out probability: 0.11144531
```

Name	

ZigZag Education supporting

A Level AQA Computer Science Pap

Summer 2020

Food Magnate Simulation

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

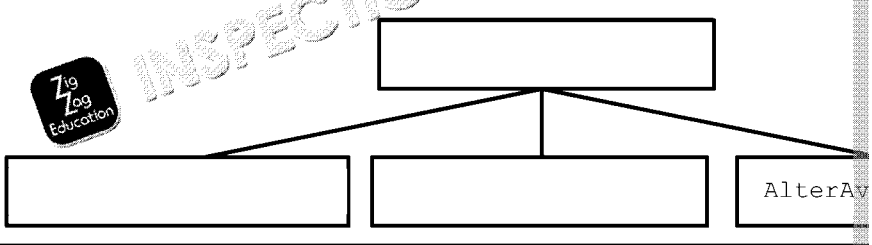
- Answer **all** questions
- The marks available for each question are shown in brackets

- 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:

Programming Theory Question

Answer all questions. Remember to save this document

Q		Answer
1	(a)	
	(b)	
	(c)	
	(d)	
	(e)	
	(f)	
	(g)	
	(h)	
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		

Programming Tasks

Answer all questions. Remember to save this document r

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