

M250 Revision 2021

M250 Exam 2018J

Phil Molyneux

23 May 2021

M250 Exam Revision

Agenda & Aims

- ▶ Welcome and introductions
- ▶ Revision strategies
- ▶ M250 Exam — Online Friday 11 June 2021 – see *Assessment*
- ▶ M250 2018J exam (June 2019)
- ▶ Topics and discussion for each question
- ▶ Exam techniques
- ▶ These slides and notes are at www.pmolyneux.co.uk/OU/M250FolderSync/M250ExamRevision/ at M250Prsntn2020JExamRevision/
- ▶ **Recording**   ✓

Agenda

Introductions

Adobe Connect

Instructions 2018J

Qs

Solns

Exam Reminders

White Slide

M269 Exam Revision

Introductions & Revision strategies

- ▶ Introductions
- ▶ What other exams are you doing this year ?
- ▶ Each give one exam tip to the group

Adobe Connect

Interface — Student Quick Reference

M250 Revision
2021

Phil Molyneux

Participant Quick Reference Guide

The screenshot shows the Adobe Connect interface with the following labeled components:

- Share pod:** Located at the top left, containing icons for Meeting, Speaker volume, Audio set up, Webcam, and a microphone icon.
- Connect microphone:** A button located below the microphone icon in the Share pod.
- Status:** A text box below the Connect microphone button containing the text: "Status: raise hand, agree, disagree, step away, speak louder, speak softer, speed up, slow down, laughter, applause".
- Nothing is being shared:** A message displayed in the main central area of the interface.
- Adobe Connect Help:** A button located at the top right of the interface.
- Connection status:** A button located at the top right of the interface, next to the Adobe Connect Help button.
- Video pod:** A section on the right side of the interface containing a "Start My Webpage" button.
- Attendee pod:** A section on the right side of the interface containing a list of attendees, including "Dar Span 2".
- Chat pod:** A section on the right side of the interface containing a chat area with a text input field and a "Everyone" button.

Agenda

Adobe Connect

Student View

- Settings
- Student & Tutor Views
- Sharing Screen & Applications
- Ending a Meeting
- Invite Attendees
- Layouts
- Chat Pods

Instructions 2018J

Qs

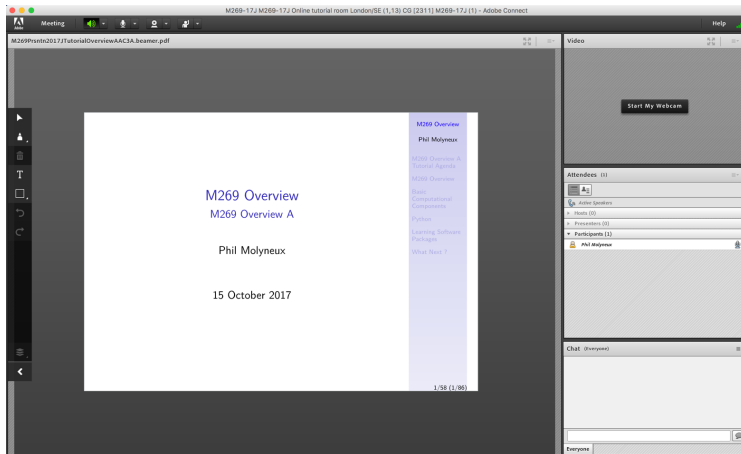
Solns

Exam Reminders

White Slide

Adobe Connect

Interface — Student View



M250 Revision
2021

Phil Molyneux

Agenda

Adobe Connect

Student View

- Settings
- Student & Tutor Views
- Sharing Screen & Applications
- Ending a Meeting
- Invite Attendees
- Layouts
- Chat Pods

Instructions 2018J

Qs

Solns

Exam Reminders

White Slide

Agenda

Adobe Connect

Student View

Settings

Student & Tutor Views

Sharing Screen &
Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Instructions 2018J

Qs

Solns

Exam Reminders

White Slide

- ▶ **Everybody: Audio Settings** Meeting > Audio Setup Wizard. . .
- ▶ **Audio** Menu bar > Audio > Microphone rights for Participants ✓
- ▶ Do not *Enable single speaker mode*
- ▶ **Drawing Tools** Share pod menu bar > Draw (1 slide/screen)
- ▶ Share pod menu bar > Menu icon > Enable Participants to draw ✓ gray
- ▶ Meeting > Preferences > Whiteboard > Enable Participants to draw ✓
- ▶ Cancel hand tool . . . Do not enable green pointer. . .
- ▶ Meeting > Preferences > Attendees Pod ✗ *Raise Hand notification*
- ▶ Meeting > Preferences > Display Name Display First & Last Name
- ▶ **Cursor** Meeting > Preferences > General tab > Host Cursors
Show to all attendees ✓ (default *Off*)
- ▶ Meeting > Preferences > Screen Share > Cursor > Show Application Cursor
- ▶ **Webcam** Menu bar > Webcam > Enable Webcam for Participants ✓
- ▶ **Recording** Meeting > Record Meeting. . . ✓

Agenda

Adobe Connect

Student View

Settings

Student & Tutor Views

Sharing Screen &
Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Instructions 2018J

Qs

Solns

Exam Reminders

White Slide

► Tutor Access

TutorHome > M269 Website > Tutorials

Cluster Tutorials > M269 Online tutorial room

Tutor Groups > M269 Online tutor group room

Module-wide Tutorials > M269 Online module-wide room

► Attendance

TutorHome > Students > View your tutorial timetables

► Beamer Slide Scaling 440% (422 x 563 mm)

► Clear Everyone's Status

Attendee Pod > Menu > Clear Everyone's Status

► Grant Access and send link via email

Meeting > Manage Access & Entry > Invite Participants...

► Presenter Only Area

Meeting > Enable/Disable Presenter Only Area

Adobe Connect

Keystroke Shortcuts

- ▶ Keyboard shortcuts in Adobe Connect
- ▶ **Toggle Mic**  + **M** (Mac), **Ctrl** + **M** (Win) (On/Disconnect)
- ▶ **Toggle Raise-Hand status**  + **E**
- ▶ **Close dialog box**  (Mac), **Esc** (Win)
- ▶ **End meeting**  + ****

Agenda

Adobe Connect

Student View

Settings

Student & Tutor Views

Sharing Screen &
Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Instructions 2018J

Qs

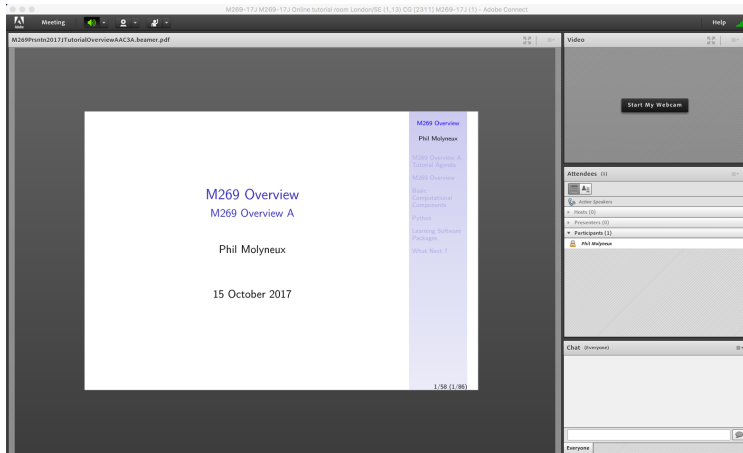
Solns

Exam Reminders

White Slide

Adobe Connect Interface

Student View (default)



M250 Revision
2021

Phil Molyneux

Agenda

Adobe Connect

Student View

Settings

Student & Tutor Views

Sharing Screen &
Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Instructions 2018J

Qs

Solns

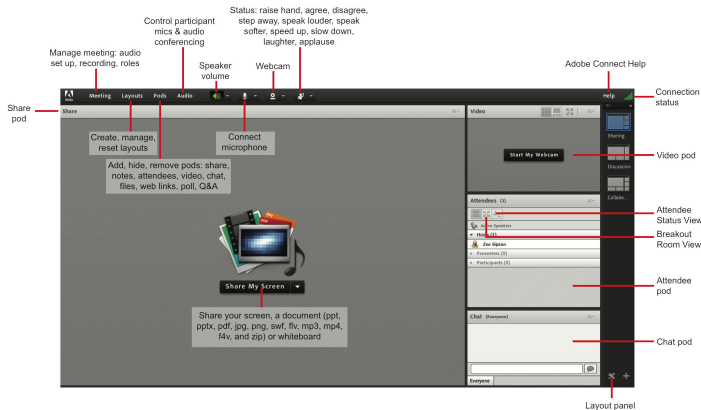
Exam Reminders

White Slide

Adobe Connect Interface

Tutor View

Host Quick Reference Guide



M250 Revision
2021

Phil Molyneux

Agenda

Adobe Connect

Student View

Settings

Student & Tutor Views

Sharing Screen &
Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Instructions 2018J

Qs

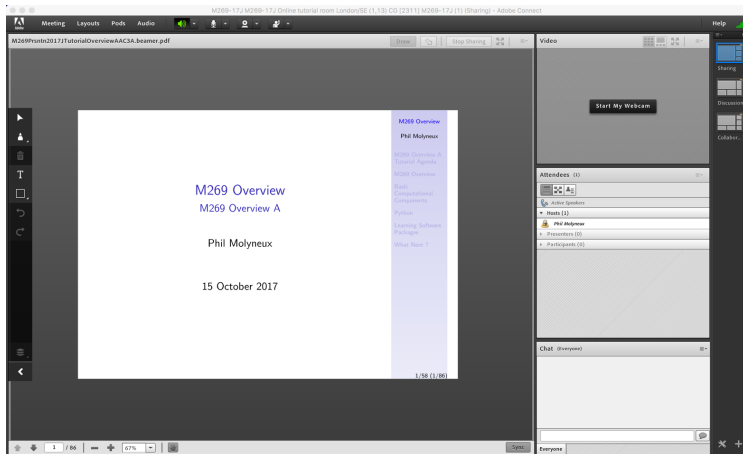
Solns

Exam Reminders

White Slide

Adobe Connect Interface

Tutor View



M250 Revision
2021

Phil Molyneux

Agenda

Adobe Connect

Student View

Settings

Student & Tutor Views

Sharing Screen &
Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Instructions 2018J

Qs

Solns

Exam Reminders

White Slide

Adobe Connect Interface

Sharing Screen & Applications

- ▶ **Share My Screen** > **Application tab** > **Terminal** for **Terminal**
- ▶ **Share menu** > **Change View** > **Zoom in** for mismatch of screen size/resolution (Participants)
- ▶ (Presenter) Change to 75% and back to 100% (solves participants with smaller screen image overlap)
- ▶ Leave the application on the original display
- ▶ Beware blue hatched rectangles — from other (hidden) windows or contextual menus
- ▶ Presenter screen pointer affects viewer display — beware of moving the pointer away from the application
- ▶ First time: **System Preferences** > **Security & Privacy** > **Privacy** > **Accessibility**

Agenda

Adobe Connect

Student View

Settings

Student & Tutor Views

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Instructions 2018J

Qs

Solns

Exam Reminders

White Slide

- ▶ *Notes for the tutor only*
- ▶ **Student:** Meeting > Exit Adobe Connect
- ▶ **Tutor:**
- ▶ **Recording** Meeting > Stop Recording ✓
- ▶ **Remove Participants** Meeting > End Meeting... ✓
 - ▶ Dialog box allows for message with default message:
 - ▶ *The host has ended this meeting. Thank you for attending.*
- ▶ **Recording availability** *In course Web site for joining the room, click on the eye icon in the list of recordings under your recording — edit description and name*
- ▶ **Meeting Information** Meeting > Manage Meeting Information — can access a range of information in Web page.
- ▶ **Attendance Report** see course Web site for joining room

Agenda

Adobe Connect

Student View

Settings

Student & Tutor Views

Sharing Screen &
Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Instructions 2018J

Qs

Solns

Exam Reminders

White Slide

Adobe Connect

Invite Attendees

- ▶ **Provide Meeting URL** Menu Meeting Manage Access & Entry
Invite Participants...
- ▶ **Allow Access without Dialog** Menu Meeting
Manage Meeting Information provides new browser window with *Meeting Information* Tab bar Edit Information
- ▶ Check *Anyone who has the URL for the meeting can enter the room*
- ▶ Default *Only registered users and accepted guests may enter the room*
- ▶ **Reverts to default next session but URL is fixed**
- ▶ Guests have blue icon top, registered participants have yellow icon top — same icon if URL is open
- ▶ See [Start, attend, and manage Adobe Connect meetings and sessions](#)

Agenda

Adobe Connect

Student View
Settings
Student & Tutor Views
Sharing Screen & Applications
Ending a Meeting

Invite Attendees

Layouts
Chat Pods

Instructions 2018J

Qs

Solns

Exam Reminders

White Slide

Agenda

Adobe Connect

Student View
Settings
Student & Tutor Views
Sharing Screen &
Applications
Ending a Meeting
Invite Attendees

Layouts

Chat Pods

Instructions 2018J

Qs

Solns

Exam Reminders

White Slide

- ▶ **Creating new layouts** example *Sharing* layout
- ▶ **Menu** > **Layouts** > **Create New Layout...** > **Create a New Layout dialog**
> **Create a new blank layout** and name it *PMolyMain*
- ▶ New layout has no Pods but does have Layouts Bar open (see Layouts menu)
- ▶ **Pods**
- ▶ **Menu** > **Pods** > **Share** > **Add New Share** and resize/position — initial name is *Share n*
- ▶ **Rename Pod** **Menu** > **Pods** > **Manage Pods...** > **Manage Pods**
> **Select** > **Rename** or **Double-click & rename**
- ▶ Add Video pod and resize/reposition
- ▶ Add Attendance pod and resize/reposition
- ▶ Add Chat pod — name it *PMolyChat* — and resize/reposition

- ▶ Dimensions of **Sharing** layout (on 27-inch iMac)
 - ▶ Width of Video, Attendees, Chat column 14 cm
 - ▶ Height of Video pod 9 cm
 - ▶ Height of Attendees pod 12 cm
 - ▶ Height of Chat pod 8 cm
- ▶ **Duplicating Layouts** does *not* give new instances of the Pods and is probably not a good idea (apart from local use to avoid delay in reloading Pods)

Agenda

Adobe Connect

Student View

Settings

Student & Tutor Views

Sharing Screen &
Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Instructions 2018J

Qs

Solns

Exam Reminders

White Slide

Adobe Connect

Chat Pods

- ▶ **Format Chat text**
- ▶ Chat Pod menu icon My Chat Color
- ▶ Choices: Red, Orange, Green, Brown, Purple, Pink, Blue, Black
- ▶ Note: Color reverts to Black if you switch layouts
- ▶ Chat Pod menu icon Show Timestamps

Agenda

Adobe Connect

Student View
Settings
Student & Tutor Views
Sharing Screen &
Applications
Ending a Meeting
Invite Attendees
Layouts

Chat Pods

Instructions 2018J

Qs

Solns

Exam Reminders

White Slide

M250 2018J Exam

Qs

- ▶ M250 Object-oriented Java Programming
- ▶ Presentation 2018J Exam
- ▶ Date Monday, 10 June 2019 Time 10:00–13:00
- ▶ You should attempt **ALL** questions
- ▶ **Note** see the original exam paper for exact wording and formatting — these slides and notes may change some wording and formatting

▶ Go to Solns

M250 2018J Exam

Solns

- ▶ The solutions given below are not official solutions
- ▶ For some questions, alternatives are given — a student would only have to provide one
- ▶ No marks are given for code comments
- ▶ You may assume any import statements required, unless otherwise indicated.
- ▶ You may assume that methods receive sensible values when a message is sent, unless otherwise indicated.
- ▶ When writing code, you will not be penalised for minor errors, as long as the meaning is clear.

▶ [Go to Qs](#)

M250 2018J Exam

Q 1 (40 marks total)

- ▶ **Scenario** Equity is a union of more than 43000 performers. All performers in Equity have a professional name, known as their *equity name* which is unique to them, and can choose to join a local branch of Equity.
- ▶ Performers can belong to a local branch which organises regular meetings, for example on the second Saturday of each month.
- ▶ This question asks you to write parts of the class **Performer**, whose purpose is to model this scenario.
- ▶ Assume a class **Branch** which has two private **String** instance variables, **name**, **address**, a two-argument constructor allowing the branch name and address to be initialised, an **equals** method, getter methods for **name** and **address** and a setter method for **address**.
- ▶ Q 1 continued on next slide

M250 2018J Exam

Q 1(a)(i)

(a)(i) Write a class **Performer** with the following: **(9 marks)**

- ▶ a private instance variable of type **String** called **equityName**
- ▶ a private instance variable of type **double** called **payRate**, which will be used to hold the agreed rate of pay for that performer
- ▶ a private instance variable of type **Branch** called **branch** which will refer to the instance of **Branch** which that performer has joined
- ▶ a public **class** variable of type **double** called **minPayRate** which is the minimum pay rate agreed by Equity for performers.
- ▶ Q 1(a)(i) continued on next slide

▶ Go to Soln 1

M250 2018J Exam

Q 1(a)(i) (contd)

- ▶ a public single-argument constructor which initializes `equityName` to the argument string `aName`, sets `branch` to `null` and sets `payRate` to `minPayRate`
- ▶ a public setter method for `payRate`
- ▶ a public getter method for `branch`
- ▶ a public setter method for `branch`
- ▶ a public getter method for `equityName`
- ▶ Q 1(a) continued on next slide

▶ Go to Soln 1(a)(i)

M250 2018J Exam

Q 1(a)(ii)

- (ii) Write a public instance method `isInSameBranchAs()` that has a `Performer` argument.
- ▶ This method will return `true` if the receiver and the argument `Performer` objects are members of the same branch, and `false` otherwise. **(5 marks)**
 - ▶ Q 1(a) continued on next slide

▶ Go to Soln 1(a)(ii)

M250 2018J Exam

Q 1(a)(iii)

- (iii) Write a public instance method `getFirstName()` that has no arguments.
- ▶ This method will return a `String` consisting of all the characters in the `equityName`, up to but not including the first space. You may assume that there is a space in the `equityName`. **(5 marks)**
 - ▶ Q 1 continued on next slide

▶ [Go to Soln 1\(a\)\(iii\)](#)

M250 2018J Exam

Q 1(b)

(b) Given the code developed in part (a), assume that the following code is part of a method and is executed:

```
Branch b1; // 1
b1 = new Branch("Kent", "The_Alexander_Centre"); // 2
Branch b2; // 3
b2 = new Branch("Dorset", "Wessex_fm_Studios"); // 4
Performer.minPayRate = 9.50; // 5
Performer p1 = new Performer("Happy_Bunny"); // 6
Performer p2 = new Performer("Silly_Sausage"); // 7
p1.setPayRate(10.00); // 8
p2.setPayRate(20.00); // 9
p1.setBranch(b1); // 10
p2.setBranch(b1); // 11
System.out.println(p1.isInSameBranchAs(p2)); // 12
```

► Q 1(b) continued on next slide

► Go to Soln 1(b)

M250 2018J Exam

Q 1(b) (contd)

- ▶ In the numbered lines of code above, identify all the examples of the following, stating the line number(s) on which they occur. If there are no examples, state *None* explicitly. **(7 marks)**

- (i) messages are sent
- (ii) reference variables are declared
- (iii) primitive variables are declared
- (iv) object construction
- (v) operators are used
- (vi) formal arguments are declared
- (vii) actual arguments are used

- ▶ Q 1 continued on next slide

▶ Go to Soln 1(b)

M250 2018J Exam

Q 1(c)

- (c) For the class `Performer`, write the public instance method `equals()` that overrides the `equals()` method inherited from `Object`.
- ▶ This method will return `true` if the `equityName` of the receiver is the same as the `equityName` of the argument object, and otherwise return `false`. **(5 marks)**
 - ▶ Q 1 continued on next slide

▶ Go to Soln 1(c)

M250 2018J Exam

Q 1(d)

- (d) Based on the **Performer** class written so far, answer the following questions:
- (i) What is the nature of the object-oriented relationship between the classes **Performer** and **Branch**? Explain your answer. **(2 marks)**
 - (ii) Consider line `// 5` in part (b) above. Why can the value of **minPayRate** be set at this point when no **Performer** objects have been constructed? **(2 marks)**
 - (iii) Give two examples of how scope applies to the **Performer** class. One example should relate to an instance variable and the other should relate to a formal argument. **(5 marks)**
- (40 marks total)**

► Go to Soln 1(d)

M250 2018J Exam

Q 2 (30 marks)

- **Scenario** This question concerns a number of vehicle classes and the **Drivable** interface that specifies some common behaviours.
- (a) **Drivable** is a Java interface that specifies three methods **accelerate()**, **brake()** and **stop()**. These methods take no argument and return no value. Write down the Drivable interface. **(3 marks)**
- Q 2 continued on next slide

► Go to Soln 2

(b) In this part of the question you will develop code for the `Vehicle` class. The class `Vehicle` inherits directly from `Object` and implements the `Drivable` interface.

(i) Write down the header for the `Vehicle` class. **(1 mark)**

(ii) Suppose `Vehicle` has a single private instance variable `speed` of type `int`. `Vehicle` implements the methods of the `Drivable` interface according to the following rules.

- ▶ `accelerate()` causes `speed` to be increased by 1.
- ▶ `brake()` causes `speed` to be decreased by 1, as long as it is greater than 0, otherwise it leaves it unchanged.
- ▶ `stop()` causes `speed` to be repeatedly decreased by 1 until it reaches 0.
- ▶ Write the code for these three methods. **(5 marks)**
- ▶ Q 2 continued on next slide

- (c) In this part of the question you will develop code for the `Car` class. The class `Car` is a subclass of `Vehicle`.

`Car` has two extra `int` instance variables `maxSpeed` and `increment`. **(7 marks)**

- (i) When an instance of `Car` receives the message `accelerate()`, it increases its `speed` by `increment` if that would not take the `speed` over `maxSpeed`, otherwise `speed` is left unchanged.

Write the `accelerate()` method for `Car`.

- (ii) What is the benefit of adding the `@Override` annotation to the `accelerate()` method for `Car`?
- (iii) Suppose that we want to keep a count of the number of `Car` objects that have been created. Explain using code fragments how we could achieve this.

► Q 2 continued on next slide

M250 2018J Exam

Q 2(d)

(d) Suppose that a class called `SpeedBoat`, which is unrelated to `Car`, also implements the `Drivable` interface, and that a class called `Service` has a public constructor that takes a formal argument of type `Drivable`. **(4 marks)**

(i) Briefly explain why lines `//1` and `//2` below are valid:

```
Car c = new Car();  
SpeedBoat sb = new SpeedBoat();  
Service s1 = new Service(c); //1  
Service s2 = new Service(sb); //2
```

(iii) Suppose that we want to keep a count of the number of `Car` objects that have been created. Explain using code fragments how we could achieve this.

► Q 2 continued on next slide

► Go to Soln 2(d)

M250 2018J Exam

Q 2(e),(f)

- (e) Describe **three** differences between **abstract classes** and **interfaces**. **(6 marks)**
- (f) Suppose that **SportsCar** is a subclass of **Car**. Describe what needs to be added to the class **SportsCar** (if anything) so that **SportsCar** will implement the interface **Drivable**. Briefly justify your answer. **(4 marks)**

(30 marks total)

► [Go to Soln 2\(e\)](#)

M250 2018J Exam

Q 3 (30 marks)

- **Scenario** Caravan owners who belong to a club make bookings in advance for their stays on various sites, giving their estimated time of arrival for each stay on a site.

The club wants to look at the pattern of estimated arrival times for all their caravan sites for a particular weekend so that they can organise staffing appropriately.

Two classes, [Booking](#) and [CaravanSite](#), have already been partially completed.

- Q 3 continued on next slide

► [Go to Soln 3](#)

M250 2018J Exam

Q 3 (contd)

- ▶ The class `Booking` already has the following instance variables, constructor and getters:
- ▶ A private instance variable `makeAndModel` of type `String` which represents the make and model of the caravan e.g. `"Swift Basecamp"`,
- ▶ A private instance variable `owner` of type `String`, which represents an owner name e.g. `"Sue Smith"`,
- ▶ A private instance variable `estArrivalHour` of type `int`, which represents the estimated arrival hour as a whole number using the 24-hour clock (e.g. 16 is used to represent 4pm),
- ▶ Q 3 continued on next slide

▶ Go to Soln 3

M250 2018J Exam

Q 3 (contd)

- ▶ A three-argument constructor that takes arguments of types `String`, `String` and `int` and uses them to set the instance variables,
- ▶ Getter methods for `makeAndModel`, `owner` and `estArrivalHour`.
- ▶ The class `CaravanSite` already has the following instance variables:
- ▶ A private instance variable `siteName` of type `String`, which represents the unique name of the caravan site (e.g. `"Park Coppice"`),
- ▶ A private instance variable `maxVans` of type `int`, which represents the maximum number of caravans that can be accommodated on that site.
- ▶ Q 3 continued on next slide

- (a) In this part of the question you will develop additional code for the `CaravanSite` class.
- (i) Write down the declaration of a private instance variable called `bookings`, which should be declared as a `List` of `Booking` elements, representing bookings currently made for the site, in the order the bookings were made. **(1 mark)**
- (ii) Write a two-argument constructor for `CaravanSite` that takes a `String` argument representing the name of the caravan site, and an `int` representing the maximum number of caravans that can be accommodated, and initialises the instance variables accordingly. The constructor should also initialise bookings with a suitable empty collection. **(3 marks)**
- Q 3(a) continued on next slide

M250 2018J Exam

Q 3(a) (contd)

- (iii) Write a public instance method `addBooking()` that takes a `Booking` argument representing the booking of a caravan.

As long as the number of bookings already made is less than the maximum number of caravans the site can accommodate, the `Booking` is added to bookings.

If there is not enough room then a suitable message is printed.

In both cases the remaining number of vans that can still be accommodated after this booking is returned.

(4 marks)

- Q 3 continued on next slide

► Go to Soln 3(a)(iii)

- (b) In this part of the question you will develop extra code for the `Booking` class so that instances of `Booking` may be sorted from earliest to latest estimated arrival hour.

Assume the `equals()` and `hashCode()` methods for `Booking` have already been written.

- (i) Write down the new class header for `Booking`, which must now implement an appropriate interface. **(1 mark)**
- (ii) Write a `compareTo(Booking)` method for `Booking` that will allow ordering of `Booking` instances as above. **(3 marks)**

► Q 3 continued on next slide

► Go to Soln 3(b)

M250 2018J Exam

Q 3(c)

- (c) Write a public instance method `orderBookings()` for the `CaravanSite` class that takes no argument and returns no value.

This method should reorder the elements of bookings by estimated arrival hour. **(2 marks)**

- Q 3 continued on next slide

► Go to Soln 3

- (d) In this part of the question you will develop code for a further class, `CaravanClub`. This class will help to determine the pattern of estimated arrival times across all caravan sites.

The class `CaravanClub` requires a single private instance variable `arrByTime`. This is a map where the key is a particular estimated arrival hour as a whole number (e.g. 16) and the value is an **unordered set** of `Booking` with that arrival time, from all caravan sites.

- (i) Write down the declaration of a private instance variable `arrByTime` of a suitable interface type to reference the map described above. **(2 marks)**
- (ii) Write a zero argument constructor that initialises `arrByTime` to a suitable collection. **(2 marks)**

► Q 3(d) continued on next slide

M250 2018J Exam

Q 3(d) (contd)

- (iii) Write the public instance method `addSite()`. This method takes a `CaravanSite` instance as the argument and has no return value. The method adds each of the bookings for that particular site to the `arrByTime` map, according to the bookings' estimated arrival hours.

Assume that the class `CaravanSite` has a public instance method `getBookings()` that returns a list of the bookings for that site.

Note that you cannot assume that a particular estimated arrival hour exists as a key in the map. **(8 marks)**

- Q 3 continued on next slide

► Go to Soln 3(d)

M250 2018J Exam

Q 3(e)

- (e)(i) Why is it preferable to declare a collection variable in terms of an interface type, such as `List`, rather than a concrete class, such as `ArrayList`, which implements that interface? Explain your answer, making two points. (2 marks)
- (ii) Give **two** ways in which an `ArrayList` is different from an array. (2 marks)

(30 marks total)

► Go to Soln 3(e)

M250 2018J Exam

Soln 1

(a)(i) Q 1

```
1 public class Performer {
2     private String equityName ;
3     private double payRate ;
4     private Branch branch ;
5     public static double minPayRate ;

6
7     public Performer(String aName) {
8         super() ;
9         equityName = aName ;
10        branch = null ;
11        payRate = Performer.minPayRate ;
12        /* or */
13        // payRate = minPayRate ;
14    }
```

► Soln 1(a)(i) continued on next slide

► Go to Q 1(a)(i)

[Agenda](#)

[Adobe Connect](#)

[Instructions 2018J](#)

[Qs](#)

[Solns](#)

[Soln 1](#)

[Soln 1\(a\)](#)

[Soln 1\(b\)](#)

[Soln 1\(c\)](#)

[Soln 1\(d\)](#)

[Soln 2](#)

[Soln 3](#)

[Exam Reminders](#)

[White Slide](#)

M250 2018J Exam

Soln 1(a)(i) (contd)

```
1  public void setPayRate(double aPayRate) {  
2      payRate = aPayRate ;  
3  }  
  
5  public Branch getBranch() {  
6      return branch ;  
7  }  
  
9  public void setBranch(Branch aBranch) {  
10     branch = aBranch ;  
11 }  
  
13 public String getEquityName() {  
14     return equityName ;  
15 }
```

► Soln 1(a) continued on next slide

► Go to Q 1(a)(i)

M250 2018J Exam

Soln 1(a)(ii)

(ii)

```
1 public boolean isInSameBranchAs(Performer p) {  
2     return branch.equals(p.getBranch()) ;  
3     /* or */  
4     // return getBranch().equals(p.getBranch()) ;  
5 }
```

► Soln 1(a) continued on next slide

► Go to Q 1(a)(ii)

M250 2018J Exam

Soln 1(a)(iii)

(iii)

```
1 public String getFirstName() {  
2     int spaceIndex = equityName.indexOf(" ") ;  
3     /* or */  
4     // int spaceIndex = equityName.indexOf(' ') ;  
5     return equityName.substring(0,spaceIndex) ;  
6 }
```

► Soln 1 continued on next slide

► Go to Q 1(a)(iii)

M250 2018J Exam

Soln 1(b)

- (b)
 - (i) messages are sent: lines 8,9,10,11,12
 - (ii) reference variables are declared: lines 1,3, 6, 7 (note latter two include initialization)
 - (iii) primitive variables are declared: None
 - (iv) object construction: lines 2,4,6,7
 - (v) operators are used: 2,4,6,7
 - (vi) formal arguments are declared: None
 - (vii) actual arguments are used: 2,4,6,7,8,9,10,11,12

► Go to Q 1(b)

M250 2018J Exam

Soln 1(c)

(c)

```
1  @Override
2  public boolean equals(Object obj) {
3      Performer pfmr = (Performer) obj ;
4      return equityName.equals(pfmr.equityName) ;
5  }
```

- ▶ This version assumes that the object is of type **Performer**
- ▶ See below for a more robust version
- ▶ Soln 1(c) continued on next slide

▶ Go to Q 1(c)

M250 2018J Exam

Soln 1(c) (contd)

(c) Alternative, more robust version

```
1  @Override
2  public boolean equals(Object obj) {
3      if (obj == this) {
4          return true ;
5      }
6      if (!(obj instanceof Performer)) {
7          return false ;
8      }
9      Performer pfmr = (Performer) obj ;
10     return equityName.equals(pfmr.equityName) ;
11 }
```

- It is recommended to override `hashCode()` if you are overriding `equals()`
- Soln 1 continued on next slide

► Go to Q 1(c)

M250 2018J Exam

Soln 1(d)

- (d)
- (i) A **Performer** object has a **Branch** — object composition not inheritance
 - (ii) **minPayRate** can be set since it is a class (static) variable and hence already exists with the class **Performer** definition.
 - (iii) The scope of a class member such as an instance variable is the entire class (except where shadowed by another declaration with the same name — there is none here).
The scope of a formal parameter is the body of the method

► Go to Q 1(d)

M250 2018J Exam

Soln 2(a)

(a) Q 2

```
1 public interface Drivable {  
2     void accelerate() ;  
3     void brake() ;  
4     void stop() ;  
5 }
```

- ▶ The method description modifiers of **abstract** and **public** are implicit
- ▶ Soln 2 continued on next slide

▶ Go to Q 2

M250 2018J Exam

Soln 2(b)

(b)

```
1 public class Vehicle implements Drivable {
2     private int speed ;

3
4     public Vehicle() {
5         super() ;
6         speed = 0 ;
7     }

8
9     public void accelerate() {
10         speed = speed + 1 ;
11     }
12 // } // continued below
```

► Soln 2(b) continued on next slide

► Go to Q 2(b)

[Agenda](#)

[Adobe Connect](#)

[Instructions 2018J](#)

[Qs](#)

[Solns](#)

[Soln 1](#)

[Soln 2](#)

[Soln 2\(a\)](#)

[Soln 2\(b\)](#)

[Soln 2\(c\)](#)

[Soln 2\(d\)](#)

[Soln 2\(e\)](#)

[Soln 2\(f\)](#)

[Soln 3](#)

[Exam Reminders](#)

[White Slide](#)

M250 2018J Exam

Soln 2(b) (contd)

```
1  public void brake() {  
2      if (speed > 0) {  
3          speed = speed - 1 ;  
4      }  
5  }  
  
7  public void stop() {  
8      while (speed > 0) {  
9          speed = speed - 1 ;  
10     }  
11 }  
  
13 public int getSpeed() {// required later  
14     return speed ;  
15 }  
  
17 public void setSpeed(int spd) {// required later  
18     speed = spd ;  
19 }  
20 }
```

► Soln 2 continued on next slide

Agenda

Adobe Connect

Instructions 2018J

Qs

Solns

Soln 1

Soln 2

Soln 2(a)

Soln 2(b)

Soln 2(c)

Soln 2(d)

Soln 2(e)

Soln 2(f)

Soln 3

Exam Reminders

White Slide

M250 2018J Exam

Soln 2(c)

(c)

```
1 public class Car extends Vehicle {
2     private int maxSpeed ;
3     private int increment ;
4     public static int count = 0 ; // Q2(c)(iii)
5
6     public Car() {
7         super() ;
8         Car.count = Car.count + 1 ; // Q2(c)(iii)
9     }
10
11     @Override
12     public void accelerate() {
13         if ((getSpeed() + increment) <= maxSpeed) {
14             super.setSpeed(super.getSpeed() + increment) ;
15         }
16     }
17 }
```

► Soln 2(c) continued on next slide

M250 2018J Exam

Soln 2(c) (contd)

- (c)
- (ii) `@Override` gets the Java compiler to check that the method signature is correct — see Unit 6, page 15
- (iii) See comments on code above

► Soln 2 continued on next slide

► Go to Q 2(c)

M250 2018J Exam

Soln 2(d)

(d) Q 2

- (i) Both **Car** and **SpeedBoat** implement the interface **Drivable** and the **Service** constructor takes an argument of type **Drivable**
 - (ii) Actual methods will depend on the class of object at runtime
- Soln 2 continued on next slide

► Go to Q 2(d)

M250 2018J Exam

Soln 2(e)

- (e)
- ▶ Only one abstract class can be inherited but a class may implement more than one interface
 - ▶ Abstract classes can declare instance variables but interface can not
 - ▶ Up to Java 8, interfaces could not declare default methods
 - ▶ Soln 2 continued on next slide

▶ Go to Q 2(e)

M250 2018J Exam

Soln 2(f)

(f)

Nothing is required since **SportsCar** will inherit the interface fields from **Car**

► Go to Q 2

M250 Revision
2021

Phil Molyneux

[Agenda](#)

[Adobe Connect](#)

[Instructions 2018J](#)

[Qs](#)

[Solns](#)

[Soln 1](#)

[Soln 2](#)

[Soln 2\(a\)](#)

[Soln 2\(b\)](#)

[Soln 2\(c\)](#)

[Soln 2\(d\)](#)

[Soln 2\(e\)](#)

[Soln 2\(f\)](#)

[Soln 3](#)

[Exam Reminders](#)

[White Slide](#)

M250 2018J Exam

Soln 3(a)

(a) Q 3

```
1 public class CaravanSite {
2     // provided
3     private String siteName ;
4     private int maxVans ;
5
6     private List<Booking> bookings ;
7
8     public CaravanSite(String aName, int aMaxVans) {
9         super() ;
10        siteName = aName ;
11        maxVans = aMaxVans ;
12        bookings = new ArrayList<>() ;
13    }
14    // } // continued below
```

► Soln 3(a) continued on next slide

[Agenda](#)

[Adobe Connect](#)

[Instructions 2018J](#)

[Qs](#)

[Solns](#)

[Soln 1](#)

[Soln 2](#)

[Soln 3](#)

[Soln 3\(a\)](#)

[Soln 3\(b\)](#)

[Soln 3\(c\)](#)

[Soln 3\(d\)](#)

[Soln 3\(e\)](#)

[Exam Reminders](#)

[White Slide](#)

► [Go to Q 3\(a\)](#)

M250 2018J Exam

Soln 3(a) (contd)

```
1 public int addBooking(Booking aBooking) {  
2     if (bookings.size() < maxVans) {  
3         bookings.add(aBooking) ;  
4     }  
5     else {  
6         System.out.println("No_space") ;  
7     }  
8     return maxVans - bookings.size() ;  
9 }
```

► Soln 3 continued on next slide

► Go to Q 3(a)(iii)

[Agenda](#)

[Adobe Connect](#)

[Instructions 2018J](#)

[Qs](#)

[Solns](#)

[Soln 1](#)

[Soln 2](#)

[Soln 3](#)

[Soln 3\(a\)](#)

[Soln 3\(b\)](#)

[Soln 3\(c\)](#)

[Soln 3\(d\)](#)

[Soln 3\(e\)](#)

[Exam Reminders](#)

[White Slide](#)

M250 2018J Exam

Soln 3(b)

(b) Q 3

```
1 public class Booking implements Comparable<Booking> {  
2     // provided  
3     private String makeAndModel ;  
4     private String owner ;  
5     private int estArrivalHour ;  
  
7     public int compareTo(Booking aBooking) {  
8         return estArrivalHour - aBooking.estArrivalHour ;  
9         /* or */  
10        // Integer.compare(estArrivalHour,  
11        //                    aBooking.estArrivalHour)  
12    }  
13 }
```

► Soln 3(b) continued on next slide

► Go to Q 3(b)

M250 Revision
2021

Phil Molyneux

Agenda

Adobe Connect

Instructions 2018J

Qs

Solns

Soln 1

Soln 2

Soln 3

Soln 3(a)

Soln 3(b)

Soln 3(c)

Soln 3(d)

Soln 3(e)

Exam Reminders

White Slide

M250 2018J Exam

Soln 3(b) (contd)

(b) Q 3 provided parts

```
1  public Booking(String aMandM
2                      ,String anOwner
3                      ,int anHour) {
4      super() ;
5      makeAndModel = aMandM ;
6      owner = anOwner ;
7      estArrivalHour = anHour ;
8  }
```

► Soln 3(b) continued on next slide

► Go to Q 3(b)

M250 2018J Exam

Soln 3(b) (contd)

(b) Q 3 provided parts

```
1  public String getMakeAndModel() {  
2      return makeAndModel ;  
3  }  
  
5  public String getOwner() {  
6      return owner ;  
7  }  
  
9  public int getEstArrivalHour() {  
10     return estArrivalHour ;  
11 }
```

► Soln 3(b) continued on next slide

► Go to Q 3(b)

[Agenda](#)

[Adobe Connect](#)

[Instructions 2018J](#)

[Qs](#)

[Solns](#)

[Soln 1](#)

[Soln 2](#)

[Soln 3](#)

[Soln 3\(a\)](#)

[Soln 3\(b\)](#)

[Soln 3\(c\)](#)

[Soln 3\(d\)](#)

[Soln 3\(e\)](#)

[Exam Reminders](#)

[White Slide](#)

M250 2018J Exam

Soln 3(b) (contd)

(b) Q 3 provided parts

```
1  @Override
2  public boolean equals(Object obj) {
3      Booking bkg = (Booking) obj ;
4      return makeAndModel.equals(bkg.makeAndModel)
5          && owner.equals(bkg.owner) ;
6  }

8  @Override
9  public int hashCode() {
10     return (20 + makeAndModel.hashCode())
11         * owner.hashCode() ;
12 }
```

► Soln 3 continued on next slide

► Go to Q 3(b)

M250 2018J Exam

Soln 3(c)

(c) Q 3

```
1  public void orderBookings() {  
2      Collections.sort(bookings) ;  
3  }  
  
5  public List<Booking> getBookings() {  
6      return bookings ;  
7  }
```

► Soln 3 continued on next slide

► Go to Q 3(c)

[Agenda](#)

[Adobe Connect](#)

[Instructions 2018J](#)

[Qs](#)

[Solns](#)

[Soln 1](#)

[Soln 2](#)

[Soln 3](#)

[Soln 3\(a\)](#)

[Soln 3\(b\)](#)

[Soln 3\(c\)](#)

[Soln 3\(d\)](#)

[Soln 3\(e\)](#)

[Exam Reminders](#)

[White Slide](#)

M250 2018J Exam

Soln 3(d)

(d) Q 3

```
1 public class CaravanClub {
2     private Map<Integer,Set<Booking>> arrByTime ;

4     public CaravanClub() {
5         arrByTime = new HashMap<>() ;
6     }

8     public void addSite(CaravanSite aSite) {
9         for (Booking aBooking : aSite.getBookings()) {
10             Integer hour = aBooking.getEstArrivalHour() ;
11             if (!(arrByTime.containsKey(hour))) {
12                 arrByTime.put(hour, new HashSet<>()) ;
13             }
14             arrByTime.get(hour).add(aBooking) ;
15         }
16     }
17 }
```

► Soln 3 continued on next slide

► Go to Q 3(d)

Agenda

Adobe Connect

Instructions 2018J

Qs

Solns

Soln 1

Soln 2

Soln 3

Soln 3(a)

Soln 3(b)

Soln 3(c)

Soln 3(d)

Soln 3(e)

Exam Reminders

White Slide

M250 2018J Exam

Soln 3(e)

(e) Q 3

- (i) The interface is the real type of the variable, parameter, method or other field and should be used instead of the implementation class — this enables flexibility and maintainability
See page 76 of Unit 10 *Sets and Maps* and Item 64, page 280 Bloch (2017)
- (ii) [ArrayList](#) is expandable unlike [Array](#) — it implements the [List](#) interface which has different fields and methods to [Array](#)
See Item 28, page 126 Bloch (2017)

► Go to Q 3(e)

[Agenda](#)

[Adobe Connect](#)

[Instructions 2018J](#)

[Qs](#)

[Solns](#)

[Soln 1](#)

[Soln 2](#)

[Soln 3](#)

[Soln 3\(a\)](#)

[Soln 3\(b\)](#)

[Soln 3\(c\)](#)

[Soln 3\(d\)](#)

[Soln 3\(e\)](#)

[Exam Reminders](#)

[White Slide](#)

Exam

Reminders

- ▶ Read the [Exam arrangements booklet](#)
- ▶ Before the exam — check the date, time and location (and how to get there)
- ▶ At the exam centre – arrive early
- ▶ Bring photo ID with signature
- ▶ Use black or [blue](#) pens (not erasable and not pencil) — see [Cult Pens](#) for choices — pencils for preparing diagrams (HB or blacker)
- ▶ Practice writing by hand
- ▶ In the exam — Read the questions — carefully — before and after answering them
- ▶ Don't get stuck on a question — move on, come back later
- ▶ But do make sure you have attempted all questions
- ▶ ... and finally [Good Luck](#)

Exam Revision