M250 Exam Revision

M250 Tutorial 07

Contents

1 M250 Exam Revision: Agenda **2**

2 Adobe Connect **3**
- 2.1 Interface . . . 3
- 2.2 Settings . . . 4
- 2.3 Sharing Screen & Applications . . . 5
- 2.4 Ending a Meeting . . . 6
- 2.5 Invite Attendees . . . 6
- 2.6 Layouts . . . 7
- 2.7 Chat Pods . . . 8
- 2.8 Web Graphics . . . 8
- 2.9 Recordings . . . 9

3 Spec 2021 Rubric **9**

4 Spec 2021 Questions **9**
- 4.1 Q 1 . . . 9
- 4.2 Soln 1 . . . 10
- 4.3 Q 2 . . . 11
- 4.4 Soln 2 . . . 11
- 4.5 Q 3 . . . 12
- 4.6 Soln 3 . . . 13
- 4.7 Q 4 . . . 14
- 4.8 Soln 4 . . . 14
- 4.9 Q 5 . . . 15
- 4.10 Soln 5 . . . 15
- 4.11 Q 6 . . . 16
- 4.12 Soln 6 . . . 16
- 4.13 Q 7 . . . 17
- 4.14 Soln 7 . . . 19
- 4.15 Q 8 . . . 20
- 4.16 Soln 8 . . . 20
- 4.17 Q 9 . . . 21
- 4.18 Soln 9 . . . 23
- 4.19 Q 10 . . . 24
- 4.20 Soln 10 . . . 25

5 Spec 2021 Solns **26**

6 Prsntn 2018J Qs **26**
- 6.1 Qs . . . 26
- 6.2 Q 1 . . . 26
  - 6.2.1 Q 1(a) . . . 26
  - 6.2.2 Q 1(b) . . . 27

6.2.3 Q 1(c) . . . 28
6.2.4 Q 1(d) . . . 28
6.3 Q 2 . . . 28
6.3.1 Q 2(a) . . . 28
6.3.2 A 2(b) . . . 28
6.3.3 Q 2(c) . . . 29
6.3.4 Q 2(d) . . . 29
6.3.5 Q 2(e)(f) . . . 29
6.4 Q 3 . . . 30
6.4.1 Q 3(a) . . . 30
6.4.2 Q 3(b) . . . 31
6.4.3 Q 3(c) . . . 31
6.4.4 Q 3(d) . . . 31
6.4.5 Q 3(e) . . . 32

7 Prsntn 2018J Solns 32
7.1 Solns . . . 32
7.2 Soln 1 . . . 32
7.2.1 Soln 1(a) . . . 32
7.2.2 Soln 1(b) . . . 33
7.2.3 Soln 1(c) . . . 34
7.2.4 Soln 1(d) . . . 34
7.3 Soln 2 . . . 34
7.3.1 Soln 2(a) . . . 34
7.3.2 Soln 2(b) . . . 35
7.3.3 Soln 2(c) . . . 35
7.3.4 Soln 2(d) . . . 36
7.3.5 Soln 2(e) . . . 36
7.3.6 Soln 2(f) . . . 36
7.4 Soln 3 . . . 36
7.4.1 Soln 3(a) . . . 36
7.4.2 Soln 3(b) . . . 37
7.4.3 Soln 3(c) . . . 37
7.4.4 Soln 3(d) . . . 38
7.4.5 Soln 3(e) . . . 38

8 What Next ? 38

9 References 39
9.1 Java Documentation . . . 39
9.2 Books Phil Likes . . . 40
References . . . 41

1 M250 Exam Revision: Agenda

- Introductions
- Adobe Connect reminders
- *Adobe Connect* — if you or I get cut off, wait till we reconnect (or send you an email)
- M250 Specimen Exam from 2021

- M250 Exam 2019 from Presentation 2018J
- Revision strategies and exam techniques

Introductions — Phil

- *Name* Phil Molyneux
- *Background*
  - Undergraduate: Physics and Maths (Sussex)
  - Postgraduate: Physics (Sussex), Operational Research (Brunel), Computer Science (University College, London)
  - Worked in Operational Research, Business IT, Web technologies, Functional Programming
- *First programming languages* Fortran, BASIC, Pascal
- *Favourite Software*
  - Haskell — pure functional programming language
  - Text editors TextMate, Sublime Text — previously Emacs
  - Word processing in LaTeX — all these slides and notes
  - Mac OS X
- *Learning style* — I read the manual before using the software

Introductions — You

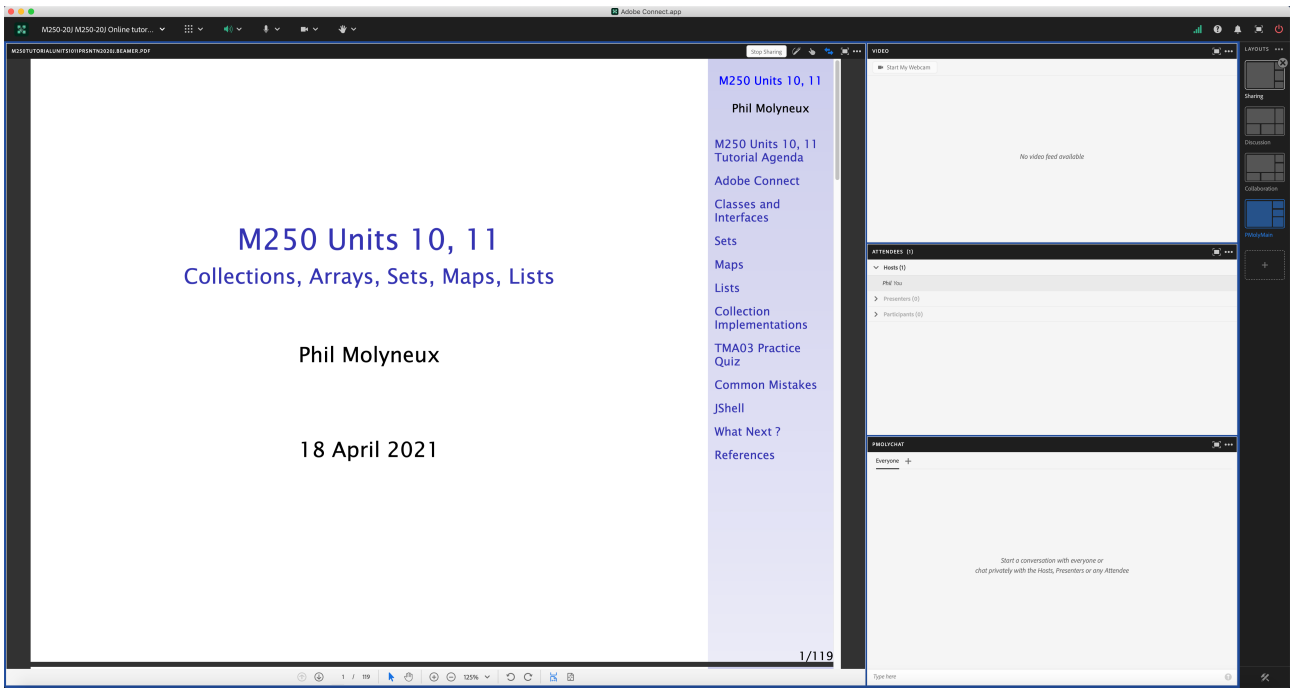
- *Name* ?
- What other exams are you taking this year ?
- Give one revision tip and exam tip to the group

ToC

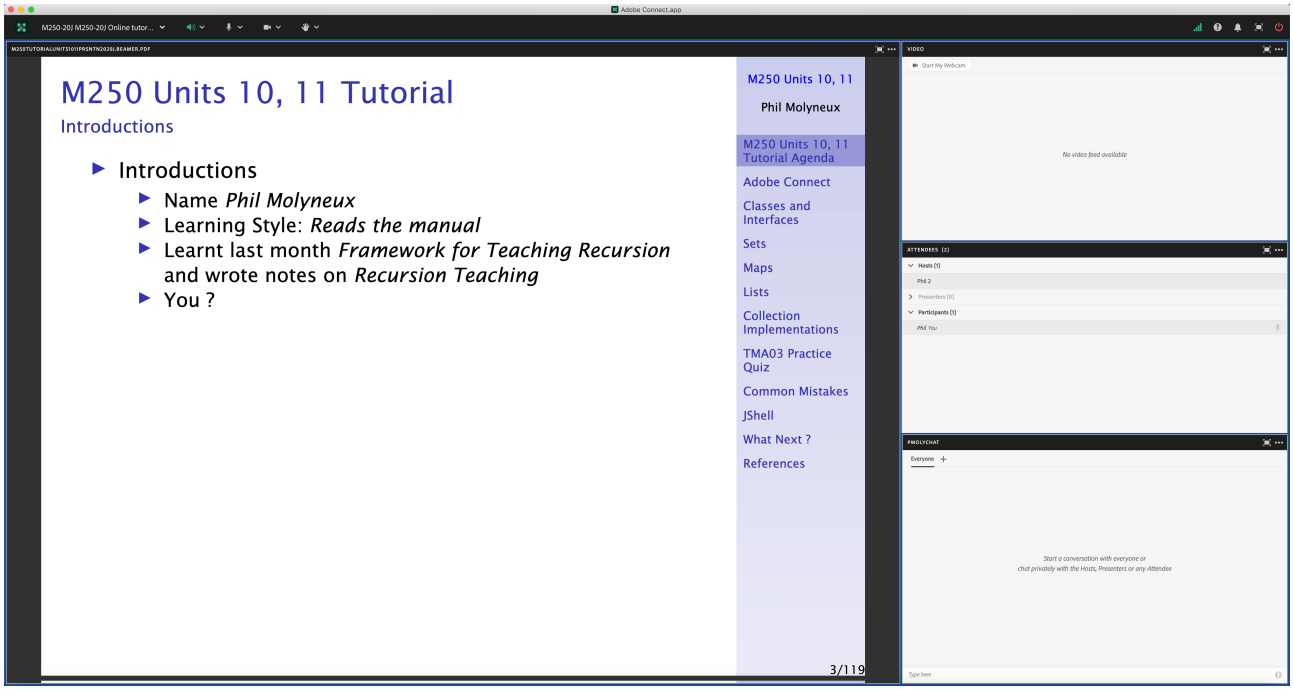
2 Adobe Connect Interface and Settings

2.1 Adobe Connect Interface

Adobe Connect Interface — Host View



Adobe Connect Interface — Participant View



2.2 Adobe Connect Settings

Adobe Connect — Settings

- **Everybody** *Menu bar* › *Meeting* › Speaker & Microphone Setup
- *Menu bar* › *Microphone* › Allow Participants to Use Microphone ✔
- Check Participants see the entire slide including slide numbers bottom right **Workaround**
  - *Disable Draw* *Share pod* › *Menu bar* › *Draw icon*
  - *Fit Width* *Share pod* › *Bottom bar* › *Fit Width icon* ✔

- *Meeting* > Preferences > General > **Host Cursor** > Show to all attendees
- *Menu bar* > *Video* > Enable **Webcam** for Participants ✔
- Do not *Enable single speaker mode*
- Cancel hand tool
- Do not enable green pointer
- **Recording** Meeting > Record Session ✔
- **Documents** Upload PDF with drag and drop to share pod
- Delete *Meeting* > Manage Meeting Information > Uploaded Content and *check filename* > *click on delete*

Adobe Connect — Access

- **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** ⌘ + \

2.3 Adobe Connect — 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 blued 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

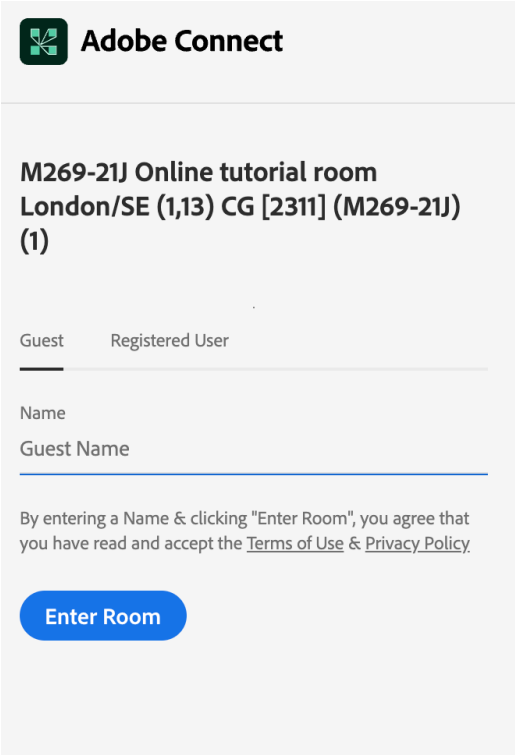
2.4 Adobe Connect — Ending a Meeting

- *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.
- **Delete File Upload** Meeting > Manage Meeting Information > Uploaded Content tab select file(s) and click Delete
- **Attendance Report** see course Web site for joining room

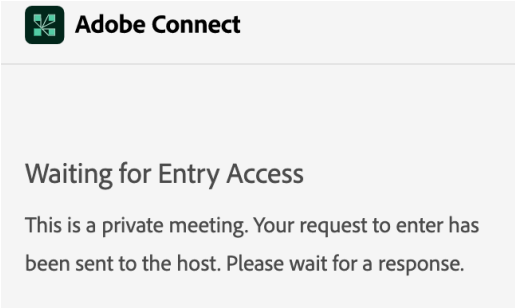
2.5 Adobe Connect — Invite Attendees

- **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
- Click on the link sent in email from the Host
- Get the following on a Web page

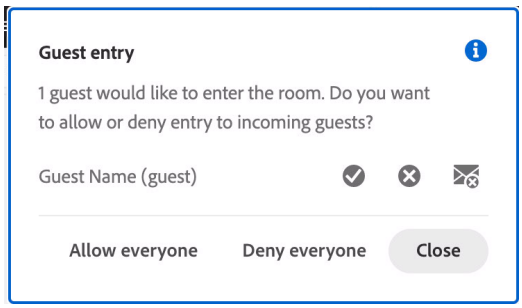
- As *Guest* enter your name and click on Enter Room



- See the *Waiting for Entry Access* for *Host* to give permission



- *Host* sees the following dialog in *Adobe Connect* and grants access



2.6 Layouts

- **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 *PMolyShare*
- **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 — rename 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)
- **Auxiliary Layouts** name *PMolyAux0n*
  - Create new Share pod
  - Use existing Chat pod
  - Use same Video and Attendance pods

2.7 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

2.8 Graphics Conversion for Web

- Conversion of the screen snaps for the installation of Anaconda on 1 May 2020
- Using GraphicConverter 11
- File > Convert & Modify > Conversion > Convert
- Select files to convert and destination folder
- Click on Start selected Function or ⌘+↵

2.9 Adobe Connect Recordings

- *Menu bar* > *Meeting* > Preferences > Video
- Aspect ratio > Standard (4:3) (not Wide screen (16:9) default)
- Video quality > Full HD (1080p not High default 480p)
- **Recording** *Menu bar* > *Meeting* > Record Session ✔
- **Export Recording**
- *Menu bar* > *Meeting* > Manage Meeting Information
- *New window* > Recordings > *check Tutorial* > Access Type button
- check Public > check Allow viewers to download
- **Download Recording**
- *New window* > Recordings > *check Tutorial* > Actions > Download File

ToC

3 Specimen Exam for 2021 — Rubric

- Time limit 3 hours 30 minutes
- Answer all questions
- Part 1 — 4 Short Questions (25 marks)
- Part 2 — A simple class (2 questions 15 marks)
- Part 3 — Class relationships (2 questions 30 marks)
- Part 4 — Collections (2 questions 30 marks)
- Note that the order of sub-questions varies from attempt to attempt — so my slides may vary in presentation from the version you used

ToC

4 Spec 2021 Questions

4.1 M250 From 2021 Specimen 2021 Exam Q 1

- The following code does not compile. Why not? Select all the reasons that lead to compilation errors.

```
Public class Robot
{
private int x;

public Robot()
{
s = 1;
}

public getX()
{
return x;
}
```

```
public void DecreaseX()
{
x++;
}
```

- Select one or more:

1. The method DecreaseX does not follow naming conventions.
2. The constructor uses an undeclared variable.
3. The method getX should have an argument.
4. The method DecreaseX needs to decrement x.
5. The keyword this must be used to access the field x.
6. There is a brace (curly bracket) missing.
7. The constructor should declare a return type.
8. The method getX should declare a return type.
9. The code is not indented correctly.
10. The class header is not valid.
11. The instance variable x is not explicitly initialised.

Go to Soln 1

ToC

4.2 M250 From 2021 Specimen 2021 Exam Soln 1

- The following lead to compilation errors:

2. The constructor uses an undeclared variable.
6. There is a brace (curly bracket) missing.
8. The method getX should declare a return type.
10. The class header is not valid.

- Code that does compile is given below

Go to Q 1

- The following does compile

```
public class Robot {
  private int x ;

  public Robot() {
    x = 1 ;
  }

  public int getX() {
    return x ;
  }

  public void DecreaseX() {
    x++ ;
  }
}
```

Go to Q 1

ToC

4.3 M250 From 2021 Specimen 2021 Exam Q 2

- Given the following declarations and initialisations, select the two correct options below.

```
1   String fish1 = "FISH";
2   String fish2 = "FiSh".toUpperCase();
3   String fish3 = fish1;
4   String fish = "fish";

6   System.out.println(fish1 == fish2);          // line 1
7   System.out.println(fish1.equals(fish2));     // line 2
8   System.out.println(fish1 == fish3);          // line 3
9   System.out.println(fish == fish1);           // line 4
10  System.out.println(fish.equals(fish1));      // line 5
```

- Select one or more:

1. The result of executing line 1 is `true` because `fish1` and `fish2` both reference strings consisting of the four characters `F`, `I`, `S` and `H`.
2. The result of executing line 2 is `true` because `fish1` and `fish2` both reference a string consisting of, in that order, the four characters `F`, `I`, `S` and `H`.
3. The result of executing line 3 is `true` because `fish1` and `fish3` both reference the same `String` object consisting of the four characters `F`, `I`, `S` and `H`.
4. The result of executing line 4 is `true` because `fish` and `fish1` both reference the same string consisting of the four characters `F`, `I`, `S` and `H`.
5. The result of executing line 5 is `true` because `fish` and `fish1` both reference the same string consisting of the four characters `F`, `I`, `S` and `H`.

Go to Soln 2

ToC

4.4 M250 From 2021 Specimen 2021 Exam Soln 2

- Answers

```
1   String fish1 = "FISH";
2   String fish2 = "FiSh".toUpperCase();
3   String fish3 = fish1;
4   String fish = "fish";

6   System.out.println(fish1 == fish2);          // line 1 false
7   System.out.println(fish1.equals(fish2));     // line 2 true
8   System.out.println(fish1 == fish3);          // line 3 true
9   System.out.println(fish == fish1);           // line 4 false
10  System.out.println(fish.equals(fish1));      // line 5 false
```

Go to Q 2

ToC

4.5 M250 From 2021 Specimen 2021 Exam Q 3

- Given the following class modelling a music CD, answer the four sets of questions about it.

```java
class CD {
  private String artist ;
  private String title ;
  private int minutes ;

  public CD(String anArtist, String aTitle, int numMinutes) {
    artist = anArtist ;
    title = aTitle ;
    minutes = numMinutes ;
  }

  public String toString() {
    return "Artist: " + artist + " Title: " + title + " Playing time: " + minutes ;
  }
```

- Code continued:

```java
  public void hours() {
    if (minutes < 60) {
      System.out.println("Less than one hour") ;
    }
    else {
      int hrs ;
      hrs = minutes / 60 ;
      System.out.println("hours " + hrs) ;
    }
  }
}
```

1 Match the following features to their correct names

(a) `public CD(String anArtist, String aTitle, int numMinutes)`

(b) `private String title ;`

(c) `minutes = 60`

(d) `int hrs ;`

(e) `/`

(f) `anArtist`

Choose...		
expression	operand	signature
method header	actual parameter	formal parameter
local variable declaration	constructor header	operator
literal	field declaration	

2 Which of the following features occur? (Tick the correct ones)

(a) method chaining

(b) multiple inheritance

(c) overloading

(d) overriding

(e) composition

(f) polymorphism

3 How many are there of each of the following? (Type in a digit, not a word)

(a) different operators ☐ (don't count repeats of the same operator)

(b) methods ☐

(c) primitive type instance variables ☐

(d) reference type instance variables ☐

4 Which sets of variables have the same scope?

Select **true** if the variables have the same scope, otherwise select **false**.

(a) `artist, title` **true/false**

(b) `artist, title, minutes` **true/false**

(c) `hrs, minutes` **true/false**

(d) `anArtist, aTitle, numMinutes` **true/false**

Go to Soln 3

ToC

4.6 M250 From 2021 Specimen 2021 Exam Soln 3

(a) `public CD(String anArtist, String aTitle, int numMinutes)` constructor header

(b) `private String title ;` field declaration

(c) `minutes = 60` expression

(d) `int hrs ;` local variable declaration

(e) `/` operator

(f) `anArtist` formal parameter

2 Which of the following features occur? (Tick the correct ones)

(a) method chaining

(b) multiple inheritance

(c) overloading

(d) overriding yes

(e) composition yes

(f) polymorphism

3 How many are there of each of the following? (Type in a digit, not a word)

(a) different operators 4 (don't count repeats of the same operator) {=,+,<,/} Do not forget (.) is a separator not an operator, see Java Language Specificati 3.11 Separators, 3.12 Operators

(b) methods 2

(c) primitive type instance variables 1

(d) reference type instance variables 2

4 Which sets of variables have the same scope?

Select **true** if the variables have the same scope, otherwise select **false**.

(a) `artist, title` true/**false**

(b) `artist, title, minutes` true/**false**

(c) `hrs, minutes` **true**/false

(d) `anArtist, aTitle, numMinutes` true/**false**

ToC

4.7 M250 From 2021 Specimen 2021 Exam Q 4

- Write a public method in the class `Test` that has the signature `concatenateThese(int, int)` and does not return any value.
- The method concatenates all of the elements between the given indexes of the `nums` array into a single string and prints that string out (see the example below).
- You do not have to perform any checks on the parameters to see whether they are in bounds for the array.
- For example

Test	Result
Test t = new Test(new int[]{1,3,7,9,10}); t.concatenateThese(2,4);	7910

- Complete your code in the following:

```
public class Test {
  private int[] nums;
  public Test(int[] vals) {
      nums = vals;
  }
  // Write your concatenateThese(int, int) method here
}
```

Go to Soln 4

ToC

4.8 M250 From 2021 Specimen 2021 Exam Soln 4

- Sample answer

```
public class Test {
  private int[] nums;
  public Test(int[] vals) {
      nums = vals;
  }
  // Write your concatenateThese(int, int) method here
  public void concatenateThese(int x, int y) {
    String numsStr = "" ;
    for (int i = x; i <= y; i++) {
      numsStr = numsStr + this.nums[i] ;
    }
    System.out.println(numsStr) ;
  }
}
```

Go to Q 4

ToC

4.9 M250 From 2021 Specimen 2021 Exam Q 5

- **Scenario** This question concerns a class called House which is to be developed to model some aspects of a house.
- Write a class to complete the requirements in (a)-(g) below:

(a) The class is to be called `House`.

(b) The class requires two private instance variables called `material` of type `String`, and `age` of type `int`.

(c) Add a public constructor to the class that takes two parameters. The first parameter is of type `String` and the second parameter is of type `int`. Use the first parameter to set the `material` field, and the second parameter to set the `age` field.

(d) Write standard getter methods for the two fields.

(e) Write standard setter methods for the two fields.

(f) Write a public method called `about` that returns a string of the following form:

A *material* house of age *age*

- The *material* and *age* should be replaced by the actual `material` and `age` of the house.

(g) Write a public method with the signature `equals(House)` that returns `true` if the fields of the actual parameter have the same values as the fields of this object, and returns `false` otherwise.

- Below is an example test case for this class.
- For example

Test	Result
House a = new House("brick", 23); System.out.println(a.getMaterial());	brick

- Complete your code in the following:

```
// write your answer here
```

Go to Soln 5

ToC

4.10 M250 From 2021 Specimen 2021 Exam Soln 5

- Possible answer

```
// (a) class header

public class House {
  // (b) private instance variables
  private String material ;
  private int age ;
```

```java
    // (c) public constructor
    public House(String aMaterial, int anAge) {
      material = aMaterial ;
      age = anAge ;
    }

    // (d) standard getter methods
    public String getMaterial() {
      return material ;
    }

    public int getAge() {
      return age ;
    }
```

```java
    // (e) standard setter methods
    public void setMaterial(String aMaterial) {
      material = aMaterial ;
    }

    public void setAge(int anAge) {
       age = anAge ;
    }

    // (f) public method called about that returns a string
    public String about() {
      return "A␣" + material + "␣house␣of␣age␣" + age ;
    }

    // (g) public method with the signature equals(House)
    public boolean equals(House aHouse) {
      return this.getMaterial().equals(aHouse.getMaterial()) && this.age == aHouse.getAge() ;
    }
  }
```

- Note in (g) return of `Boolean` instead of `boolean` loses marks

Go to Q 5

ToC

4.11 M250 From 2021 Specimen 2021 Exam Q 6

- Based on the `House` class just described, select the correct answers below:

(a) The class overrides [0,1,2,...] method(s).

(b) The class overloads [0,1,2,...] method(s).

(c) The class [*is not/is*] a subclass of `Object`.

(d) The class [*does not demonstrate/demonstrates*] information hiding.

(e) The class [*needs to/does not need to*] use at least one external method call.

Go to Soln 6

ToC

4.12 M250 From 2021 Specimen 2021 Exam Soln 6

- Sample answers:

(a) The class overrides [**0**] method(s).

(b) The class overloads [**1**] method(s).

The `equals` method (since not same signature as `equals` inherited from `Object`)

(c) The class **is** a subclass of `Object`.

(d) The class **demonstrates** information hiding.

(e) The class **needs to** use at least one external method call.

The `equals` method of `String`

Go to Q 6

ToC

4.13 M250 From 2021 Specimen 2021 Exam Q 7

- **Scenario** This question concerns a class called `Child` which is to be developed as a subclass of the class `Person` (click to view this file), which has already been developed.
- Persons have a first name, a last name, wear white shirts by default, have a number of friends, and have an amount of money.
- Children may be in a playing state, or not playing.
- The Java library class `java.awt.Color` is used to represent shirt colours such as `Color.WHITE`.
- Note that printing out a colour produces output such as `java.awt.Color[r=0,g=0,b=255]` (which represents `Color.BLUE` in this case). The three numbers represent components of Red, Green, and Blue colour.
- **Develop only the class `Child`**

(a) Add the class `Child` below, making it a subclass of `Person`.

(b) Add a private instance variable to the class called `playing` of type `boolean`.

(c) Add a public constructor for `Child` whose first parameter is the child's first name and whose second parameter is the child's second name.

The constructor should initialise the child's first and last names using the received arguments.

The instance variable `playing` should be set to `true`. The initial `money` should be set to `10`.

(d) Add a standard getter method for `playing` called `isPlaying`.

(e) Add a public method `play`, which takes no arguments and returns no value.

The method sets `playing` to `true` and increments the child's number of friends by `1`.

(f) Add a public method `work`, which takes no arguments and returns no value.

The method sets sets `playing` to `false` and and decrements the child's number of friends by `1`.

(Don't worry about the value becoming negative.)

(g) Add a public method `getNickname` that returns a nickname for the child based on their first and last names at the time the method is called.

The method returns the first three letters of the child's first name concatenated to the last three letters of the child's last name in lowercase.

(You can assume the names are long enough.)

For example, if the child's first name is `"Betsy"` and their last name is `"Corble"` the method will return the string `"Betble"`.

(h) Add a public method `buySnack` which does not return a value and has a single parameter of type `int` representing the cost of a snack.

If the child has enough money to buy the snack then the method decreases the money the child owns by the argument received, otherwise it just prints

```
I need money
```

(i) Add a public method `goHome` which does not return a value and takes no arguments.

If the child has no friends then the method prints

```
I'm going home
```

Otherwise the method prints

```
Bye
```

as many times as the child has friends.

(i) Add a public `setShirtColour` method to override the inherited method of that name.

The child's method behaves in the same way as the inherited method provided that the child is not playing.

When a child is playing, its `setShirtColour` method behaves as follows:

If the child is wearing a shirt that is `Color.WHITE` then the method prints

```
I'm changing now
```

before setting the child's shirt colour to the received argument.

If the child is not wearing a shirt that is `Color.WHITE` then the method prints

```
I'm wearing play clothes already
```

but doesn't change the shirt colour.

- Below is an example test case for this class.
- **For example**

Test	Result
Child f = new Child("Jan", "Feb");	1
System.out.println(f.getNumFriends());	false
f.work();	0
System.out.println(f.isPlaying());	
System.out.println(f.getNumFriends());	

Go to Soln 7

ToC

4.14 M250 From 2021 Specimen 2021 Exam Soln 7

- Sample answer

```java
import java.awt.Color ;


// (a) Child class header
public class Child extends Person {
  // (b) private instance variable
  private boolean playing ;

  // (c) Child constructor
  public Child(String aFirstName, String aLastName) {
    super(aFirstName,aLastName) ;
    playing = true ;
    this.setMoney(10) ;
  }
```

```java
  // (d) getter
  public boolean isPlaying() {
    return this.playing ;
  }

  // (e) setter
  public void play() {
    this.playing = true ;
    this.setNumFriends(this.getNumFriends() + 1) ;
  }
```

```java
// (f) public method work
public void work() {
  this.playing = false ;
  this.setNumFriends(this.getNumFriends() - 1) ;
}

// (g) public method getNickname
public String getNickname() {
  String lstNm = this.getLastName() ;
  int lenLstNm = lstNm.length() ;
  String fstNm = this.getFirstName() ;
  String fst3  = fstNm.substring(0,3) ;
  String lst3  = lstNm.substring(lenLstNm - 3) ;
  return fst3 + lst3 ;
}
```

```java
// (h) public method buySnack
public void buySnack(int snkCst) {
  int mny = this.getMoney() ;
  if (snkCst <= mny) {
    this.setMoney(mny - snkCst) ;
  } else {
    System.out.println("I_need_money") ;
  }
}

// (i) public method goHome
public void goHome() {
  int nmFrnds = this.getNumFriends() ;
  if (nmFrnds > 0) {
    for (int i = 1; i <= nmFrnds; i++) {
      System.out.println("Bye") ;
    }
  } else {
    System.out.println("I'm_going_home") ;
  }
}
```

```java
// (j) public method setShirtColour
@Override
public void setShirtColour(Color aColour) {
  Color shrtClr = this.getShirtColour() ;
  if (shrtClr.equals(Color.WHITE)) {
    System.out.println("I'm_changing_now") ;
```

```
      super.setShirtColour(aColour) ;
    } else {
      System.out.println("I'm wearing play clothes already") ;
    }
  }

}
```

ToC

4.15 M250 From 2021 Specimen 2021 Exam Q 8

- Answer parts (a)–(d) below.

(a) Consider the following code based on the person and child scenario and the code developed in this question.

Select all of the following statements that will compile.

(i) `Child c = new Child("Celia", "Goth");`

(ii) `Person p = new Child("Penny", "Bun");`

(iii) `Person p = new Person("Kim", "Wilde");`

(iv) `Person p = new Object("Janet", "Becker");`

(v) `Person p = new Child("Penny");`

(vi) `Child c = new Person("Tom", "Sawyer");`

(b) Suppose that the following further code is added to the class `Child`

```
  public String playingString() {
     return (playing ? "not playing" : "playing") ;
  }
```

Select all of the true statements in this scenario:

(i) The `playingString` method contains a logical error.

(ii) The `playingString` method overloads the toString method in the Object class.

(iii) The `playingString` method will not compile because it contains a syntax error.

(iv) The `playingString` method overloads the toString method in the Person class.

(v) The `playingString` method will not compile because it is missing the `@Override` annotation.

(vi) The `playingString` method will cause an exception.

ToC

4.16 M250 From 2021 Specimen 2021 Exam Soln 8

(a) Consider the following code based on the person and child scenario and the code developed in this question.

Select all of the following statements that will compile.

(i) `Child c = new Child("Celia", "Goth");` yes

(ii) `Person p = new Child("Penny", "Bun");` yes

(iii) `Person p = new Person("Kim", "Wilde");` yes

(iv) `Person p = new Object("Janet", "Becker");`

(v) `Person p = new Child("Penny");`

(vi) `Child c = new Person("Tom", "Sawyer");`

(b) Suppose that the following further code is added to the class `Child`

```
public String playingString() {
   return (playing ? "not_playing" : "playing") ;
}
```

Select all of the true statements in this scenario:

(i) The `playingString` method contains a logical error. yes

(ii) The `playingString` method overloads the toString method in the Object class.

(iii) The `playingString` method will not compile because it contains a syntax error.

(iv) The `playingString` method overloads the toString method in the Person class.

(v) The `playingString` method will not compile because it is missing the `@Override` annotation.

(vi) The `playingString` method will cause an exception.

Go to Q 8

ToC

4.17 M250 From 2021 Specimen 2021 Exam Q 9

- **Scenario** A concert hall hosts musical concerts.

  A concert has a programme of musical performances.

  Each item of music in a programme has a title and a composer.

- The class Music (click to view this file) has already been developed.

- Please note that the answer box below contains **two** classes to complete.

  You can only use import statements at the top of the answer box.

- The answer box below includes some methods of the class `Concert` that you should not alter, as well as a wrapper for the `ConcertHall` class.

(a) **Develop only the class `Concert` in this part.**

(i) Add a declaration for a private instance variable called `programme`, which should be declared as a `List` containing elements of type `Music`.

(ii) Add a public constructor for `Concert` that takes two string parameters representing the date of the concert and the concert name, and initialises the related variables accordingly.

The constructor should also initialise `programme` to a suitable empty collection.

(iii) Add a standard getter method for the `programme` collection.

(iv) Write a public instance method `getConcertLength` that takes no parameters and returns the length of the concert in minutes.

The method will need to loop through the `Music` items in `programme`, and add up all their performance times then return the total.

(v) Write a public instance method `addProgrammeItem` that takes an argument of type Music and returns no value.

If the running time of the concert will not exceed `MAX_LENGTH` by adding the music to the programme list then it is added, otherwise the error message

```
Running time exceeded
```

is printed instead.

(vi) The concert hall owners want to be able to sort concerts by their `concertName`.

Modify the `Concert` class so that it implements the appropriate interface, and then implement the `compareTo` method that will allow the ordering required.

(b) **Develop only the class `ConcertHall` in this part.**

(i) Add a *public* instance variable `whatsOn` to the ConcertHall class that will be used to map between sorted composers' names and sorted sets of names of their music that are performed in a concert.

(For example, when populated the map might include a mapping from the name `"Elgar"` to a sorted set of Elgar's music including `"Engima Variations"` and `"Sospiri"`.)

Note that this field needs to be made public for testing purposes.

(ii) Add a public `ConcertHall` constructor that initialises `whatsOn` to a suitable collection type (initially empty).

(iii) Add a public instance method `addConcert` that takes a parameter of type `Concert` and does not return a value.

This method's job is to populate the `whatsOn` map according to the contents of the concert.

Remember that a Concert has a `programme` of music.

When the `addConcert` method is finished running the `whatsOn` map should contain a mapping for each composer whose music is in the concert programme, with the value being the sorted set of the composer's music in the programme.

- Below is an example test case for this class.
- **For example**

Test	**Result**
//check constructor executes and initialisation Concert c = new Concert("2021-12-20", "Happy days"); System.out.println(c.getDate()); System.out.println(c.getConcertName()); System.out.println(c.getProgramme());	2021-12-20 Happy Days []

Go to Soln 9

ToC

4.18 M250 From 2021 Specimen 2021 Exam Soln 9

- Sample answer from file Concert.java

```java
import java.util.* ;

//Scroll down to see the ConcertHall class below the Concert class
/**
 * The class Concert models a musical event at a concert hall.
 * Complete the class according to the instructions in part (a)
 */
// (a)(vi) interface imlementation
class Concert implements Comparable<Concert> {
  private String concertName;
  private String date ; // in "yyyy-mm-dd" format
  public static final int MAX_LENGTH = 120 ;
  // (a)(i) private instance variable
  private List<Music> programme ;
```

```java
// (a)(ii) public constructor
public Concert(String aDate, String aConcertName) {
  date = aDate ;
  concertName = aConcertName ;
  programme = new ArrayList<>() ;
}

/**
 * Getter for the date of the concert
 */
public String getDate() {
  return this.date ;
}

/**
 * Getter for the name of the concert
 */
public String getConcertName() {
  return this.concertName;
}
```

```java
// (a)(iii) getter for programme
public List<Music> getProgramme() {
  return this.programme ;
}

// (a)(iv) public instance method getConcertLength
public int getConcertLength() {
  List<Music> aProgramme = this.getProgramme() ;
  int concertLength = 0 ;
  for (Music progItem : aProgramme) {
    concertLength = concertLength + progItem.getPerformanceTime() ;
  }
  return concertLength ;
}
```

```java
// (a)(v) public instance method addProgrammeItem
public void addProgrammeItem(Music progItem) {
  int concertLength = this.getConcertLength() ;
  int progItemLength = progItem.getPerformanceTime() ;
  if (concertLength + progItemLength <= MAX_LENGTH) {
    this.getProgramme().add(progItem) ;
  } else {
    System.out.println("Running time exceeded") ;
  }
}

// (a)(vi) natural ordering of concerts
public int compareTo(Concert aConcert) {
  return (this.getConcertName().compareTo(aConcert.getConcertName())) ;
}
```

```java
  /**
   * A simple equals method
   */
```

```java
    public boolean equals(Object o) {
      Concert c = (Concert) o;

      return this.getDate().equals(c.getDate()) && this.getConcertName().equals(c.getConcertName));
    }

    /**
     * return a hash code for this object based on its date and name
     */
    public int hashCode() {
      return new Integer(this.getDate()).hashCode()*101 + this.getConcertName().hashCode();
    }
  }
```

```java
  /**
   * This class models a concert hall that hosts concerts of music
   * Complete this class using the instructions in part (b)
   */
  class ConcertHall {
    // Add code for ConcertHall here

    // (b)(i) public instance variable whatsOn
    // SortedMap<composer,SortedSet<title>>
    public SortedMap<String,SortedSet<String>> whatsOn ;

    // (b)(ii) public constructor
    public ConcertHall() {
      whatsOn = new TreeMap<String,SortedSet<String>>() ;
    }
```

```java
    // (b)(iii) public instance method addConcert
    public void addConcert(Concert aConcert) {
      List<Music> aProgramme = aConcert.getProgramme() ;
      SortedSet<String> ts ;
      for (Music progItem : aProgramme) {
        if (this.whatsOn.containsKey(progItem.getComposer())) {
          ts = this.whatsOn.get(progItem.getComposer()) ;
        } else {
          ts = new TreeSet<String>() ;
          this.whatsOn.put(progItem.getComposer(),ts) ;
        }
        ts.add(progItem.getTitle()) ;
      }
    }

  }
```

- **Errors in the development of the answer**

(1) Wrong bracket in method calls) not (

(2) Forgot import java.util.*

(3) getConcertLength() got programme from wrong place

(4) Forgot implements clause

(5) Problem with compareTo()

(6) implements Comparable should have been implements Comparable<Concert>

(7) Did not expect @Override

Go to Q 9

ToC

4.19 M250 From 2021 Specimen 2021 Exam Q 10

- Thinking about the scenario of the concert hall in the previous question:

(a) Select **two** reasons why it is preferable to declare the `whatsOn` collection using an interface type, such as a `Map`, rather than a concrete class such as `HashMap`, which implements that interface.

(i) A HashMap is abstract while a Map is concrete.

(ii) A Map provides more opportunity for reuse, due to substitutability of subtypes.

(iii) Using a Map allows us to change the implementation type more easily later on.

(iv) A Map supports multiple inheritance while a HashMap does not.

(v) A Map is more efficient than a HashMap.

(b) Select **two** reasons why a set is appropriate for the values in the `whatsOn` map, while a *list* was chosen for the `programme` in the `Concert` class:

(i) Titles of music by a composer are unique, so a set is appropriate for storing them.

(ii) A set is more efficient for storing music titles associated with a composer than a list.

(iii) A set maintains the order of items added to it so is best for a music programme.

(iv) A concert hall has a set of music, so composition with sets and lists is appropriate.

(v) A programme of music has a particular playing order, so a list is appropriate for the programme.

Go to Soln 10

4.20 M250 From 2021 Specimen 2021 Exam Soln 10

(a) Select **two** reasons why it is preferable to declare the `whatsOn` collection using an interface type, such as a `Map`, rather than a concrete class such as `HashMap`, which implements that interface.

(i) A HashMap is abstract while a Map is concrete.

(ii) A Map provides more opportunity for reuse, due to substitutability of subtypes. yes

(iii) Using a Map allows us to change the implementation type more easily later on. yes

(iv) A Map supports multiple inheritance while a HashMap does not.

(v) A Map is more efficient than a HashMap.

(b) Select **two** reasons why a set is appropriate for the values in the `whatsOn` map, while a *list* was chosen for the `programme` in the `Concert` class:

(i) Titles of music by a composer are unique, so a set is appropriate for storing them. yes

(ii) A set is more efficient for storing music titles associated with a composer than a list.

(iii) A set maintains the order of items added to it so is best for a music programme.

(iv) A concert hall has a set of music, so composition with sets and lists is appropriate.

(v) A programme of music has a particular playing order, so a list is appropriate for the programme. yes

Go to Q 10

ToC

5 Spec 2021 Solns

ToC

6 Prsntn 2018J Qs

6.1 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

ToC

6.2 M250 2018J Exam Q 1

- **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`.

6.2.1 Q 1(a)

(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.
- 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

(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)**

(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)**

ToC

6.2.2 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
```

- 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

ToC

6.2.3 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)**

ToC

6.2.4 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)**

Total **(40 marks)**

Go to Soln 1

ToC

6.3 M250 2018J Exam Q 2

6.3.1 Q 2(a)

- **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)**

ToC

6.3.2 A 2(b)

(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)**

ToC

6.3.3 Q 2(c)

(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.

ToC

6.3.4 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.

ToC

6.3.5 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)**

Total **(30 marks)**

Go to Soln 2

ToC

6.4 M250 2018J Exam Q 3

- **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.

- 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),
- 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.

6.4.1 Q 3(a)

(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)**

(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)**

ToC

6.4.2 Q 3(b)

(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)**

ToC

6.4.3 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)**

ToC

6.4.4 Q 3(d)

(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)**

(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)**

ToC

6.4.5 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)**

Total **(30 marks)**

Go to Soln 3

ToC

7 Prsntn 2018J Solns

7.1 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

ToC

7.2 M250 2018J Exam Soln 1

7.2.1 Soln 1(a)

(a)(i) **Q 1**

```
public class Performer {
  private String equityName ;
  private double payRate ;
  private Branch branch ;
  public static double minPayRate ;

  public Performer(String aName) {
    super() ;
    equityName = aName ;
```

```
10        branch = null ;
11        payRate = Performer.minPayRate ;
12        /* or */
13        // payRate = minPayRate ;
14      }
```

```
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      }
```

(ii)

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

(iii)

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

ToC

7.2.2 Soln 1(b)

(b)

(i) messages are sent: lines 8,9,10,11,12

(ii) reference variables are declared: lines 1,3,6,7

(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

ToC

7.2.3 Soln 1(c)

(c)

```
@Override
public boolean equals(Object obj) {
  Performer pfmr = (Performer) obj ;
  return equityName.equals(pfmr.equityName) ;
}
```

- This version assumes that the object is of type `Performer`
- See below for a more robust version

(c) Alternative, more robust version

```
@Override
public boolean equals(Object obj) {
  if (obj == this) {
    return true ;
  }
  if (!(obj instanceof Performer)) {
    return false ;
  }
  Performer pfmr = (Performer) obj ;
  return equityName.equals(pfmr.equityName) ;
}
```

- It is recommended to override `hashcode()` if you are overriding `equals()`

ToC

7.2.4 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

ToC

7.3 M250 2018J Exam Soln 2

7.3.1 Soln 2(a)

(a) **Q 2**

```
public interface Drivable {
  void accelerate() ;
  void brake() ;
  void stop() ;
}
```

- The method description modifiers of `abstract` and `public` are implicit

ToC

7.3.2 Soln 2(b)

(b)

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

  public Vehicle() {
    super() ;
    speed = 0 ;
  }

  public void accelerate() {
    speed = speed + 1 ;
  }
// } // continued below
```

```
  public void brake() {
    if (speed > 0) {
      speed = speed - 1 ;
    }
  }

  public void stop() {
    while (speed > 0) {
      speed = speed - 1 ;
    }
  }

  public int getSpeed() {// required later
    return speed ;
  }

  public void setSpeed(int spd) {// required later
    speed = spd ;
  }
}
```

ToC

7.3.3 Soln 2(c)

(c)

```
public class Car extends Vehicle {
  private int maxSpeed ;
  private int increment ;
  public static int count = 0 ; // Q2(c)(iii)

  public Car() {
    super() ;
    Car.count = Car.count + 1 ; // Q2(c)(iii)
  }

  @Override
  public void accelerate() {
    if ((getSpeed() + increment) <= maxSpeed) {
      super.setSpeed(super.getSpeed() + increment) ;
    }
  }
}
```

(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

ToC

7.3.4 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

ToC

7.3.5 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

ToC

7.3.6 Soln 2(f)

(f)

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

Go to Q 2

ToC

7.4 M250 2018J Exam Soln 3

7.4.1 Soln 3(a)

(a) **Q 3**

```java
public class CaravanSite {
  // provided
  private String siteName ;
  private int maxVans ;

  private List<Booking> bookings ;

  public CaravanSite(String aName, int aMaxVans) {
    super() ;
    siteName = aName ;
    maxVans = aMaxVans ;
    bookings = new ArrayList<>() ;
  }
// } // continued below
```

```java
  public int addBooking(Booking aBooking) {
    if (bookings.size() < maxVans) {
      bookings.add(aBooking) ;
    }
    else {
      System.out.println("No_space") ;
    }
    return maxVans - bookings.size() ;
  }
```

ToC

7.4.2 Soln 3(b)

(b) **Q 3**

```
public class Booking implements Comparable<Booking> {
  // provided
  private String makeAndModel ;
  private String owner ;
  private int estArrivalHour ;

  public int compareTo(Booking aBooking) {
    return estArrivalHour - aBooking.estArrivalHour ;
    /* or */
    // Integer.compare(estArrivalHour,
    //                 aBooking.estArrivalHour)
  }
}
```

(b) **Q 3** provided parts

```
  public Booking(String aMandM
                 ,String anOwner
                 ,int anHour) {
    super() ;
    makeAndModel = aMandM ;
    owner = anOwner ;
    estArrivalHour = anHour ;
  }
```

(b) **Q 3** provided parts

```
  public String getMakeAndModel() {
    return makeAndModel ;
  }

  public String getOwner() {
    return owner ;
  }

  public int getEstArrivalHour() {
    return estArrivalHour ;
  }
```

(b) **Q 3** provided parts

```
  @Override
  public boolean equals(Object obj) {
    Booking bkg = (Booking) obj ;
    return makeAndModel.equals(bkg.makeAndModel)
        && owner.equals(bkg.owner) ;
  }

  @Override
  public int hashCode() {
    return (20 + makeAndModel.hashCode())
         * owner.hashCode() ;
  }
```

ToC

7.4.3 Soln 3(c)

(c) **Q 3**

```
  public void orderBookings() {
    Collections.sort(bookings) ;
  }
```

```
  public List<Booking> getBookings() {
    return bookings ;
  }
```

ToC

7.4.4 Soln 3(d)

(d) **Q 3**

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

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

  public void addSite(CaravanSite aSite) {
    for (Booking aBooking : aSite.getBookings()) {
      Integer hour = aBooking.getEstArrivalHour() ;
      if (!(arrByTime.containsKey(hour))) {
        arrByTime.put(hour, new HashSet<>()) ;
      }
      arrByTime.get(hour).add(aBooking) ;
    }
  }
}
```

ToC

7.4.5 Soln 3(e)

(e) **Q 3**

(i) The interface is the real type of the variable, parameter, method of 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 Bloch (2017, Item 64, page 280)

(ii) `ArrayList` is expandable unlike `Array` — it implements the `List` interface which has different fields and methods to `Array`

See Bloch (2017, Item 28, page 126)

Go to Q 3

ToC

8 What Next ?

Programming, Debugging, Psychology

Although programming techniques have improved immensely since the early days, the process of finding and correcting errors in programming — known graphically if inelegantly as *debugging* — still remains a most difficult, confused and unsatisfactory operation. The chief impact of this state of affairs is psychological. Although we are happy to pay lip-service to the adage that to err is human, most of us like to make a small private reservation about our own performance on special occasions when we really try. It

is somewhat deflating to be shown publicly and incontrovertibly by a machine that even when we do try, we in fact make just as many mistakes as other people. If your pride cannot recover from this blow, you will never make a programmer.

Christopher Strachey, Scientific American 1966 vol 215 (3) September pp112–124

- To err is human, to really foul things up requires a computer.
- Attributed to Paul R. Ehrlich in 101 Great Programming Quotes
- Attributed to Bill Vaughn in Quote Investigator
- Derived from Alexander Pope (1711, An Essay on Criticism)
- *To Err is Humane; to Forgive, Divine*
- This also contains

  A little learning is a dangerous thing;

  Drink deep, or taste not the Pierian Spring
- In programming, this means you have to *read the fabulous manual* (RTFM)

Units 1–5, TMA01

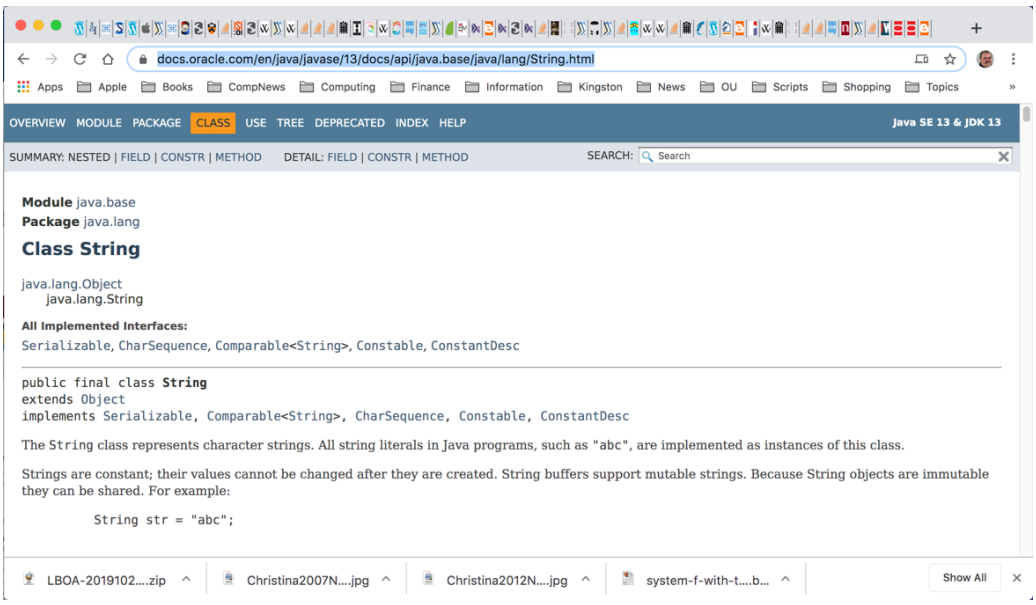
- Tutorial: Exam revision: Online 10:00 Sunday 10 May 2026
- Exam Wednesday, 3 June 2026 Remote exam

ToC

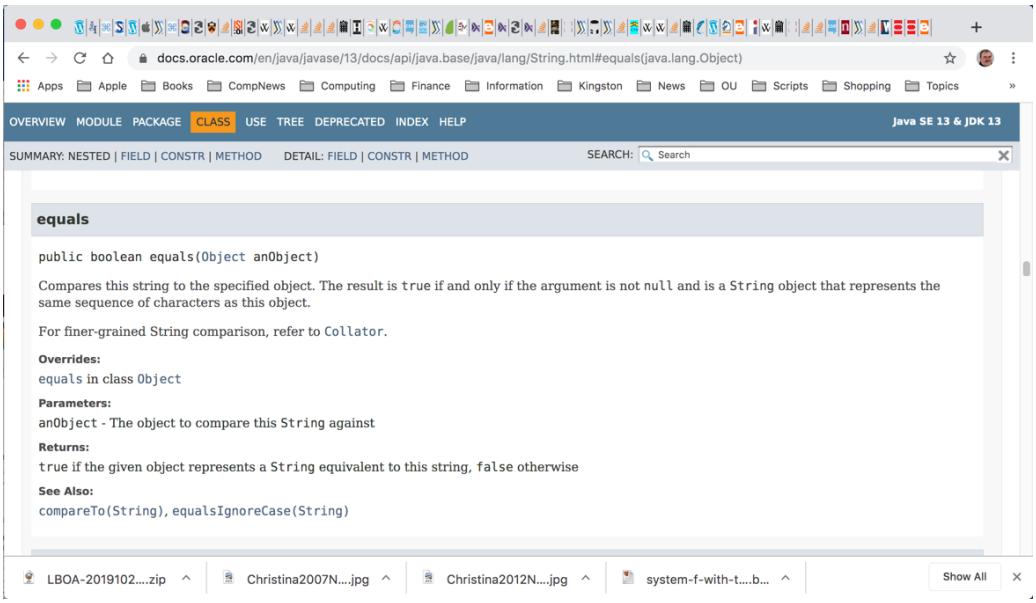
9 Web Links & References

9.1 Java Documentation

- Java Documentation — BlueJ has JDK 7 embedded, JDK 13 is current (2019)
- JDK 13 Documentation
- Java Platform API Specification
- Java Language Specification
- JDK Documentation > API Documentation > java.base
  - java.lang — fundamental classes for the Java programming language
  - java.util — Collections framework



- `Strings` are *immutable* objects
- See java.lang.StringBuilder for *mutable* strings
- In a *functional programming approach* everything is immutable — it makes life simpler (but at a cost)



- Remember `(==)` tests for *identity* — what does this mean ?

ToC

9.2 Books Phil Likes

- M250 is self contained — you do not need further books — but you might like to know about some:
- Sestoft (2016) — the best short reference
- Evans and Flanagan (2018) — the best longer reference
- Barnes and Kölling (2016) — the BlueJ book — see www.bluej.org for documentation and tutorial

- Bloch (2017) — guide to best practice

ToC

References

Barnes, David J. and Michael Kolling (2009). *Objects First with Java*. Pearson Education, fourth edition. ISBN 0-13-606086-2. URL http://www.bluej.org/objects-first/.

Barnes, David J. and Michael Kolling (2011). *Objects First with Java*. Pearson Education, fifth edition. ISBN 0132835541. URL http://www.bluej.org/objects-first/.

Barnes, David J. and Michael Kölling (2016). *Objects First with Java*. Pearson, sixth edition. ISBN 1292159049. URL http://www.bluej.org/objects-first/. 40

Bloch, Joshua (2017). *Effective Java*. Addison-Wesley Professional, third edition. ISBN 9780134685991. 38, 41

Darwin, Ian F (2014). *Java Cookbook*. O'Reilly, third edition. ISBN 9781449337049.

Evans, Benjamin J and David Flanagan (2014). *Java In A Nutshell*. O'Reilly, sixth edition. ISBN 1449370829. URL https://github.com/kittylyst/javanut6-examples.

Evans, Benjamin J and David Flanagan (2018). *Java In A Nutshell*. O'Reilly, seventh edition. ISBN 1492037257. 40

Felleisen, Matthias and Daniel P. Friedman (1998). *A Little Java, A Few Patterns*. MIT Press. ISBN 0262561158. URL http://felleisen.org/matthias/BALJ-index.html.

Gosling, James; Bill Joy; Guy L. Steele Jr.; Gilad Bracha; and Alex Buckley (2014). *The Java Language Specification, Java SE 8 Edition (Java Series) (Java (Addison-Wesley))*. Addison Wesley, eighth edition. ISBN 013390069X. URL https://docs.oracle.com/en/java/javase/12/index.html.

Naftalin, Maurice and Philip Wadler (2006). *Java Generics and Collections*. O'Reilly. ISBN 059610247X.

Schildt, Herbert (2018a). *Java: A Beginner's Guide*. McGraw-Hill, eighth edition. ISBN 1260440214. URL http://mhprofessional.com/9781260440218-usa-java-a-beginners-guide-eighth-edition-group.

Schildt, Herbert (2018b). *Java: The Complete Reference, Eleventh Edition*. McGraw-Hill, eleventh edition. ISBN 1260440230. URL http://mhprofessional.com/9781260440232-usa-java-the-complete-reference-eleventh-edition-group.

Sestoft, Peter (2002). *Java Precisely*. MIT Press. ISBN 0-262-69276-7.

Sestoft, Peter (2005). *Java Precisely*. MIT, second edition. ISBN 0262693259.

Sestoft, Peter (2016). *Java Precisely*. MIT, third edition. ISBN 0262529076. URL http://www.itu.dk/people/sestoft/javaprecisely/. 40

Thimbleby, Harold (1999). A critique of Java. *Software: Practice and Experience*, 29(5):457–478.

Waldo, Jim (2010). *Java: The Good Parts*. O'Reilly. ISBN 9780596803735. URL http://shop.oreilly.com/product/9780596803742.do.

section WebLinksReferences (end)

Author Phil Molyneux Written 10 May 2026 *Printed 10th May 2026*
Subject dir: ⟨*baseURL*⟩/OU/Courses/Computing/M250/M250Presentations/M250Prsntn2025J
Topic path:
/M250Prsntn2025JTutorials/M250Tutorial20260510ExamRevPrsntn2025J/M250Tutorial20260510ExamRevPrsntn2025J.pdf