Java: Composition, Inheritance, Interfaces

M250 25J Tutorial 05

Phil Molyneux

15 February 2026

Java: Composition, Inheritance, Interfaces
Phil Molyneux
Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

M250 Java: Composition, Inheritance, Interfaces

Tutorial Agenda

- Introductions
- Adobe Connect reminders
- *Adobe Connect* — if you or I get cut off, wait till we reconnect (or send you an email)
- Composition
- Inheritance, subclasses, superclasses
- Interfaces
- String Formatting
- JShell (optional)
- Some useful Web & other references
- Time: about 1 to 2 hours
- Do ask questions or raise points.
- Slides/Notes M250Tutorial20240218InheritanceInterfacesPrsntn2023J/

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

Tutorial

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

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

Tutorial

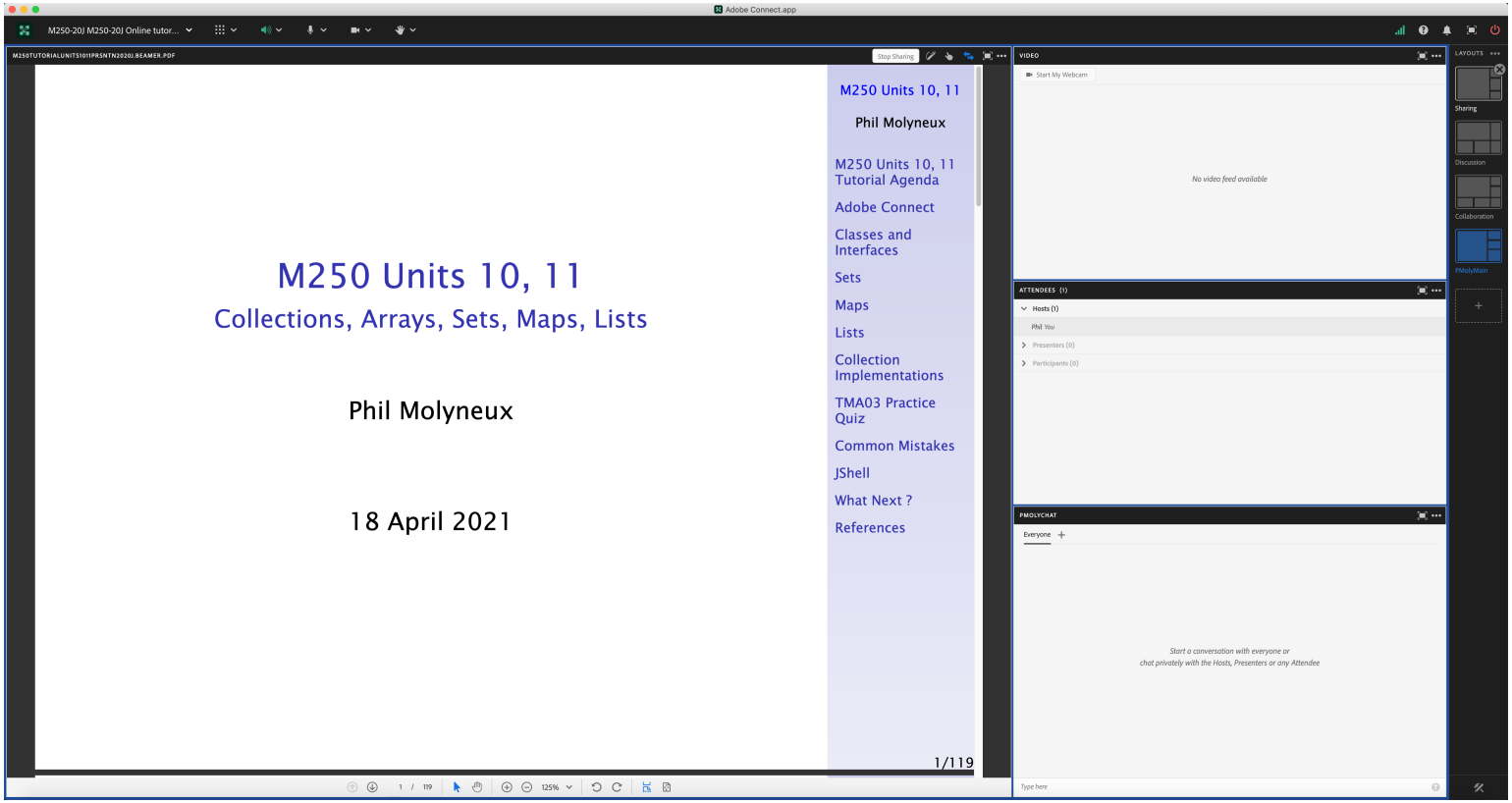
Introductions — You

- *Name ?*
- *Favourite software/Programming language ?*
- *Favourite text editor or integrated development environment (IDE)*
- List of text editors, Comparison of text editors and Comparison of integrated development environments
- *Other OU courses ?*
- *Anything else ?*

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

Adobe Connect

Interface — Host View



Tutorial Agenda

Adobe Connect

- Interface
- Settings
- Sharing Screen & Applications
- Ending a Meeting
- Invite Attendees
- Layouts
- Chat Pods
- Web Graphics
- Recordings

Composition

Inheritance

Interfaces

String Formatting

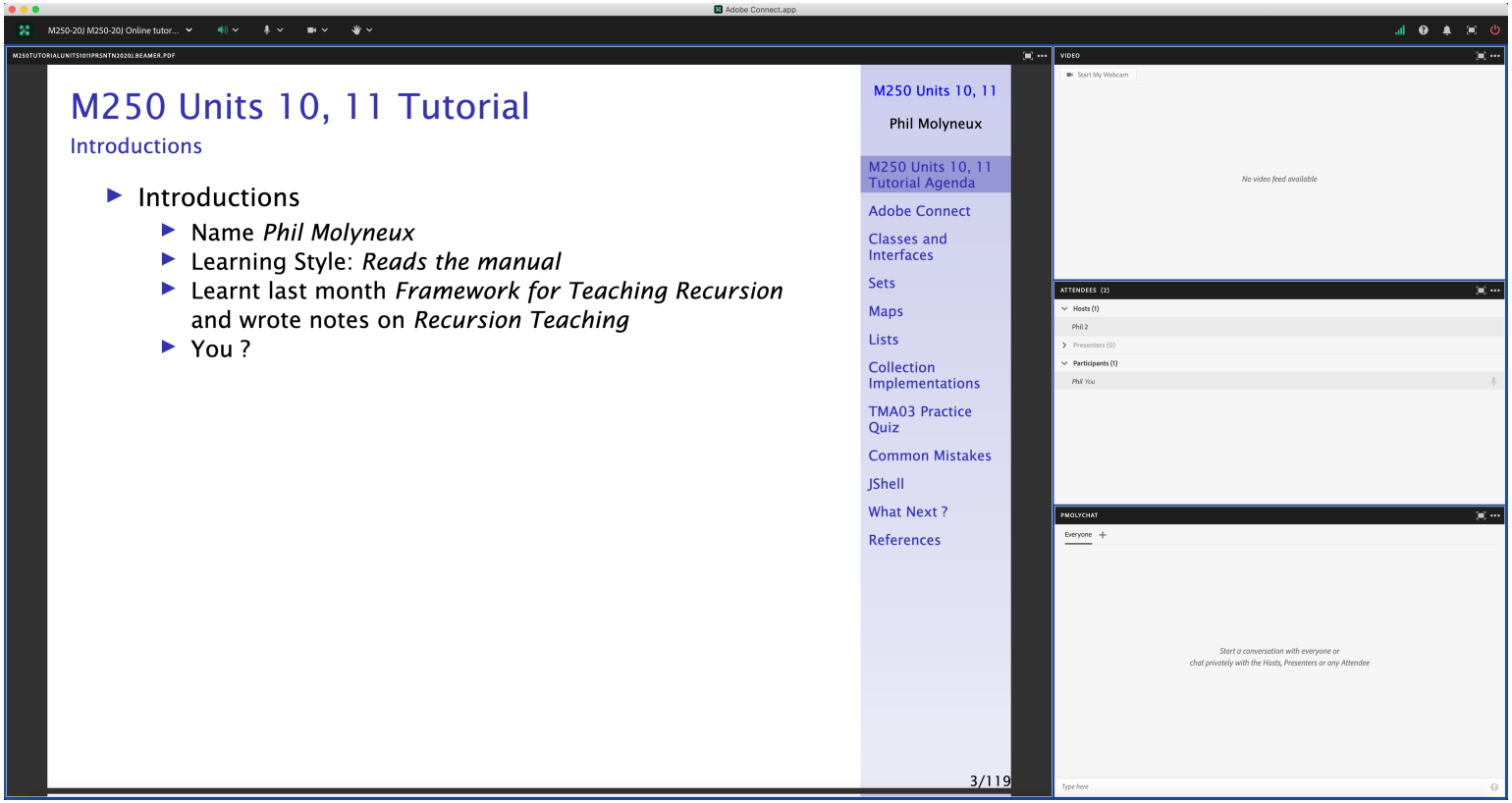
JShell

What Next ?

References

Adobe Connect

Interface — Participant View



Tutorial Agenda

Adobe Connect

Interface

Settings

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Web Graphics

Recordings

Composition

Inheritance

Interfaces

String Formatting

JShell

What Next ?

References

Adobe Connect

Settings

- **Everybody** *Menu bar* > *Meeting* > Speaker & Microphone Setup
- *Menu bar* > *Microphone* > Allow Participants to Use Microphone ✔
- Check Participants see the entire slide **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*

Tutorial Agenda
Adobe Connect
Interface
Settings
Sharing Screen & Applications
Ending a Meeting
Invite Attendees
Layouts
Chat Pods
Web Graphics
Recordings
Composition
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

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

Tutorial Agenda
Adobe Connect
Interface
Settings
Sharing Screen & Applications
Ending a Meeting
Invite Attendees
Layouts
Chat Pods
Web Graphics
Recordings
Composition
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

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** ⌘ + \

Tutorial Agenda
Adobe Connect
Interface
Settings
Sharing Screen & Applications
Ending a Meeting
Invite Attendees
Layouts
Chat Pods
Web Graphics
Recordings
Composition
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

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 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

Tutorial Agenda
Adobe Connect
Interface
Settings
Sharing Screen & Applications
Ending a Meeting
Invite Attendees
Layouts
Chat Pods
Web Graphics
Recordings
Composition
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

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

Tutorial Agenda
Adobe Connect
Interface
Settings
Sharing Screen & Applications
Ending a Meeting
Invite Attendees
Layouts
Chat Pods
Web Graphics
Recordings
Composition
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

Adobe Connect

Invite Attendees

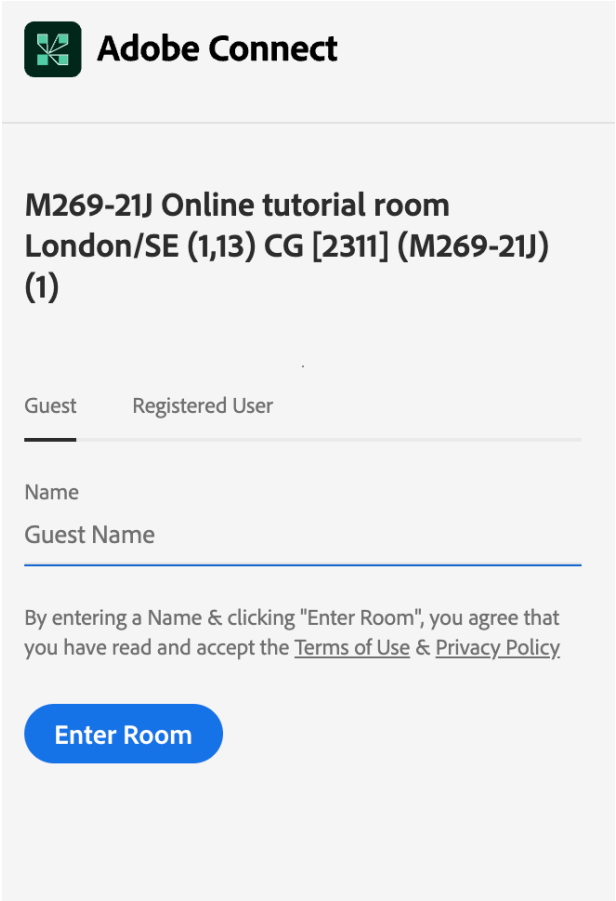
- **Provide Meeting URL** Menu > Meeting > Manage Access & Entry > Invite Participants...
- **Allow Access without Dialog** Menu > Meeting > Manage Meeting Information provides new browser window with *Meeting Information* Tab bar > Edit Information
- Check *Anyone who has the URL for the meeting can 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

Tutorial Agenda
Adobe Connect
Interface
Settings
Sharing Screen & Applications
Ending a Meeting
Invite Attendees
Layouts
Chat Pods
Web Graphics
Recordings
Composition
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

Adobe Connect

Entering a Room as a Guest (1)

- 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



Tutorial Agenda

Adobe Connect

- Interface
- Settings
- Sharing Screen & Applications
- Ending a Meeting
- Invite Attendees
- Layouts
- Chat Pods
- Web Graphics
- Recordings

Composition

Inheritance

Interfaces

String Formatting

JShell

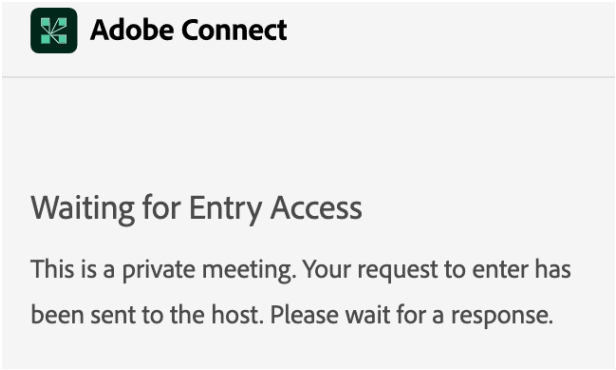
What Next ?

References

Adobe Connect

Entering a Room as a Guest (2)

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

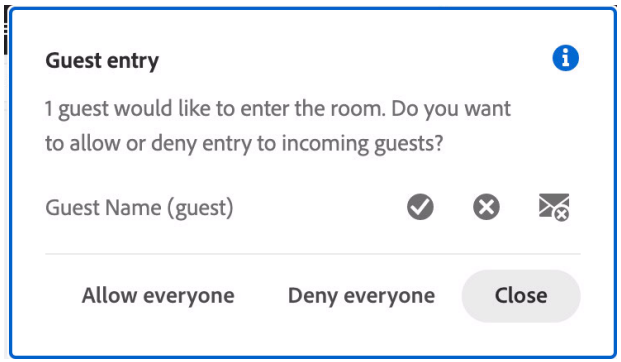


Tutorial Agenda
Adobe Connect
Interface
Settings
Sharing Screen & Applications
Ending a Meeting
Invite Attendees
Layouts
Chat Pods
Web Graphics
Recordings
Composition
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

Adobe Connect

Entering a Room as a Guest (3)

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



Tutorial Agenda
Adobe Connect
Interface
Settings
Sharing Screen & Applications
Ending a Meeting
Invite Attendees
Layouts
Chat Pods
Web Graphics
Recordings
Composition
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

Adobe Connect

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

Java: Composition, Inheritance, Interfaces

Phil Molyneux

Tutorial Agenda

Adobe Connect
- Interface
- Settings
- Sharing Screen & Applications
- Ending a Meeting
- Invite Attendees
- Layouts
- Chat Pods
- Web Graphics
- Recordings

Composition

Inheritance

Interfaces

String Formatting

JShell

What Next ?

References

Adobe Connect

Layouts

- 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

Tutorial Agenda
Adobe Connect
Interface
Settings
Sharing Screen & Applications
Ending a Meeting
Invite Attendees
Layouts
Chat Pods
Web Graphics
Recordings
Composition
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

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

Tutorial Agenda
Adobe Connect
Interface
Settings
Sharing Screen & Applications
Ending a Meeting
Invite Attendees
Layouts
Chat Pods
Web Graphics
Recordings
Composition
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

Graphics Conversion

PDF to PNG/JPG

- 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 ⌘ + ⏎

Tutorial Agenda
Adobe Connect
Interface
Settings
Sharing Screen & Applications
Ending a Meeting
Invite Attendees
Layouts
Chat Pods
Web Graphics
Recordings
Composition
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

Adobe Connect Recordings

Exporting 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

Java: Composition, Inheritance, Interfaces

Phil Molyneux

Tutorial Agenda

Adobe Connect

Interface

Settings

Sharing Screen & Applications

Ending a Meeting

Invite Attendees

Layouts

Chat Pods

Web Graphics

Recordings

Composition

Inheritance

Interfaces

String Formatting

JShell

What Next ?

References

Composition

TMA02 Practice Quiz

- These questions below are similar to TMA02
- Create the complete solution in BlueJ first and ensure it compiles
- This exercise uses CodeRunner to check your answers — it has to compile for CodeRunner to work
- Note that if asked to produce a string it must be exactly as given
- The quiz can be attempted any number of times without penalty
- The exercise models a `Car` class that is composite since is has an `Engine` component

TMA02 Practice Quiz

Initial Code

- Open BlueJ and create a new Project
- Project > New Project...
- There may be a problem navigating folders — in that case use the text box
- Create new classes Edit > New Class... Engine and Car
- The javadoc comments are omitted for space here

```
1  public class Engine {

53 }

55 public class Car {

95 }
```

Engine Class

Instance Variables and Constructor

- The `Engine` class requires a single `int` instance variable `revs` that represents the revolutions (revs) per minute of the engine. It also needs three `int` constants:
- `MAXREVS` which is set to 6000 (above that, the engine will blow up),
- `MINREVS` which is set to 1000 (below that, the engine will stall) and
- `REVSINC` which is set to 1000 (which, for the purpose of this question, is the amount by which `revs` can be increased or decreased).
- Declare the variables and write a public, zero-argument constructor for `Engine` that sets `revs` to 0.

Tutorial Agenda
Adobe Connect
Composition
TMA02 Practice Quiz
Engine Class
Car Class
Example Problems
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

Engine Class

Instance Variables and Constructor

```
2   private int revs ;
3   private final int MAXREVS = 6000 ;
4   private final int MINREVS = 1000 ;
5   private final int REVSINC = 1000 ;

7   public Engine() {
8     this.revs = 0 ;
9   }
```

- Note that constants are covered in Unit 7 Section 4.4 page 119
- **final** objects cannot be changed
- **immutable** contents of the object cannot be changed — so final and immutable are not the same thing
- See Stack Overflow: Immutable and Final in Java
- See Stack Overflow: Properties of Immutable Objects

Java: Composition, Inheritance, Interfaces

Phil Molyneux

Tutorial Agenda

Adobe Connect

Composition

TMA02 Practice Quiz

Engine Class

Car Class

Example Problems

Inheritance

Interfaces

String Formatting

JShell

What Next ?

References

Engine Class

Getter Method(s)

- The `Engine` class requires a getter method `getRevs()` for `revs` which returns the value of `revs`.
- Write `getRevs()`.

```
11  public int getRevs() {
12    return this.revs ;
13  }
```


Tutorial Agenda
Adobe Connect
Composition
TMA02 Practice Quiz
Engine Class
Car Class
Example Problems
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

Engine Class

isRunning() Method

- The `Engine` class requires a method `isRunning()` which returns `true` if `revs` is positive (ie the engine is running) or `false` if not.
- Write `isRunning()`.

```
15  public boolean isRunning() {
16    return this.revs > 0 ;
17  }
```


Tutorial Agenda
Adobe Connect
Composition
TMA02 Practice Quiz
Engine Class
Car Class
Example Problems
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

Engine Class

startEngine() Method

- The `Engine` class requires a method `startEngine()` that returns a `boolean`. If the engine is already running, then this method returns `false`. If the engine is not already running and the value of `revs` is `0` then `revs` is set to the minimum rev value and the method returns `true`, otherwise the method returns `false`.
- Write `startEngine()`.

```
19  public boolean startEngine() {
20    if (this.isRunning()) {
21      return false ;
22    } else {
23      this.revs = MINREVS ;
24      return true ;
25    }
26  }
```

Engine Class

incRevs() Method

- The `Engine` class requires a method `incRevs()` that returns a `boolean`. If it is possible to increase `revs` by `REVSINC` without exceeding `MAXREVS` than that is done and `true` returned. If not, then `revs` is set to `0` and `false` returned (modelling the engine blowing up).
- Write `incRevs()`.

```
28  public boolean incRevs() {
29    if (this.getRevs() + REVSINC <= MAXREVS) {
30      this.revs = this.getRevs() + REVSINC ;
31      return true ;
32    } else {
33      this.revs = 0 ;
34      return false ;
35    }
36  }
```

Java: Composition, Inheritance, Interfaces
Phil Molyneux
Tutorial Agenda
Adobe Connect
Composition
TMA02 Practice Quiz
Engine Class
Car Class
Example Problems
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

Engine Class

decRevs() Method

- The `Engine` class requires a method `decRevs()` that returns a `boolean`. If it is possible to decrease `revs` by `REVSINC` without going below `MINREVS` than that is done and `true` returned. If not, then `revs` is set to `0` and `false` returned (modelling the engine stalling).
- Write `decRevs()`.

```
38  public boolean decRevs() {
39    if (this.getRevs() - REVSINC >= MINREVS) {
40      this.revs = this.getRevs() - REVSINC ;
41      return true ;
42    } else {
43      this.revs = 0 ;
44      return false ;
45    }
46  }
```

Engine Class

reduceToIdling() Method

- The Engine class requires a method reduceToIdling() that reduces revs from whatever value it is now down to MINREVS, repeatedly reducing it by REVSINC, (modelling reducing the revs until the engine is idling). So for example if revs was 4000, then reduceToIdling() would change revs to 3000, then 2000, then 1000, and then not reduce it any more.
- Write reduceToIdling(). The method does not return a value.

```
48  public void reduceToIdling() {
49    while (this.getRevs() - REVSINC >= MINREVS) {
50      this.decRevs() ;
51    }
52  }
```

Java: Composition, Inheritance, Interfaces
Phil Molyneux
Tutorial Agenda
Adobe Connect
Composition
TMA02 Practice Quiz
Engine Class
Car Class
Example Problems
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

Car Class

Instance Variable(s) and Constructor(s)

- The Car class requires a single instance variable eng of type Engine.
- This is where composition occurs because a Car *has-an* Engine.
- Write a public constructor for Car that takes a single argument of type Engine and sets eng to that argument.
- Declare the instance variable and write the constructor.

```
56  private Engine eng ;

58  public Car(Engine eng) {
59    this.eng = eng ;
60  }
```


Tutorial Agenda
Adobe Connect
Composition
TMA02 Practice Quiz
Engine Class
Car Class
Example Problems
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

Car Class

start() Method

- The Car class requires a method start(). This method sends a message to eng to tell it to start, and if unsuccessful, reports this as a String output "Engine is already running".
- There is no return value.

```
62  public void start() {
63    if (!(this.eng.startEngine())) {
64      System.out.println("Engine␣is␣already␣running") ;
65    }
66  }
```


Tutorial Agenda
Adobe Connect
Composition
TMA02 Practice Quiz
Engine Class
Car Class
Example Problems
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

Car Class

getRevs() Method

- The `Car` class requires a method `getRevs()` that returns the `int` value of `revs` (from `eng`).
- This value has to be obtained by sending a message to `eng` (because an instance of class `Car` doesn't know the value itself).
- Write `getRevs()`.

```
68  public int getRevs() {
69    return this.eng.getRevs() ;
70  }
```


Tutorial Agenda
Adobe Connect
Composition
TMA02 Practice Quiz
Engine Class
Car Class
Example Problems
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

Car Class

accelerate() Method

- The `Car` class requires a method `accelerate()`. This method first checks that the engine is running (sends a message to `eng`). If it is not running then this is reported as a `String` output `"You've not started the engine yet"`). If it is running then it attempts to increase the revs (by sending another message to `eng`). If this increase is unsuccessful then this is reported as a `String` output `"You blew up the engine!"`.
- Write `accelerate()`. The method does not return a value.

```
72  public void accelerate() {
73    if (!(this.eng.isRunning())) {
74      System.out.println("You've not started the engine yet") ;
75    } else if (!(this.eng.incRevs())) {
76      System.out.println("You blew up the engine!") ;
77    }
78  }
```

Car Class

decelerate() Method

- The `Car` class requires a method `decelerate()`. This method first checks that the engine is running (sends a message to `eng`). If it is not running then this is reported as a `String` output `"You've not started the engine yet"`). If it is running then it attempts to decrease the revs (by sending another message to `eng`). If this decrease is unsuccessful then this is reported as a `String` output `"Stalled"`.
- Write `decelerate()`. The method does not return a value.

```
80  public void decelerate() {
81    if (!(this.eng.isRunning())) {
82      System.out.println("You've not started the engine yet") ;
83    } else if (!(this.eng.decRevs())) {
84      System.out.println("Stalled") ;
85    }
86  }
```


Java: Composition, Inheritance, Interfaces

Phil Molyneux

Tutorial Agenda

Adobe Connect

Composition

TMA02 Practice Quiz

Engine Class

Car Class

Example Problems

Inheritance

Interfaces

String Formatting

JShell

What Next ?

References

Car Class

stop() Method

- The Car class requires a method stop(). This method first checks that the engine is running (sends a message to eng). If it is not running then this is reported as a String output "You've not started the engine yet"). If it is running then it reduces the engine revs gradually to the idling revs — sends another message to eng to do this.
- Write stop(). The method does not return a value.

```
88  public void stop() {
89    if (!(this.eng.isRunning())) {
90      System.out.println("You've not started the engine yet") ;
91    } else {
92      this.eng.reduceToIdling() ;
93    }
94  }
```

Java: Composition, Inheritance, Interfaces
Phil Molyneux
Tutorial Agenda
Adobe Connect
Composition
TMA02 Practice Quiz
Engine Class
Car Class
Example Problems
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

TMA02 Practice Quiz

Example Problems (1)

- Initially for the constants I had

```
final int MAXREVS = 6000 ;
final int MINREVS = 1000 ;
final int REVSINC = 1000 ;
```

- For each field, I had the *Specification* (not *Compilation*) error:
- The field XXX is missing modifier `private`

TMA02 Practice Quiz

Example Problems (2)

- Reading Unit 7 Section 4.4, I thought the constants should be class (static) fields, so tried:

```
private static final int MAXREVS = 6000 ;
private static final int MINREVS = 1000 ;
private static final int REVSINC = 1000 ;
```

- For each field, I had the *Specification* (not *Compilation*) error:
- The field XXX should not have modifier `static`

TMA02 Practice Quiz

Example Problems (3a)

- My next version compiled and met the specification but generated a large number of error messages
- The first error test had

```
// This tests startEngine() returns true when revs is 0
Engine e = new Engine();
System.out.println(e.getRevs());
System.out.println(e.startEngine());
```

- This should have reported `0`, `true` but reported `0`, `false` (scroll right in the window to see **Expected** and **Got**)
- Decided to try the file in JShell (see below)

Java: Composition, Inheritance, Interfaces

Phil Molyneux

Tutorial Agenda

Adobe Connect

Composition

TMA02 Practice Quiz

Engine Class

Car Class

Example Problems

Inheritance

Interfaces

String Formatting

JShell

What Next ?

References

TMA02 Practice Quiz

Example Problems (3b)

- Using JShell with the file containing the two classes but generated a large number of error messages

```
jshell> /open M250TMA02PracticeQuiz.java

jshell> /list
```

```
jshell> Engine en = new Engine()
en ==> Engine@68de145

jshell> System.out.println(en.getRevs())
0

jshell> System.out.println(en.startEngine())
false

jshell> System.out.println(en.getRevs())
0
```

- Looks like `startEngine()` is a problem

Tutorial Agenda
Adobe Connect
Composition
TMA02 Practice Quiz
Engine Class
Car Class
Example Problems
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

TMA02 Practice Quiz

Example Problems (3c)

- What is wrong with the following definition pf `startEngine()`?

```
19  public boolean startEngine() {
20    if (!(this.isRunning())) {
21      return false ;
22    } else {
23      this.revs = MINREVS ;
24      return true ;
25    }
26  }
```

- Other error messages involved `startEngine()` so fix this first and see what happens

TMA02 Practice Quiz

Example Problems (3d)

- `if` condition round the wrong way — a common error

```
19  public boolean startEngine() {
20    if (this.isRunning()) {
21      return false ;
22    } else {
23      this.revs = MINREVS ;
24      return true ;
25    }
26  }
```

- Re-compiled and passed all the tests

Classes

Overview and Structure

- A **class** represents a concept, a template for creating instances (objects)
- An **object** is an instance of a concept (a class)
- A classDeclaration of `class C` has the form

```
classModifiers class C extendsClause implementsClause
  classBody
```

- extendsClause and implementsClause refer to superclasses and interface (see later in M250)
- For a top-level class classModifiers may be a list of `public` and at most one of `abstract` or `final`

Java: Composition, Inheritance, Interfaces
Phil Molyneux
Tutorial Agenda
Adobe Connect
Composition
Inheritance
Classes
Class Hierarchy
Interfaces
String Formatting
JShell
What Next ?
References

Classes

Overview and Structure (2)

- The classBody contains declarations of fields, constructors, methods, nested classes, nested interfaces, and initialiser blocks (M250 mainly uses the first three)
- The declarations *may* appear in any order but you should use the order suggested in *M250 Code Conventions*

```
{
  fieldDeclarations
    /* class (static) variables */
    /* instance variables */
  constructorDeclarations
  methodDeclarations
}
```

- A source file may begin with package (not used in M250) and import declarations (to be covered later)

Classes

Example Declaration

```
class Point {
  int x, y ;

  Point(int x, int y) {
    this.x = x ;
    this.y = y ;
  }

  void move(int dx, int dy) {
    x = x + dx ;
    y = y + dy ;
  }

  public String toString() {
    return ("(" + x + "," + y + ")") ;
  }
}
```


Tutorial Agenda
Adobe Connect
Composition
Inheritance
Classes
Class Hierarchy
Interfaces
String Formatting
JShell
What Next ?
References

Encapsulation

Data Hiding

	Member Visibility			
Accessible to	**Public**	**Protected**	**Default**	**Private**
Defining class	Yes	Yes	Yes	Yes
Class in same package	Yes	Yes	Yes	No
Subclass in different package	Yes	Yes	No	No
Nonsubclass different package	Yes	No	No	No

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Classes
Class Hierarchy
Interfaces
String Formatting
JShell
What Next ?
References

Class Methods and Class Variables

Class Methods and Hiding

	Superclass instance method	**Superclass class method**
Subclass instance method	Overrides super	Compile fail
Subclass class method	Compile fail	Hides sub

Class Methods and Class Variables

Overriding and Hiding (1)

```
1  class Foo {
2      public static void method() {
3          System.out.println("in Foo");
4      }
5  }

7  class Bar extends Foo {
8      public static void method() {
9          System.out.println("in Bar");
10     }
11 }
```

- Example of a static (class) method *hiding* another static method
- Example from CodeRanch: Overriding vs Hiding
- What is the difference between *overriding* and *hiding*?

Class Methods and Class Variables

Overriding and Hiding (2)

```
17 /**
18  * Class Foo defines
19  * class method classMethod()
20  * instance method instanceMethod()
21  */
22 class Foo {
23   public static void classMethod() {
24     System.out.println("classMethod()␣in␣Foo");
25   }

27   public void instanceMethod() {
28     System.out.println("instanceMethod()␣in␣Foo");
29   }
30 }
```

- Now declare Bar as a subclass of Foo

Class Methods and Class Variables

Overriding and Hiding (3)

```
32 /**
33  * Class Bar is a subclass of Foo also defining
34  * class method classMethod()
35  * instance method instanceMethod()
36  */
37 class Bar extends Foo {
38   public static void classMethod() {
39     System.out.println("classMethod()␣in␣Bar");
40   }

42   public void instanceMethod() {
43     System.out.println("instanceMethod()␣in␣Bar");
44   }
45 }
```

- Now run a test job with these declarations

Class Methods and Class Variables

Overriding and Hiding (4)

```
9 class OverrideHideEG {
10   public static void main(String[] args) {
11     Foo f = new Bar();
12     f.instanceMethod();
13     f.classMethod();
14   }
15 }
```

- The output is:

```
instanceMethod() in Bar
classMethod() in Foo
```

- The instance method *overides* the instance method from Foo
- The class method of the instance is hidden since f is declared of type Foo

Compiling and Running a Java Program

Mistakes we have made

- The above code was in the file `OverrideHideEG.java`

```
<Java><139> ls
OverrideHideEG.java
<Java><140> javac OverrideHideEG
error: Class names, 'OverrideHideEG',
are only accepted
if annotation processing is explicitly requested
1 error
<Java><141> javac OverrideHideEG.java
<Java><142> java OverrideHideEG.java
error: can't find main(String[]) method in class: Foo
<Java><143> java OverrideHideEG
instanceMethod() in Bar
classMethod() in Foo
```

- Why didn't the error message say *File not found*?
- See StackOverflow: javac error

Compiling and Running a Java Program

Mistakes we have made (2)

- Using `OverrideHideEG.java` in JShell

```
jshell> /open OverrideHideEG.java

jshell> /list

   1 : class OverrideHideEG {
         public static void main(String[] args) {
           // stuff removed
         }
       }
       // further stuff removed

jshell> OverrideHideEG.main(new String[0])
instanceMethod() in Bar
classMethod() in Foo
```

- How to run a whole Java file added as a snippet in JShell?

Class Methods and Class Variables

Overriding and Hiding (5)

- The class method is *overloaded* and the choice of which method to invoke is made at compile time
- The correct version of an overriden method is chosen at run time based on the run time type of the object on which the method is invoked
- See Bloch (2017, page 239, Item 52)

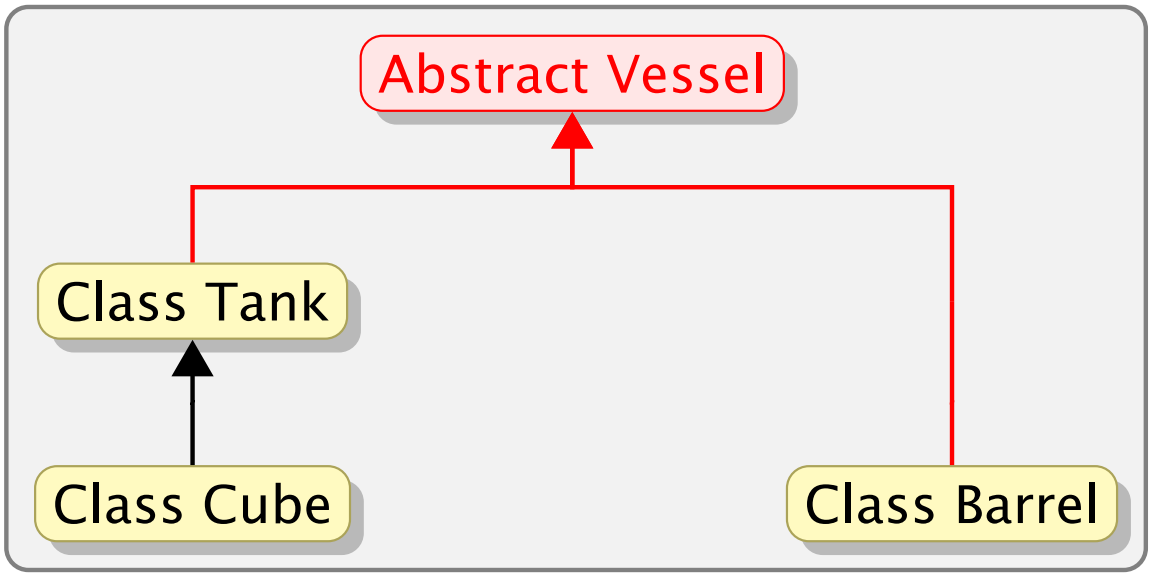
Abstract Classes, Subclasses and Overriding

Class Hierarchy Example

- The abstract class `Vessel` models the notion of a vessel for liquids
- It has a field `contents` representing its actual contents
- An abstract method `capacity()` for computing its maximal capacity and a method for filling in more, but only up to its capacity (the excess will be lost)
- The abstract class has subclasses `Tank` (a `Rectangular cuboid` vessel), `Cube` (a cubic vessel, subclass of `Tank`), and `Barrel` (a cylindrical vessel)
- The subclasses implement the `capacity()` method, they inherit the `contents` field and the `fill()` method from the superclass, and they override the `toString()` method inherited (inherited from class `Object`) to print each vessel object appropriately

Abstract Classes, Subclasses and Overriding

Class Hierarchy Example — Diagram



- The red rectangles denote abstract classes (which may implement various interfaces)
- Yellow rectangles denote concrete classes extending abstract classes and (possibly) implementing interfaces
- Note UML style diagrams have more detail — see UML Class and Object Diagrams Overview

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Classes
Class Hierarchy
Interfaces
String Formatting
JShell
What Next ?
References

Class Hierarchy

Vessel Example (1)

```
43 abstract class Vessel {
44   double contents ;

46   public Vessel() {
47     contents = 0 ;
48   }

50   abstract double capacity() ;

52   public void fill(double amount) {
53     contents
54       = Math.min(contents + amount, capacity()) ;
55   }
56 }
```

- **Abstract** classes cannot be instantiated but can be extended
- Intended to be a superclass of several classes that have common features

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Classes
Class Hierarchy
Interfaces
String Formatting
JShell
What Next ?
References

Class Hierarchy

Vessel Example (2)

```
58 class Tank extends Vessel {
59   double depth, width, height ;

61   public Tank(double depth
62               , double width, double height) {
63     super() ;
64     this.depth = depth ;
65     this.width = width ;
66     this.height = height ;
67   }

69   @Override
70   double capacity() {
71     return depth * width * height ;
72   }

74   @Override
75   public String toString() {
76     return ("tank␣(" + depth + ","
77             + width + "," + height + ")") ;
78   }
79 }
```


Tutorial Agenda
Adobe Connect
Composition
Inheritance
Classes
Class Hierarchy
Interfaces
String Formatting
JShell
What Next ?
References

Class Hierarchy

Vessel Example (3)

```
81 class Cube extends Tank {

83   public Cube(double side) {
84     super(side, side, side) ;
85   }

87   @Override
88   public String toString() {
89     return ("cube␣(" + depth + ")") ;
90   }
91 }
```

Class Hierarchy

Vessel Example (4)

```
93 class Barrel extends Vessel {
94   double radius, height ;

96   Barrel(double radius, double height) {
97     super() ;
98     this.radius = radius ;
99     this.height = height ;
100  }

102  @Override
103  double capacity() {
104    return (height * Math.PI * radius * radius) ;
105  }

107  @Override
108  public String toString() {
109    return ("barrel (" + radius + ", " + height + ")") ;
110  }
111 }
```

Class Hierarchy

Vessel Example (5)

```
 9 class VesselEG {
10   public static void main(String[] args) {
11     Vessel v1 = new Barrel(3, 10) ;
12     Vessel v2 = new Tank(10, 20, 12) ;
13     Vessel v3 = new Cube(4) ;

15     Vessel[] vs = { v1, v2, v3 } ;

17     v1.fill(90) ;
18     v1.fill(10) ;
19     v2.fill(100) ;
20     v3.fill(80) ;
```

```
40   }
41 }
```


Tutorial Agenda
Adobe Connect
Composition
Inheritance
Classes
Class Hierarchy
Interfaces
String Formatting
JShell
What Next ?
References

Class Hierarchy

Vessel Example (6)

```
22    double totalCap = 0 ;
23    for (int i = 0; i < vs.length; i++) {
24      totalCap = totalCap + vs[i].capacity() ;
25    }
26    System.out.println("Total capacity is "
27                       + totalCap) ;

29    double totalConts = 0 ;
30    for (int i = 0; i < vs.length; i++) {
31      totalConts = totalConts + vs[i].contents ;
32    }
33    System.out.println("Total contents is "
34                       + totalConts) ;

36    for (int i = 0; i < vs.length; i++) {
37      System.out.println("vessel number " + i
38                         + ": " + vs[i]) ;
39    }
```


Tutorial Agenda
Adobe Connect
Composition
Inheritance
Classes
Class Hierarchy
Interfaces
String Formatting
JShell
What Next ?
References

Class Hierarchy

Vessel Example (7)

```
jshell> /open VesselEG.java

jshell> VesselEG.main(new String[0])
Total capacity is 2746.743338823081
Total contents is 264.0
vessel number 0: barrel (3.0, 10.0)
vessel number 1: tank (10.0,20.0,12.0)
vessel number 2: cube (4.0)

jshell>
```


Tutorial Agenda
Adobe Connect
Composition
Inheritance
Classes
Class Hierarchy
Interfaces
String Formatting
JShell
What Next ?
References

Interfaces

Interface Declarations (1)

- An **interface** describes fields and methods but does not implement them
- It defines a *type* by specifying the behaviour of objects (whereas classes specify types by how objects are constructed)

```
interfaceModifiers interface I extendsClause
  fieldDescriptions
  methodDescriptions
  methodDeclarations // Java 8 not used in M250
  classDeclarations        // not used in M250
  interfaceDeclarations    // not used in M250
```

- Notes and examples based on Sestoft (2016, section 13) *Java Precisely*

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
Interface Declarations
Interfaces — Example
String Formatting
JShell
What Next ?
References

Interfaces

Interface Declarations (2)

- An interface may be declared at top level
- *interfaceModifiers* may be `public` or absent
- The *extendsClause* may be absent or `extends I1,I2,...` where the `I1,I2,...` are super-interfaces
- A *fieldDescription* declares a named constant

```
fieldDescModifiers type f = initializer ;
```

- *fieldDescModifiers* is implicitly a list of `public`, `final`, `static`
- *methodDescription* for method `m` has the form

```
methodDescModifiers returnType m(formalList)
                                  throwsClause ;
```

- *methodDescModifiers* is implicitly `abstract`, `public`

Interfaces

Example (1a)

```
import java.awt.Color ;
import java.awt.Graphics ;

interface Colored {
  Color getColor() ;
}

interface Drawable {
  void draw(Graphics g) ;
}

interface ColoredDrawable extends Colored, Drawable {
  // empty body but inherits from Colored and Drawable
}
```

- The `Colored` interface describes method `getColor()`
- Interface `Drawable` method `draw`
- `ColoredDrawable` describes both
- The methods are implicitly `public`

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
Interface Declarations
Interfaces — Example
String Formatting
JShell
What Next ?
References

Interfaces

Example (1b)

```
class ColoredPoint extends Point implements Colored {
  Color c;

  ColoredPoint(int x, int y, Color c) {
    super(x, y) ;
    this.c = c ;
  }

  @Override
  public Color getColor() {
    return c ;
  }
}
```

- The methods `getColor()` and `draw` must be public as described in the interface declarations

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
Interface Declarations
Interfaces — Example
String Formatting
JShell
What Next ?
References

Interfaces

Example (1c)

```
class ColoredDrawablePoint extends ColoredPoint
                           implements ColoredDrawable {

  ColoredDrawablePoint(int x, int y, Color c) {
    super(x, y, c) ;
  }

  @Override
  public void draw(Graphics g) {
    g.fillRect(x, y, 1, 1) ;
  }
}
```

- The method `fillRect()` is from package `java.awt.Graphics` in the module `java.desktop`
- `fillRect(int x,int y,int width,int height)`

Interfaces

Example (1d)

```
class ColoredRectangle implements ColoredDrawable {
  int x1, x2, y1, y2;
  // (x1, y1) upper left, (x2, y2) lower right corner
  Color c;

  ColoredRectangle(int x1, int y1, int x2, int y2
                   , Color c) {
    this.x1 = x1 ; this.y1 = y1 ;
    this.x2 = x2 ; this.y2 = y2 ;
    this.c = c ;
  }

  @Override
  public Color getColor() {
    return c ;
  }

  @Override
  public void draw(Graphics g) {
    g.drawRect(x1, y1, x2-x1, y2-y1) ;
  }
}
```

String Formatting

Introduction (1)

- A common task is to control the layout justification and alignment formats for numeric, strings and date/time data
- Here is an example of a simple interest calculation (using JShell)

```
jshell> double intrst = 343.17 * 2.4 / 100
intrst ==> 8.236080000000001

jshell> System.out.println("Interest is " + intrst + " units")
Interest is 8.236080000000001 units
```

- We *could* write code to round the result to 2 decimal places but Java provides several ways of formatting strings by providing a *format string* (or *template string*) with embedded *format specifiers*

String Formatting

Introduction (2)

- Using `printf`

```
jshell> System.out.printf("Interest␣is␣%.2f␣units%n", intrst)
Interest is 8.24 units
```

- Using `String.format`

```
jshell> String intrstOut = (
   ...> String.format("Interest␣is␣%.2f␣units%n", intrst) )
intrstOut ==> "Interest␣is␣8.24␣units\n"

jshell> System.out.print(intrstOut)
Interest is 8.24 units
```

- `%.2f` and `%n` are *format specifiers*
- Notice the interest is rounded to 2 decimal places
- The `%n` results in a *line separator*

String Formatting

String Formatting Methods

- `String.format(fmt,v1,...,vn)` returns a String produced from `fmt` by replacing format specifiers with the strings resulting from formating the values `V1,...,vn`
- `strm.printf(fmt,v1,...,vn)` where `strm` is a PrintWriter or PrintStream, constructs a string as above, outputs it to `strm` , and returns `strm`.
- `strm.format(fmt,v1,...,vn)` behaves as `strm.printf(fmt,v1,...,vn)`
- If a value `vi` is the wrong type for a given format specifier (or the format specifier is ill-formed) then an error is generated

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
JShell
What Next ?
References

String Formatting

Formatting Specifiers

- A formatting specifier for numeric, character, and general types has the form:

```
%[index$][flags][width][.precision]conversion
```

- The *index* is an integer 1,2,... indicating the value v_{index} to format.
- The *conversion* indicates what operation is used to format the value.
- The *width* indicates the minimum number of characters used to format the value.
- The *flags* indicate how the width should be used
- *precision* limits the output, such as number of fractional digits
- The brackets [] are meta-characters indicating optional parts
- The only mandatory parts are the percent sign (%) and the *conversion*

Formatting Specifiers

Conversions

- Table of *conversions* on numbers, characters (C) and general types (G)
- I integers, F floats, IF both.
- Uppercase conversion such as X produces uppercase

Format	*conversion*	*flags*	*precision*	Type
Decimal	d	-+ 0,(		I
Octal	o	-#0		I
Hexadecimal	x, X	-#0		I
Hex significand, exponent	a, A	-#+ 0		F
General: scientific, fractional	g, G	-#+ 0,(	Max. sig, digits	IF
Fixed-point number	f	-#+ 0,(	Fractional digits	IF
Scientific notation	e, E	-#+ 0,(	Fractional digits	IF
Unicode character [1]	c, C	-		C
Boolean: `false`, `true`	b, B	-		Boolean
Hex hashcode or `null`	h, H	-		G
Determined by `formatTo` method	s, S	-		G
A percent symbol (%)	%	(none)		
Platform specific newline	n	(none)		

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
JShell
What Next ?
References

Formatting Specifiers

Flags

Flag	Result
-	Left-justified
#	Conversion-dependent alternate form
+	Always include a sign
␣	Includes leading space for positive values
0	Zero-padded
,	Locale-specific grouping separators
(	Enclose negative numbers in parentheses

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
JShell
What Next ?
References

String Formatting

Documentation

- See java.util.Formatter for the full details
- See also the `man` entry for `printf` in Unix and Text.Printf in Haskell
- See also Python: printf-style String Formatting and:
  - Python: Language Reference: Section 2.4.3 Formatted string literals and PEP 498 — Literal String Interpolation
  - Python: str.format() and Python: Format String Syntax
  - Python: Template strings and PEP 292 — Simpler String Substitutions
- Note that *printf* is an approach to string formatting rather than an absolute standard — there are many variations.
- See Wikipedia: printf format string

Java 2021 Formatting 2022 Exs

Q 1

- Given an array of integers, none with more than three digits, some negative and some positive

  print them one per line in a right justified column with some sample text before and after the column
- For example, given the array

```
{-123, 123, 23, -23}
```

could print the array index, `idx` and values in a column as

```
idx 0 value is -123 units
idx 1 value is  123 units
idx 2 value is   23 units
idx 3 value is  -23 units
```

Go to Soln 1

Java: Composition, Inheritance, Interfaces
Phil Molyneux
Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
Q 1
Soln 1
Q 2
Soln 2
Q 3
Soln 3
Q 4
Soln 4
Q 5
Soln 5
Q 6
Soln 6
Q 7
Soln 7
JShell
What Next ?
References

Java Formatting 2022 Exs

Soln 1 (a)

- Sample answer in FormattingTest01.java

```
10  public static int[] egArrayInt01 = {-123, 123, 23, -23 } ;

12  public static void testSpace() {
13    System.out.println("Array␣egArrayInt01␣=␣"
14                       + Arrays.toString(egArrayInt01)) ;
15    int aryLen = egArrayInt01.length ;
16    for (int idx = 0; idx < aryLen; idx++) {
17      int num = egArrayInt01[idx] ;
18      System.out.format("idx␣%d␣value␣is␣%␣4d␣units%n",idx,num) ;
19    }
20  }
```

- Soln 1 continued on next slide

Go to Q 1

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
Q 1
Soln 1
Q 2
Soln 2
Q 3
Soln 3
Q 4
Soln 4
Q 5
Soln 5
Q 6
Soln 6
Q 7
Soln 7
JShell
What Next ?
References

Java Formatting 2022 Exs

Soln 1 (b)

- Using static method `testSpace()` in class `FormattingTest01` in `jshell`

```
jshell> FormattingTest01.testSpace()
Array egArrayInt01 = [-123, 123, 23, -23]
idx 0 value is -123 units
idx 1 value is  123 units
idx 2 value is   23 units
idx 3 value is  -23 units
```

Go to Q 1

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
Q 1
Soln 1
Q 2
Soln 2
Q 3
Soln 3
Q 4
Soln 4
Q 5
Soln 5
Q 6
Soln 6
Q 7
Soln 7
JShell
What Next ?
References

Java 2021 Formatting 2022 Exs

Q 2

- Given the following array

  format the values as in the previous question but with + in front of positive numbers

```
57  public static int[] egArrayInt03 = {-123, 123, +23, -23 } ;
```

- Notice that one value already has + in front.

Go to Soln 2

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
Q 1
Soln 1
Q 2
Soln 2
Q 3
Soln 3
Q 4
Soln 4
Q 5
Soln 5
Q 6
Soln 6
Q 7
Soln 7
JShell
What Next ?
References

Java Formatting 2022 Exs

Soln 2

- Here is the test outout — the code is in the following answer
- Note that the default printing of the array removes the provided + sign

```
jshell> FormattingTest01.testFmtStr1()
Array = [-123, 123, 23, -23]

idx 0 value is -123 units
idx 1 value is +123 units
idx 2 value is  +23 units
idx 3 value is  -23 units
```

Go to Q 2

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
Q 1
Soln 1
Q 2
Soln 2
Q 3
Soln 3
Q 4
Soln 4
Q 5
Soln 5
Q 6
Soln 6
Q 7
Soln 7
JShell
What Next ?
References

Java 2021 Formatting 2022 Exs

Q 3

- Given the following array

  format the values as in the previous question but with () around negative numbers

```
57  public static int[] egArrayInt03 = {-123, 123, +23, -23 } ;
```

Go to Soln 3

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
Q 1
Soln 1
Q 2
Soln 2
Q 3
Soln 3
Q 4
Soln 4
Q 5
Soln 5
Q 6
Soln 6
Q 7
Soln 7
JShell
What Next ?
References

Java Formatting 2022 Exs

Soln 3 (a)

- The first solution has a width of 4 — notice the layout of the first number

```
jshell> FormattingTest01.testFmtStr2()
Array = [-123, 123, 23, -23]

idx 0 value is (123) units
idx 1 value is  123 units
idx 2 value is   23 units
idx 3 value is (23) units
```

- The second version changes the width to 5

```
jshell> FormattingTest01.testFmtStr2a()
Array = [-123, 123, 23, -23]

idx 0 value is (123) units
idx 1 value is   123 units
idx 2 value is    23 units
idx 3 value is  (23) units
```

- Soln 3 continued on next slide

Go to Q 3

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
Q 1
Soln 1
Q 2
Soln 2
Q 3
Soln 3
Q 4
Soln 4
Q 5
Soln 5
Q 6
Soln 6
Q 7
Soln 7
JShell
What Next ?
References

Java Formatting 2022 Exs

Soln 3 (b)

- The code

```
57  public static int[] egArrayInt03 = {-123, 123, +23, -23 } ;
58  public static String fmtStr1 = "idx␣%d␣value␣is␣%+4d␣units%n" ;
59  public static String fmtStr2 = "idx␣%d␣value␣is␣%(4d␣units%n" ;
60  public static String fmtStr2a = "idx␣%d␣value␣is␣%(5d␣units%n" ;

62  public static void testFmtStr(String fmtStr, int[] ary) {
63    System.out.printf("Array␣=␣%s%n%n",
64                      Arrays.toString(ary)) ;
65    int aryLen = ary.length ;
66    for (int idx = 0; idx < aryLen; idx++) {
67      int num = ary[idx] ;
68      System.out.format(fmtStr,idx,num) ;
69    }
70  }
```

- Soln 3 continued on next slide

Go to Q 3

Java: Composition, Inheritance, Interfaces
Phil Molyneux
Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
Q 1
Soln 1
Q 2
Soln 2
Q 3
Soln 3
Q 4
Soln 4
Q 5
Soln 5
Q 6
Soln 6
Q 7
Soln 7
JShell
What Next ?
References

Java Formatting 2022 Exs

Soln 3 (c)

- Test harness code

```
72  public static void testFmtStr1() {
73    testFmtStr(fmtStr1, egArrayInt03) ;
74  }

76  public static void testFmtStr2() {
77    testFmtStr(fmtStr2, egArrayInt03) ;
78  }

80  public static void testFmtStr2a() {
81    testFmtStr(fmtStr2a, egArrayInt03) ;
82  }
```

Go to Q 3

Java: Composition, Inheritance, Interfaces
Phil Molyneux
Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
Q 1
Soln 1
Q 2
Soln 2
Q 3
Soln 3
Q 4
Soln 4
Q 5
Soln 5
Q 6
Soln 6
Q 7
Soln 7
JShell
What Next ?
References

Java 2021 Formatting 2022 Exs

Q 4

- Write a format specifier to format 1123456 as 1,123,456

Go to Soln 4

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
Q 1
Soln 1
Q 2
Soln 2
Q 3
Soln 3
Q 4
Soln 4
Q 5
Soln 5
Q 6
Soln 6
Q 7
Soln 7
JShell
What Next ?
References

Java Formatting 2022 Exs

Soln 4

- Sample answer

```
jshell> String str1 = String.format("%,d%n",1123456)
str1 ==> "1,123,456\n"

jshell> PrintStream strm1 = System.out.format("%,d%n",1123456)
1,123,456
strm1 ==> java.io.PrintStream@61064425
```

Go to Q 4

Java: Composition, Inheritance, Interfaces
Phil Molyneux
Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
Q 1
Soln 1
Q 2
Soln 2
Q 3
Soln 3
Q 4
Soln 4
Q 5
Soln 5
Q 6
Soln 6
Q 7
Soln 7
JShell
What Next ?
References

Java 2021 Formatting 2022 Exs

Q 5

- Write a format specifier that will take a single argument 1234.1234 and format it both right and left justified in a width of 20

  for example

```
|           1234.1234|
|1234.1234            |
```

Go to Soln 5

Java: Composition, Inheritance, Interfaces

Phil Molyneux

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
Q 1
Soln 1
Q 2
Soln 2
Q 3
Soln 3
Q 4
Soln 4
Q 5
Soln 5
Q 6
Soln 6
Q 7
Soln 7
JShell
What Next ?
References

Java Formatting 2022 Exs

Soln 5

- This uses the (-) and an index specifier

```
jshell> String.format("|%1$20.4f|%n|%1$-20.4f|%n",1234.1234)
$39 ==> "|          1234.1234|\n|1234.1234          |\n"

jshell> System.out.format("|%1$20.4f|%n|%1$-20.4f|%n",1234.1234)
|          1234.1234|
|1234.1234           |
$41 ==> java.io.PrintStream@61064425
```

- Notice how the same argument is used more than once

Go to Q 5

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
Q 1
Soln 1
Q 2
Soln 2
Q 3
Soln 3
Q 4
Soln 4
Q 5
Soln 5
Q 6
Soln 6
Q 7
Soln 7
JShell
What Next ?
References

Java 2021 Formatting 2022 Exs

Q 6

- Given an array with at least 11 integers, format a character bar chart similar to the followin

```
Array = [0, 3, 5, 6, 9, 11, 13, 10, 8, 7, 5, 3]
```

- Output

```
 0  (0)
 1 ### (3)
 2 ##### (5)
 3 ###### (6)
 4 ######### (9)
 5 ########### (11)
 6 ############# (13)
 7 ########## (10)
 8 ######## (8)
 9 ####### (7)
10 ##### (5)
11 ### (3)
```

Go to Soln 6

Java: Composition, Inheritance, Interfaces
Phil Molyneux
Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
Q 1
Soln 1
Q 2
Soln 2
Q 3
Soln 3
Q 4
Soln 4
Q 5
Soln 5
Q 6
Soln 6
Q 7
Soln 7
JShell
What Next ?
References

Java Formatting 2022 Exs

Soln 6 (a)

- We write the code in two parts
- The first part generates a string with the number of display characters
- The second part outputs the character bar chart
- The first version of `genNumChars()` uses an ordinary `for` loop

```
28  public static String genNumChars(int n, char ch) {
29    String str = "" ;
30    for (int idx = 0; idx < n; idx++) {
31      str = str + ch ;
32    }
33    return str ;
34  }
```

- Soln 6 continued on next slide

Go to Q 6

Java: Composition, Inheritance, Interfaces
Phil Molyneux
Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
Q 1
Soln 1
Q 2
Soln 2
Q 3
Soln 3
Q 4
Soln 4
Q 5
Soln 5
Q 6
Soln 6
Q 7
Soln 7
JShell
What Next ?
References

Java Formatting 2022 Exs

Soln 6 (b)

- The second version, genNumChars01() uses a library method from java.lang.String

```
36  public static String genNumChars01(int n, char ch) {
37    String str1 = String.valueOf(ch) ;
38    String str2 = str1.repeat(n) ;
39    return str2 ;
40  }
```

- Soln 6 continued on next slide

Go to Q 6

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
Q 1
Soln 1
Q 2
Soln 2
Q 3
Soln 3
Q 4
Soln 4
Q 5
Soln 5
Q 6
Soln 6
Q 7
Soln 7
JShell
What Next ?
References

Java Formatting 2022 Exs

Soln 6 (c)

- The second part, `printTableFromArray()` outputs the character bar chart

```
42  public static int[] egArrayInt02
43    = {0,3,5,6,9,11,13,10,8,7,5,3} ;
44  public static char dsplyCh = '#' ;

46  public static void printTableFromArray(int[] ary) {
47    System.out.printf("Array = %s%n%n",
48                      Arrays.toString(ary)) ;
49    int aryLen = ary.length ;
50    for (int idx = 0; idx < aryLen; idx++) {
51      int num = ary[idx] ;
52      String str = genNumChars01(num,dsplyCh) ;
53      System.out.format("%2d %s (%d)%n",idx,str,num) ;
54    }
55  }
```

Go to Q 6

Java: Composition, Inheritance, Interfaces
Phil Molyneux
Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
Q 1
Soln 1
Q 2
Soln 2
Q 3
Soln 3
Q 4
Soln 4
Q 5
Soln 5
Q 6
Soln 6
Q 7
Soln 7
JShell
What Next ?
References

Formatting 2022 Exs

Q 7

(a) Write a format specifier that takes a single argument 1234.1264 and formates it as 1,234.13 in a field of width 20

(b) Write a format specifier that takes a single argument 1123456 and formats it as 1,123,456.00 in a field of width 15

Go to Soln 7

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
Q 1
Soln 1
Q 2
Soln 2
Q 3
Soln 3
Q 4
Soln 4
Q 5
Soln 5
Q 6
Soln 6
Q 7
Soln 7
JShell
What Next ?
References

Formatting 2022 Exs

Soln 7

(a) Write a format specifier that takes a single argument 1234.1264 and formats it as 1,234.13 in a field of width 20

```
jshell> String str1 = String.format("%,20.2f%n",1234.1264)
str1 ==> "␣␣␣␣␣␣␣␣␣␣␣1,234.13\n"
```

(b) Write a format specifier that takes a single argument 1123456 and formats it as 1,123,456.00 in a field of width 15

```
jshell> String str1 = String.format("%,15.2f%n",(float) 1123456)
str1 ==> "␣␣␣1,123,456.00\n"
```

▸ Go to Q 7

Java: Composition, Inheritance, Interfaces
Phil Molyneux
Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
String Formatting Methods
Formatting Data Types
String Formatting Exercises
Q 1
Soln 1
Q 2
Soln 2
Q 3
Soln 3
Q 4
Soln 4
Q 5
Soln 5
Q 6
Soln 6
Q 7
Soln 7
JShell
What Next ?
References

Java Shell, JShell

References

- JShell is a Java *read-eval-print loop (REPL)* introduced in 2017 with JDK 9
- Java Shell User's Guide (Release 12, March 2019)
- Tools Reference: jshell
- JShell Tutorial (30 June 2019)
- How to run a whole Java file added as a snippet in JShell? (15 July 2019)

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

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

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

What Next ?

To err is human ?

- 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)

What Next ?

Block 2, TMA02

- TMA02 Thursday 6 March 2025
- Tutorial: Collections and file I/O: Online 10:00 Sunday 16 March 2025
- TMA03 Thursday 8 May 2025
- Tutorial: Exam revision: Online 10:00 Sunday 11 May 2025

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References

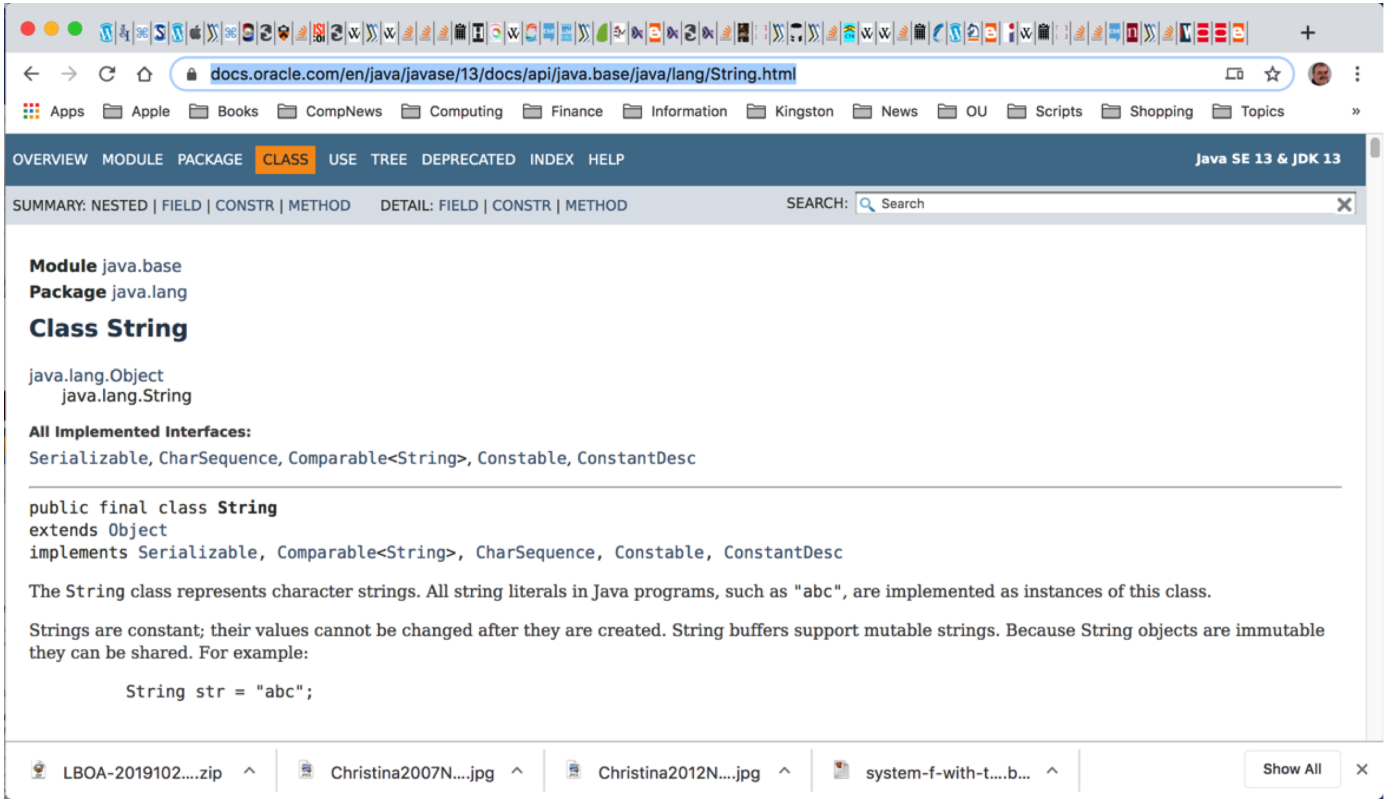
M250

Web Links

- 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

Java

API Documentation (1)

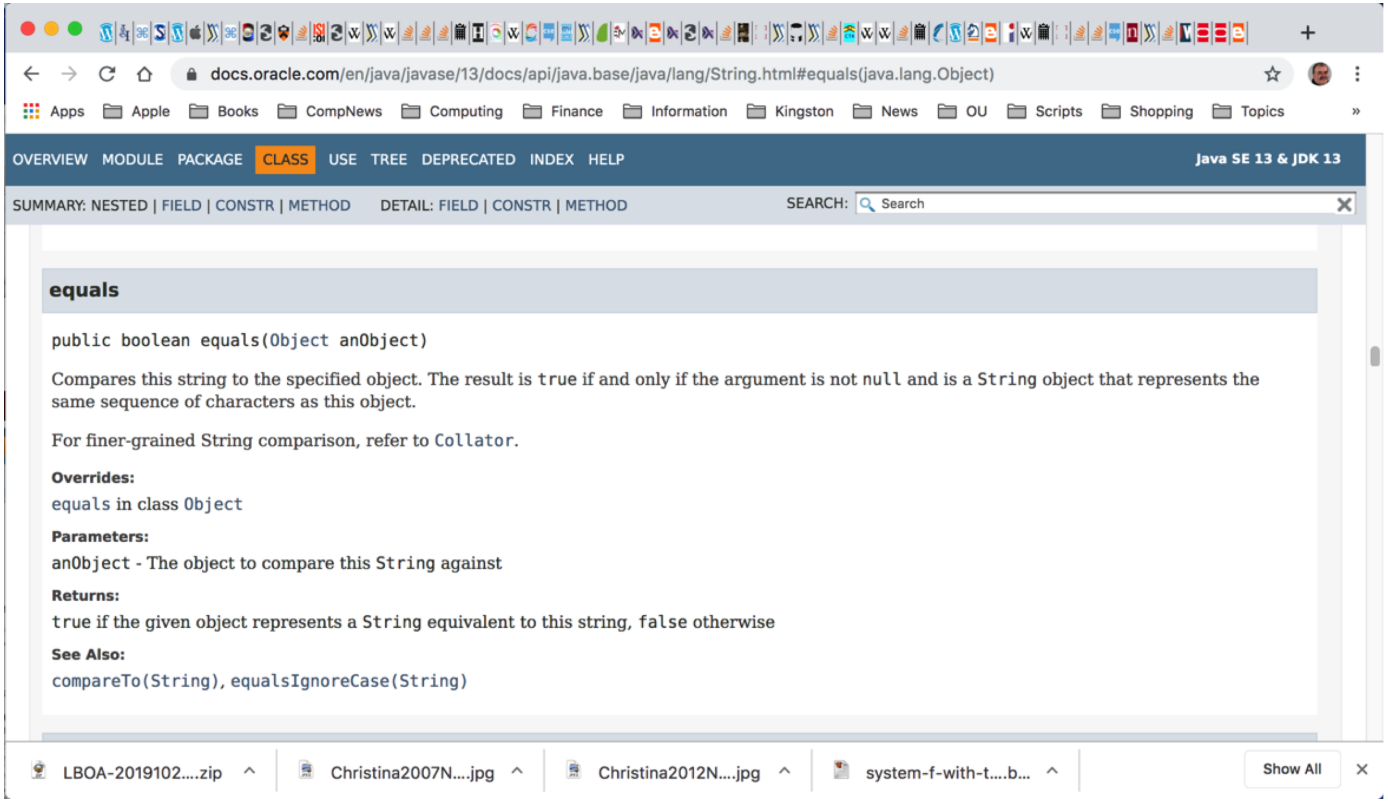


- `Strings` are *immutable* objects
- See java.lang.StringBuilder for *mutable* strings
- In a *functional programming approach* everything is immutable — it makes life simpler (but at a cost)

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References
Java Documentation
Books Phil Likes

Java

API Documentation (2)



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

M250

Books Phil Likes

- M250 is self contained — you do not need further books — but you might like to know about some:
- Sestoft (2016) *Java Precisely* — the best short reference
- Evans, Flanagan (2018) — the best longer reference *Java in a Nutshell*

  Evans, clark, Flanagan (2023) *Java in a Nutshell*
- Barnes, Kölling (2016) *Object First with Java* — the BlueJ book — see www.bluej.org for documentation and tutorial
- Bloch (2017) *Effective Java* — guide to best practice

Tutorial Agenda
Adobe Connect
Composition
Inheritance
Interfaces
String Formatting
JShell
What Next ?
References
Java Documentation
Books Phil Likes