

Java: Selection, Iteration, Inheritance, Composition

M250 Tutorial 04

Contents

1	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	5
2.5	Invite Attendees	6
2.6	Layouts	7
2.7	Chat Pods	7
2.8	Web Graphics	8
2.9	Recordings	8
3	Statements: Summary	8
3.1	Statements Overview	8
3.2	Expression & Block Statements	9
3.3	Selection Statements	9
3.4	Iteration Statements	10
3.4.1	for Statement	10
3.4.2	while Statement	11
3.4.3	foreach Statement	12
3.4.4	Removing While Iterating	12
3.5	Returns, Exits and Exceptions	13
3.6	assert Statement	15
4	Composition	15
4.1	Initialising a Lollipop Object	16
4.2	Circle,Rectangle	17
4.3	Lollipop Class	19
4.4	Lollipop Interaction	22
4.5	Composition Summary	23
5	JShell	23
6	What Next ?	23
7	References	24
7.1	Java Documentation	24
7.2	Books Phil Likes	25
	References	26

1 M250 Java: Selection, Iteration, Inheritance, Composition: Tutorial Agenda

- Introductions
- Adobe Connect reminders
- *Adobe Connect* — if you or I get cut off, wait till we reconnect (or send you an email)
- Statements: Select, Iteration and others
- Composition
- JShell (optional)
- Some useful Web & other references
- Time: about 1 hour
- Do ask questions or raise points.
- Slides/Notes [M250Tutorial20250119CompositionPrsntn2024J](#)

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 [L^AT_EX](#) — all these slides and notes
 - [Mac OS X](#)
- *Learning style* — I read the manual before using the software

Introductions — You

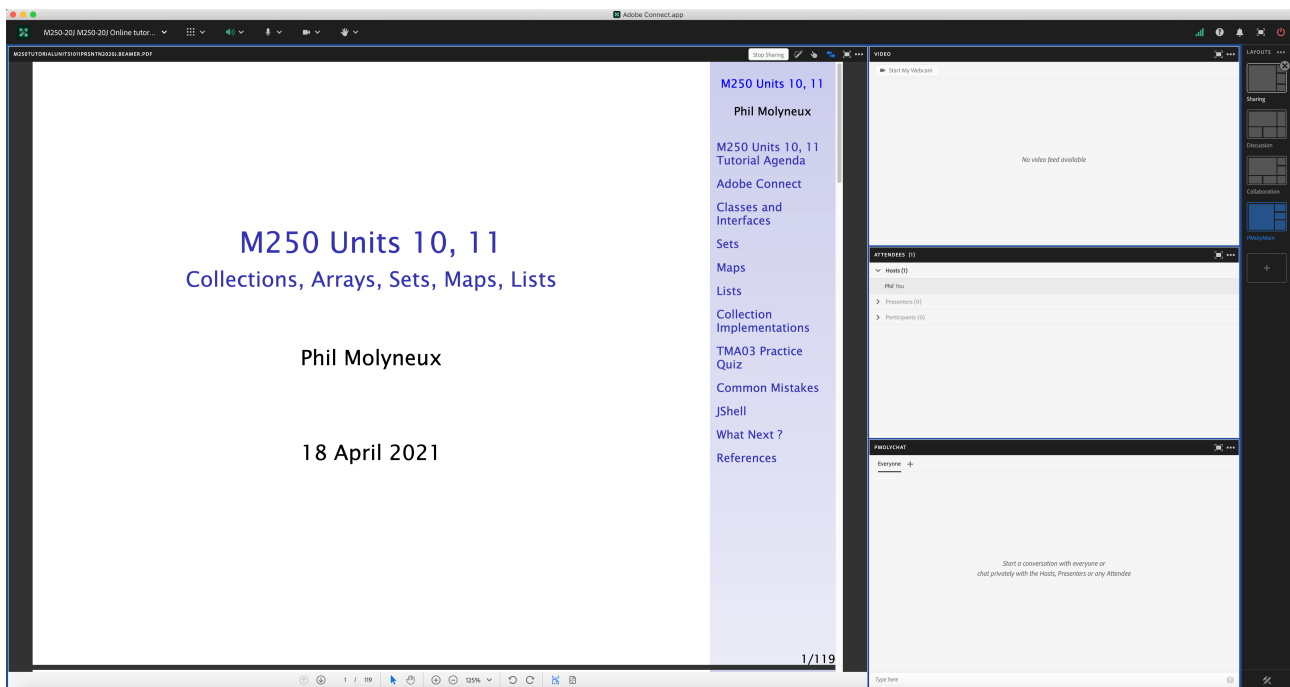
- *Name* ?
- *Favourite software/Programming language* ?
- *Favourite text editor or integrated development environment (IDE)*
- [List of text editors](#), [Comparison of text editors](#) and [Comparison of integrated development environments](#)

- Other OU courses ?
- Anything else ?

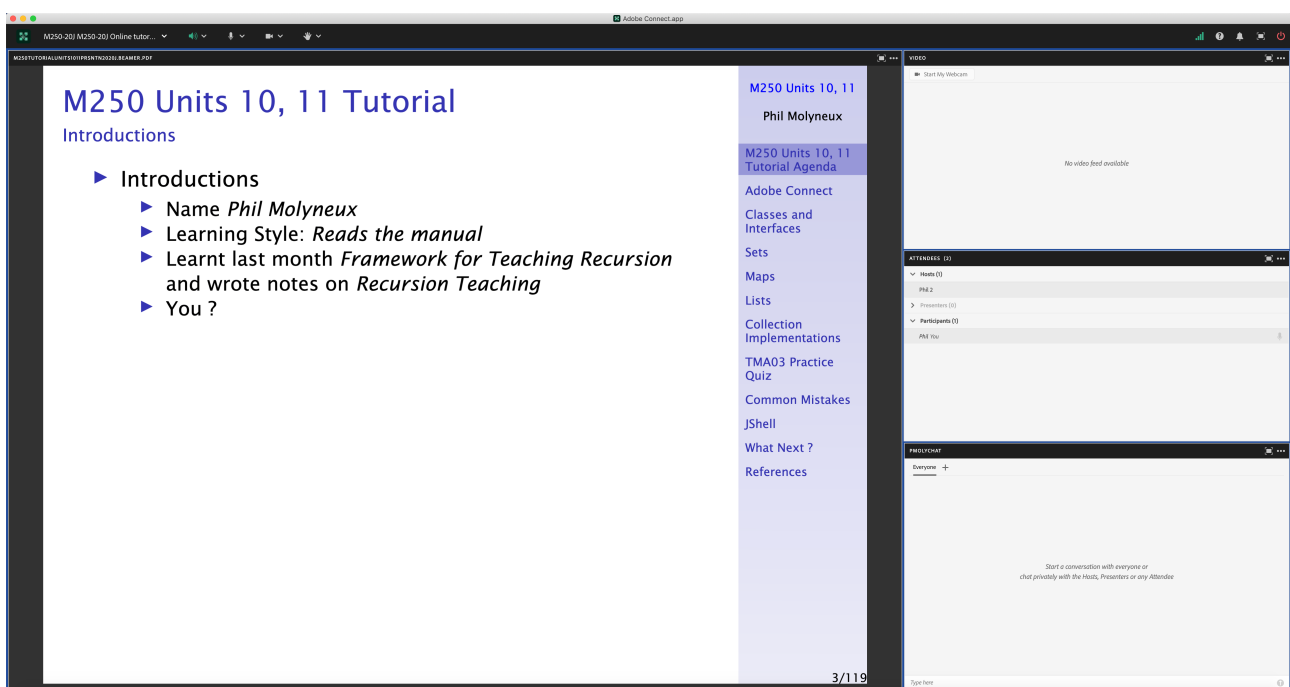
2 Adobe Connect Interface and Settings

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

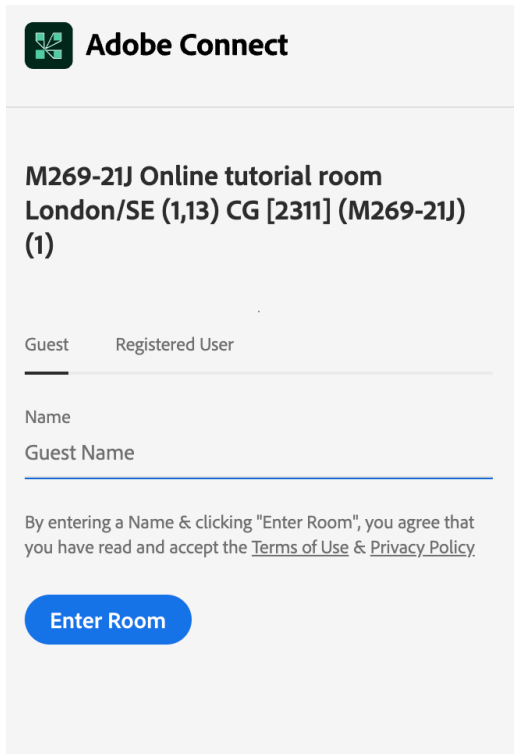
- **Share My Screen**  **Application tab**  **Terminal** for [Terminal](#)
- **Share menu**  **Change View**  **Zoom in** for mismatch of screen size/resolution (Participants)
- (Presenter) Change to 75% and back to 100% (solves participants with smaller screen image overlap)
- Leave the application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

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**



Adobe Connect

M269-21J Online tutorial room
London/SE (1,13) CG [2311] (M269-21J)
(1)

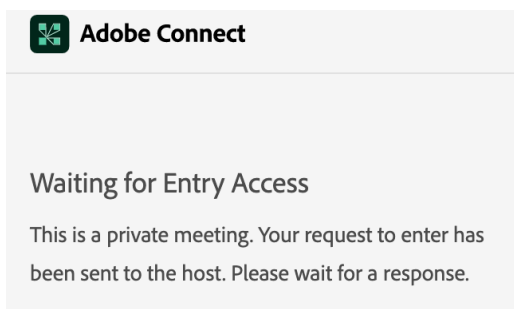
Guest Registered User

Name
 Guest Name

By entering a Name & clicking "Enter Room", you agree that you have read and accept the [Terms of Use](#) & [Privacy Policy](#).

Enter Room

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

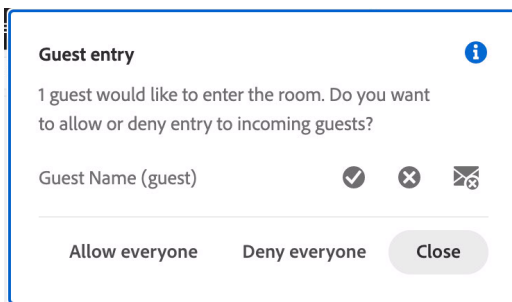


Adobe Connect

Waiting for Entry Access

This is a private meeting. Your request to enter has been sent to the host. Please wait for a response.

- *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 *PMolyAuxOn*
 - 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

3 Statements: Summary

3.1 Statements Overview

- A statement may change the computer's state: value of variables, fields, array elements, the contents of files and so on — the execution of a statement may:
- terminate normally (and execution continues with the next statement, if any) or
- terminate abruptly by throwing an exception or
- exit by executing a `return` statement (if inside a method or constructor) or
- exit a switch or loop by executing a `break` statement or
- exit the current iteration of a loop and start a new iteration by executing a `continue` statement or
- does not terminate at all (eg, `while (true) {}`)

3.2 Expression & Block Statements

- An expression statement is an `expression` followed by a `;`

```
expression ;
```

- The only forms of `expression` that may be used here are assignments, increment and decrements, method call, and object creation
- A block statement is a sequence of variable declarations, class declarations and statements

```
{
  variableDeclarations
  classDeclarations
  statements
}
```

- An empty statement consists of `;` only — it is equivalent to the block statement `{ }`

3.3 Selection Statements

- The `if` statement has the form

```
if (condition)
  trueBranch
```

- The `if-else` statement has the form

```
if (condition)
  trueBranch
else
  falseBranch
```

- The `condition` must have type `boolean` or `Boolean`
- `trueBranch` and `falseBranch` are statements
- What is wrong with the following

```
if (dataAvailable) ;
  processData() ;
```

```
if (dataAvailable)
  processData() ;
  reportResults() ;
```

```
if (dataAvailable)
  processData() ;
  reportResults() ;
else
  reportNoData() ;
```

```
if (dataAvailable) ;
  processData() ;
```

- The `trueBranch` is an empty statement `;`

```
if (dataAvailable)
  processData() ;
  reportResults() ;
```

- `reportResults()` ; will always be executed

```
if (dataAvailable)
    processData() ;
    reportResults() ;
else
    reportNoData() ;
```

- Will not compile
- **Moral** Always use block statements
- A `switch` statement has the form

```
switch (expression) {
    case constant1: branch1
    case constant2: branch2
    ...
    default: branchN
}
```

- `expression` must be of type `int`, `short`, `char`, `byte` or a boxed version of these or `String` or an enum type
- Each `constant` must be a compile-time constant expression, consisting only of literals, `final` variables, `final` fields declared with explicit field initialisers or an unqualified enum value
- (not used in M250)

3.4 Iteration Statements

3.4.1 for Statement

- A `for` statement has the form

```
for (initialization ; condition ; step)
    body
```

- `initialization` is a `variableDeclaration` or an expression
- `condition` is an expression of type `boolean` or `Boolean`
- `step` is an expression
- `body` is a statement
- `initialization` and `step` may be comma-separated lists of expressions
- `initialization`, `condition` and `step` may be empty. An empty `condition` is equivalent to `true`
- The `for` statement is executed as follows
 1. The `initialization` is executed
 2. The `condition` is evaluated. If it is `false`, the loop terminates.
 3. If it is `true` then
 - (a) the `body` is executed
 - (b) the `step` is executed
 - (c) execution continues at (2.)

- What does the following code do ?

```
for (int i = 1 ; i <= 4 ; i++) {
    for (int j = 1 ; j <= i ; j++) {
        System.out.print("*") ;
    }
    System.out.println() ;
}
```

```
jshe11> for (int i = 1 ; i <= 4 ; i++) {
...>     for (int j = 1 ; j <= i ; j++) {
...>         System.out.print("*") ;
...>     }
...>     System.out.println() ;
...> }
...>

*
**
***
****
```

3.4.2 while Statement

- A **while** statement has the form

```
while (condition)
    body
```

- **condition** is an expression of type **boolean** or **Boolean** and **body** is a statement
- It is executed as follows:

1. The **condition** is evaluated. If it is **false**, the loop terminates
2. If it is **true**, then
 - (a) The **body** is executed
 - (b) Execution continues at (1.)

- Example linear search with **while** loop

```
String[] wdays =
    {"Monday", "Tuesday", "Wednesday",
     "Thursday", "Friday", "Saturday", "Sunday"} ;

int wdayno(String wday) {
    int i = 0 ;
    while (i < wdays.length
           && ! wday.equals(wdays[i])) {
        i++ ;
    }
    if (i < wdays.length) {
        return i ;
    } else {
        return -1 ;
    }
} ;
```

```
String[] wdays =
    {"Monday", "Tuesday", "Wednesday",
     "Thursday", "Friday", "Saturday", "Sunday"} ;

int wdayno(String wday) {
    int i = 0 ;
    while (i < wdays.length
           && ! wday.equals(wdays[i])) {
        i++ ;
    }
    if (i < wdays.length) {
```

```

    return i ;
  } else {
    return -1 ;
  }
} ;

```

```

jshe11> int d1 = wdayno("Friday") ;
d1 ==> 4

```

```

jshe11> int d2 = wdayno("Dimanche") ;
d2 ==> -1

```

- Write code using a **while** statement that is equivalent to a **for** loop statement
- Write code using a **while** statement that is equivalent to a **for** loop statement

```

initialization
while (condition) {
  body
  step
}

```

```

for (initialization ; condition ; step)
  body

```

- Note that this is different behaviour to the **for** statement in Python where assignments to variables in the suite of the loop does not change the assignments made in the target list
- See [Python: for statement](#)

3.4.3 foreach Statement

- A variant of the **for** statement to iterate over elements of a collection (or *iterable*)

```

for (ElementType x : expression)
  body

```

- `expression` must have type `Iterable<t>` where `t` is some subtype of `ElementType` or an array
- All collections are directly iterable since `Collection<t>` has superinterface `Iterable<t>`
- Entries of a map can be iterated over because interface `Map<K,V>` describes a method `entrySet` that returns a `Set<Map.Entry<K,V>>` which implements `Iterable<Map.Entry<K,V>>`

3.4.4 Removing While Iterating

- Removing elements from a collection while iterating over it is fraught with problems
- Likely to generate errors (`ConcurrentModificationException`)
- Proper way is to use an `Iterator`
- Example from [Barnes and Kölling \(2016, 134\)](#)

```

Iterator<Track> it = tracks.iterator() ;
while (it.hasNext()) {
  Track t = it.next() ;
  String artist = t.getArtist() ;
  if (artist.equals(artistToRemove)) {
    it.remove() ;
  }
}

```

3.5 Returns, Exits and Exceptions

- A **return** statement with an expression argument has the form:

```
return expression ;
```

- This form of **return** must occur in the body of a method (not constructor) whose return type is a supertype or boxed or unboxed version of the type of *expression*
- The **return** statement is executed as follows:
- *expression* is evaluated to some value *v*
- It then exits the method and continues execution at the method call expression that called the method
- The value of that expression will be *v*, possibly after application of a widening, boxing or unboxing conversion
- **wdayno** using a **for** loop

```
int wdayno(String wday) {
    for (int i = 0 ; i < wdays.length ; i++) {
        if (wday.equals(wdays[i])) {
            return i ;
        }
    }
    return -1 ;
}
```

- Notice that the final **return** is after the **for** loop
- What is the effect of the code below?

```
int wdayno(String wday) {
    for (int i = 0 ; i < wdays.length ; i++) {
        if (wday.equals(wdays[i])) {
            return i ;
        }
    }
    return -1 ;
}
```

```
jshell> int wdayno(String wday) {
...>     for (int i = 0 ; i < wdays.length ; i++) {
...>         if (wday.equals(wdays[i])) {
...>             return i ;
...>         }
...>     }
...>     return -1 ;
...> }
...>
| Error:
| missing return statement
|     int wdayno(String wday) {
|         ^
```

- A **break** statement is legal only inside a loop or switch and has one of the forms

```
break ;
break labelName ;
```

- Executing **break** exits the innermost enclosing loop or switch and continues execution after that loop or switch
- A **continue** statement is legal only inside a loop and has one of the forms

```
continue ;
continue labelName ;
```

- Executing `continue` terminates the current iteration of the innermost enclosing loop and continues execution at the `step` in `for` loops or the `condition` in `while` and `do-while` loops
- A `label` statement has the form

```
labelName : statement
```

- The scope of `labelName` is `statement`, where it can be used in `break` or `continue`
- Use of labels is evidence of poor program design
- Just don't
- A `throw` statement has the form:

```
throw expression ;
```

- The type of `expression` must be a subtype of class `Throwable`
- The `throw` statement is executed as follows:
- `expression` is evaluated to obtain an exception object `v`
- If it is `null` then a `NullPointerException` is thrown
- Otherwise the exception object `v` is thrown
- The enclosing block statement terminates abruptly
- The thrown exception may be caught by a dynamically enclosing `try-catch` statement
- If the exception is not caught then the entire program execution will be aborted
- A `try-catch` statement is used to catch particular exceptions thrown by a code block
- It has the following form:

```
try
    body
catch (E1 x1) catchBody1
catch (E21 | E22 | ... | E2k x2) catchBody2
...
finally finallyBody
```

- All the various `bodies` are block statements
- There can be zero or more `catch` clauses and the `finally` clause may be absent, but there must be at least one `catch` or `finally` clause

```
class WeekdayException extends Exception {
    public WeekdayException(String wday) {
        super("Illegal weekday: " + wday) ;
    }
}

int wdayno(String wday) throws WeekdayException {
    for (int i = 0; i < wdays.length; i++) {
        if (wday.equals(wdays[i])) {
            return i ;
        }
    }
}
```

```

    }
    }
    throw new WeekdayException(wday) ;
}

```

```

jshell> class WeekdayException extends Exception {
...>     public WeekdayException(String wday) {
...>         super("Illegal weekday:_" + wday) ;
...>     }
...> }
...>

jshell> int wdayno(String wday) throws WeekdayException {
...>     for (int i = 0; i < wdays.length; i++) {
...>         if (wday.equals(wdays[i])) {
...>             return i ;
...>         }
...>     }
...>     throw new WeekdayException(wday) ;
...> }

jshell> int d4 = wdayno("Dimanche")
| Exception REPL.dJShelld31dWeekdayException:
|     Illegal weekday: Dimanche
|     at wdayno (#25:7)
|     at (#27:1)

```

3.6 assert Statement

- The `assert` statement has one of the following forms:

```

assert booleanExpression ;
assert booleanExpression : expression ;

```

- `booleanExpression` must have type `boolean` or `Boolean`
- `expression` must be of type `boolean`, `char`, `double`, `float`, `int`, `long`, a boxed version of these or `Object`
- When assertions are enabled at run-time, every execution of the `assert` command will evaluate `booleanExpression`
- If the result is `true`, program execution continues normally
- If the result is `false`, the assertion fails, and an `AssertionError` will be thrown
- In the second form, `expression` will be evaluated, and its value passed to the appropriate `AssertionError` constructor
- See Unit 8 section 7

```

assert x > 2 : "x_was_" + x ;

```

4 Composition

- Unit 4 Section 8 and the Composition Supplement discuss *composition* and compare it to *inheritance*
- *Composition* is preferred where there is a *has-a* or *is-part-of* relation

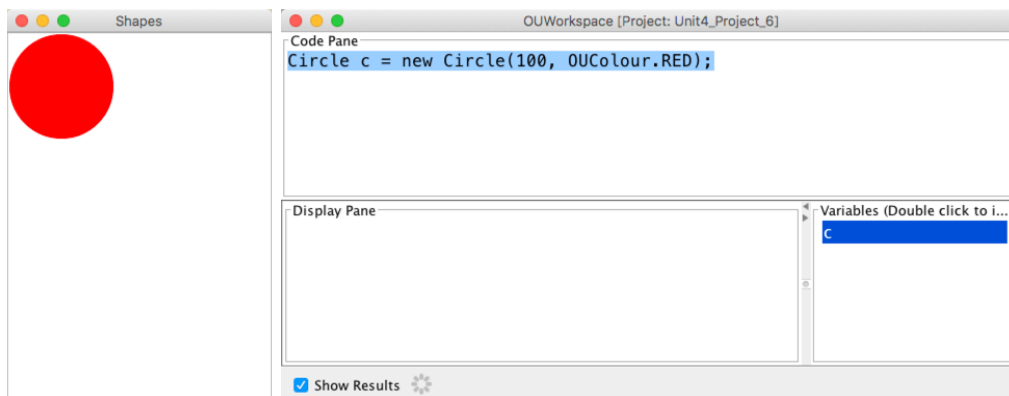
4.1 Initialising a Lollipop Object

- Lollipop Example 1 (a)
- Initialise the component objects
- Use them to initialise the composite object

```
Circle c = new Circle(100, OUColour.RED);
Rectangle r = new Rectangle(10, 100, OUColour.PINK);
Lollipop lo = new Lollipop(c, r);
```

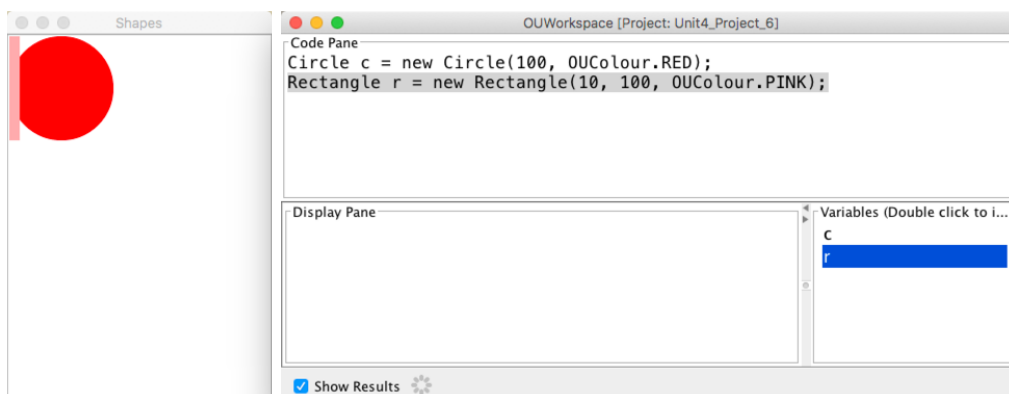
- Lollipop Example 1 (b)
- Initialise the **sweet** in **OUWorkspace**
- **Graphical Display** > **Open**

```
Circle c = new Circle(100, OUColour.RED);
```



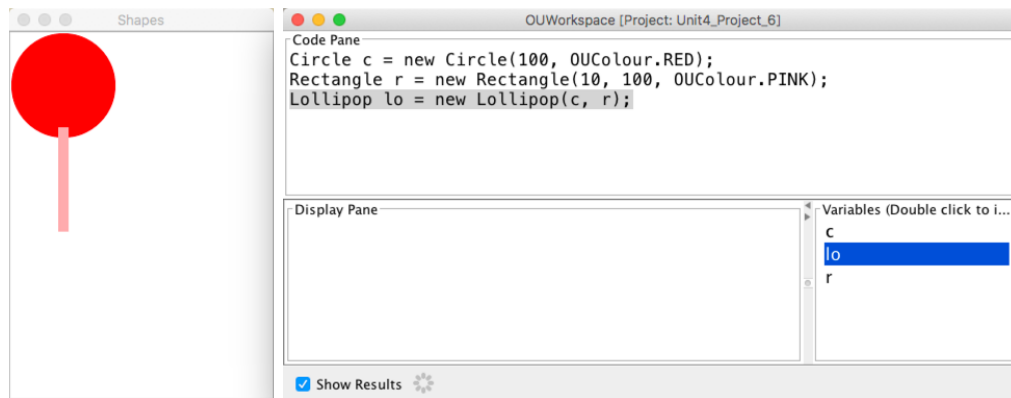
- Lollipop Example 1 (c)
- Initialise the **stick** in **OUWorkspace**
- **Graphical Display** > **Open**

```
Rectangle r = new Rectangle(10, 100, OUColour.PINK);
```



- Lollipop Example 1 (d)
- Initialise the **Lollipop** in **OUWorkspace**
- **Graphical Display** > **Open**

```
Lollipop lo = new Lollipop(c, r);
```



- **Lollipop Example 2 (a)**
- Pass in anonymous objects as actual arguments
- To make the **Lollipop** visible we have to create references to the components in **OUWorkspace**

```
Lollipop lo
= new Lollipop(new Circle(100, OUColour.RED),
               new Rectangle(10, 100, OUColour.PINK));
// lo created but not visible
Circle c = lo.getSweet();
Rectangle r = lo.getStick();
```

- **Lollipop Example 3 (a)**
- The component object is initialised by the composite object
- To make the **Lollipop** visible we have to create references to the components in **OUWorkspace**

```
Lollipop lo = new Lollipop();
// lo created but with default values
// but not yet visible
Circle c = lo.getSweet();
Rectangle r = lo.getStick();
```

4.2 Circle and Rectangle Classes

```
import ou.*;

public class Circle extends OUAnimatedObject {
    /* Instance variables */
    private OUColour colour ;
    private int xPos ;
    private int yPos ;
    private int diameter ;
```

```
public Circle() {
    super() ;
    this.colour = OUColour.BLUE ;
    this.xPos = 0 ;
    this.yPos = 0 ;
    this.diameter = 30 ;
}

public Circle(int aDiameter, OUColour aColour) {
    super() ;
    this.diameter = aDiameter ;
    this.colour = aColour ;
    this.xPos = 0 ;
    this.yPos = 0 ;
}
```

```

/* Instance methods */

public void setDiameter(int aDiameter) {
    this.diameter = aDiameter ;
    this.update() ;
}

public int getDiameter() {
    return this.diameter ;
}

public void setColour (OUColour aColour) {
    this.colour = aColour ;
    this.update() ;
}

public OUColour getColour () {
    return this.colour ;
}

```

```

public void setXPos(int x) {
    this.xPos = x ;
    this.update() ;
}

public int getXPos() {
    return this.xPos ;
}

public void setYPos(int y) {
    this.yPos = y ;
    this.update() ;
}

public int getYPos() {
    return this.yPos ;
}

```

```

public String toString() {
    return ("An_instance_of_class_"
        + this.getClass().getName()
        + ":_position_"
        + this.getXPos() + "," + this.getYPos()
        + "):_diameter_" + this.getDiameter()
        + ",_colour_" + this.getColour()) ;
}
}

```

```

import ou.*;

public class Rectangle extends OUAnimatedObject {
    /* Instance variables */

    private OUColour colour ;
    private int xPos ;
    private int yPos ;
    private int width ;
    private int height ;

```

```

public Rectangle() {
    super() ;
    this.colour = OUColour.PURPLE ;
    this.xPos = 0 ;
    this.yPos = 0 ;
    this.width = 40 ;
    this.height = 20 ;
}

public Rectangle(int aWidth, int aHeight,
    OUColour aColour) {
    super() ;
    this.width = aWidth ;
    this.height = aHeight ;
}

```

```
this.colour = aColour ;  
this.xPos = 0 ;  
this.yPos = 0 ;  
}
```

```
/* Instance methods */  
  
public void setWidth(int aWidth) {  
    this.width = aWidth ;  
    this.update() ;  
}  
  
public void setHeight(int aHeight) {  
    this.height = aHeight ;  
    this.update() ;  
}  
  
public int getWidth() {  
    return this.width ;  
}  
  
public int getHeight() {  
    return this.height ;  
}
```

```
public void setColour (OUColour aColour) {  
    this.colour = aColour ;  
    this.update() ;  
}  
  
public OUColour getColour () {  
    return this.colour ;  
}
```

```
public void setXPos(int x) {  
    this.xPos = x ;  
    this.update() ;  
}  
  
public int getXPos() {  
    return this.xPos ;  
}  
  
public void setYPos(int y) {  
    this.yPos = y ;  
    this.update() ;  
}  
  
public int getYPos() {  
    return this.yPos ;  
}
```

```
public String toString() {  
    return ("An_instance_of_class_"  
        + this.getClass().getName()  
        + ":_position_"  
        + this.getXPos() + ",_" + this.getYPos()  
        + "):_width_" + this.getWidth()  
        + ",_height_" + this.getHeight()  
        + ",_colour_" + this.getColour() ) ;  
}  
}
```

4.3 Lollipop Class

- Lollipop Class in [Lollipop.java](#)
- Instance and Class variables

```
import ou.*;

public class Lollipop {
    private Circle sweet ;
    private Rectangle stick ;
    private int licks ;

    private static final OUColour SWEET_COLOUR
        = OUColour.RED ;
    private static final OUColour STICK_COLOUR
        = new OUColour(200,200,200) ;
    private static final int SIZE = 100 ;
    //sweet diameter and stick height
    private static final int STICK_WIDTH = 10 ;
```

item Lollipop Constructor with defaults

```
public Lollipop() {

    Circle c = new Circle(SIZE, SWEET_COLOUR) ;

    c.setXPos(75) ;
    c.setYPos(75) ;

    this.sweet = c ;

    this.stick
        = new Rectangle(STICK_WIDTH
                        , SIZE
                        , STICK_COLOUR) ;

    this.attachStick() ;

    this.licks = 0 ;
}
```

- Lollipop Constructor with two arguments

```
public Lollipop(Circle aSweet, Rectangle aStick) {
    this.sweet = aSweet ;
    this.stick = aStick ;

    this.attachStick() ;

    this.licks = 0 ;
}
```

- Lollipop attachStick()

```
/**
 * Attach a stick to an existing sweet.
 * The sweet must have been initialised already.
 */

private void attachStick() {
    int radius = this.sweet.getDiameter() / 2 ;

    //move stick to the right to reach sweet centre
    this.stick.setXPos(this.sweet.getXPos()
                      + radius
                      - this.stick.getWidth() / 2) ;

    //move stick down near the bottom of the sweet
    this.stick.setYPos(this.sweet.getYPos()
                      + this.sweet.getDiameter()
                      - STICK_WIDTH) ;
}
```

- Lollipop Horizontal, Vertical Movement

```
/**
 * Method to move a lollipop horizontally.
```

```

* The direction depends on the sign of the argument.
*/
public void horiz(int xinc) {
    this.sweet.setXPos(this.sweet.getXPos() + xinc) ;
    this.stick.setXPos(this.stick.getXPos() + xinc) ;
}

/**
* Method to move a lollipop vertically.
* The direction depends on the sign of the argument.
*/
public void vert(int yinc) {
    this.sweet.setYPos(this.sweet.getYPos() + yinc) ;
    this.stick.setYPos(this.stick.getYPos() + yinc) ;
}

```

• Lollipop Eat Sweet

```

/**
* When you lick a lollipop, its sweet shrinks and
* its stick changes colour
* to get closer to the sweet's colour.
*/
public void lick() {
    if (this.sweet.getDiameter() > 1) {
        this.licks = this.licks + 1 ;

        this.sweet.setDiameter(this.sweet.getDiameter()
                               - 2) ;

        //Move the sweet so it stays on the stick
        //If we used attachStick,
        //the lollipop would move when licked.
        //This is because the circle is drawn relative
        //to the top-left corner of its bounding box.

        this.sweet.setXPos(this.sweet.getXPos() + 1) ;
        this.sweet.setYPos(this.sweet.getYPos() + 2) ;
    }
}

```

• Lollipop Eat Sweet (contd)

```

//Transfer some colour to the stick.

OUColour stickCol = this.stick.getColour();
int str = stickCol.getRed() ;
int stg = stickCol.getGreen() ;
int stb = stickCol.getBlue() ;

OUColour sweetCol = this.sweet.getColour();
int swr = sweetCol.getRed() ;
int swg = sweetCol.getGreen() ;
int swb = sweetCol.getBlue() ;

```

• Lollipop Eat Sweet (contd)

```

//Now add some colour
//from the sweet to the stick!
//Fudge factor:
//1/50th of the difference between
//the colours of the sweet and the stick
//is added to the stick colour
OUColour newCol
    = new OUColour(str + (swr - str) / 50,
                   stg + (swg - stg) / 50,
                   stb + (swb - stb) / 50) ;

this.stick.setColour(newCol) ;
} else {
    OUDialog.alert("It's all gone!") ;
}
}

```

- Lollipop getStick, getSweet, getLicks

```
/**
 * Enable workspace to see the stick part
 */
public Rectangle getStick() {
    return this.stick ;
}

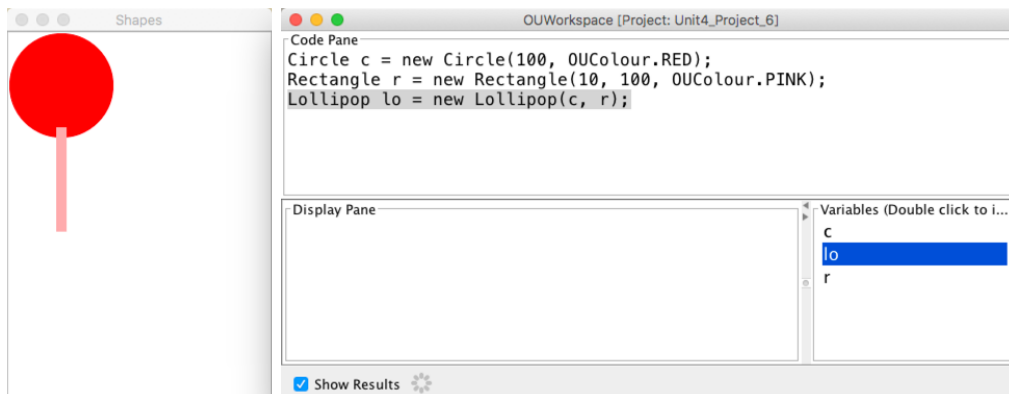
/**
 * Enable workspace to see the sweet part
 */
public Circle getSweet() {
    return this.sweet ;
}

public int getLicks() {
    return this.licks ;
}
}
```

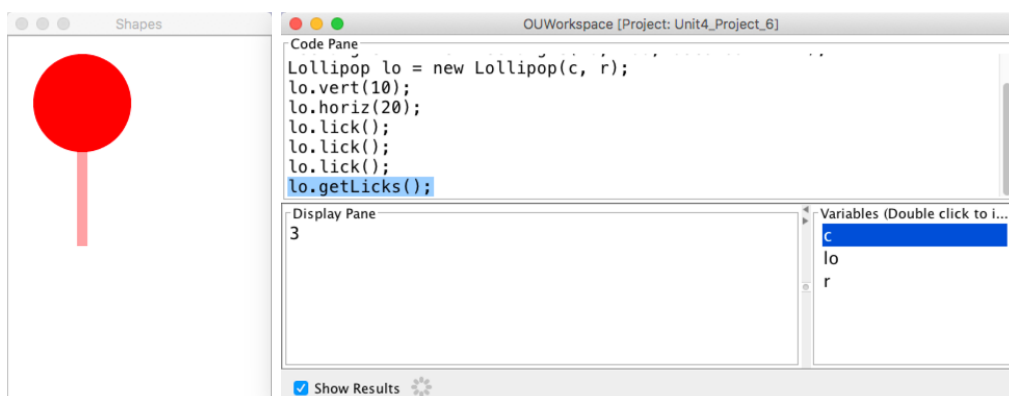
4.4 Lollipop Interaction

- Lollipop Example 1 (a) (contd)

```
lo.vert(10);
lo.horiz(20);
lo.lick();
lo.lick();
lo.lick();
lo.getLicks();
```



- Lollipop Example 1 (a) (contd)
- After the above code is executed



- Lollipop Example 1 (a) (contd)

```
lo.vert(10);  
lo.horiz(20);  
lo.lick();  
lo.lick();  
lo.lick();  
lo.getLicks();
```

- We can see the instance of the [Lollipop](#) has moved down, right
- The [sweet](#) has shrunk and the [stick](#) has changed colour (slightly)
- Notice that the [sweet](#) is now displayed overlapping the [stick](#) — what code should we have had to avoid this ?

4.5 Composition Summary

- Composition is a relationship between classes in which component objects form part of composite objects.
- Composite object classes have instance variables that are of their component object class types.
- When initialising a composite object, its component parts also need to be suitably initialised.
- Anonymous objects can be used to avoid storing unnecessary references to objects that might break encapsulation.
- Favour composition over inheritance — see [Bloch \(2017, Item18, page 87\)](#)
- Composition [has-a](#) relationship
- Inheritance [is-a](#) relationship

5 JShell

- [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)

[Go to Table of Contents](#)

6 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

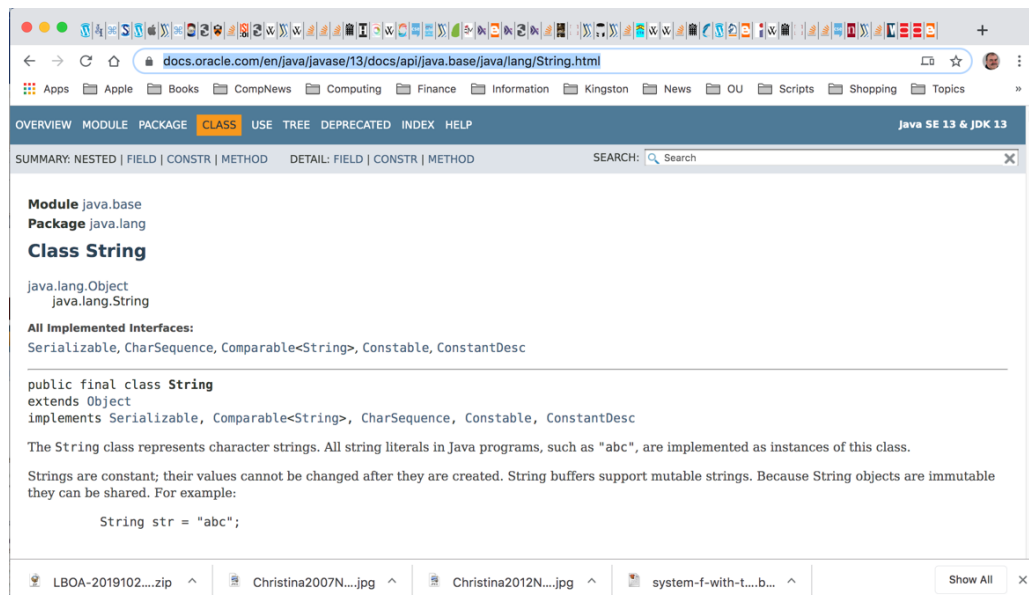
- Tutorial Online 10:00 Sunday 16 February 2025 Inheritance and Interfaces
- TMA02 Thursday 6 March 2025
- Tutorial Online 10:00 Sunday 16 March 2025 Collections
- TMA03 Thursday 8 May 2025
- Tutorial Online Sunday 11 May 2025 Exam revision

[Go to Table of Contents](#)

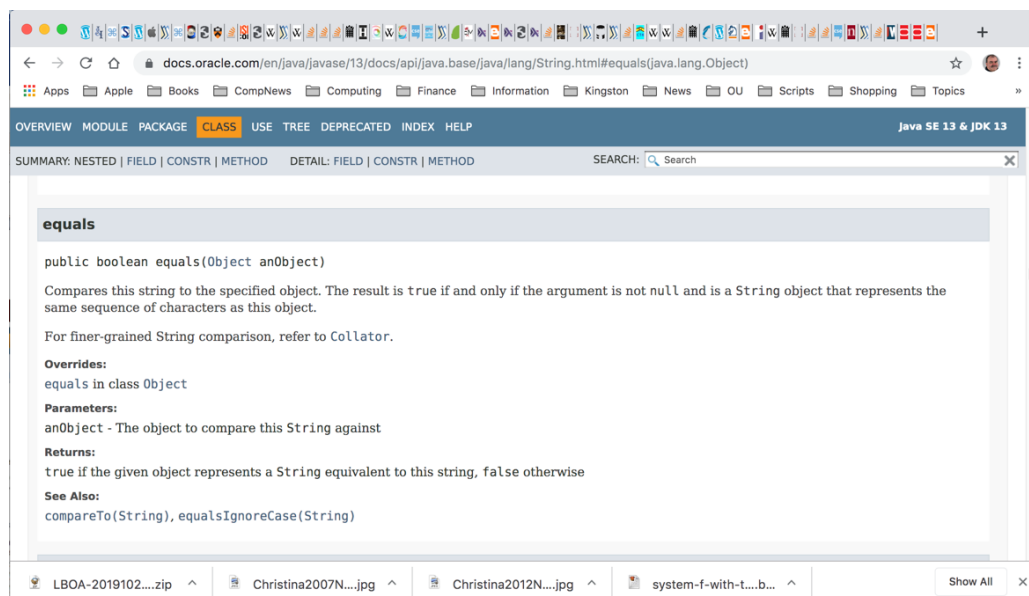
7 Web Links & References

7.1 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 ?

[Go to Table of Contents](#)

7.2 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

[Go to Table of Contents](#)

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/>. 12, 25
- Bloch, Joshua (2017). *Effective Java*. Addison-Wesley Professional, third edition. ISBN 9780134685991. 23, 26
- 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/kittylust/javanut6-examples>.
- Evans, Benjamin J and David Flanagan (2018). *Java In A Nutshell*. O'Reilly, seventh edition. ISBN 1492037257. 25
- 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/>. 25
- 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>.

[Go to Table of Contents](#)

Author Phil Molyneux Written 19 January 2025 *Printed 16th January 2025*

Subject dir: `<baseURL>/OU/Courses/Computing/M250/M250Presentations/M250Prsntn2024J`

Topic path:

`/M250Prsntn2024JTutorials/M250Tutorial20250119CompositionPrsntn2024J/M250Tutorial20250119CompositionPrsntn2024J.pdf`