

Java Classes & Statements

M250 Tutorial 03

Contents

| | | |
|----------|--|-----------|
| 1 | Agenda | 1 |
| 2 | Adobe Connect | 3 |
| 2.1 | Interface | 3 |
| 2.2 | Settings | 3 |
| 2.3 | Sharing Screen & Applications | 5 |
| 2.4 | Ending a Meeting | 5 |
| 2.5 | Invite Attendees | 5 |
| 2.6 | Layouts | 6 |
| 2.7 | Chat Pods | 7 |
| 2.8 | Web Graphics | 7 |
| 2.9 | Recordings | 8 |
| 3 | Classes: Introduction | 8 |
| 3.1 | Classes Overview and Structure | 8 |
| 3.2 | TMA01 Practice Quiz | 9 |
| 3.3 | Field, Method and Constructor Declarations | 15 |
| 4 | Statements: Summary | 16 |
| 4.1 | Statements Overview | 16 |
| 4.2 | Expression & Block Statements | 16 |
| 4.3 | Selection Statements | 17 |
| 4.4 | Iteration Statements | 18 |
| 4.4.1 | foreach statement | 20 |
| 4.5 | Returns, Exits and Exceptions | 21 |
| 4.6 | assert Statement | 24 |
| 5 | JShell | 24 |
| 6 | What Next ? | 25 |
| 7 | Web Links & References | 25 |
| 7.1 | Java Documentation | 25 |
| 7.2 | Books Phil Likes | 27 |
| | References | 28 |

1 Agenda

- Introductions
- Adobe Connect reminders
- *Adobe Connect* — if you or I get cut off, wait till we reconnect (or send you an email)
- Classes: Introduction

- Statements: Select, Iteration and others
- JShell (optional)
- Some useful Web & other references
- Time: about 1 hour
- Do ask questions or raise points.
- Slides/Notes [M250Tutorial20241117ClassesStmntsPrsntn2024J](#)

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

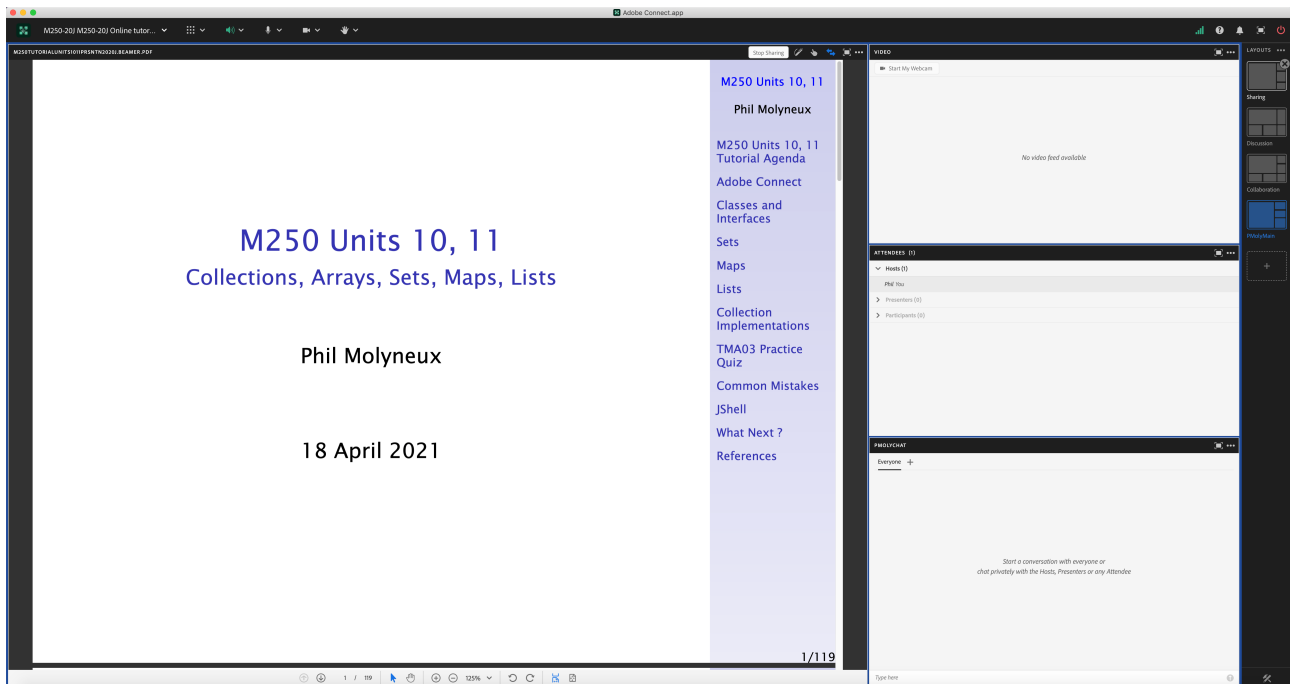
Introductions — You

- *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 ?*

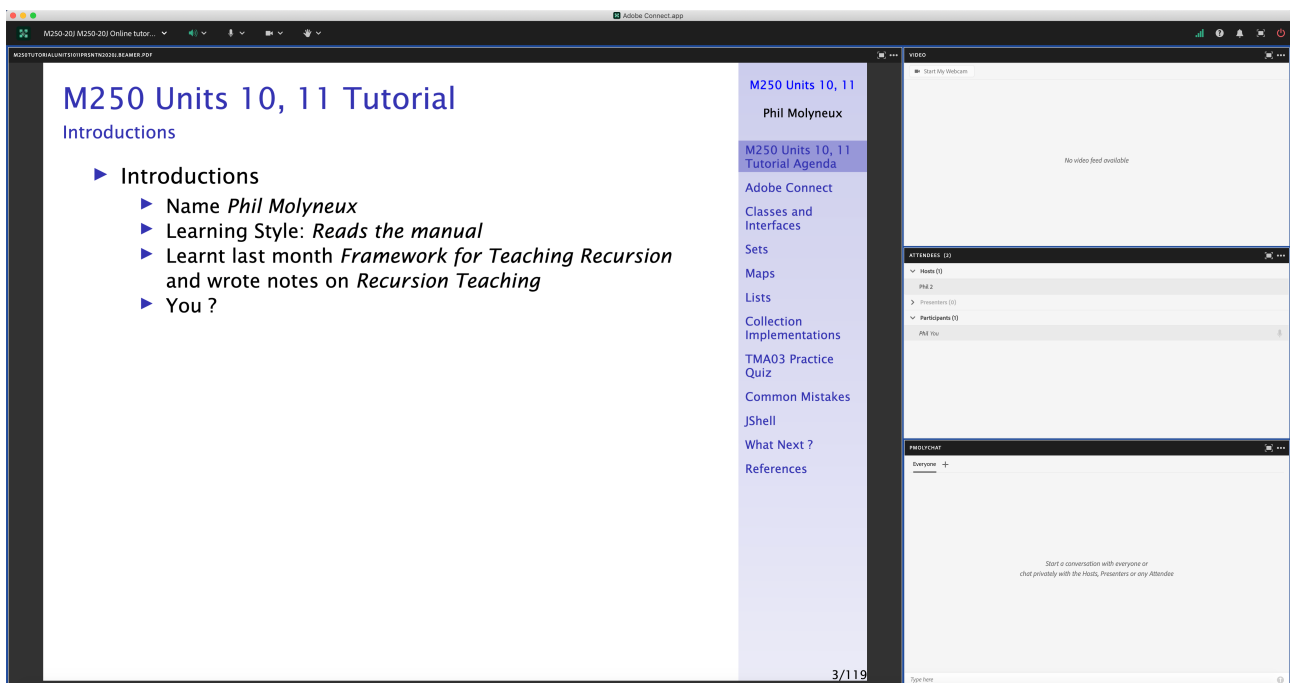
2 Adobe Connect Interface and Settings

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**

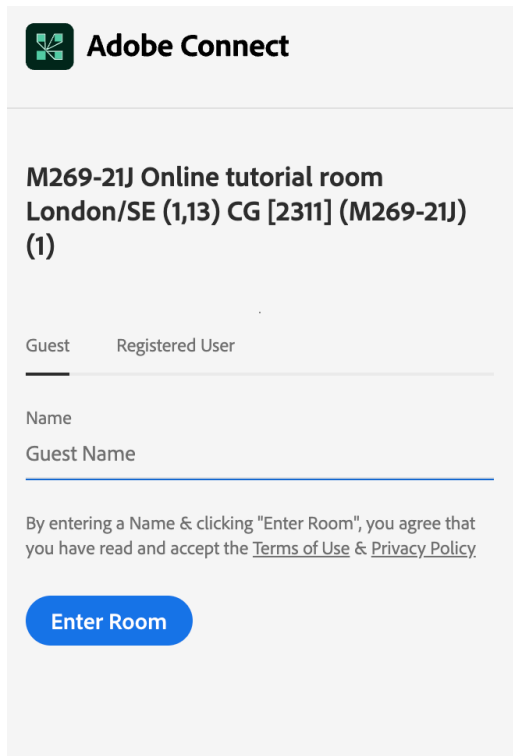
2.4 Adobe Connect — Ending a Meeting

- *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)

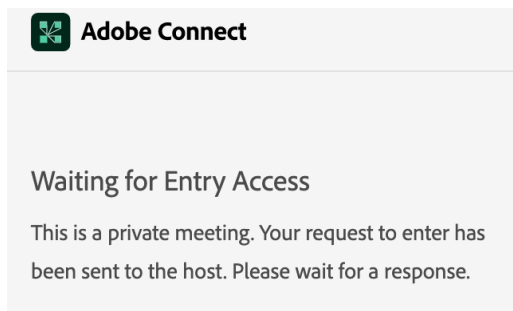
Guest Registered User

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

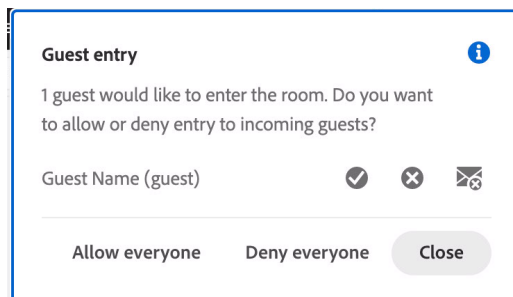


Adobe Connect

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



Guest entry

1 guest would like to enter the room. Do you want to allow or deny entry to incoming guests?

Guest Name (guest) ✓ ✕ ✉

Allow everyone Deny everyone Close

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**

[ToC](#)

3 Classes: Introduction

3.1 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**
- 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)


```

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 + ")" ;
    }
}

```

- The **Point** class is declared to have two instance fields **x** and **y**, one constructor, and two instance methods

```

jshe11> Point p1 = new Point(3,4)
p1 ==> (3,4)

jshe11> p1 = p1.move(1,2)
| Error:
| incompatible types: void cannot be converted to Point
| p1 = p1.move(1,2)
|      ^-----^

jshe11> p1.move(1,2)

jshe11> p1
p1 ==> (4,6)

```

- Notice the error message — **move()** works by side effect

[ToC](#)

3.2 TMA01 Practice Quiz

- 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 class **Edit** > **New Class...** M250Colour

```

/**
 * Write a description of class M250Colour here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class M250Colour
{

```

```

}

```

- (a)(i) Write a **private** instance field **String hexColour**
- (a)(ii) Write a constructor for **M250Colour** initialising **hexColour** to **"#000000"**

```

10 // instance variables
11 private String hexColour ;
13

```

```

14  * Constructor for objects of class M250Colour
15  */
16  public M250Colour()
17  {
18      // initialise instance variables
19      hexColour = "#000000" ;
20  }

```

- (a)(iii) Write a getter method for `hexColour`

```

23  /**
24   * Returns the value of hexColour of the receiver
25   *
26   * @return hexColour of the receiver
27   */
28  public String getHexColour(){
29      return this.hexColour ;
30  }

```

- Notice I prefer K&R layout
- (b)(i) Write a public method `isValidLength(String hStr)` to check `hStr` has 7 characters

```

48  /**
49   * Returns true if the input String has length 7
50   *
51   * @return true if the input String has length 7
52   */
53  public boolean isValidLength(String hStr){
54      final int hexStrLen = 7 ;
55      return hStr.length() == hexStrLen ;
56  }

```

- Note alternative

```

public boolean isValidLength(String hStr){
    return hStr.length() == 7;
}

```

- (b)(ii) Write `isValidFirst(String hStr)` to check the first character is '#'

```

63  public boolean isValidFirst(String hStr){
64      final char hexStrPrefix = '#' ;
65      return hStr.length() > 0
66             && (hStr.charAt(0) == hexStrPrefix) ;
67  }

```

- Alternative

```

public boolean isValidFirst(String hStr){
    return hStr.length() > 0
           && (hStr.charAt(0) == '#');
}

```

- (b)(iii) Write a method `isValidCharacters(String hStr)` to check the rest of the characters are valid hex

```

74  public boolean isValidCharacters(String hStr){
75      boolean validChr ;
76      int hStrLen = hStr.length() ;
77      char hStrCharAtI ;

79      for (int i = 1 ; i <= hStrLen - 1 ; i++) {
80          hStrCharAtI = hStr.charAt(i) ;
81          validChr
82              = (('0' <= hStrCharAtI
83                 && hStrCharAtI <= '9')
84                 || ('A' <= hStrCharAtI

```

```

85         && hStrCharAtI <= 'F')) ;
86     if (!validChr) {
87         return false ;
88     }
89 }
90 return true ;
91 }

```

- (b)(iii) What are the errors in:

```

public boolean isValidCharacters(String h){
    for (int position = 1; position < 7; position++){
        if ((h.charAt(position) >= 0
            && h.charAt(position) <= 9)
            || (h.charAt(position) >= 'A'
            && h.charAt(position) <= 'F')){
            return true;
        }
    }
    return false;
}

```

1. In the code for `isValidCharacters()` there is a `for` loop with an `if` condition — if the condition is `true` for at least one character in the 6 (six) characters then the whole lot are regarded as valid — the loop will only return `false` if the `if` condition always evaluates to `false`
2. The condition is comparing a character to the values denoted by `0` and `9` and not the characters `'0'` and `'9'` — why does this not generate an error? Because in Java characters are regarded as numeric types — so in the comparison, the character is coerced to its value as a Unicode code point — for example, `'2'` has Unicode code point 50 so is coerced to 50

- (b)(iii) Alternative

```

public boolean isValidCharactersA(String hStr)
{
    String validValues = "0123456789ABCDEF" ;
    int hStrLen = hStr.length() ;
    String hSubStr ;

    for(int index = 1; index <= hStrLen - 1; index++){
        {
            hSubStr = hStr.substring(index, index + 1) ;
            if (!validValues.contains(hSubStr)) {
                return false ;
            }
        }
    }
    return true ;
}

```

- (b)(iii) Alternative with Regular Expressions

```

public boolean isValidCharactersB(String hStr)
{
    return hStr.matches("[A-F0-9]+") ;
}

```

- `matches()` is an instance method of the class `String`
- `"[A-F0-9]+"` string representing a `regular expression`
- `.` metacharacter matches any single character
- `[A-F0-9]` is a character class matching any one of A to F or 0 to 9
- `+` matches the preceding pattern 1 or more times

- Class `Pattern` in Package `java.util.regex` describes the syntax
- (b)(iv) Write a method `isValidHexColour(String hStr)` that combines the three checks

```

98 public boolean isValidHexColour(String hStr){
99     return isValidLength(hStr)
100         && isValidFirst(hStr)
101         && isValidCharacters(hStr) ;
102 }

```

- (b)(iv) Write a method `isValidHexColour(String hStr)` using regular expressions

```

public boolean isValidHexColour(String hStr) {
    return hStr.matches("#[A-F0-9]{6}") ;
}

```

- `{6}` matches 6 copies of the preceding regular expression

```

jshe11> public boolean isValidHexColour(String hStr) {
...>     return hStr.matches("#[A-F0-9]{6}") ;
...> }
...>
| created method isValidHexColour(String)

jshe11> boolean b1 =
...>     isValidHexColour("#FFAABB")
b1 ==> true

jshe11> boolean b2 =
...>     isValidHexColour("FFAABB")
b2 ==> false

jshe11> boolean b3 =
...>     isValidHexColour("#FAB")
b3 ==> false

jshe11> boolean b4 =
...>     isValidHexColour("#FFAABBCC")
b4 ==> false

```

- (c) Write a setter method for `M250Colour` that outputs an appropriate message

```

35 public void setHexColour(String hStr){
36     boolean validStr = isValidHexColour(hStr) ;
37     String msg ;
38
39     if (validStr) {
40         msg = ("Colour_" + hStr + "_is_valid") ;
41         this.hexColour = hStr ;
42     } else {
43         msg = ("Colour_" + hStr + "_is_not_valid") ;
44     }
45     System.out.println(msg) ;
46 }

```

- Not returning a boolean by incomplete expression

```

jshe11> public boolean isValidLength(String hStr){
...>     return hStr.length();
...> }
...>
| Error:
| incompatible types: int cannot be converted to boolean
|         return hStr.length();
|         ^-----^

```

- Using assignment where you meant equality test

```

jshe11> public boolean isValidLength(String hStr){
...>     return hStr.length() = 7;

```

```

...>    }
...>
| Error:
| unexpected type
|   required: variable
|   found:    value
|       return hStr.length() = 7;
|           ^-----^

```

- Example usage in JShell
- `mClr` is a reference to an object — it is displayed in JShell in the form `<className>@<hexDigits>`
- See `Object toString()` method for an explanation

```

jshell1> M250Colour mClr = new M250Colour()
mClr ==> M250Colour@68de145

jshell1> String str1 = mClr.getHexColour()
str1 ==> "#000000"

jshell1> mClr.setHexColour("#FF0000")
Colour #FF0000 is valid

jshell1> String str2 = mClr.getHexColour()
str2 ==> "#FF0000"

jshell1> String str3 = mClr.getClass().getName()
+ '@' + Integer.toHexString(mClr.hashCode())
str3 ==> "M250Colour@68de145"

```

- The `toString()` method for class `Object` returns a string in the form `<className>@<hexDigits>`
- `getClass().getName() + '@' + Integer.toHexString(hashCode())`
- See [StackOverflow: Object reference](#)
- See `Object toString() Method in Java`
- Example usage in JShell
- Note that `isValidLength()`, `isValidFirst()`, `isValidCharacters()`, `isValidHexColour()` are *instance methods*

```

jshell1> M250Colour mClr = new M250Colour()
mClr ==> M250Colour@306a30c7

jshell1> boolean b1 = isValidLength("asdf")
| Error:
| cannot find symbol
|   symbol:   method isValidLength(java.lang.String)
|   boolean b1 = isValidLength("asdf");
|           ^-----^

jshell1> boolean b1 = mClr.isValidLength("asdf")
b1 ==> false

```

- what *might* (almost certainly) wrong with the following:

```

public boolean isValidHexColour(String h){

    if (isValidCharacters(h) == true
        && isValidFirst(h) == true
        && isValidLength(h) == true){
        return true;
    } else {
        return false;
    }
}

```

1. What happens if the string is empty ?
2. If the first character is not valid, it is not worth checking the rest
 - The following does not compile — what is the error message and why ?

```
public boolean isValidCharactersA(String h) {
    for (int i = 1; i < 8; i++) {
        if (!( (h.charAt(i) >= 48
                && h.charAt(i) <= 57)
              || (h.charAt(i) >= 65
                && h.charAt(i) <= 70))) {
            return false;
        }
        return true;
    }
}
```

- Here is the error message — but why ?

```
jshell> public boolean isValidCharactersA(String h) {
...>     for (int i = 1; i < 8; i++) {
...>         if (!( (h.charAt(i) >= 48
...>                 && h.charAt(i) <= 57)
...>               || (h.charAt(i) >= 65
...>                 && h.charAt(i) <= 70))) {
...>             return false;
...>         }
...>         return true;
...>     }
...> }
...>
| Error:
| missing return statement
| public boolean isValidCharactersA(String h) {
|                                     ^
```

- If a method is declared to have a return type, then the method must return a value — it must not be possible for execution to reach the end of a method body without executing a `return` statement (see *Java Language Specification (JLS)* Section 8.4.7 (Edition 13) *Method Body* for full details, but a bit formal)
- Why is the compiler saying *Missing return* when we can see two and the code is bound to hit one ?
- The compiler has to work for *every* syntactically valid program so it has to have some *effectively computable* rules
- We go back to *Java Language Specification (JLS)* Section 14.21 (Edition 13) *Unreachable Statements* and try and work out what the Java compiler is expected to do with *for* statements
- Essentially a *for* statement *can complete normally* if the statement is reachable and the condition is not a constant `true`
- So in terms of program flow the compiler doesn't know whether the loop terminates or not
- The analysis of the compiler is a syntactic check on where the program execution could go
- to work out whether an arbitrary block of code or statement would or would not terminate is equivalent to solving the *Halting problem* which we know is not solvable (see M269)
- So the code is missing a `return` statement *after* the `for` loop

- However, if the compiler had accepted the code, then it would still have returned `true` if the first character was valid

[ToC](#)

3.3 Field, Method and Constructor Declarations

- A field holds a value inside an object (if non-static) or a class (if static)
- A fieldDeclaration has one of the forms

```
fieldModifiers type fieldName1, fieldName2, ... ;
fieldModifiers type fieldName1 = initializer1, ... ;
```

- fieldModifiers may be a list of the modifiers `static`, `final`, `transient` and `volatile` (last two not in M250) and at most one of the access modifiers `private`, `protected`, `public`
- A field `f` in a class `C` declared `static` is a class field and can be referred to as `C.f` or `o.f` where `o` is an expression of type `C` — in the declaration of `C` it can be referred to as `f`
- A field not declared `static` is an instance field

| Accessible to | Member Visibility | | | |
|-------------------------------|-------------------|-----------|---------|---------|
| | 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 |

Table from [Evans and Flanagan \(2014, p 126\)](#)

- A method must be declared inside a class
- A methodDeclaration declaring method `m` has the form

```
methodModifiers returnType m(formalList) throwsClause
methodBody
```

- The formalList is a comma-separated list of zero or more *formal parameter declarations*, of one of the forms

```
parameterModifier type parameterName
parameterModifier type... parameterName
```

- The parameterModifier may be `final` meaning that the parameter cannot be modified inside the method
- The second form of parameter declaration can only appear last and declares a parameter array (TODO: check if this is used in M250)
- The purpose of a constructor in class `C` is to initialize new objects (instances) of the class
- A constructorDeclaration in class `C` has the form

```
constructorModifiers C(formalList) throwsClause
constructorBody
```

- The `constructorModifiers` may be a list of at most one of `private`, `protected` and `public`
- Constructors may be overloaded in the same way as methods
- The constructor signature (a list of parameter types in `formalList`) is used to distinguish constructors in the same class
- A constructor may call another overloaded constructor in the same class using the syntax

```
this(actualList)
```

- A class that does not explicitly declare a constructor implicitly declares a public, argumentless *default constructor* whose only (implicit) action is to call the superclass constructor

```
public C() { super() ; }
```

- A class `C` may be declared a *subclass* of class `B` by an `extendsClause` of the form

```
class C extends B {...}
```

- The very first action of a constructor in `C` may be an explicit call to a constructor in superclass `B`
- If a constructor `C(...)` does not explicitly call `super(...)` as its first action, then it implicitly calls the argumentless default constructor `B()` as its first action, as if by `super()`

[ToC](#)

4 Statements: Summary

4.1 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) {}`)

[ToC](#)

4.2 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 *{ }*

[ToC](#)

4.3 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 **branch** is a sequence of statements, usually terminated by **break** or **return** (if inside a method or constructor) or **continue** (inside a loop).
- If a **branch** is not exited by **break**, **return**, or **continue**, then execution continues with the next **branch** in the switch regardless of the **case** clauses, until a **branch** exits or the switch ends
- (not used in M250)

[ToC](#)

4.4 Iteration Statements

- 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() ;
...> }
...>
*
**
***
****
```

- 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](#)

4.4.1 foreach statement

```

for (tx x : expression)
    body

```

- The *expression* must have type `Iterable<t>` where `t` is a subtype of type `tx`
- Iterators obtained from *expression* will generate elements that can be assigned to `x`
- *expression* must be a statement

```

int[] primes = new int[] {2,3,5,7,11,13,17,19,23,29} ;
for (int n : primes) {
    System.out.println(n) ;
}

```

```

int[] iarr = new int[] { 2, 3, 5, 7, 11 } ;
int sum = 0 ;
for (int p : iarr) {
    sum = sum + p ;
}
System.out.println("sum_=" + sum) ;

```

```

jshe11> System.out.println("sum_=" + sum) ;
sum = 28

```

- Note that `print` and `println` convert a value to textual representation and outputs it to a `PrintWriter` or `PrintStream` (`System.out` is a `PrintStream`)
- Python programmers may be tempted by the following but see the error

```
jshe11> System.out.println("sum_=_", sum) ;
| Error:
| no suitable method found for println(java.lang.String,int)
|     method java.io.PrintStream.println() is not applicable
|     (actual and formal argument lists differ in length)
//and lots of other error messages
```

- Explicitly going through an iterable using **for**
- Example 79 from Sestoft (2016, page 57)

```
Iterable<Integer> ible = fromTo(13, 17);
for (Iterator<Integer> iter = ible.iterator(); iter.hasNext(); /* none */) {
    int i = iter.next();
    System.out.println(i);
}
```

- Method **fromTo** generates an Iterable collection

```
jshe11> Iterable<Integer> ible = fromTo(13, 17);
...> for (Iterator<Integer> iter = ible.iterator(); iter.hasNext(); /* none */) {
...>     int i = iter.next();
...>     System.out.println(i);
...> }
...>
ible ==> 1FromToIterable@7aec35a
13
14
15
16
17
```

- Method **fromTo** generates an Iterable collection
- Example 143 from Sestoft (2016, page 113)

```
public Iterable<Integer> fromTo(final int m, final int n) {
    class FromToIterator implements Iterator<Integer> {
        private int i = m;
        public boolean hasNext() { return i <= n; }
        public Integer next() {
            if (i <= n)
                return i++;
            else
                throw new NoSuchElementException();
        }
        public void remove() { throw new UnsupportedOperationException(); }
    }
    class FromToIterable implements Iterable<Integer> {
        public Iterator<Integer> iterator() {
            return new FromToIterator();
        }
    }
    return new FromToIterable();
}
```

[ToC](#)

4.5 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)
```

[ToC](#)

4.6 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 ;
```

[ToC](#)

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)

[ToC](#)

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](#))

Chps 1–4, TMA01

- Chps 4–5, Iteration, collections; Functional Java (optional)
- Tutorial 03 10:00 Sunday 16 November 2025 online
- TMA01 Thursday 11 December 2025
- Chps 6,7 Arrays
- Tutorial 04 10:00 Sunday 18 January 2026 online
- Chps 8–10 Inheritance
- Tutorial 05 10:00 Sunday 15 February 2026 online
- TMA02 Thursday 5 March 2026

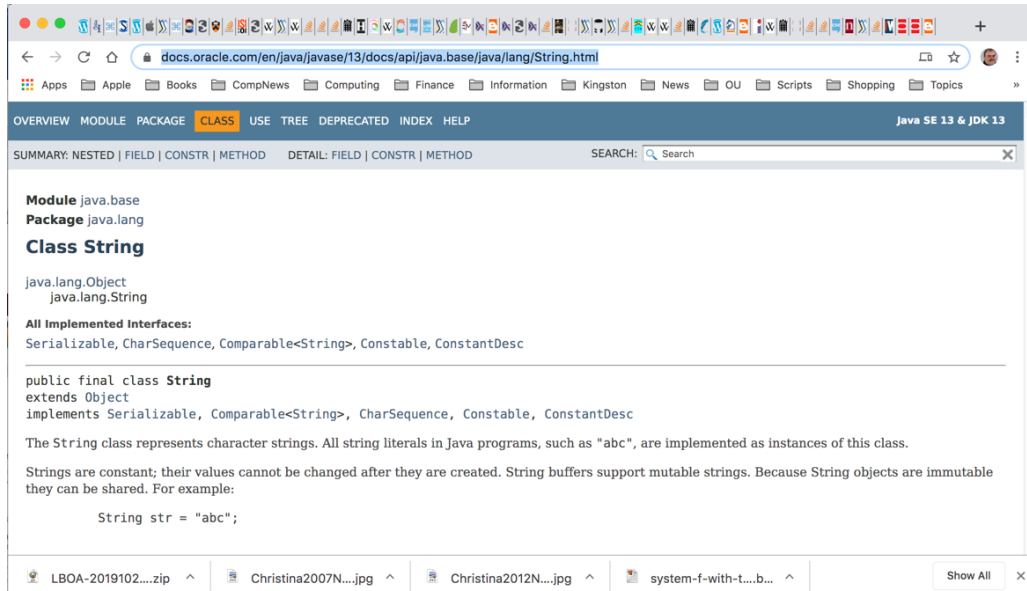
[ToC](#)

7 Web Links & References

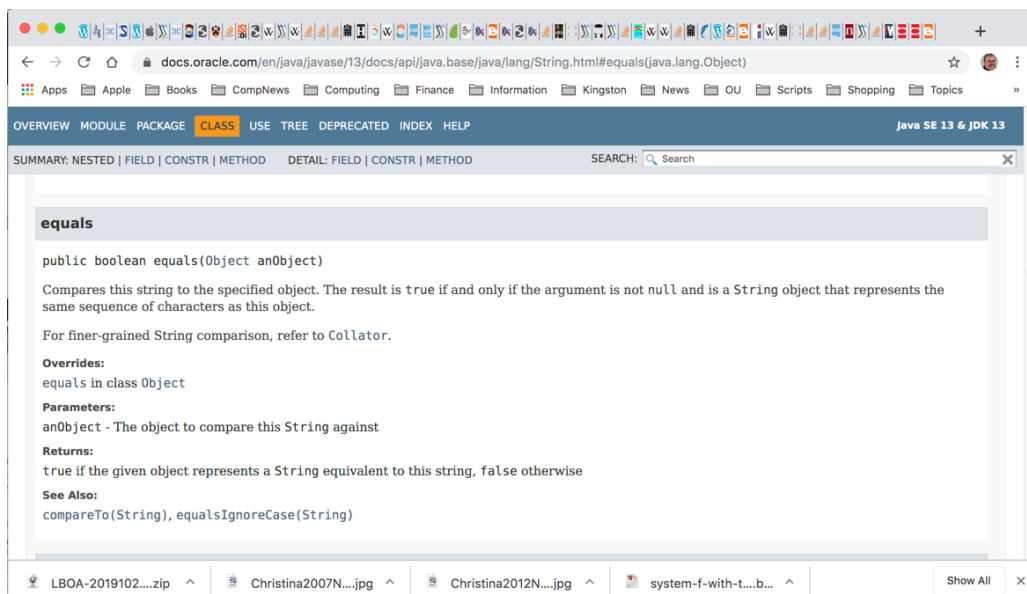
7.1 Java Documentation

- [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 ?

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

[ToC](#)

References

- Barnes, David J. and Michael Kolling (2009). *Objects First with Java*. Pearson Education, fourth edition. ISBN 0-13-606086-2. URL <http://www.bluej.org/objects-first/>.
- Barnes, David J. and Michael Kolling (2011). *Objects First with Java*. Pearson Education, fifth edition. ISBN 0132835541. URL <http://www.bluej.org/objects-first/>.
- Barnes, David J. and Michael Kölling (2016). *Objects First with Java*. Pearson, sixth edition. ISBN 1292159049. URL <http://www.bluej.org/objects-first/>. 27
- Bloch, Joshua (2017). *Effective Java*. Addison-Wesley Professional, third edition. ISBN 9780134685991. 27
- 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/kittylst/javanut6-examples>. 15
- Evans, Benjamin J and David Flanagan (2018). *Java In A Nutshell*. O'Reilly, seventh edition. ISBN 1492037257. 27
- 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/>. 27

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>.

[ToC](#)