

# Java: Collections, Arrays, Sets, Maps, Lists

## M250 Tutorial 06

### Contents

|           |                                         |           |
|-----------|-----------------------------------------|-----------|
| <b>1</b>  | <b>Tutorial Agenda</b>                  | <b>2</b>  |
| <b>2</b>  | <b>Adobe Connect</b>                    | <b>3</b>  |
| 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 . . . . .                     | 8         |
| 2.8       | Web Graphics . . . . .                  | 8         |
| 2.9       | Recordings . . . . .                    | 8         |
| <b>3</b>  | <b>Classes and Interfaces</b>           | <b>9</b>  |
| <b>4</b>  | <b>Sets</b>                             | <b>9</b>  |
| <b>5</b>  | <b>Maps</b>                             | <b>10</b> |
| <b>6</b>  | <b>Lists</b>                            | <b>11</b> |
| <b>7</b>  | <b>Collection Implementations</b>       | <b>11</b> |
| <b>8</b>  | <b>TMA03 Practice Quiz</b>              | <b>12</b> |
| 8.1       | Information . . . . .                   | 12        |
| 8.2       | Question 1 . . . . .                    | 13        |
| 8.2.1     | Q 1 Sample Usage . . . . .              | 17        |
| 8.3       | Question 2 . . . . .                    | 21        |
| 8.3.1     | Q 2 Sample Usage . . . . .              | 25        |
| 8.4       | Question 3 . . . . .                    | 27        |
| 8.4.1     | Q 3 Sample Usage . . . . .              | 30        |
| <b>9</b>  | <b>Common Mistakes</b>                  | <b>32</b> |
| 9.1       | Array to List . . . . .                 | 32        |
| 9.2       | Using a TreeSet . . . . .               | 33        |
| 9.3       | Remove List Elements . . . . .          | 34        |
| <b>10</b> | <b>JShell</b>                           | <b>35</b> |
| <b>11</b> | <b>What Next ?</b>                      | <b>35</b> |
| <b>12</b> | <b>References</b>                       | <b>36</b> |
| 12.1      | Java Documentation . . . . .            | 36        |
| 12.2      | Books Phil Likes . . . . .              | 37        |
|           | References . . . . .                    | 38        |

# 1 Java: Collections, Arrays, Sets, Maps, Lists: Tutorial Agenda

- Introductions
- Adobe Connect reminders
- *Adobe Connect* — if you or I get cut off, wait till we reconnect (or send you an email)
- Collections framework
- Arrays
- Sets, Maps
- Lists
- Review of TMA03 Practice Quiz
- Common Mistakes
- JShell (optional)
- Some useful Web & other references
- Time: about 1 to 2 hours
- Do ask questions or raise points.
- Slides/Notes [M250Tutorial06Collections](#)

## 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<sup>A</sup>T<sub>E</sub>X](#) — 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* ?

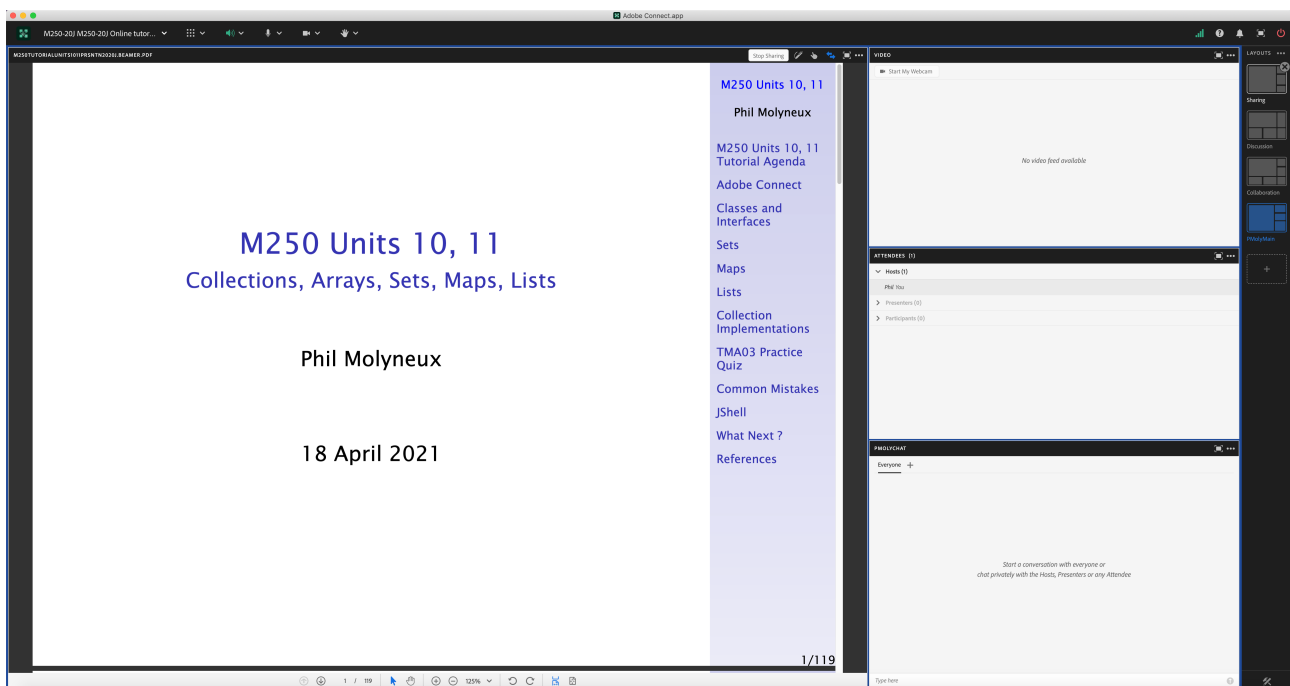
- Favourite *text editor* or *integrated development environment (IDE)*
- List of text editors, Comparison of text editors and Comparison of integrated development environments
- Other OU courses ?
- Anything else ?



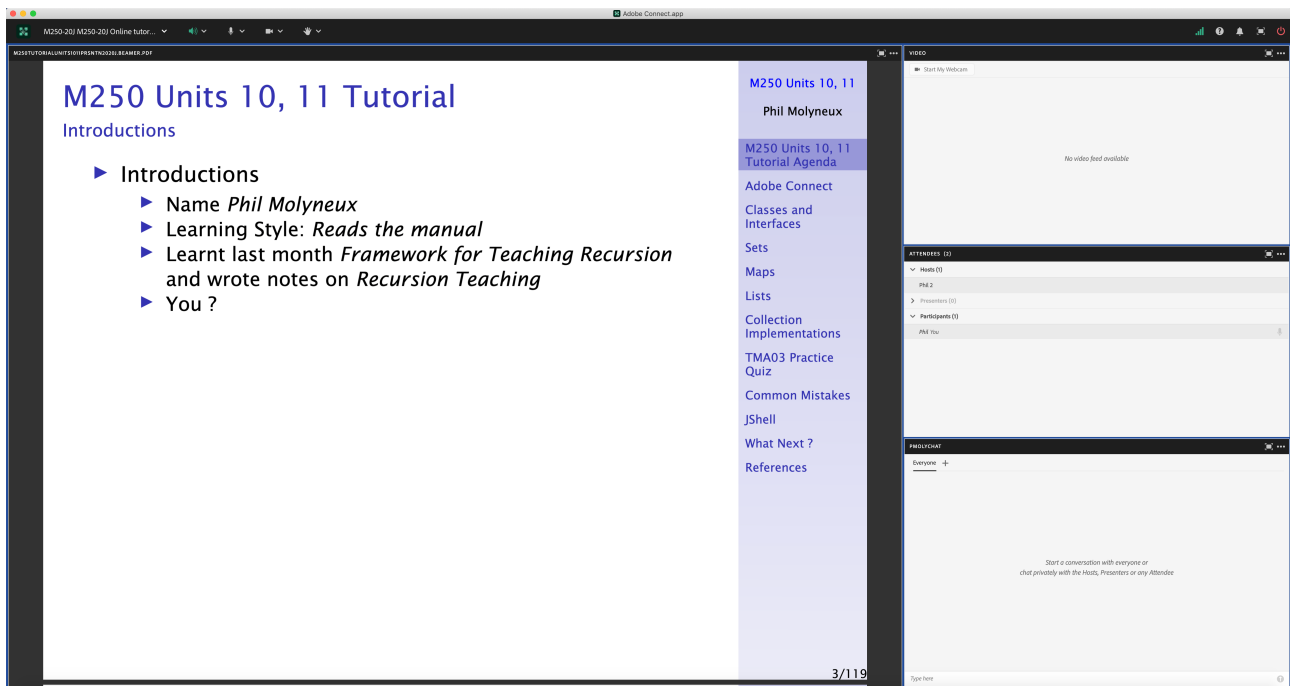
## 2 Adobe Connect Interface and Settings

### 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**

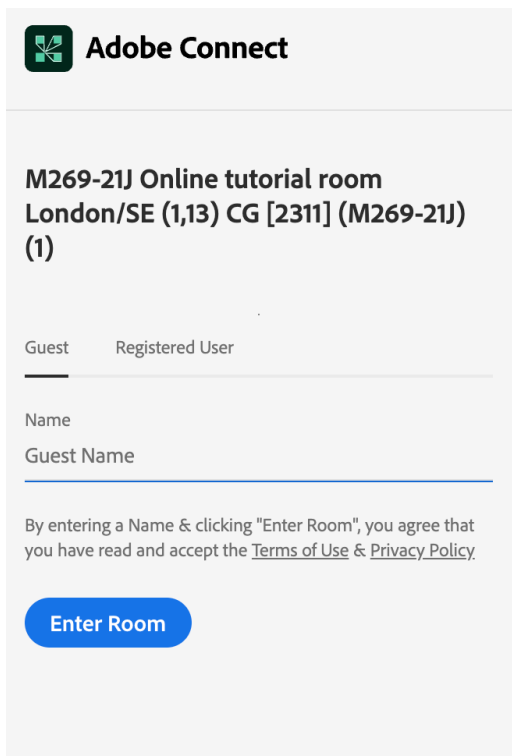
## 2.4 Adobe Connect — Ending a Meeting

- *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)**

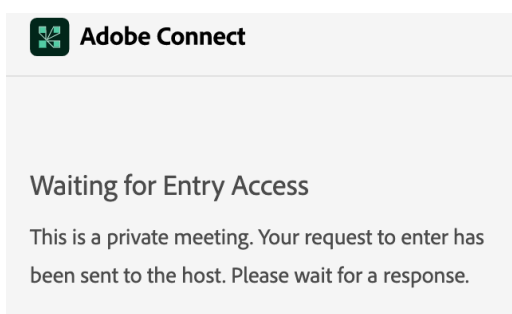
Guest   Registered User

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

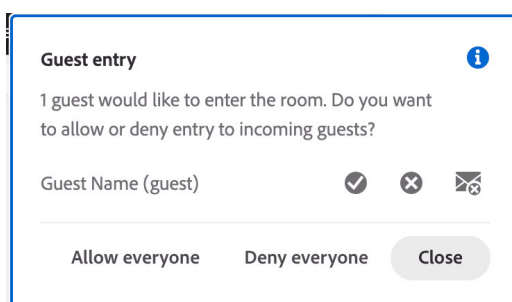


**Adobe Connect**

**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 and Interfaces

- Classes and Interfaces were introduced in Unit 6 and there is a reminder on page 105 of Unit 10
- It is worth discussing of the roles of Classes and Interfaces — some students will be finding the detail gets in the way of some broad concepts
- Focus on the *Java type system*
- Question: How does a Class define a type ?
- Question: How does an Interface define a type ?
- Page 105 of Unit 10 gives the view of this
- **Class**: tells you how to construct a thing of a new type
- **Interface**: to be of this type you have to implement the specified actions
- There are parallels in other languages but you have to be careful of the use of terminology here

[ToC](#)

### 4 Sets

- A **Set** is a collection with no order, no duplicates, no index and varying size
- Discuss a number of examples similar to Unit 10
- The examples below use **JShell**, a **Read-Eval-Print loop (REPL)** tool available for Java
- **Java Shell User's Guide** describes its usage
- Note: **JShell** is not directly in M250 (it arrived in JDK 9) but for demonstrations students only need to know:
  - Java statements and class definitions can be executed at the prompt *jshell*> and continuation prompt ...>
  - The result is reported on the line following

- Most common libraries are automatically imported

```
Set<String> keywordSet = new HashSet<String>();
```

```
jshell> String[] pArray =
...> { "d", "a", "c", "a", "b", "a" }
pArray ==> String[6] { "d", "a", "c", "a", "b", "a" }

jshell> Set<String> qSet =
...> new TreeSet<String>(Arrays.asList(pArray))
qSet ==> [a, b, c, d]

jshell> boolean b = qSet.add("bb")
b ==> true

jshell> qSet
qSet ==> [a, b, bb, c, d]
```

[ToC](#)

## 5 Maps

- Mapping keys to values — sometimes called *Dictionaries*
- Exercise: mapping file names to content types — what part of the filename gives us the information?
- Mapping file extensions to file types

```
jshell> Map<String, String> fileTypeMap =
...> new HashMap<String, String>()
fileTypeMap ==> {}

jshell> String retVal = fileTypeMap.put("java", "Java")
retVal ==> null

jshell> String retVal = fileTypeMap.put("py", "Python")
retVal ==> null

jshell> String retVal = fileTypeMap.put("lhs", "Haskell")
retVal ==> null

jshell> String retVal = fileTypeMap.put("hs", "Haskell")
retVal ==> null
```

- Repeating a key in `put` overwrites an entry but reports the previous value

```
jshell> String retVal
      = fileTypeMap.put("lhs", "Literate_Haskell")
retVal ==> "Haskell"

jshell> fileTypeMap
fileTypeMap ==>
{java=Java, lhs=Literate Haskell,
 py=Python, hs=Haskell}

jshell> Set<String> fTypes =
...> fileTypeMap.keySet()
fTypes ==> [java, lhs, py, hs]
```

- Possible further discussion of citation keys for bibliographies — see [JabRef](#) (implemented in Java) or [BibDesk](#)

[ToC](#)

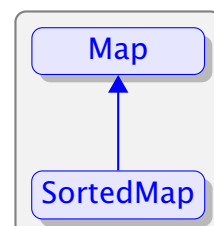
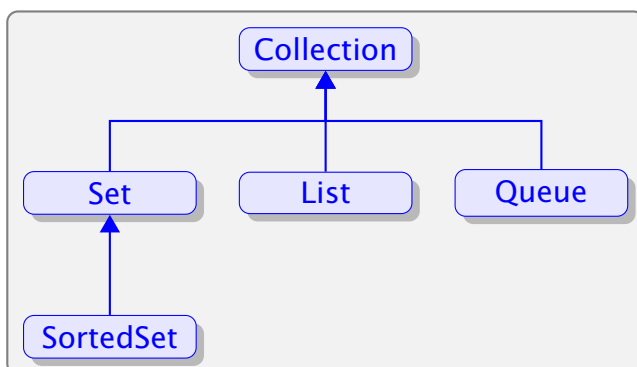
## 6 Lists

- Lists implement the idea of a sequence of items
- Dynamic size — items can be added, removed or modified (though you can have lists of fixed size)
- Ordered and indexed by integers (starting at 0)
- Duplicates allowed
- Summary in *M250 Exam Handbook* page 24
- [List](#) interface implemented by [ArrayList](#) and [LinkedList](#)
- Covered in Unit 11 — would not have time for more than a brief mention in this session

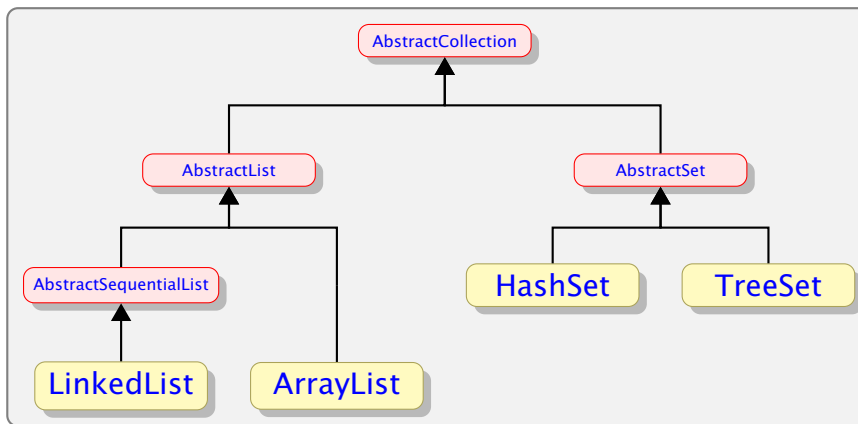
[ToC](#)

## 7 Collection Implementations

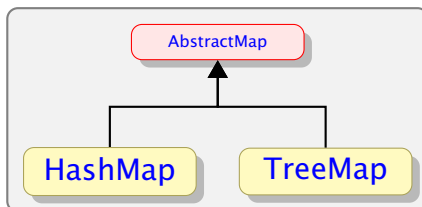
- This section discusses the hierarchy of interfaces, abstract classes and concrete classes that make up the *Collections Framework*
- It follows Unit 10 with some similar exercises
- Classes that implement the collection interfaces typically have names in the form of *<Implementation-style><Interface>*
- Note that the diagrams may have some conventions that I may have missed — see, for example, [UML Class and Object Diagrams Overview](#)
- M250 follows some conventions from [Javadoc](#) — see [Javadoc Guide](#)



- The blue rectangles denote interfaces and subinterfaces.
- **Exercise** Using *M250 Exam Handbook* to find some details of [Set](#) (page 26) and [SortedSet](#) (page 27)



- The red rectangles denote abstract classes which implement various interfaces
- Yellow rectangles denote concrete classes extending abstract classes and (possibly) implementing interfaces
- Note that [TreeSet](#) also implements the [SortedSet](#) interface



- Note that [TreeMap](#) also implements the [SortedMap](#) interface
- **Exercise** Using *M250 Exam Handbook* to find some details of [HashMap](#) (page 30) and [TreeMap](#) (page 31)
- Timing: **10 mins**
- Note: I would prefer to have a diagram with interface, abstract classes and concrete classes all in one diagram but this would take some time to produce — see, for example, Sestoft, *Java Precisely* (2016) section 22, page 102

[ToC](#)

## 8 TMA03 Practice Quiz

### 8.1 Information

- The quiz is intended to help with TMA03 Q2 and exam Q3
- There are three questions about the usage of [List](#), [Map](#) and [Set](#)
- The code can be checked with *Precheck* and *Check*
- *Precheck* checks that the code compiles and is not missing some features
- *Check* checks the functionality
- You are advised to develop your code in BlueJ first
- The quiz can be repeated any number of times to improve the score

## 8.2 Question 1

- This CodeRunner question concerns an animal shelter that keeps records about animals brought to the shelter.
- We will model this using two classes, `Animal` and `Shelter`, and we have provided incomplete code for the Shelter class in the answer box.
- Your task is to complete the `Shelter` class. You do not need to add anything to the `Animal` class.
- Before you start, (1) read over the provided `Animal` class.
- **Animal Class**

```
147 class Animal {
148     // instance variables
149     private int week ; // 1 to 52
150     private String kind ;
151     private String name ;
152     private String description ;
153
154     /**
155      * Constructor for objects of class Animal
156      */
157     public Animal(int aWeek
158                  ,String aKind
159                  ,String aName
160                  ,String aDescription) {
161         this.week = aWeek ;
162         this.kind = aKind ;
163         this.name = aName ;
164         this.description = aDescription ;
165     }
```

```
230 }
```

```
167 /**
168  * getter for week
169  */
170 public int getWeek() {
171     return this.week ;
172 }
173
174 /**
175  * getter for type
176  */
177 public String getKind() {
178     return this.kind ;
179 }
180
181 /**
182  * getter for name
183  */
184 public String getName() {
185     return this.name ;
186 }
```

```
188 /**
189  * getter for description
190  */
191 public String getDescription() {
192     return this.description ;
193 }
194
195 /**
196  * setter for description
```

```

197  * Note - concatenates new description to end of existing one
198  */
199  public void setDescription(String moreDescription) {
200      this.description = this.description
201          + " " + moreDescription ;
202  }

```

```

204  /**
205   * A simple equals method
206   */
207  @Override
208  public boolean equals(Object o) {
209      Animal anm1 = (Animal) o ;
210      return ( getKind().equals(anm1.getKind())
211          && getWeek() == anm1.getWeek()
212          && getName().equals(anm1.getName())) ;
213  }

215  /**
216   * hashCode
217   */
218  @Override
219  public int hashCode() {
220      return getName().length();
221  }

223  @Override
224  public String toString() {
225      return (this.getKind()
226          + " " + this.getName()
227          + ":" + this.getDescription()) ;
228  }

```

- (2) Note that in this question an instance of `Shelter` holds data about a number of `Animal` objects in an `ArrayList`. For example, the `ArrayList` might contain the following data:

| Index | Value                                                                                             |
|-------|---------------------------------------------------------------------------------------------------|
| 1     | <code>Animal</code> object with week 50, kind "cat", name "Billy", description "Black diabetic"   |
| 2     | <code>Animal</code> object with week 50, kind "cat", name "Zoe", description "Black and white"    |
| 3     | <code>Animal</code> object with week 51, kind "dog", name "Rover", description "mongrel"          |
| 4     | <code>Animal</code> object with week 52, kind "tortoise", name "Speedy", description "Horsefield" |

- Note that the `ArrayList` stores entries in the order in which they were added.
- (a) (i) Declare an additional private instance variable `animals` in the `Shelter` class, capable of referencing an `ArrayList` whose values are `Animal` objects, as in the example table above.
- Add a standard getter method for the `animals` collection.
- (ii) The class `Shelter` should now have these instance variables:
    - `animals`, which you added in part (i) above
    - `currentWeek` of type `int`, which is the current week of the year, and will be used when calculating how long an animal has been in the shelter.
  - Amend the provided zero-argument constructor for `Shelter` so that when a new instance of `Shelter` is created `animals` is assigned a suitable empty `ArrayList` object and `currentWeek` is set to 1.

```

9 class Shelter {
10     private int currentWeek ;
11     private List<Animal> animals ;

13     public Shelter () {
14         super() ;
15         this.animals = new ArrayList<Animal>() ;
16         this.currentWeek = 1 ;
17     }

```

- (a) (iii) Complete the skeleton instance method `addAnimal()` for the class `Shelter`, with the header

```

public void addAnimal(String aKind
                     ,String aName
                     ,String aDescription)

```

- The method should use its three arguments to create an instance of `Animal`, using the value of `currentWeek` for its `week`, then add the `Animal` to the `animals` `ArrayList`.

```

27 public void addAnimal(String aKind
28                     ,String aName
29                     ,String aDescription) {
30     Animal anm1
31     = new Animal(this.currentWeek
32                 , aKind, aName, aDescription) ;
33     this.animals.add(anm1) ;
34 }

```

- (b) (i) Write a public instance method with the signature `inTheLastMonth(Animal)` for the `Shelter` class.

- The method should determine whether the `Animal` received as an argument has been brought into the shelter in the last month — that is, if the number of weeks between `currentWeek` and the animal's `week` is 4 or less

Note, however, that when `currentWeek` gets to 52, then the next `currentWeek`'s value is 1.

You'll need to watch out for this when calculating how many weeks it has been since the animal arrived. For two examples, see below.

| <code>currentWeek</code> in <code>Shelter</code> | Animal's <code>week</code> value | Weeks since arrived |
|--------------------------------------------------|----------------------------------|---------------------|
| 52                                               | 50                               | 2                   |
| 1                                                | 50                               | 3                   |

- The method should return `true` if it has been less than or equal to four weeks since the animal arrived; otherwise it should return `false`. You can assume that an animal is never in the shelter for more than a year.

```

36 public boolean inTheLastMonth(Animal anm1) {
37     int currWeek = this.currentWeek ;
38     int animalWeek = anm1.getWeek() ;
39     return ( ( currWeek - animalWeek <= 4
40             && currWeek - animalWeek >= 0)
41             || (currWeek - animalWeek + 52 <= 4) ) ;
42 }

```

- (b) (ii) Write a public instance method `showRecentAnimals()` for the `Shelter` class.

- For each `Animal` in the collection referenced by `animals`, if the animal was brought into the shelter in the last month (it has been less than or equal to four weeks since the animal arrived) then data about that animal should be printed to the standard output, with the details of each such animal on a separate line.
- If no animals have been brought in in the last month then *No recent animals* should be printed.
- For example, using the data from the table in part (a), the output from `showRecentAnimals()` when `currentWeek` is 3 should be:

```
dog Rover:mongrel
tortoise Speedy:Horsefield
```

```
44 public void showRecentAnimals() {
45     boolean noRecentAnimals = true ;

46     for (Animal anm1 : this.animals) {
47         if (inTheLastMonth(anm1)) {
48             System.out.println(anm1.getKind()
49                               + " " + anm1.getName()
50                               + ":" + anm1.getDescription()) ;
51             noRecentAnimals = false ;
52         }
53     }

54     if (noRecentAnimals) {
55         System.out.println("No recent animals") ;
56     }
57 }
58
59 }
```

(c) Write a public method `homed()` in the `Shelter` class with the header

```
public boolean homed(int aWeek
                    ,String aKind
                    ,String aName)
```

- This method should determine whether or not the `ArrayList` referenced by `animals` contains an `Animal` with a `week`, `kind` and `name` matching the method arguments.
- If there is such an animal, it should be removed from the list and `true` should be returned. Otherwise `false` should be returned.
- This version uses `List` operations

```
61 public boolean homed(int aWeek
62                    ,String aKind
63                    ,String aName) {
64     Animal anm1 = new Animal(aWeek,aKind,aName,"No_Desc") ;
65     boolean anm1IsIn
66     = this.animals.contains(anm1) ;
67     if (anm1IsIn) {
68         this.animals.remove(anm1) ;
69     }
70     return anm1IsIn ;
71 }
```

- `remove()` here is from the `Collections` Interface
- It takes an object as argument and removes a single instance of the element, if present
- It returns `true` if it succeeds
- It uses `equals()` to check elements
- We could have made the above method shorter — how ?

- This version uses an [Iterator](#)

```

73 public boolean homedA(int aWeek
74                     ,String aKind
75                     ,String aName) {
76
77     Iterator<Animal> animalIter = this.animals.iterator() ;
78     while (animalIter.hasNext()) {
79         Animal anml = animalIter.next() ;
80         if ( anml.getWeek() == aWeek
81             && anml.getKind() == aKind
82             && anml.getName() == aName) {
83             animalIter.remove() ;
84             return true ;
85         }
86     }
87     return false ;
88 }

```

- The code below may work but it is unpredictable
- See [Iterating through a Collection, avoiding ConcurrentModificationException when removing objects in a loop](#)
- See [java.base > java.util > Class ConcurrentModificationException](#)

```

90 public boolean homedB(int aWeek
91                     ,String aKind
92                     ,String aName) {
93
94     for (Animal anml : this.animals) {
95         if ( anml.getWeek() == aWeek
96             && anml.getKind() == aKind
97             && anml.getName() == aName) {
98             this.animals.remove(anml) ;
99             return true ;
100         }
101     }
102     return false ;
103 }

```

### 8.2.1 Q 1 Sample Usage

- The code is in [M250TMA03PracticeQuizSolnA.java](#) and we use [jShell](#) to do evaluations — see [Java Shell User's Guide](#)
- We have several classes in one file — see [Java: Multiple class declarations in one file](#)

```

import java.util.* ;

class M250TMA03PracticeQuizSolnA {
    public static void main(String[] args) {
        // further code here or in Utilities
    }
}

```

```
class Shelter {
```

```
class Animal {
```

```
class Utilities {
```

- Open the Java file at the jShell prompt — it will compile it

```

jshell> /open M250TMA03PracticeQuizSolnA.java
jshell> Shelter shelter01 = new Shelter()
shelter01 ==> Shelter@46f7f36a

```

```

jsshell> Animal anm101 = new Animal(50,"cat","Billy","No_Description")
anm101 ==> Animal@5

jsshell> Animal anm102 = new Animal(51,"dog","Rover","No_Description")
anm102 ==> Animal@5

jsshell> Animal anm103 = new Animal(1,"rabbit","Roger","No_Description")
anm103 ==> Animal@5

jsshell> shelter01.populate()

jsshell> shelter01.showRecentAnimals()
dog Rover:mongrel
tortoise Speedy:Horsefield

jsshell> /exit
| Goodbye

```

- **Shelter** has **populate()** and **toString()** definitions to facilitate sample usage

```

108 public void populate() {
109     this.currentWeek = 50 ;
110     this.addAnimal("cat","Billy","Black_diabetic") ;
111     this.addAnimal("cat","Zoe","Black_and_white") ;
112     this.currentWeek = 51 ;
113     this.addAnimal("dog","Rover","mongrel") ;
114     this.currentWeek = 52 ;
115     this.addAnimal("tortoise","Speedy","Horsefield") ;
116     this.currentWeek = 3 ;
117 }

119 @Override
120 public String toString() {
121     String outStr = "" ;
122     outStr = outStr + "currentWeek_is_" + currentWeek ;
123     for (Animal anm1 : this.animals) {
124         outStr = outStr + "\n" + anm1.toString() ;
125     }
126     return outStr ;
127 }

```

```

jsshell> /open M250TMA03PracticeQuizSolnA.java

jsshell> Shelter shelter01 = new Shelter()
shelter01 ==> currentWeek is 1

jsshell> shelter01.populate()

jsshell> shelter01
shelter01 ==> currentWeek is 3
cat Billy:Black diabetic
cat Zoe:Black and white
dog Rover:mongrel
tortoise Speedy:Horsefield

```

```

302 public static void testHomed() {
303     Shelter shelter01 = new Shelter() ;
304     shelter01.populate() ;
305     System.out.println("At_start_shelter01:_\n"
306         + shelter01.toString()) ;
307     System.out.println("Deleting_existing_animal:_cat_Billy") ;
308     boolean homedRetVa101
309         = shelter01.homed(50,"cat","Billy") ;
310     System.out.println("After_deletion_shelter01:_\n"
311         + shelter01.toString()) ;
312     System.out.println("Deleting_non-existing_animal:_dog_Spot") ;
313     boolean homedRetVa102
314         = shelter01.homed(51,"dog","Spot") ;
315     System.out.println("After_non-existing_animal_shelter01:_\n"
316         + shelter01.toString()) ;
317 }

```

```
jsShell> Utilities.testHomed()
At start shelter01: currentWeek is 3
cat Billy:Black diabetic
cat Zoe:Black and white
dog Rover:mongrel
tortoise Speedy:Horsefield
Deleting existing animal: cat Billy
After deletion shelter01: currentWeek is 3
cat Zoe:Black and white
dog Rover:mongrel
tortoise Speedy:Horsefield
Deleting non-existing animal: dog Spot
After non-existing animal shelter01: currentWeek is 3
cat Zoe:Black and white
dog Rover:mongrel
tortoise Speedy:Horsefield

jsShell>
```

```
231 public static List<Integer> testRemoveForeach() {
232     List<Integer> intList01
233     = Utilities.sampleIntList01() ;
234     System.out.println("intList01_at_call_is_"
235         + intList01.toString());
236     for (Integer iNum : intList01) {
237         if (iNum % 2 == 1) {
238             intList01.remove(iNum) ;
239         }
240     }
241     System.out.println("intList01_at_return_is_"
242         + intList01.toString());
243     return intList01 ;
244 }
```

```
jsShell> List<Integer> intListA = Utilities.testRemoveForeach()
intList01 at call is [1, 2, 3, 4, 5, 6]
| Exception java.util.ConcurrentModificationException
|     at ArrayList$Itr.checkForComodification (ArrayList.java:1042)
|     at ArrayList$Itr.next (ArrayList.java:996)
|     at Utilities.testRemoveForeach (#5:20)
|     at (#6:1)
```

```
246 public static List<Integer> testRemoveForLoop() {
247     List<Integer> intList01
248     = Utilities.sampleIntList01() ;
249     System.out.println("intList01_at_call_is_"
250         + intList01.toString());
251     for (int idx = 0 ; idx < intList01.size() ; idx++) {
252         if (intList01.get(idx) % 2 == 1) {
253             intList01.remove(idx) ;
254         }
255     }
256     System.out.println("intList01_at_return_is_"
257         + intList01.toString());
258     return intList01 ;
259 }
```

```
jsShell> List<Integer> intListB = Utilities.testRemoveForLoop()
intList01 at call is [1, 2, 3, 4, 5, 6]
intList01 at return is [2, 4, 6]
intListB ==> [2, 4, 6]
```

- This version is to remove every element from the list
- But what happens ...

```
261 public static List<Integer> testRemoveForLoop01() {
262     List<Integer> intList01
263     = Utilities.sampleIntList01() ;
264     System.out.println("intList01_at_call_is_"
265         + intList01.toString());
266     for (int idx = 0 ; idx < intList01.size() ; idx++) {
```

```

267     intList01.remove(idx) ;
268 }
269 System.out.println("intList01_at_return_is_"
270                   + intList01.toString() ;
271 return intList01 ;
272 }

```

```

jshell> List<Integer> intListC = Utilities.testRemoveForLoop01()
intList01 at call is [1, 2, 3, 4, 5, 6]
intList01 at return is [2, 4, 6]
intListC ==> [2, 4, 6]

```

- Try with the list of strings
- As before only every other element is removed — why ?

```

274 public static List<String> testRemoveForLoop02() {
275     List<String> strList01
276     = Utilities.sampleStrList01() ;
277     System.out.println("strList01_at_call_is_"
278                       + strList01.toString() ;
279     for (int idx = 0 ; idx < strList01.size() ; idx++) {
280         strList01.remove(idx) ;
281     }
282     System.out.println("strList01_at_return_is_"
283                       + strList01.toString() ;
284     return strList01 ;
285 }

```

```

jshell> List<String> strListA = Utilities.testRemoveForLoop02()
strList01 at call is [a, b, c, d]
strList01 at return is [b, d]
strListA ==> [b, d]

```

| Iteration | strList01    | strList01.size() | idx | Deleted |
|-----------|--------------|------------------|-----|---------|
| Loop 1    | [a, b, c, d] | 4                | 0   | a       |
| Loop 2    | [b,c,d]      | 3                | 1   | c       |
| Loop 3    | [b,d]        | 2                | 2   | -       |

- At the beginning of Loop 3,

```
(idx < strList01.size()) == false
```

- Hence the for loop terminates with

```
strList01 == [b,d]
```

- The Iterator works as we want

```

287 public static List<String> testRemoveIterator() {
288     List<String> strList01
289     = Utilities.sampleStrList01() ;
290     System.out.println("strList01_at_call_is_"
291                       + strList01.toString() ;
292     Iterator<String> strIter = strList01.iterator() ;
293     while (strIter.hasNext()) {
294         String str = strIter.next() ;
295         strIter.remove() ;
296     }
297     System.out.println("strList01_at_return_is_"
298                       + strList01.toString() ;
299     return strList01 ;
300 }

```

```

jshell> List<String> strListB = Utilities.testRemoveIterator()
strList01 at call is [a, b, c, d]
strList01 at return is []
strListB ==> []

```

[ToC](#)[ToC](#)

### 8.3 Question 2

- This CodeRunner question concerns an animal shelter that keeps records about animals brought to the shelter.
- We will model this using two classes, `Animal` and `Shelter`, and we have provided incomplete code for these classes in the answer box.
- Your task is to complete those classes.
- Before you start, (1) read over the provided `Animal` class. The class has four instance variables:
  - `week` (of type `int`), which is a number from 1 to 52 denoting the week of the year the animal was brought to the shelter.
  - `kind`, `name` and `description`, which are of type `String`, and which store the kind of animal, its name and its description.
- Also note the provided constructor and methods of the `Animal` class and what they do.
- (2) Note that an instance of `Shelter` is used to hold data about a number of `Animal` objects, in a map. For example, the map might contain the following data:

| Key | Value                                                                                                                                                 |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2   | <code>Animal</code> object with <code>week</code> 50, <code>kind</code> "cat", <code>name</code> "Billy", <code>description</code> "Black diabetic"   |
| 1   | <code>Animal</code> object with <code>week</code> 50, <code>kind</code> "cat", <code>name</code> "Zoe", <code>description</code> "Black and white"    |
| 3   | <code>Animal</code> object with <code>week</code> 51, <code>kind</code> "dog", <code>name</code> "Rover", <code>description</code> "mongrel"          |
| 4   | <code>Animal</code> object with <code>week</code> 52, <code>kind</code> "tortoise", <code>name</code> "Speedy", <code>description</code> "Horsefield" |

- The order of the keys in the map shown above is just for illustration. The map does not store entries in any particular order.
- (a) (i) Declare an additional private instance variable `animals` in the `Shelter` class, capable of referencing a map whose keys are integers and whose values are `Animal` objects, as in the example table above.

Add a standard getter method for the `animals` collection.

- (a) (ii) The class `Shelter` should now have these instance variables:
- `animals`, which you added in part (i) above
  - `currentWeek` of type `int`, which is the current week of the year, and will be used when calculating how long an animal has been in the shelter.
  - `currentId` of type `int` which is the key for the last animal which was admitted to the shelter

- Amend the provided zero-argument constructor for `Shelter` so that when a new instance of `Shelter` is created `animals` is assigned a suitable empty map object and `currentWeek` is set to 1 and the `currentId` is set to 0.

```

9  class Shelter {
10     private int currentWeek ;
11     private int currentId ;
12     private Map<Integer, Animal> animals ;

14     public Shelter() {
15         super() ;
16         this.animals = new HashMap<>() ;
17         this.currentWeek = 1 ;
18         this.currentId = 0 ;
19     }

```

- (a) (iii) Complete the skeleton instance method `addAnimal()` for the class `Shelter`, with the header

```

public void addAnimal(String aKind
                     ,String aName
                     ,String aDescription)

```

- The method should use its three arguments to create an instance of `Animal`, using the value of `currentWeek` for its `week`, then add the `Animal` to the `animals` map, using the next value of `currentId` as the key.
- `currentId` will need to be kept updated so that each animal gets a unique ID.
- The first animal should have a `currentId` of 1.

```

33 public void addAnimal(String aKind
34                      ,String aName
35                      ,String aDescription) {
36     Animal anm1
37     = new Animal(this.currentWeek
38                  , aKind, aName, aDescription) ;
39     this.currentId = this.currentId + 1 ;
40     this.animals.put(this.currentId, anm1) ;
41 }

```

- (b) (i) Write a public instance method with the signature `inTheLastMonth(Animal)` for the `Shelter` class.

- The method should determine whether the `Animal` received as an argument has been brought into the shelter in the last month — that is, if the number of weeks between `currentWeek` and the animal's `week` is 4 or less

Note, however, that when `currentWeek` gets to 52, then the next `currentWeek`'s value is 1.

You'll need to watch out for this when calculating how many weeks it has been since the animal arrived. For two examples, see below.

| currentWeek in Shelter | Animal's week value | Weeks since arrived |
|------------------------|---------------------|---------------------|
| 52                     | 50                  | 2                   |
| 1                      | 50                  | 3                   |

- The method should return `true` if it has been less than or equal to four weeks since the animal arrived; otherwise it should return `false`. You can assume that an animal is never in the shelter for more than a year.

```

43 public boolean inTheLastMonth(Animal anm1) {
44     int currWeek = this.currentWeek ;
45     int animalWeek = anm1.getWeek() ;
46     return ( ( currWeek - animalWeek <= 4
47         && currWeek - animalWeek >= 0)
48         || (currWeek - animalWeek + 52 <= 4) ) ;
49 }

```

(b) (ii) Write a public instance method `showRecentAnimals()` for the `Shelter` class.

- For each `Animal` in the collection referenced by `animals`, if the animal was brought into the shelter in the last month (it has been less than or equal to four weeks since the animal arrived) then data about that animal should be printed to the standard output, with the details of each such animal on a separate line.
- If no animals have been brought in in the last month then *No recent animals* should be printed.
- For example, using the data from the table in part (a), the output from `showRecentAnimals()` when `currentWeek` is 3 should be:

```

dog Rover: mongrel
tortoise Speedy: Horsefield

```

```

public void showRecentAnimals() {
    boolean noRecentAnimals = true ;

    for (Animal anm1 : this.animals.values()) {
        if (inTheLastMonth(anm1)) {
            System.out.println(anm1.getKind()
                + " " + anm1.getName()
                + ":" + anm1.getDescription()) ;
            noRecentAnimals = false ;
        }
    }

    if (noRecentAnimals) {
        System.out.println("No recent animals") ;
    }
}

```

(c) Now turn to the `Animal` class.

- Two animals with the same `week`, `kind` and `name` should be considered to be the same. (`description` is irrelevant). So we need to override the `equals()` method inherited from `Object`.
- Whenever we override the inherited `equals()` method we also need to provide a `hashCode()` method compatible with the redefined `equals()`.
- (i) Write an `equals()` method to override that inherited from `Object`, which returns `true` if the `week`, `kind` and `name` for two `Animal` objects are the same, and `false` otherwise.
- (ii) Write a `hashCode()` method to override that inherited from `Object`, which returns the number of characters in the `name` of an `Animal` object.

```

193 /**
194  * A simple equals method
195  */
196 @Override
197 public boolean equals(Object o) {
198     Animal anm1 = (Animal) o ;
199     return ( getKind().equals(anm1.getKind())
200         && getWeek() == anm1.getWeek()
201         && getName().equals(anm1.getName()) ) ;

```

```

202 }
203
204 /**
205  * hashCode
206  */
207 @Override
208 public int hashCode() {
209     return getName().length() ;
210 }

```

(d) Return to the Shelter class. A public method with the header (below) is required

```

public boolean homed(int aWeek
                    ,String aKind
                    ,String aName)

```

- This method should determine whether or not the map referenced by `animals` contains an `Animal` with a `week`, `kind` and `name` matching the method arguments.
- If there is such an animal, its key-value pair should be removed from the map and `true` should be returned. Otherwise `false` should be returned.
- Write the `homed()` method.
- This version uses `Map` operations

```

68 public boolean homed(int aWeek
69                     ,String aKind
70                     ,String aName) {
71     Animal anm1 = new Animal(aWeek,aKind,aName,"No_Desc") ;
72     boolean anm1WasRemoved
73     = this.animals.values().remove(anm1) ;
74     return anm1WasRemoved ;
75 }

```

- This version uses an `Iterator`
- Note that we iterate over the keys not the Map itself
- We *can* iterate over a map — see below

```

77 public boolean homedA(int aWeek
78                     ,String aKind
79                     ,String aName) {
80
81     Animal anm1ToGo = new Animal(aWeek,aKind,aName,"No_Desc") ;
82     Iterator<Integer> animalKeyIter
83     = this.animals.keySet().iterator() ;
84     while (animalKeyIter.hasNext()) {
85         Animal anm1In = this.animals.get(animalKeyIter.next()) ;
86         if (anm1ToGo.equals(anm1In)) {
87             animalKeyIter.remove() ;
88             return true ;
89         }
90     }
91     return false ;
92 }

```

- The `entrySet()` method of the `Map` interface returns a `Set` view of the mappings contained in the map
- Any changes we make to the set will be reflected in the map

```

public boolean homedC(int aWeek
                    ,String aKind
                    ,String aName) {

    Animal anm1ToGo = new Animal(aWeek,aKind,aName,"No_Desc") ;
    Set<Map.Entry<Integer,Animal>> animalEntrySet

```

```

    = this.animals.entrySet() ;
    Iterator<Map.Entry<Integer,Animal>> animalEntryIter
    = animalEntrySet.iterator() ;
    while (animalEntryIter.hasNext()) {
        Animal anm1In = animalEntryIter.next().getValue() ;
        if (anm1ToGo.equals(anm1In)) {
            animalKeyIter.remove() ;
            return true ;
        }
    }
    return false ;
}

```

- The code below may work but it is unpredictable
- See [Iterating through a Collection, avoiding ConcurrentModificationException when removing objects in a loop](#)
- See [java.base > java.util > Class ConcurrentModificationException](#)

```

94 public boolean homedB(int aWeek
95                     ,String aKind
96                     ,String aName) {
97
98     for (Integer anm1Key : this.animals.keySet()) {
99         Animal anm1 = this.animals.get(anm1Key) ;
100         if ( anm1.getWeek() == aWeek
101             && anm1.getKind() == aKind
102             && anm1.getName() == aName) {
103             this.animals.remove(anm1Key) ;
104             return true ;
105         }
106     }
107     return false ;
108 }

```

### 8.3.1 Q 2 Sample Usage

- The code is in [M250TMA03PracticeQuizSolnB.java](#) and we use [jShell](#) to do evaluations — see [Java Shell User's Guide](#)
- We have several classes in one file — see [Java: Multiple class declarations in one file](#)

```

import java.util.* ;

class M250TMA03PracticeQuizSolnB {
    public static void main(String[] args) {
        // further code here or in Utilities
    }
}

```

```
class Shelter {
```

```
class Animal {
```

```
class Utilities {
```

- Open the Java file at the jShell prompt — it will compile it

```

jshe11> /open M250TMA03PracticeQuizSolnB.java

jshe11> Shelter shelter01 = new Shelter()
shelter01 ==> currentWeek is 1

jshe11>

```

- Note that in the first example usage shown in Q 1 the value of the shelter was displayed as follows

```
jshe11> /open M250TMA03PracticeQuizSolnA.java

jshe11> Shelter shelter01 = new Shelter()
shelter01 ==> Shelter@46f7f36a

jshe11>
```

- Note that in the first example usage shown in Q 1 the value of the shelter was displayed as follows

```
jshe11> /open M250TMA03PracticeQuizSolnA.java

jshe11> Shelter shelter01 = new Shelter()
shelter01 ==> Shelter@46f7f36a

jshe11>
```

- The original example had the default `toString()` definition from `Object` which gives the textual representation of an object as its class, the @ sign character, and the unsigned hexadecimal representation of the hash code of the object

```
x.getClass().getName() + "@" + Integer.toHexString(x.hashCode())
```

- `Shelter` has `populate()` and `toString()` definitions to facilitate sample usage

```
113 public void populate() {
114     this.currentWeek = 50 ;
115     this.addAnimal("cat","Billy","Black_diabetic") ;
116     this.addAnimal("cat","Zoe","Black_and_white") ;
117     this.currentWeek = 51 ;
118     this.addAnimal("dog","Rover","mongrel") ;
119     this.currentWeek = 52 ;
120     this.addAnimal("tortoise","Speedy","Horsefield") ;
121     this.currentWeek = 3 ;
122 }

124 @Override
125 public String toString() {
126     String outStr = "" ;
127     outStr = outStr + "currentWeek_is_" + currentWeek ;
128     for (Integer anmlKey : this.animals.keySet()) {
129         Animal anml = this.animals.get(anmlKey) ;
130         outStr = outStr + "\n"
131             + "ID_" + anmlKey
132             + "_" + anml.toString() ;
133     }
134     return outStr ;
135 }
```

- `Utilities` class has further definitions to facilitate sample usage

```
234 public static void testHomed() {
235     Shelter shelter01 = new Shelter() ;
236     shelter01.populate() ;
237     System.out.println("At_start_shelter01:_\n"
238         + shelter01.toString()) ;
239     System.out.println("Deleting_existing_animal:_cat_Billy") ;
240     boolean homedRetVa101
241         = shelter01.homed(50,"cat","Billy") ;
242     System.out.println("After_deletion_shelter01:_\n"
243         + shelter01.toString()) ;
244     System.out.println("Deleting_non-existing_animal:_dog_Spot") ;
245     boolean homedRetVa102
246         = shelter01.homed(51,"dog","Spot") ;
247     System.out.println("After_non-existing_animal_shelter01:_\n"
248         + shelter01.toString()) ;
249 }
```

```
jshe11> shelter01.populate()

jshe11> System.out.println(shelter01)
currentWeek is 3
ID 1 cat Billy:Black diabetic
ID 2 cat Zoe:Black and white
ID 3 dog Rover:mongrel
ID 4 tortoise Speedy:Horsefield
```

```
jshe11> Utilities.testHomed()
At start shelter01: currentWeek is 3
ID 1 cat Billy:Black diabetic
ID 2 cat Zoe:Black and white
ID 3 dog Rover:mongrel
ID 4 tortoise Speedy:Horsefield
Deleting existing animal: cat Billy
After deletion shelter01: currentWeek is 3
ID 2 cat Zoe:Black and white
ID 3 dog Rover:mongrel
ID 4 tortoise Speedy:Horsefield
Deleting non-existing animal: dog Spot
After non-existing animal shelter01: currentWeek is 3
ID 2 cat Zoe:Black and white
ID 3 dog Rover:mongrel
ID 4 tortoise Speedy:Horsefield

jshe11>
```

[ToC](#)[ToC](#)

## 8.4 Question 3

- This CodeRunner question concerns an animal shelter that keeps records about animals brought to the shelter.
- We will model this using two classes, `Animal` and `Shelter`, and we have provided incomplete code for the `Shelter` class in the answer box.
- Your task is to complete the `Shelter` class. In this question you do not need to add anything to the `Animal` class.
- Before you start, The `Animal` class is provided in the answer box already.
- (2) Note that in this question an instance of `Shelter` holds data about a number of `Animal` objects in a `Set`. For example, the `Set` might contain the following data:

---

`Animal` object with week 50, kind "cat", name "Billy", description "Black diabetic"

---

`Animal` object with week 50, kind "cat", name "Zoe", description "Black and white"

---

`Animal` object with week 51, kind "dog", name "Rover", description "mongrel"

---

`Animal` object with week 52, kind "tortoise", name "Speedy", description "Horsefield"

---

- The `Set` does not store entries in any particular order. Sets do not allow duplicate entries.
- (a) (i) Declare an additional private instance variable `animals` in the `Shelter` class, capable of referencing a `Set` whose values are `Animal` objects, as in the example

table above.

Add a standard getter method for the `animals` collection and for the `currentWeek` instance variable.

- (ii) The class `Shelter` should now have these instance variables:
  - `animals`, which you added in part (i) above
  - `currentWeek` of type `int`, which is the current week of the year, and will be used when calculating how long an animal has been in the shelter.
- Amend the provided zero-argument constructor for `Shelter` so that when a new instance of `Shelter` is created `animals` is assigned a suitable empty `Set` object and `currentWeek` is set to 1.

```

9  class Shelter {
10     private int currentWeek ;
11     private Set<Animal> animals ;

13     public Shelter () {
14         super() ;
15         this.animals = new HashSet<Animal>() ;
16         this.currentWeek = 1 ;
17     }

```

- (a) (iii) Complete the skeleton instance method `addAnimal()` for the class `Shelter`, with the header

```

public void addAnimal(String aKind
                     ,String aName
                     ,String aDescription)

```

- The method should use its three arguments to create an instance of `Animal`, using the value of `currentWeek` for its `week`, then add the `Animal` to the `animals` `Set`.

```

27     public void addAnimal(String aKind
28                          ,String aName
29                          ,String aDescription) {
30         Animal anm1
31         = new Animal(this.currentWeek
32                     , aKind, aName, aDescription) ;
33         this.animals.add(anm1) ;
34     }

```

- (b) (i) Write a public instance method with the signature `inTheLastMonth(Animal)` for the `Shelter` class.

- The method should determine whether the `Animal` received as an argument has been brought into the shelter in the last month — that is, if the number of weeks between `currentWeek` and the animal's `week` is 4 or less

Note, however, that when `currentWeek` gets to 52, then the next `currentWeek`'s value is 1.

You'll need to watch out for this when calculating how many weeks it has been since the animal arrived. For two examples, see below.

| <code>currentWeek</code> in <code>Shelter</code> | Animal's <code>week</code> value | Weeks since arrived |
|--------------------------------------------------|----------------------------------|---------------------|
| 52                                               | 50                               | 2                   |
| 1                                                | 50                               | 3                   |

- The method should return `true` if it has been less than or equal to four weeks since the animal arrived; otherwise it should return `false`. You can assume that an animal is never in the shelter for more than a year.

```

36 public boolean inTheLastMonth(Animal anml) {
37     int currWeek = this.currentWeek ;
38     int animalWeek = anml.getWeek() ;
39     return ( ( currWeek - animalWeek <= 4
40         && currWeek - animalWeek >= 0)
41         || (currWeek - animalWeek + 52 <= 4) ) ;
42 }

```

(b) (ii) Write a public instance method `showRecentAnimals()` for the `Shelter` class.

- For each `Animal` in the collection referenced by `animals`, if the animal was brought into the shelter in the last month (it has been less than or equal to four weeks since the animal arrived) then data about that animal should be printed to the standard output, with the details of each such animal on a separate line.
- If no animals have been brought in in the last month then *No recent animals* should be printed.
- For example, using the data from the table in part (a), the output from `showRecentAnimals()` when `currentWeek` is 3 should be:

```

dog Rover:mongrel
tortoise Speedy:Horsefield

```

```

44 public void showRecentAnimals() {
45     boolean noRecentAnimals = true ;
46
47     for (Animal anml : this.animals) {
48         if (inTheLastMonth(anml)) {
49             System.out.println(anml.getKind()
50                 + " " + anml.getName()
51                 + ":" + anml.getDescription()) ;
52             noRecentAnimals = false ;
53         }
54     }
55
56     if (noRecentAnimals) {
57         System.out.println("No_recent_animals") ;
58     }
59 }

```

(c) Write a public method `homed()` in the `Shelter` class with the header

```

public boolean homed(int aWeek
                    ,String aKind
                    ,String aName)

```

- This method should determine whether or not the `Set` referenced by `animals` contains an `Animal` with a `week`, `kind` and `name` matching the method arguments.
- If there is such an animal, it should be removed from the set and `true` should be returned. Otherwise `false` should be returned.
- This version uses `Set` operations

```

61 public boolean homed(int aWeek
62                     ,String aKind
63                     ,String aName) {
64     Animal anml = new Animal(aWeek,aKind,aName,"No_Desc") ;
65     boolean anmlIsIn
66         = this.animals.contains(anml) ;
67     if (anmlIsIn) {
68         this.animals.remove(anml) ;

```

```

69     }
70     return anmlIsIn ;
71 }

```

- This version uses an [Iterator](#)

```

73 public boolean homedA(int aWeek
74                     ,String aKind
75                     ,String aName) {
76
77     Iterator<Animal> animalIter = this.animals.iterator() ;
78     while (animalIter.hasNext()) {
79         Animal anml = animalIter.next() ;
80         if ( anml.getWeek() == aWeek
81             && anml.getKind() == aKind
82             && anml.getName() == aName) {
83             animalIter.remove() ;
84             return true ;
85         }
86     }
87     return false ;
88 }

```

- The code below may work but it is unpredictable
- See [Iterating through a Collection, avoiding ConcurrentModificationException when removing objects in a loop](#)
- See [java.base > java.util > Class ConcurrentModificationException](#)

```

90 public boolean homedB(int aWeek
91                     ,String aKind
92                     ,String aName) {
93
94     for (Animal anml : this.animals) {
95         if ( anml.getWeek() == aWeek
96             && anml.getKind() == aKind
97             && anml.getName() == aName) {
98             this.animals.remove(anml) ;
99             return true ;
100         }
101     }
102     return false ;
103 }

```

### 8.4.1 Q 3 Sample Usage

- The code is in [M250TMA03PracticeQuizSolnC.java](#) and we use [jShell](#) to do evaluations — see [Java Shell User's Guide](#)
- We have several classes in one file — see [Java: Multiple class declarations in one file](#)

```

import java.util.* ;

class M250TMA03PracticeQuizSolnC {
    public static void main(String[] args) {
        // further code here or in Utilities
    }
}

```

```
class Shelter {
```

```
class Animal {
```

```
class Utilities {
```

- Open the Java file at the jShell prompt — it will compile it

```
jshe11> /open M250TMA03PracticeQuizSolnC.java

jshe11> Shelter shelter01 = new Shelter()
shelter01 ==> currentWeek is 1

jshe11>
```

- Note that in the first example usage shown in Q 1 the value of the shelter was displayed as follows

```
jshe11> /open M250TMA03PracticeQuizSolnA.java

jshe11> Shelter shelter01 = new Shelter()
shelter01 ==> Shelter@46f7f36a

jshe11>
```

- Note that in the first example usage shown in Q 1 the value of the shelter was displayed as follows

```
jshe11> /open M250TMA03PracticeQuizSolnA.java

jshe11> Shelter shelter01 = new Shelter()
shelter01 ==> Shelter@46f7f36a

jshe11>
```

- The original example had the default `toString()` definition from `Object` which gives the textual representation of an object as its class, the @ sign character, and the unsigned hexadecimal representation of the hash code of the object

```
x.getClass().getName() + "@" + Integer.toHexString(x.hashCode())
```

- `Shelter` has `populate()` and `toString()` definitions to facilitate sample usage

```
108 public void populate() {
109     this.currentWeek = 50 ;
110     this.addAnimal("cat","Billy","Black_diabetic") ;
111     this.addAnimal("cat","Zoe","Black_and_white") ;
112     this.currentWeek = 51 ;
113     this.addAnimal("dog","Rover","mongrel") ;
114     this.currentWeek = 52 ;
115     this.addAnimal("tortoise","Speedy","Horsefield") ;
116     this.currentWeek = 3 ;
117 }

119 @Override
120 public String toString() {
121     String outStr = "" ;
122     outStr = outStr + "currentWeek_is_" + currentWeek ;
123     for (Animal anml : this.animals) {
124         outStr = outStr + "\n" + anml.toString() ;
125     }
126     return outStr ;
127 }
```

- `Utilities` class has further definitions to facilitate sample usage

```
226 public static void testHomed() {
227     Shelter shelter01 = new Shelter() ;
228     shelter01.populate() ;
229     System.out.println("At_start_shelter01:_"
230         + shelter01.toString()) ;
231     System.out.println("Deleting_existing_animal:_cat_Billy") ;
232     boolean homedRetVal01
233         = shelter01.homed(50,"cat","Billy") ;
234     System.out.println("After_deletion_shelter01:_"
235         + shelter01.toString()) ;
236     System.out.println("Deleting_non-existing_animal:_dog_Spot") ;
```

```

237     boolean homedRetVal02
238     = shelter01.homed(51,"dog","Spot") ;
239     System.out.println("After_non-existing_animal_shelter01:_"
240                       + shelter01.toString() ) ;
241 }

```

```

jshell> shelter01.populate()

jshell> System.out.println(shelter01)
currentWeek is 3
ID 1 cat Billy:Black diabetic
ID 2 cat Zoe:Black and white
ID 3 dog Rover:mongrel
ID 4 tortoise Speedy:Horsefield

```

```

jshell> Utilities.testHomed()
At start shelter01: currentWeek is 3
cat Zoe:Black and white
cat Billy:Black diabetic
dog Rover:mongrel
tortoise Speedy:Horsefield
Deleting existing animal: cat Billy
After deletion shelter01: currentWeek is 3
cat Zoe:Black and white
dog Rover:mongrel
tortoise Speedy:Horsefield
Deleting non-existing animal: dog Spot
After non-existing animal shelter01: currentWeek is 3
cat Zoe:Black and white
dog Rover:mongrel
tortoise Speedy:Horsefield

jshell>

```

[ToC](#)[ToC](#)[ToC](#)

## 9 Common Mistakes

- All programming languages have some sharp edges or subtle points, including [Java](#)
- This section discusses some common mistakes
- The examples below use [JShell](#), a [Read-Eval-Print loop \(REPL\)](#) tool available for Java
- [Java Shell User's Guide](#) describes its usage
- Note: [JShell](#) is not directly in M250 (it arrived in JDK 9) but for demonstrations students only need to know:
  - Java statements and class definitions can be executed at the prompt `jshell>` and continuation prompt `...>`
  - The result is reported on the line following
  - Most common libraries are automatically imported
- The examples refer to the *M250 Units* and *M250 Exam Handbook* for further points

### 9.1 Converting an Array to a List

```
jshe11> String[] xArray = {"a","b","c","d"}
xArray ==> String[4] { "a", "b", "c", "d" }

jshe11> List<String> yList = Arrays.asList(xArray)
yList ==> [a, b, c, d]

jshe11> yList.add("e")
| Exception java.lang.UnsupportedOperationException
|   at AbstractList.add (AbstractList.java:153)
|   at AbstractList.add (AbstractList.java:111)
|   at (#20:1)
```

- See *M250 Exam Handbook* section 5.3 *Collection utility classes* Class `Arrays` (page 32)
- `asList` returns an `ArrayList` which is of *fixed size* — this `ArrayList` is a private static class inside `Arrays` — it is not the `java.util.ArrayList` class
- Solution: the `ArrayList` constructor can accept a `Collection` type, which is also a super type for `java.util.Arrays.ArrayList`

```
jshe11> String[] xArray = {"a","b","c","d"}
xArray ==> String[4] { "a", "b", "c", "d" }

jshe11> List<String> yList =
...> new ArrayList<String>(Arrays.asList(xArray))
yList ==> [a, b, c, d]

jshe11> boolean b = yList.add("e")
b ==> true

jshe11> yList
yList ==> [a, b, c, d, e]
```

[ToC](#)

## 9.2 Using a TreeSet

```
jshe11> String[] pArray =
...> {"d", "a", "c", "a", "b", "a"}
pArray ==> String[6] { "d", "a", "c", "a", "b", "a" }

jshe11> Set<String> qSet =
...> new TreeSet<String>(Arrays.asList(pArray))
qSet ==> [a, b, c, d]

jshe11> String elmnt = qSet.first()
| Error:
| cannot find symbol
|   symbol: method first()
|   String elmnt = qSet.first();
|                   ^-----^
```

- `TreeSet` implements the `SortedSet` interface
- `first()` is a method implemented by `TreeSet`
- So what is wrong ?

```
jshe11> String[] pArray =
...> {"d", "a", "c", "a", "b", "a"}
pArray ==> String[6] { "d", "a", "c", "a", "b", "a" }

jshe11> SortedSet<String> rSet =
...> new TreeSet<String>(Arrays.asList(pArray))
rSet ==> [a, b, c, d]

jshe11> String elmnt = rSet.first()
elmnt ==> "a"
```

- `element` was declared to be of type `Set`
- `first()` is not in the protocol of `Set`
- Declare the set to be of type `SortedSet`
- (See SAQ 10 in Unit 10)

[ToC](#)

### 9.3 Remove Elements from List Inside Loop

```
jshe11> List<String> yList =
...> new ArrayList<String>(Arrays.asList(xArray))
yList ==> [a, b, c, d]

jshe11> for (String str : yList) {
...> if (str.equals("a")) {
...>     yList.remove(str);
...> }
...> }
| Exception java.util.ConcurrentModificationException
|   at ArrayList$Itr.checkForComodification
|       (ArrayList.java:1042)
|   at ArrayList$Itr.next (ArrayList.java:996)
|   at (#67:1)
```

- Unit 9 page 33 describes the *for-each* statement — a note on page 35 mentions that the collection should not be modified in the loop — hence the error
- When iterating over a collection or map, the underlying collection should not be modified except through the iterator's `remove` method. If it is modified in any other way, the result is unpredictable.
- If we have just one element to remove, here is an alternative using `list` methods

```
jshe11> List<String> yList =
...> new ArrayList<String>(Arrays.asList(xArray))
yList ==> [a, b, c, d]

jshe11> int idx = yList.indexOf("a")
idx ==> 0

jshe11> String str = yList.remove(0)
str ==> "a"

jshe11> yList
yList ==> [b, c, d]
```

- The `Iterable` interface provides the `iterator` method — see `Iterator` and `ListIterator` interfaces
- See *M269 Exam Handbook page 24* (and mentioned in Exercise 2 Unit 10, p 87, solution p 236)

```
jshe11> List<String> yList =
...> new ArrayList<String>(Arrays.asList(xArray))
yList ==> [a, b, c, d]

jshe11> Iterator<String> iter = yList.iterator()
iter ==> java.util.ArrayList$Itr@12bc6874

jshe11> while (iter.hasNext()) {
...>     String str = iter.next();
...>     if (str.equals("a")) {
...>         iter.remove();
...>     }
...> }
```

```
jshell> yList
ys ==> [b, c, d]
```

```
jshell> String[] jArray = {"a","b","c","a","d","a"}
jArray ==> String[6] { "a", "b", "c", "a", "d", "a" }

jshell> List<String> kList =
...> new ArrayList<String>(Arrays.asList(jArray))
kList ==> [a, b, c, a, d, a]

jshell> Iterator<String> iter = ks.iterator()
iter ==> java.util.ArrayList$Itr@5d3411d

jshell> while (iter.hasNext()) {
...> String str = iter.next();
...> if (str.equals("a")) {
...>     iter.remove();
...> }
...> }

jshell> kList
kList ==> [b, c, d]
```

- The collection should not be modified other than using [remove](#) but can use [add](#) and [set](#) with [ListIterator](#)

[ToC](#)
[ToC](#)

## 10 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](#)

## 11 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

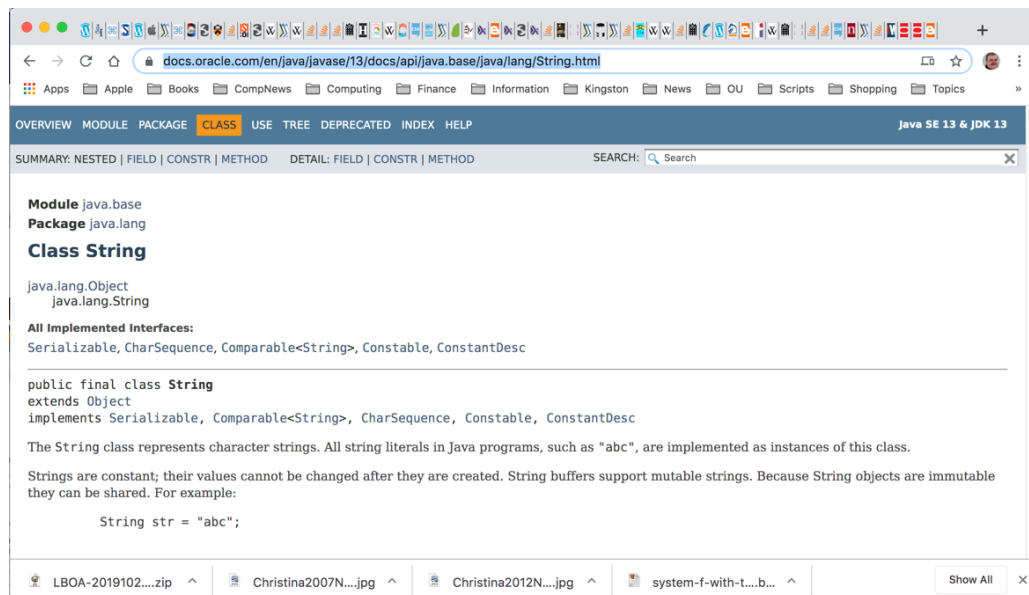
- TMA03 Thursday 8 May 2025
- Tutorial: Exam revision: Online 10:00 Sunday 11 May 2025
- Exam Friday 30 May 2025

[ToC](#)

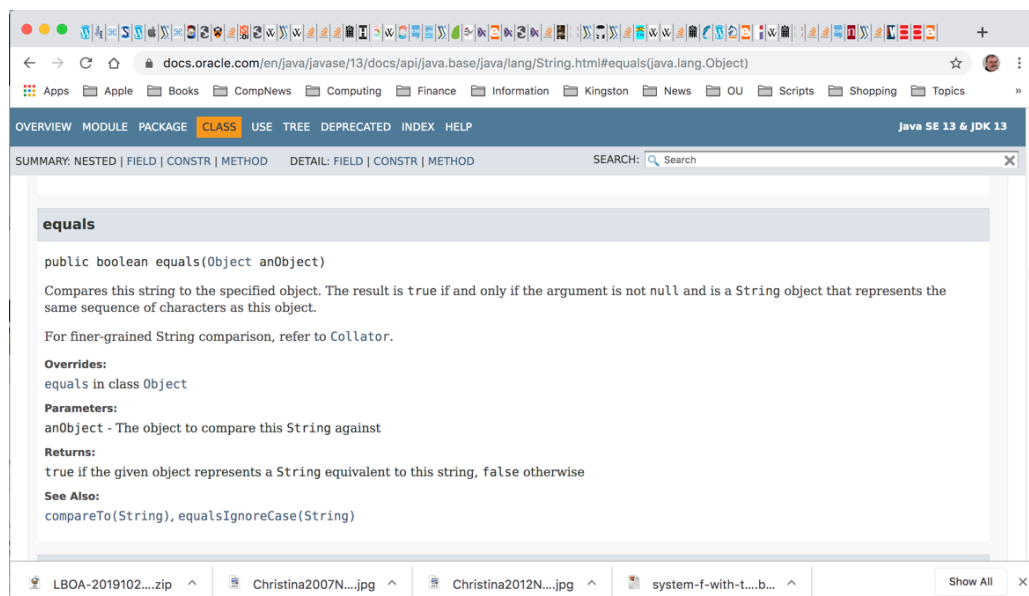
## 12 Web Links & References

### 12.1 Java Documentation

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



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



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

[ToC](#)

## 12.2 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](http://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 Kolling (2016). *Objects First with Java*. Pearson, sixth edition. ISBN 1292159049. URL <http://www.bluej.org/objects-first/>. 37
- Bloch, Joshua (2017). *Effective Java*. Addison-Wesley Professional, third edition. ISBN 9780134685991. 38
- 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. 37
- 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/>. 37
- 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](#)

---

Author Phil Molyneux    Written 16 March 2025    *Printed 13th March 2025*

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

Topic path:

`/M250Prsntn2024JTutorials/M250Tutorial20250316CollectionsPrsntn2024J/M250Tutorial20250316CollectionsPrsntn2024J.pdf`