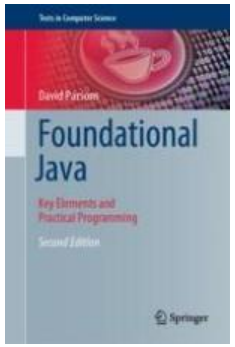


Chapter 5
Creating Objects

Foundational Java

Key Elements and Practical Programming

Reference Data Types

- Java has both primitive types and reference types
- Reference types include…
- Objects
  - User-defined types
    - e.g. BankAccount, InsurancePolicy, Client
  - HashMap
  - String
  - Many more!
- Arrays

Classes

- Classes can provide methods
  - System.out.println(…)
  - Math.random()
- Program entry point
  - main method
- Specify type of object
  - attributes (data fields)
  - operations (methods)

Object Creation - Constructors

- Objects are created by invoking a Constructor
  - Need an object reference
  - Use the ‘new’ keyword
  - Constructors are methods that construct objects
    - They have the same name as the class
    - They can accept parameters
- The return value of a constructor call is a new object

```
Object myObject = new Object();
// myObject is an Object, Object() is the constructor
```

Overloaded String Constructors

- Zero arguments

```
String s1 = new String();                    // s1 is a String
```

- Parameter argument

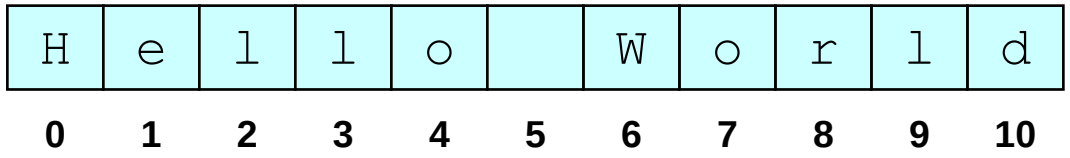
```
String s2 = new String("Hello");             // s2 is another String
```

- Assignment to a literal is the same as calling a constructor

```
String s3 = "Hello";  // same effect as String s3 = new String("Hello");
```

String Methods

- Strings are Immutable in Java
  - Cannot be modified once created
  - All operations on Strings returns new Strings
- Have methods to access, search and perform comparisons
- Contain Unicode characters
- 0 based indexing



Referencing ‘null’

- An object reference needs to be initialised, but it need not always reference an actual object
- An alternative is to initialise it to ‘null’

```
String customerName = null;
```

- Useful when we need to create a reference to an object in a different part of the program to where the actual object will be created

Object Methods

- Methods are operations that an object can perform
- They are invoked using the '.' operator:
  - objectName.methodName(parameters)
- For example, all objects have a toString() method that returns a String representation of the object

```
Object myObject = new Object();
String s = myObject.toString();
```

Some String Methods

Method	Purpose
length()	Returns an integer representing the number of characters in the String.
contains(…)	Returns ‘true’ if the character or String passed to the method is matched inside the original String, otherwise returns false
startsWith(…)	Returns ‘true’ if the character or String passed to the method is matched inside the original String, otherwise returns false
indexOf(…)	Returns the index (as an int) within the original String of the first occurrence of the specified character or String
lastIndexOf(String)	Returns the index (as an int) within the original String of the last occurrence of the specified character or String.
substring(int, int)	Returns another String that is a copy of the part of the original String between the two indexes provided.

Long Strings

- Long Strings (e.g. source code) may be broken over multiple lines by concatenating with the “+” operator

```
String HTMLString = new String(
  "<!DOCTYPE html>\n" +
  "<html>\n" +
  "\t<head>\n" +
  "\t\t<title>My HTML Page</title>\n" +
  "\t</head>\n" +
  "\t<body>\n" +
  "\t\t<h1>Welcome to my page</h1>\n" +
  "\t\t<p>This page is written in simple HTML<p>\n" +
  "\t\t<p>Since HTML files are just text markup, they can be contained in Java String objects</p>\n" +
  "\t</body>\n" +
  "</html>");
```

Text Blocks

- Text blocks using triple quotes make handling long Strings much easier

```
String myLongString = """
  <!DOCTYPE html>
  <html>
    <head>
      <title>My HTML Page</title>
    </head>
    <body>
      <h1>Welcome to my page</h1>
      <p>This page is written in simple HTML<p>
      <p>Since HTML files are just text markup, they can be contained in Java String objects</p>
    </body>
</html>
""";
```

StringBuilders

- Strings are immutable
  - cannot be changed once created
- Some String methods simply create new String objects that have a different state
- StringBuilder can change an existing String of characters.
- Default StringBuilder constructor creates a StringBuilder with a default capacity of 16 characters
  - will automatically resize itself if necessary

```
StringBuilder builder = new StringBuilder();
```

StringBuilder Methods

- Current capacity of a StringBuilder

```
int capacity = builder.capacity();
```

- Append extra characters

```
builder.append("more text");
```

- Insert characters

```
builder.insert(0, "new text")
```

StringBuffer

- Provides the same interface as the StringBuilder
  - thread-safe - less efficient
- Use StringBuilder unless multithreading is required

Exercise 5.1

- Create a class with a ‘main’ method
- In ‘main’, create a new String object that contains lower case letters
- Create another String object that contains uppercase letters
- Create a StringBuilder object that contains the first String converted to upper case
- Append a lower-case version of the second String to the StringBuilder

The ‘toString’ Method

- toString() is common to all objects in Java
- Returns a String representation of the object
- Default implementation helpful in understanding how reference types are handled in memory

```
Object myObject = new Object();
String myString = myObject.toString();
System.out.println(myString);
```

```
java.lang.Object@36d64342
```

Class name and memory hashcode (usually unique for each object)

Exercise 5.2

- Create a class with a “main” method.
- In “main”, create three objects using the Object constructor.
- Print out all the objects using “System.out.println” – note that their hash codes should appear different.
- Print out the hash code of one of the objects using the ‘hashCode” method. Note that it looks different from the one generated by “toString” because that is the hexadecimal representation of the same value.
- Use the “Integer.toHexString” method to convert the hash code of one of the objects to hexadecimal. You should see that the value looks the same as the one generated by the “toString” method.

References and Memory

- A variable stores a reference to an object
  - A reference encapsulates a pointer to the object, but does not enable the memory access traditionally associated with pointers
  - The object's physical address cannot be accessed or manipulated

Primitive Variable Assignment

- Assignment of primitive types means the stored values are copied from one variable to another

```
int var1= 5;
int var2 = 9;
var1 = var2; // value copied from var2 to var 1
var2 = 10; // does not affect var1
```

Reference Type Assignment

- Assignment of reference types redirects the reference to point to another object
- ‘a’ and ‘b’ will reference different objects

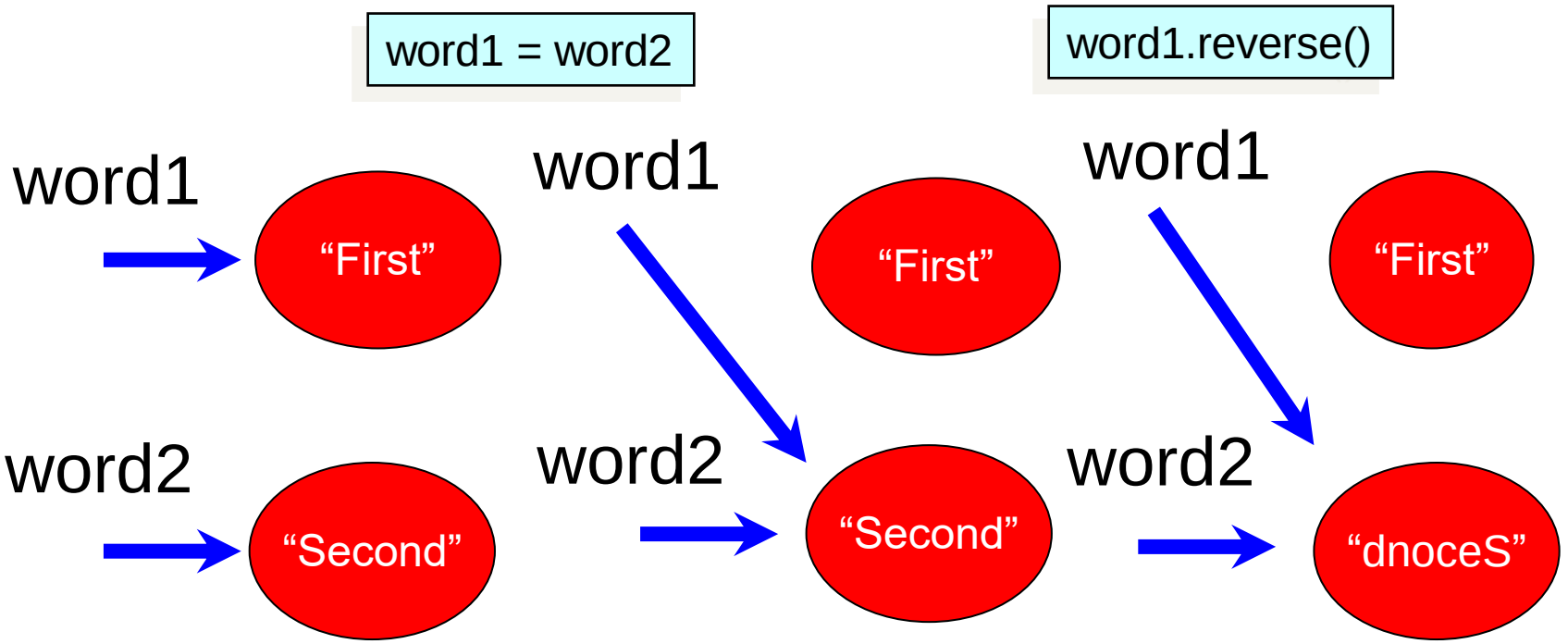
```
Object a = new Object();
Object b = new Object();
System.out.println(a);
System.out.println(b);
```

- After assignment, they will reference the same object

```
a = b;
```

Shared References

- If two references point to the same object, changes made via one reference will be seen by the second



Garbage Collection

- An object is eligible for garbage collection when there are no longer any references to it
- This occurs automatically
  - Objects cannot be deleted manually
  - You can redirect a reference to null if you no longer wish to access an object and make it available for garbage collection

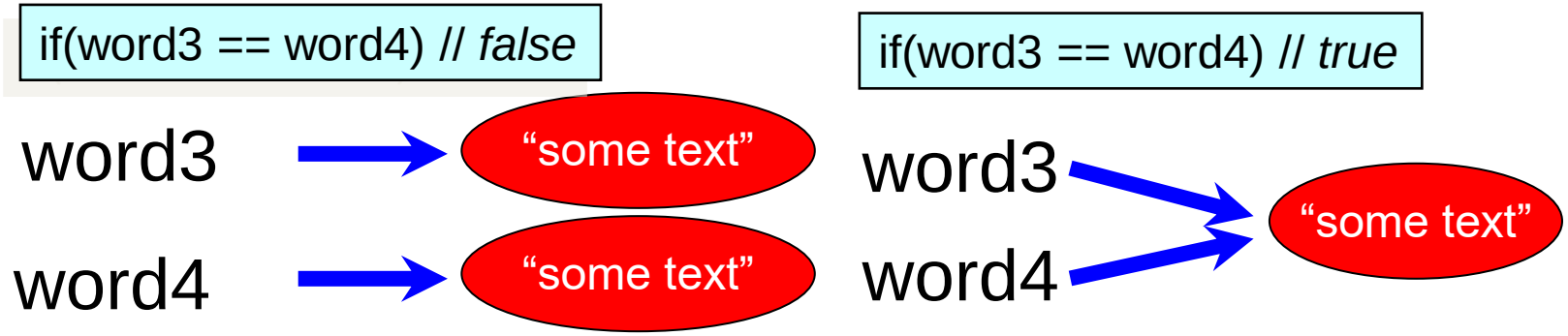
Garbage Collection is Automatic

- The garbage collector is part of the runtime system
  - It runs on a low priority thread
  - It runs automatically when the JVM needs more memory to continue executing
- You can suggest to the runtime that it might like to do some garbage collection with the ‘gc’ method (it may ignore you)

```
System.gc()
```

Object Equality

- The == and != operators can be used with objects
- They compare the object references (pointers)
  - == is true if the references are identical (if they point to the same object)
  - The state of the objects is not relevant

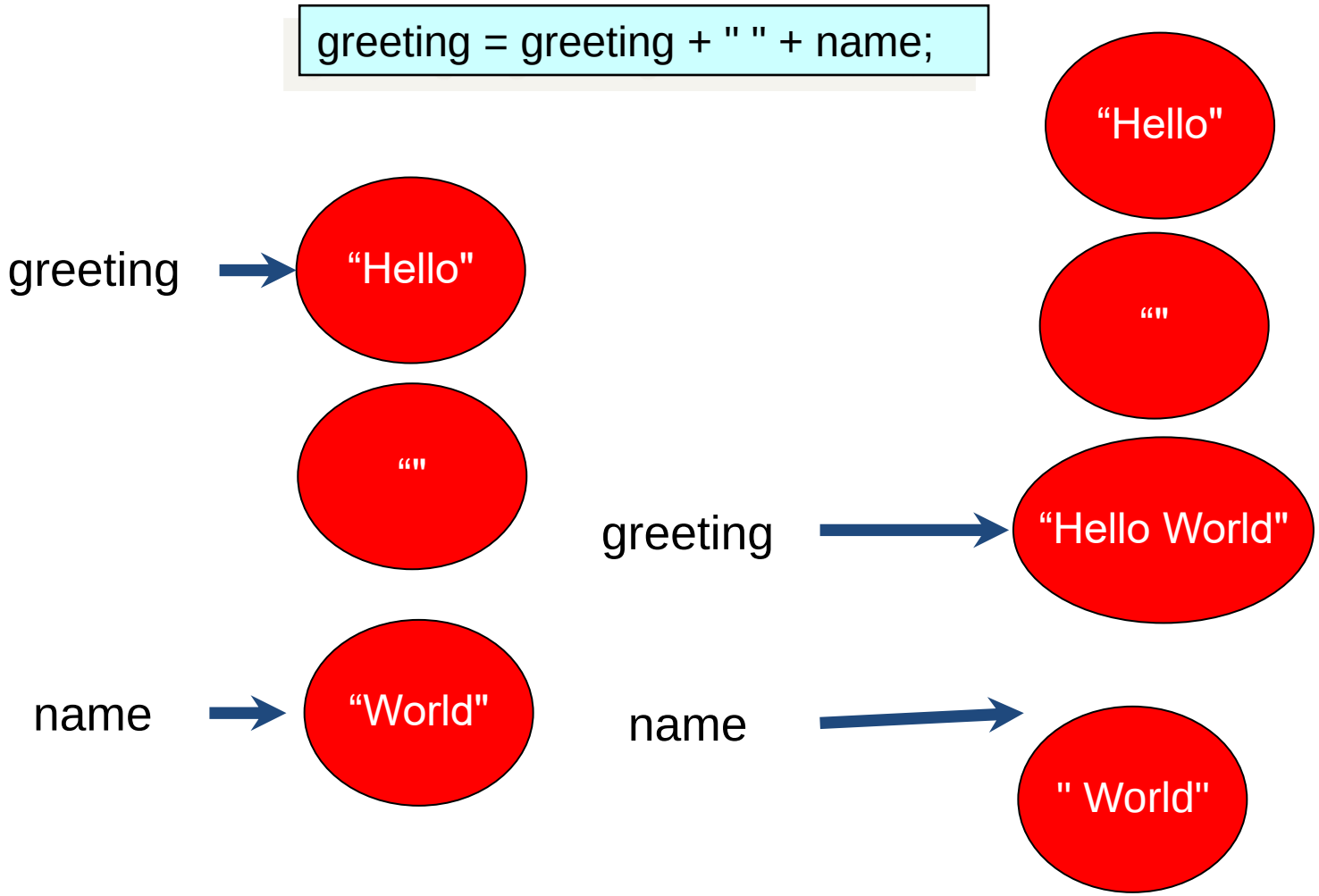


String Concatenation

- Concatenation can generate a lot of work for the garbage collector

```
String greeting = "Hello";
String name = "World";
greeting = greeting + " " + name; // result is "Hello World"
```

String Concatenation and Memory



Java Library Packages

- Java classes are all in packages
- The full name of a class includes its package name
  - For example, the full name of one of the Java Date classes (in the java.util package) is java.util.Date
  - To use it in our code, we need the fully qualified class name for both reference declarations and constructor calls:

```
java.util.Date date = new java.util.Date();
```

Importing Classes

- To access classes in another package without using the full name you can import classes from that package
  - 'import' statements come after the package declaration but before any other code in a Java source file

```
package xxxx;
import java.util.date;
```

  - The import enables the simple name of the Date to be used in the code

```
Date today = new Date();
```

Wild Cards and Multiple Imports

- You can have as many import statements as you like

```
package xxxx;
import …;
import …;
public class ….
```

- You can use .* to indicate all names in a package

```
import java.util.*;
```

- Or import individual class names (recommended)

```
import java.util.Date;
import java.util.Formatter;
```

Importing from java.util

- Date and Formatter are in the java.util package

```
import java.util.Date;
import java.util.Formatter;

public class ImportExample
{
  public static void main(String[] args) {
    Date myDate = new Date();
    Formatter myDateFormatter = new Formatter();
    myDateFormatter.format("%tR",myDate);
    System.out.println(myDateFormatter);
  }
}
```

Date and Time Formats

Format String	Output Format
%tR	Time formatted for the 24-hour clock as hours and minutes, e.g. 12:00
%tT	Time formatted for the 24-hour clock as hours, minutes and seconds, e.g. 12:00:00
%tr	Time formatted for the 12-hour clock as hours, minutes and seconds plus AM or PM e.g. 12:00:00 AM
%tD	Date formatted as day month and (short) year, e.g. 01/01/20
%tF	Date formatted as (long) year, month and day. e.g. 2020-01-01
%tc	Date and time formatted as day, date, time, zone and year, e.g. Wed Jan 01 12:00:00 EDT 2020.

Modules

- Eclipse adds a “module-info.java” file to Java projects
- Defines the module name and what is in it
- Since Java 9, Java has been divided into multiple modules
- Only the modules that are required by an application need to be deployed with that application for it to work, rather than the whole Java Runtime Environment (JRE).
- Different modules can use different versions of Java

The Core Java Packages

- The packages we have used so far (“java.lang” and “java.util”) both come from a module called “java.base”
- Since this module contains all the core packages of Java, we do not need to do anything for our code to access this module.
- The core Java packages in this module use the “java” folder and have one subfolder in the package name

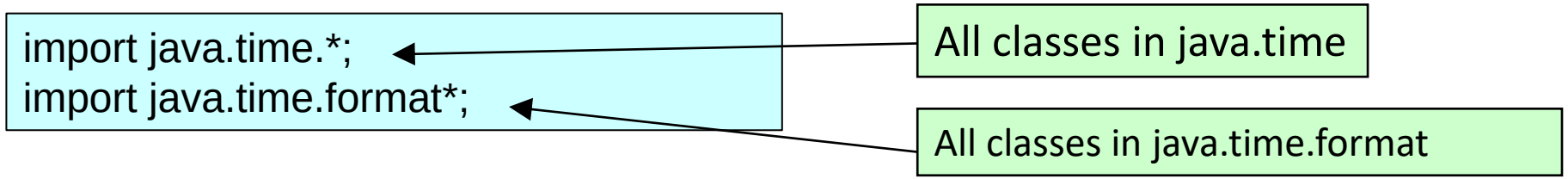
- java.io
- java.lang ← Does not have to be imported
- java.math
- java.net
- java.nio
- java.security
- java.text
- java.time
- java.util

java.base

- The “java.base” module also includes one “javax” (extension) package, “javax.net” (classes for networking applications).
- There are many sub packages. For example “java.time.format” is a sub-package of “java.time”

Subpackages

- Import statements in Java only apply to a specific package, they do not include any sub packages, so both would need to imported separately



- Eclipse will sort out your imports for you
  - In edit window pop up menu:
    - Source -> Organize Imports
  - Note: can have unwanted side effects!

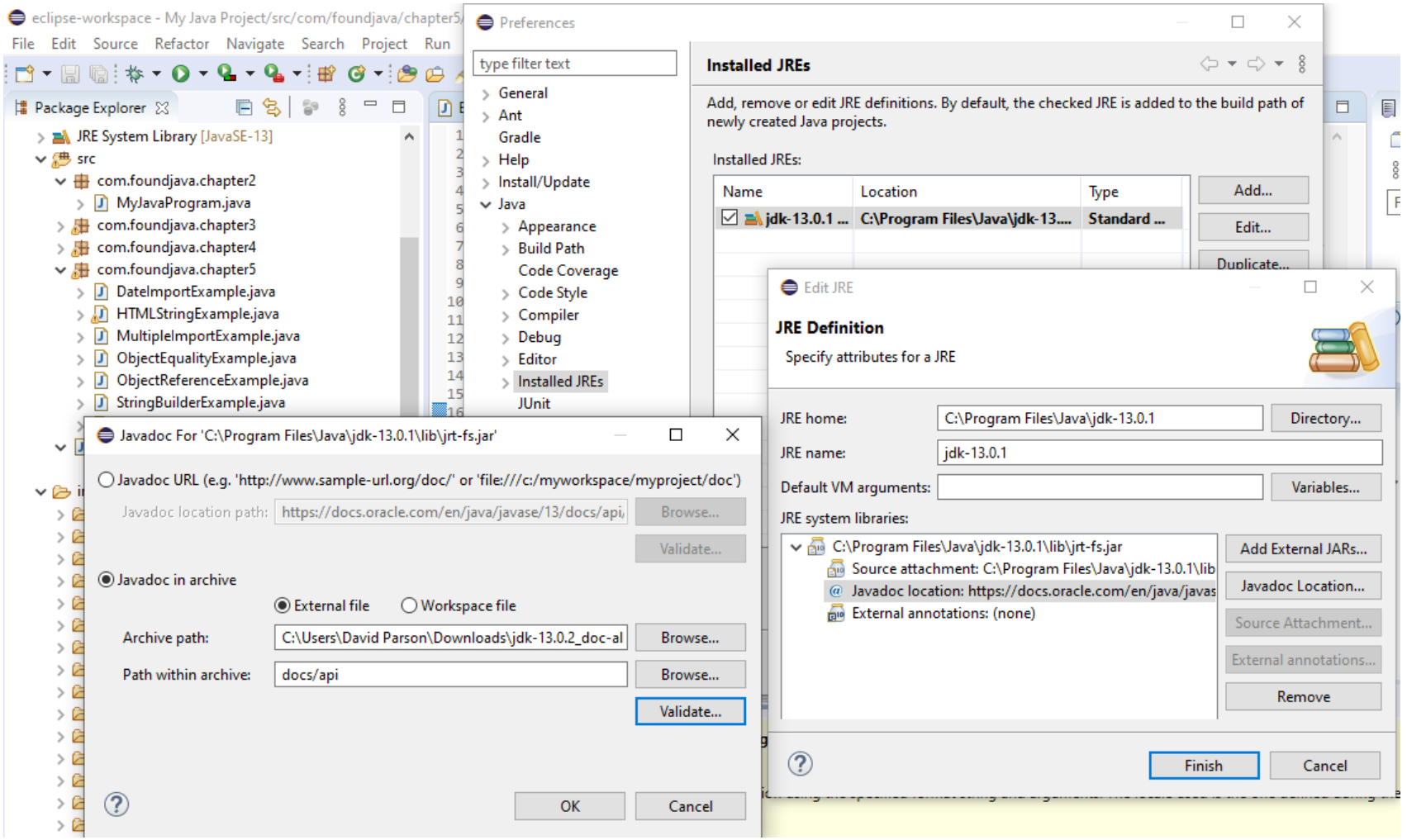
Exercise 5.3

- Create a class with a main method
- In ‘main’, create a new Date object. Use the necessary import statement
- Create a String object that contains the same date by using the toString method of the Date class
- Create another String object that contains the same date converted to uppercase
- Display the message “Today’s date is…” followed by your uppercase string using a single println statement

Using Javadoc

- To find out about the available Java classes you can use the documentation supplied
  - Can be viewed as local HTML pages
  - Separate download from JDK
- This has been created using the Javadoc tool
- Can be integrated into Eclipse by linking it with the installed JRE
  - Window -> Preferences

Javadoc in Eclipse



Exercise 5.4

- Create a class with a “main” method.
- The Point class within the “java.awt” package represents the “x” and “y” co-ordinates of a point in a two-dimensional space. In “main”, create a new ja-va.awt.Point object using the zero arguments constructor. Add the necessary “import” statement.
- Print the Point object on the console using “System.out.println” – what are its “x” and “y” coordinate values?
- Change both of its coordinate values and print it again (see the Javadoc for de-tails of which methods can be used to do this).
- Why do you think the “x” and “y” fields of Point objects are publicly accessible?

Exercise 5.5

- The Date class has a getTime method that returns (as a long integer) the number of milliseconds that have passed since January 1st 1970
- We can use this to calculate the current time by using the divide (/) and remainder (%) operators
  - there are 86,400,000 milliseconds in a day
  - there are 3,600,000 milliseconds in an hour
  - there are 60,000 milliseconds in a minute
- Use these to write some code that will tell you the current time

Summary

- Creation and use of objects and methods related to String data
  - String, StringBuilder, StringBuffer, Text Blocks
  - ‘toString’ method
- How objects are referenced in memory
- How some operators work with objects
- Garbage collection
- String immutability and concatenation
- Modules, packages and subpackages
- Importing classes and packages
- Javadoc