GETTING JAVA WORKING

Compiling and running your first Java program

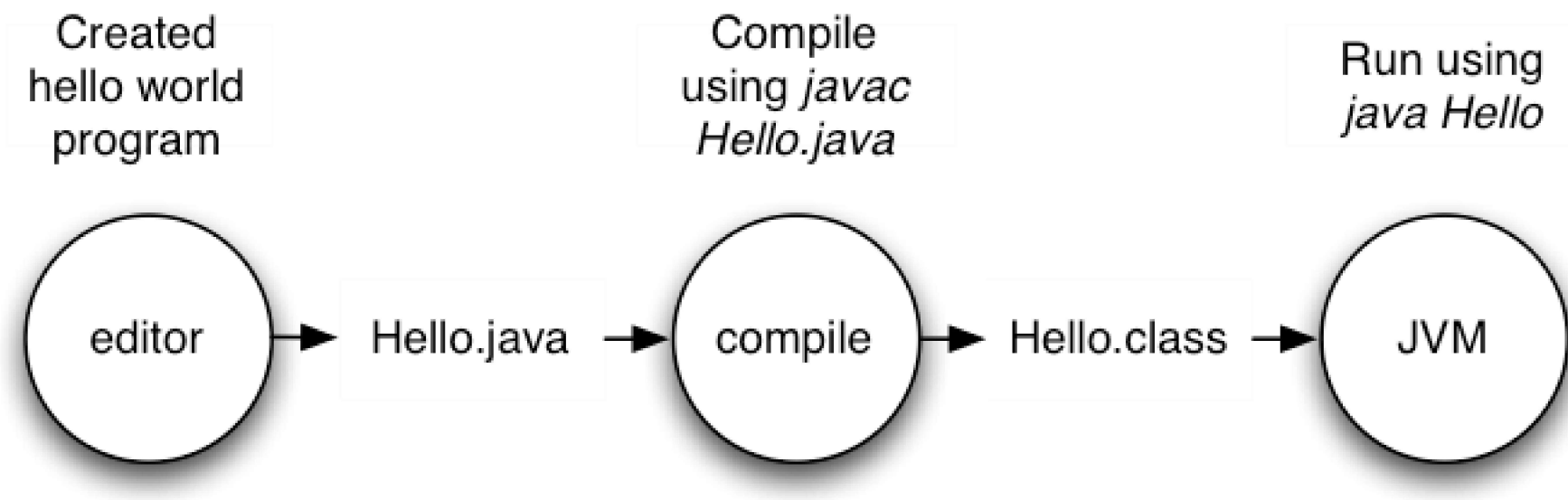
About Java

- Developed by Sun Microsystems (which is now a subsidiary of Ora first released in 1995
- Three main editions
  - Java ME - Micro Edition
  - Java SE - Standard Edition
  - Java EE - Enterprise Edition
- Version installed in our labs is Java SE 17
  - I recommend this for your own computer
- Source code is compiled to bytecode which is then run on a Java V (JVM) therefore platform independent

Installing Java on your own computer

- We recommend Amazon Corretto 17
  - https://aws.amazon.com/corretto/
  - "Download Amazon Corretto 17"
- Windows: find **Windows x64** and download the **".msi"** link
- Mac: find **macOS x64** and download the **".pkg"** link
- Linux: depends on your distribution
  - Various packages available on the Amazon Corretto page, but...
  - **sudo apt-get install openjdk-17-jdk** might be the quickest option

Edit/Compile/Run Cycle In Java



- The save and compile phases or the compile and run phases some single step within an IDE

First Java Program

- ‘Hello World’ in Java

```
public class HelloWorld {
    public static void main (String[] args) {
        System.out.println ("Hello World");
    }
}
```

- Must be saved in a file called **HelloWorld.java**
- Compile in terminal using

```
javac HelloWorld.java
```

  - Creates **HelloWorld.class**
- Run in terminal with

```
java HelloWorld
```

About HelloWorld

- We have created a Class
  - More on this later!
- The entry point to our program is the
  `public static void main (String[] args)`
  method
- The line `System.out.println(…);` prints out the text
  - "Hello World"
- Not much object orientation in HelloWorld

VARIABLES AND TYPES

The basics

Primitive vs reference

Variables

- A variable is a piece of information that we keep track of during th execution
- Variables are stored in memory
  - and can be changed as the program runs
- Before you can use a variable to store information
  - We must declare it
- Dealing with memory locations and addresses is too low level for need abstraction over the store
  - However, it is useful to remember this underlying memory model

Creating Variables

- Declaration
  - *type name*;
  - `int n;`
  - `boolean valid_input;`
- Initialisation (comes after declaration)
  - *name = initial value*;
  - `n = 17;`
  - `valid_input = true;`
- Declaration and initialisation
  - *type name = initial value*;
  - `int another_number = 8;`
  - `int sum = 5 + another_number;`
  - `String greeting = "hello";`

Primitive Types

Type name	What is it?	Size	E
boolean	true/false	1 bit	tr
byte	Signed integer	8 bits	7
char	Unicode character	16 bits	‘A
short	Signed integer	16 bits	4
int	Signed integer	32 bits	3
long	Signed integer	64 bits	3
float	Floating point number	32 bits	1
double	Floating point number	64 bits	1

See http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.htm

Type Ranges

- **boolean**
  - represents true or false using a single bit of information, 1 or 0.
- **byte**
  - 8 bit signed integer, 256 numbers, in the range -128 to +127 (-2^7 to 2^7-1)
- **short**
  - 16 bit signed integer in the range -32,768 to +32,767 (-2^{15} to $2^{15}-1$)
- **char**
  - 16 bit unsigned integer in the range 0 to 65535 (0 to 2^{16})
  - number is index of character in Unicode character table
    - 48 is ‘0’, 49 is ‘1’, 42 is ‘2’ … 65 is ‘A’, 66 is ‘B’, 67 is ‘C’ …
- **int**
  - 32 bit signed integer in the range -2,147,483,648 to +2,147,483,647 (-2^{31} to $2^{31}-1$)
- **long**
  - 64 bit signed integer in the range -9,223,372,036,854,755,808 to +9,223,372,036,854,
- All *signed integers* use two's complement representation

Type Ranges (cont.)

- Floating point numbers (IEEE 754)

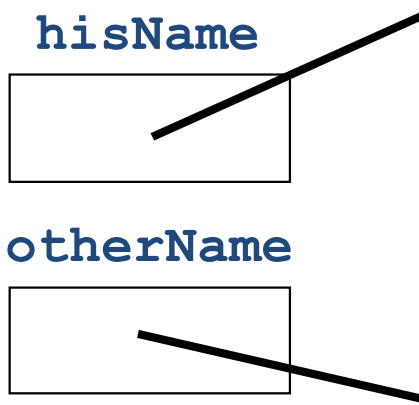
- **float**
  - 32 bit signed floating point number
  - in the range ±1.4e-45 to ±3.4e+38
    - commonly with 7 decimal values

- **double**
  - 64 bit signed floating point number
  - in the range ±4.9e-324 to ±1.8e308
    - commonly with 14 decimal values

Reference Types

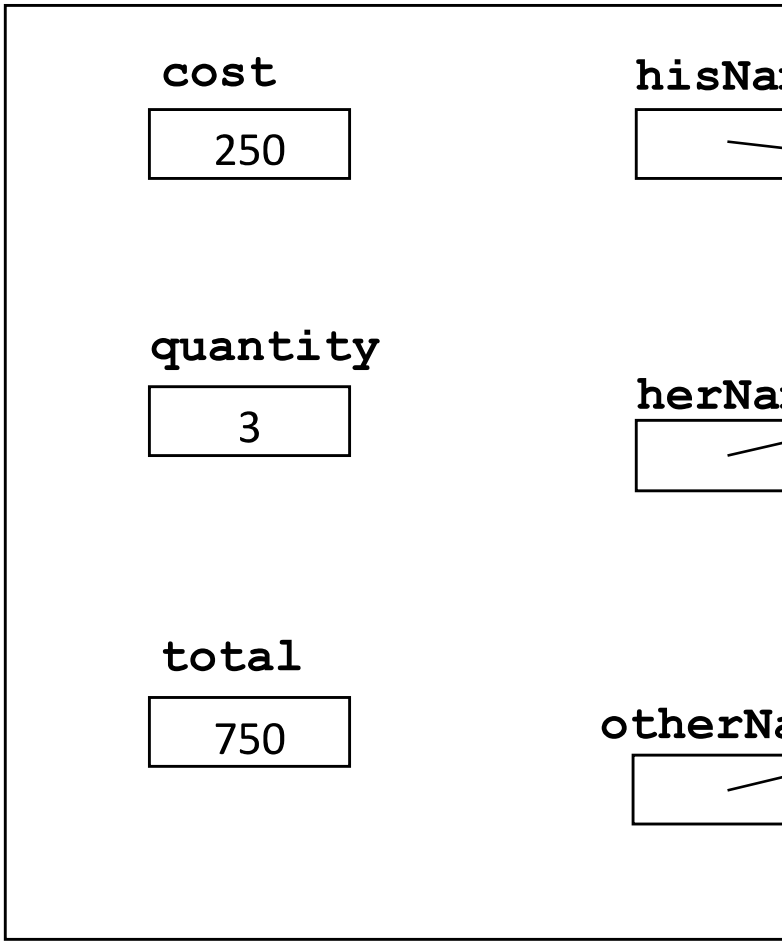
- Describe kinds of *objects*
- Many predefined reference types in Java
  - e.g. String
- You can define any number of your own
  - More on this later
- Commonly represented by arrows going from variable boxes
  - Variable contains a *reference* to the actual data
    - e.g. java.lang.String;@15db9742

```
String hisName = "Bob";

String otherName = "";
```



Representation in store

```
int cost = 250;
int quantity = 3;
int total = cost * quantity;

String hisName = "Bob";
String herName = "";
String otherName = null;
```



- Note: **null** is the default value for reference types so
  - `String otherName;` is equivalent to `String otherName = null`

METHODS

Methods

- Similar to functions, procedures, etc. from other languages

modifiers

return type

parameter list

```
public static int add (int first, int second) {
    return (first + second);
}
```

- Modifiers (for now)
  - public – means you can call the method from classes other than then one in which it i
  - static – means that the method belongs to the class and not an object (instance of the
    - you will learn more about this next week
- Return type
  - Any Java type can be used as the return type or void for nothing returned

See http://docs.oracle.com/javase/tutorial/java/javaOO/methods.html

Method Invocation

- To call (aka to *use* or to *invoke*) a method defined in the current cl

  <method_name> (<arguments>);

- To call a **static** (class) method defined in another class

  <class_name>.<method_name> (<arguments>);

- To call an **instance** (non-static) method associated with another o

  <object_name>.<method_name>(<arguments>);

Example static method invocation

- Static method within the same class

```
1.  // In HelloWorld.java
2.  public class HelloWorld {
3.      public static void printGreeting(){
4.          System.out.print("Hello World");
5.      }
6.
7.      public static void main (String[] args) {
8.          printGreeting();
9.      }
10. }
```

- Lines 3–5 define the static method **`printGreeting`**
- Line 8 calls the method **`printGreeting`**
  - This prints out the text “Hello World”

Example static method invocation (2)

- Static method in different class

```
1. // In HelloWorld.java
2. public class HelloWorld {
3.    public static void printGreeting(){
4.        System.out.print("Hello World");
5.    }
6. }
```

```
1. // In HelloMain.java
2. public class HelloMain {
3.    public static void main (String[] args) {
4.        HelloWorld.printGreeting();
5.    }
6. }
```

- Lines 3–5 in class **HelloWorld** define the static method `printGreeti`
- Line 4 in class **HelloMain** calls the method `printGreeting` defined
  - This prints out the text “Hello World”

Example instance method invocation

- Looking ahead to OO lectures next week:

```
1. // In HWPrinter.java
2. public class HWPrinter {
3.    public void printGreeting(){
4.        System.out.print("Hello World");
5.    }
6. }
```

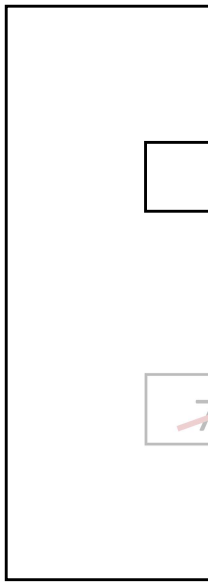
```
1. // In HelloMain.java
2. public class HelloMain {
3.    public static void main (String[] args) {
4.        HWPrinter printer = new HWPrinter();
5.        printer.printGreeting();
6.    }
7. }
```

- Lines 3–5 in class
  instance method
- Line 4 in class **Hel**
  instance (object)
  - `printer` stores
    permitting us to i
- Line 5 in class **Hel**
  printGreeting met
  HWPrinter
  - This prints out the

Method Parameters

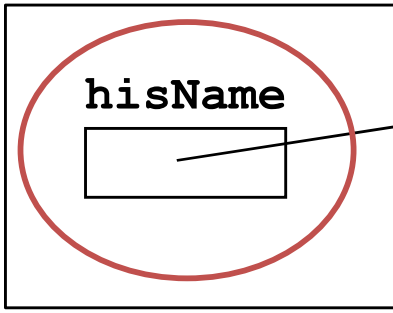
- In Java, parameters are passed to methods *by value*

```
public class PassByValue {
    public static void increment(int p1) {
        p1 = p1 + 1; // update p1
        System.out.println("inc: p1 = " + p1);
    }

    public static void main(String[] args) {
        int i = 71;
        increment(i); // call method with i
        System.out.println("main: i = " + i);
    }
}
```



Passing Objects to Methods

- Recall: For reference types (Objects) the variable contains a *refere* data
  - e.g. java.lang.String;@15db9742



- When an object is passed to a method
  - The value (i.e. the *reference*) is passed
- The Method can use the reference to access the object
  - can call its methods
  - can alter the contents

Passing Array Objects to Method

```
public static void printArray(int[] array) {
    System.out.print("[");
    for (int i = 0; i < array.length; i++) { System.out.print(" " + array[i]); }
    System.out.println(" ]");
}

public static void fillArray(int[] array) {
    for (int i = 0; i < array.length; i++) {
        array[i] = i; // use array reference to modify data
    }
}

public static void resetArray(int[] array) {
    array = new int[array.length]; // replace array reference
}

public static void main(String[] args) {
    int[] intArray = new int[10];
    fillArray(intArray);
    printArray(intArray);
    resetArray(intArray);
    printArray(intArray);
}
```

Output is:

[0 1 2 3 4 5 6 7 8 9]

[0 1 2 3 4 5 6 7 8 9]

STRINGS

Strings

- Strings are objects in Java
- They have instance methods that you can call
  - .length() – returns the length of string
  - .substring(int i) – returns substring starting at position i
  - .substring(int i, int j) – substring between positions i and j
  - .indexOf(String s) – returns index of first occurrence of s in this string, -1 if str
  - .contains(String s) – returns true if this string contains s
  - .equals(String s) – returns true if s has the same characters as this string
  - .split(String expr) – splits this string into an array of strings using the regular e

- Look at the API:
  - https://docs.oracle.com/en/java/javase/17/docs/api/
  - https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Stri

String Method Example

```
// Program code
String a = "Hello how are you?";
String b = a.substring(1, 3);

System.out.println(a);
System.out.println(b);
System.out.println(b.length());
System.out.println(a.substring(6));
System.out.println(a.substring(a.indexOf("are")));
```

Equality of Objects

- The == operator tests equality of value
  - Works as expected when used on primitives (int, float, etc.)
  - But remember: for reference types the value is the *reference* to the object, so
- When comparing objects you are best to use the methods provide
  - For example the String class has a
    `public boolean equals (String str)`
    method for testing equality
    - Use `if (string1.equals (string2)) { ... }`
      - This checks if the string objects contain the same characters
    - Do not use `if (string1 == string2) { ... }`
      - This checks if the variables reference *the same object*
  - `equals` is a standard method name for checking Object equality

CS5001 Object Oriented Modelling, Des
Programming

LIVE CLASS 1

Michael Young
School of Computer Science
University of St Andrews

Admin

- Michael Young
  - Lecturer for CS5001
  - Associate lecturer, School of Computer Science
  - cs5001-staff@st-andrews.ac.uk
- Kasim Terzic
  - Director of Postgraduate Teaching
  - Handles extensions, special circumstances
  - dopgt-cs@st-andrews.ac.uk
- Ian Miguel
  - Head of School
  - hos-cs@st-andrews.ac.uk

Who is Michael Torpey?

Wed 08/09/2021 14:20

To: Michael Young

Hi Michael,

I'm on this programme but have already had an advising appointment with Dr Torpey - do I need to book in with you as well?

Kind regards,

Flipped classroom

- Recorded Lectures
  - Entirely online – Panopto
  - “Core content”
  - Will come out in batches
- Live lectures (like these)
  - Reinforces recorded lectures
  - Interactive, with questions encouraged!
  - Not recorded
- Exercise classes
  - One per week
  - Fun exercises to practise your skills, not assessed
  - Live support from lab demonstrators
- Benefits
  - Hands-on engagemen
  - Makes best use of face
  - More communication
  - More time spent with
- Drawbacks
  - Requires self-discipline
  - Risk of technical hiccu
  - More time spent with

Discourse

- Discourse: forum for online discussion
  - Hosted by the School
  - Log in with your **CS username/password**
  - https://discourse.cs.st-andrews.ac.uk/c/CS5001
- Lectures will be accompanied by a Discourse thread
  - I'll email out the link
  - Use this thread to ask questions and discuss the material
  - I'll check periodically to answer questions and moderate
  - Read what others have asked, in case your question has been answered
- I won't answer questions on Microsoft Teams
  - You can always email me

Exercise classes

- Each week, one hour slot in the lab
- Exercise sheet posted on StudRes
  - In advance of the session
- Work through the sheet and ask questions
  - Lab demonstrators and I will be present
- Questions go from easy to very hard
  - Don’t worry if you don’t finish them all
- Not handed in

Timetable

	Monday	Tuesday	Wednesday	Thursday
Week 1		11:00 Purdie B		
Week 2		11:00 JH lab		
Week 3		11:00 Purdie B		
Week 4		11:00 JH lab		
Week 5		11:00 Purdie B		
Week 6	INDEPENDENT LEARNING WEEK			
Week 7		11:00 Purdie B		
Week 8		11:00 JH lab		
Week 9		???		

Exemption test

- If you are already very familiar with Java and OOP, you can skip th
- Exemption test is available – you must pass in order to skip
- Available shortly after this session
  - Microsoft Forms link sent to everyone by email
  - Should take around one hour
- Do not look up answers online!
  - We can't check this, but if you cheat you may find later modules very difficult
  - Your responsibility to ensure you are capable of passing
- Deadline: Thu 15th September, 17:00
  - Test will not be available after this time
- If successful, contact your adviser to re-advise onto another modu

Coursework

- Assessment via coursework (no exam!)
  - There are 4 assessed programming assignments
    http://studres.cs.st-andrews.ac.uk/CS5001/Coursework/
- Practicals are marked on:
  - Implementation of all the requirements
  - Correctness of the implementation
  - Quality of the code
  - Efficiency of the solution
  - Correct use of object orien
  - Originality of the solution
- Automated checking is provided where appropriate
  - For most practicals, you will be provided with a set of basic tests to test your program
- Coursework marked in accordance with School marking guidelines
  - Descriptors are in the Student Handbook:
    https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html

Practical 0: Hello world!

- Available immediately after this session
- Deadline this Friday
- Worth 0% of your final grade (don't worry!)
- Very simple code, based on code in recorded lecture
- Helps you understand our systems
  - Lab machines
  - Automated checker (stacscheck)
  - MMS
- Work on it in your own time
  - Help available in exercise class this Friday

Software

- Operating System
  - We recommend using Linux (e.g. for practical work)
    - Available on lab computers
      - Connecting to lab computers remotely: https://systems.wiki.cs.st-andrews.ac.uk/index.php/W
    - You can use Windows if you prefer but Automated Checking is designed for Linux!
    - Mac OS is similar to Linux

- Writing, compiling, and running Java programs
  - Write program using any text Editor (e.g. Emacs, Notepad++, Gedit)
    - Compile and run programs in Terminal window
  - Write program using an Integrated Development Environment (e.g. VSCode, Eclipse, I
    - Compile and run programs within IDE
    - Good support for Java programming

- Writing reports
  - Microsoft Office or Open/LibreOffice or LaTeX
  - Submit a PDF

Useful information

- http://docs.oracle.com/javase/tutorial/java/nutsandbolts/
- http://www.tutorialspoint.com/java/
- http://studres.cs.st-andrews.ac.uk/CS5001/
- http://academicjava.com/

Checkpoint 1

- At this point you should understand:
  - Basic Java syntax
  - Compiling and running Java programs
    - From command line
  - How to produce basic output from your applications
  - Variables & Types
    - Including the difference between reference types and prim
  - Methods, method calls
  - Strings

Pick a type!

- number_of_birds
- fuel_remaining
- size_of_class
- name_of_president
- population_of_scotland
- money_in_account
- date_of_birth
- favorite_emoji
- world_population
- movie_star_rating
- correlation_coefficie
- x
- shoe_size
- phone_number
- hair_color
- full_text_of_war_an
- efficiency_rating
- hourly_temperature
- all_names
- students

Types

- Primitive vs reference
- Size of variables

OPERATIONS

Common operations

- Integer operators
  - Everyday operators:
    - + - * addition subtraction multiplication 273+45-9*5
  - Less usual operators:
    - / division 23 / 5 = 4
    - % remainder 23 % 5 = 3
- On double:
    - / division 23.0/5.0 = 4.6
- On String:
    - + concatenation "the" + " " + "end" = "the end"
      " " + "anything" = " anything"
- Operator Precedence: () * / % + -

See http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op1.html

Comparison & Boolean Operators

Expression	Name
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal
==	equal
!=	not equal
&&	and
||	or
!	negate

See http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op2.html
Operator Precedence: http://docs.oracle.com/javase/tutorial/java/nutsandbo

Bitwise operations (on binary numbers)

Operator	Name	Example	Resul
a & b	AND	3 & 1 = b11 & b01	b01 =
a ^ b	XOR	3 ^ 1 = b11 ^ b01	b10 =
a | b	OR	3 | 1 = b11 I b01	b11 =
a >> b	signed right shift	2 >>1 = b010 >>1	b001 =
a << b	signed left shift	2 <<1 = b010 <<1	b100 =
~a	unary bitwise compliment	~((byte)127) = ~b01111111	b100000 = -128

See http://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3

CHOICE AND ITERATION

if

while

do-while

for

Choice

- Makes a decision

```
if (x == 10) {
   ...
} else if (x > 10) {
   ...
} else {
   ...
}
```

boolean condition

No specified condition

- Conditions are checked in turn
  - Only one branch is executed
  - `else` is executed if none of the conditions are true

See http://docs.oracle.com/javase/tutorial/java/nutsandbolts/if.html

Iteration – while loop

- Perform an action, 0 or more times

```
while (true) {
    ...
}
```

```
while (x==y) {
    ...
}
```

```
while ((x < 10) && (y > 10) || (z == 23)) {
    if (k == 16) {
        break;
    }
}
```

- Beware the infinite loop!
  - Make sure the value of your condition is affected by the body of the loop
  - or that your loop has a break that can be reached

See http://docs.oracle.com/javase/tutorial/java/nutsandbolts/while.h

Iteration – do-while loop

- Perform an action, 1 or more times

```
do {
    ...
} while (true);
```

```
do {
    ...
} while (x==y);
```

```
do {
    if (k == 16) {
        break;
    }
} while ((x < 10) && (y > 10) || (z == 23));
```

See http://docs.oracle.com/javase/tutorial/java/nutsandbolts/while.h

Iteration – for loop

- Iterate over a number of statements
  - Most commonly used when you know the number of times to iterate

Declare and initialize loop variable

boolean condition

Update loop variable

```
for (int i = 0; i < 10; i++) {
    ...
}

int number = 12;
for (int i = 0; i < number; i = i + 2) {
    System.out.println(i);
}
```

See http://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html

ARRAYS

Arrays

- Arrays are Objects in java
  - Fixed length collection of items
  - Can be applied to any java object or primitive
- Declared using square brackets after the item type, e.g.
  - `int[] myQuantities; float[] somePrices; Object[] the`
- Initialised with the keyword ***new*** and the size of the array, e.g.
  - `int[] numbers = new int[5];`
    - ***`numbers`*** holds a reference to array of 5 integers (all with default value of 0)
  - `Object[] myObjects = new Object[100];`
    - ***`myObjects`*** holds a reference to an array of 100 object references (all with defau
- Arrays have a length property so
  - `numbers.length` contains the number of elements in the numbers array

See http://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Array Example

- Create an array containing the first 10 elements of the Fibonacci s

```
int[] fib = new int[10];
fib[0] = 0;
fib[1] = 1;
for (int i = 2; i < fib.length; i++){
    fib[i] = fib[i - 2] + fib[i - 1];
}
```

0	1	2	3	4	5	6	7	8	9
0	1	1	2	3	5	8	13	21	3

Array Example: args

- Recall: The entry point to our program is

```
public static void main (String[] args)
```

- **`args`** is an array of the command line arguments provided when

- 'Hello Arg' in Java

```
public class HelloArg {
    public static void main (String[] args) {
        System.out.println ("Hello " + args[0]);
    }
}
```

- Compiled in the usual way
- Run in terminal with, e.g.

```
java HelloArg Bob
```

Outp
He

for-each

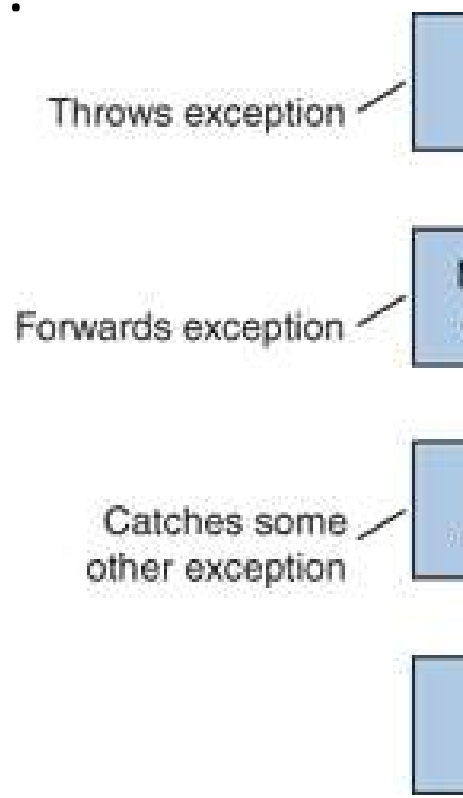
- Iterate over items in a collection

```
/* given some String[] anArray of some length */
for (String str: anArray) {
   System.out.println ("The string is " + str);
   ...
}
```

EXCEPTIONS

Exceptions

- Shorthand term for the phrase ‘exceptional event’.
- Exception occurs within a method
  - Object which describes the exception is created
- Exception Object passed up the stack looking for an exception handler
  - If no handler is found then the program terminates
- Allows normal functionality to be considered separately from exceptional behaviour
  - Separates exception handling from ‘real’ code



Exception handling

- A method or class can *throw* an exception
- An exception can be *caught* or passed up the call stack
- Exceptions are handled using a *try-catch* block

```
try {
    // your application code here
    // only executes up to the statement that generates an e
    // or until the end of the block if no exception occurs
} catch (IOException ioe) {
    // code here is executed on IOException only
} catch (Exception e) {
    // code here is executed on any other Exception
} finally {
    // code here is executed whether exception occurred or n
}
```

DOCUMENTATION

Documentation

- Java uses a tool called javadoc for source code documentation
- Can create documentation in a variety of formats such as
  - HTML - the default and industry standard
  - PDF
  - Plain text
  - XML
- Extends the commenting system with special tags
- Provides human readable information
- Annotations[*] provide machine readable information
  - Can be used for all sorts of wonderful tricks

See http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html

[*] http://docs.oracle.com/javase/tutorial/java/annotations/index.html

Javadoc - A Class

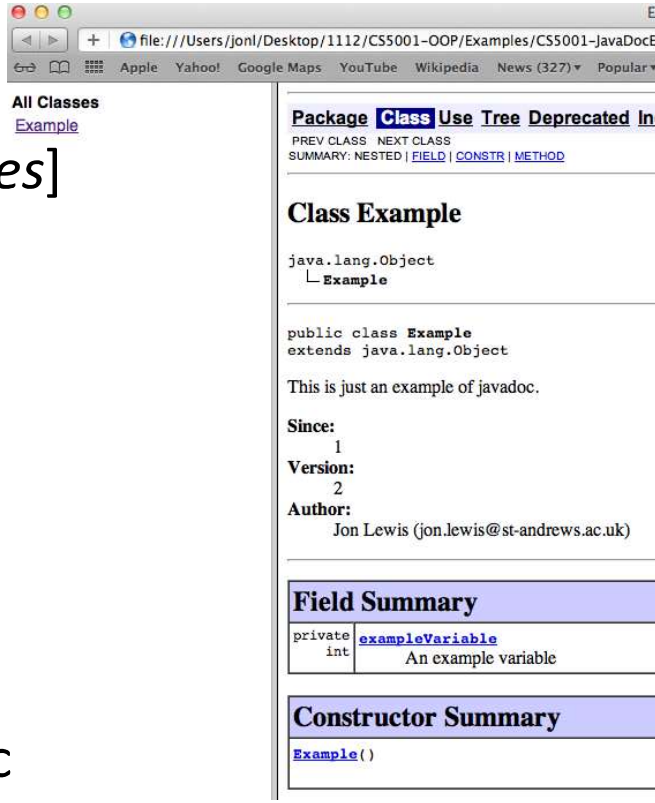
```
/**
* An example class to illustrate javadoc.
* @author CS5001 Lecturer <cs5001-staff@st-andrews.ac.u
* @version 2
* @since 1
*/
public class Example {
    /**
    * A variable created for a CS5001 lecture
    */
    private int exampleVariable;
}
```

Javadoc - A Method

```
/**
* Adds two numbers together and returns the result
* @param x First number
* @param y Second number
* @return The sum of x and y
* @throws MathException If there is a problem adding nu
*/
public int add(int x, int y) throws MathException {
    return x + y;
}
```

Generating Javadoc Webpages

- Command line:
  - javadoc {*packages*|*source-files*} [*options*] [@*argfiles*]
  - e.g. ***javadoc src/*.java -d doc***
  - Documentation will appear in the *doc* directory



- In IDEs:
  - There may be built-in support!
  - Eclipse: File > Export > Java > Javadoc
  - You may need to tell specify where to find Javadoc
    - Windows: C:\Program Files\Java\jdk-x.x.x\bin\javadoc.exe
    - Linux: /usr/local/jdk/bin/javadoc

See https://www.oracle.com/technical-resources/articles/java/javadoc-tool.htm

STYLE

Good practices for your code

Code Style, Good Practices

- Makes no difference to the computer compiling & running your co
- Makes a huge difference to developers
  - Code is easier to read
  - Code is easier to understand
  - Code is easier to maintain
  - Mistakes are easier to spot

- For this module we use an in-house style known as the *Kirby* style
  https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/program
  - Assignments will be checked against this style (as part of automated checking)
  - You should try to eliminate all style warnings prior to submission of assignmen
    - Some style warnings are not *too* important (e.g. you might not need javadoc string
    - Some style warnings are more important (e.g. indenting code, or using meaningful

Code Style, Good Practices: Naming Conven

Identifier Type	Rules for Naming	Examples
Packages	**Lowercase** Top-level domain names/country code followed by your organization's own internal naming conventions.	com.sun.eng uk.ac.standrews.cs
Classes & Interfaces	**Pascal Case** (aka upper camel case) Use whole words – avoid acronyms and abbreviations. Use appropriate plurality	class Raster; class ImageSprite; interface RasterDel

Methods	**Camel Case** (aka lower camel case or mixed case)
	Use verbs.
		run();
		runFast();
		getBackground();

Variables	**Camel Case** (aka lower camel case or mixed case) Do not start with underscore _ or dollar sign $ Short yet meaningful. Only use one-character names for temporary "throwaway" variables.	int i; char c; float m
Constants	**Uppercase** with words separated by **underscores** ("_").	static final int MIN_ static final int MAX_

CS5001 Object Oriented Modelling, Des
Programming

LIVE CLASS 2

Michael Young
School of Computer Science
University of St Andrews

Checkpoint 2

- At this point you should understand:
  - Choice, Iteration
    - if, while, do-while, for, for-each
  - Arrays
    - they're fixed type and fixed length
  - Methods, parameters, pass-by-value
  - String objects and some of their methods
  - Exception handling
    - try, catch, exception types
  - Javadoc
    - inserting documentation into your source code

Reading

- Bruce Eckel, Thinking In Java - http://www.mindview.net/Books/TIJ/
  - Online electronic copy of 3rd Edition is free
  - https://studres.cs.st-andrews.ac.uk/CS5001/ThinkingInJava/
- Java in Two Semesters. Charatan, Q; Kans, A. 2009
  - Earlier editions are also suitable, although it does not cover features of more recent Ja
- Head First Design Patterns
  - by Eric Freeman & Elisabeth Robson
  - Advanced object-oriented practices (good for later in the course)
- Extra resources:
  - http://resourcelists.st-andrews.ac.uk/modules/cs5001.html
  - http://www.tutorialspoint.com/java/
  - http://docs.oracle.com/javase/tutorial/java/nutsandbolts/

API

- The Java Application Programming Interface (API)
  - Shows javadoc for entire Java implementation
- https://docs.oracle.com/en/java/javase/17/docs/api/
- Use it!
- (It's not as scary as it looks)