

Intro

30 September 2019 11:01

<https://warwick.ac.uk/fac/sci/dcs/teaching/modules/cs118/>

Monday 11am I3

Friday 9am

R0.21

Seminar 1x per week

Two pieces of coursework

- Tuesday week 5
- Thursday week 10

Optional problem sheets -wk 3,5,8,10 (chance to get some extra guidance)

Exam

Week 1 or 2 of Term 3 (60%)

Coursework (40%)

Introduction to Java Programming – Y Daniel Liang

Exam Questions

Floating point notation

- Primitive Data types and control statements
- Iterative statements methods arrays and recursion
- Object oriented programming
- Inheritance, Abstract classes and interfaces
- Exceptions and Generics

About Java

18 September 2019 12:07

- Java is Object Oriented. Everything is an Object.
- Java is platform independent. When Java is compiled, it is not compiled into platform dependent machine code, but instead into platform-independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run.
- Java is architecture-neutral and the compiler is written in ANSI C and is therefore highly portable.
- Java is interpreted

A Java program is a collection of Objects that communicate via invoking each other's methods.

- **OBJECTS** – Have states and behaviours. An Object is an instance of a class
- **CLASS** – Can be defined as a template/blueprint that describes the behaviour/state that the objects of its type support
- **METHODS** – A behaviour. A class can contain many methods. It is in methods where the logic is written and data is manipulated
- **INSTANCE VARIABLES** – Each Object has its unique set of instance variables.

Java Code -> compile -> Bytecode -> interpret -> Machine Code

Can distribute bytecode without giving user source code. Bytecode can be decompiled

#methods are a named collection of statements that perform some task

Blocks are sections of code delimited by braces

Statements are ended by semi-colons

Readability

Comment

Open a brace. Indent the next line

Running java

Javac file.java

Java file (file.class)

Source File Declaration

18 September 2019 13:06

Source File Declaration Rules

There are some essential rules when declaring classes called source file declaration rules.

- There can only be one public class per source file.
- A source file can have multiple non-public classes
- The public class name should be the name of the source file which should be appended by .java e.g. `public class Employee{}` - source file name would be `Employee.java`
- If the class is defined within a package, then the package statement should be the first statement in the source file
- If import statements are present then they must be written between the package statement and the class declaration. If there are no package statements then the import statement should be the first line in the source file
- Import and package statements will imply to all the classes present in the source file. It is not possible to declare different import and package statements to different classes in the source file.

Creating Objects

18 September 2019 13:21

Programming in object oriented languages is an alternative way of thinking about problems. Organising code so that the data and the operations of that data are bundled together. Object types are just extended types.

```
public class Puppy {
    int puppyAge;

    public Puppy(String name) {
        // This constructor has one parameter, name.
        System.out.println("Name chosen is : " + name );
    }

    public void setAge( int age ) {
        puppyAge = age;
    }

    public int getAge( ) {
        System.out.println("Puppy's age is : " + puppyAge );
        return puppyAge;
    }

    public static void main(String []args) {
        /* Object creation */
        Puppy myPuppy = new Puppy( "tommy" );

        /* Call class method to set puppy's age */
        myPuppy.setAge( 2 );

        /* Call another class method to get puppy's age */
        myPuppy.getAge( );

        /* You can access instance variable as follows as well */
        System.out.println("Variable Value : " + myPuppy.puppyAge );
    }
}
```

Constructors

Every class has a constructor. If you do not explicitly write a constructor, the Java compiler builds a default constructor for that class.

Each time a new object is created, the constructor is invoked. They must have the same name as the class. A class can have more than one constructor.

```
// A simple constructor.
class MyClass {
    int x;

    // Following is the constructor
    MyClass(int i ) {
        x = i;
    }
}
```

You would call constructor to initialize objects as follows –

```
public class ConsDemo {
    public static void main(String args[]) {
        MyClass t1 = new MyClass( 10 );
        MyClass t2 = new MyClass( 20 );
        System.out.println(t1.x + " " + t2.x);
    }
}
```

Data Types

18 September 2019 13:21

Byte

8 bit signed two's complement int

Min val -128, Max val 127

Default 0

A byte is 4x smaller than an int

Short

16 bit signed two's complement integer

Min value is -2^{15} , Max value $2^{15}-1$ inclusive

2x smaller than int

Default 0

Int

32 bit signed two's complement integer

Min -2^{31} , Max $2^{31} - 1$ inclusive

Long

64 bit signed two's complement integer

Min -2^{63} , Max $2^{63} - 1$

Used when wider range is needed than int

Float

A single precision 32 IEEE 754 floating point

Used mainly to save memory in large arrays of floating point numbers

Never used for precise values such as currency

Double

A double precision 64 bit IEEE 754 floating point

Generally used as the default data type for decimal values

Never used for precise values such as currency

Boolean

Represents 1 bit of information

Only true or false, 0 or 1

Simple flag to track true or false conditions

Defaults to false

Char

Single 16-bit unicode character

Min 0, Max 65,535 inclusive.

User to store any character.

-128	64	32	16	8	4	2	1
1	1	1					

More Data Types

18 September 2019 13:28

Reference Datatypes

- Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, Employee, Puppy, etc.
- Class objects and various type of array variables come under reference datatype.
- Default value of any reference variable is null.
- A reference variable can be used to refer any object of the declared type or any compatible type.
- Example: `Animal animal = new Animal("giraffe");`

Java Literals

A literal is a source code representation of a fixed value. They are represented directly in the code without any computation.

Literals can be assigned to any primitive type variable. For example —

```
byte a = 68;  
char a = 'A';
```

byte, int, long, and short can be expressed in decimal(base 10), hexadecimal(base 16) or octal(base 8) number systems as well.

Prefix 0 is used to indicate octal, and prefix 0x indicates hexadecimal when using these number systems for literals. For example —

Java also supports the following special escape sequences for String and Char literals:

Notation	Character represented
<code>\n</code>	Newline (0x0a)
<code>\r</code>	Carriage return (0x0d)
<code>\f</code>	Formfeed (0x0c)
<code>\b</code>	Backspace (0x08)
<code>\s</code>	Space (0x20)
<code>\t</code>	tab
<code>\"</code>	Double quote
<code>\'</code>	Single quote
<code>\\</code>	backslash
<code>\ddd</code>	Octal character (ddd)
<code>\uxxxx</code>	Hexadecimal UNICODE character (xxxx)

Variables

18 September 2019

13:36

```
int a, b, c;           // Declares three integers, a, b, and c.
```

```
int a = 10, b = 10;   // Example of initialization
```

```
byte B = 22;          // initializes a byte type variable B.
```

```
double pi = 3.14159;  // declares and assigns a value of PI.
```

```
char a = 'a';         // the char variable a is initialized with value 'a'
```

Local Variables

- Local variables are declared in methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor, or block.
- Local variables are implemented at stack level internally.
- There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

Instance Variables

- Instance variables are declared in a class, but outside a method, constructor or any block.
- When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.
- Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.
- Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. *ObjectReference.VariableName*.

Class/Static Variables

- Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.
- Static variables are stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.

Modifier Types

18 September 2019 14:21

Access Control Modifiers

Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The four access levels are –

- Visible to the package, the default. No modifiers are needed.
- Visible to the class only (private).
- Visible to the world (public).
- Visible to the package and all subclasses (protected).

Non-Access Modifiers

Java provides a number of non-access modifiers to achieve many other functionality.

- The *static* modifier for creating class methods and variables.
- The *final* modifier for finalizing the implementations of classes, methods, and variables.
- The *abstract* modifier for creating abstract classes and methods.
- The *synchronized* and *volatile* modifiers, which are used for threads.

Operators

18 September 2019 14:22

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	A + B will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10
* (Multiplication)	Multiplies values on either side of the operator.	A * B will give 200
/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

Operator	Description	Example
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Operator	Description	Example
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
(logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A B) is true
! (logical not)	Called Logical NOT Operator. Use to reverse the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
=	bitwise inclusive OR and assignment operator.	C = 2 is same as C = C 2

Conditional Operator (? :)

Conditional operator is also known as the **ternary operator**. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide, which value should be assigned to the variable. The operator is written as –

variable x = (expression) ? value if true : value if false

Machine Code and Assembly

30 September 2019 11:30

Mc is processor specific and therefore so is assembly
Standards have become more common however

Assembly -> assemble -> Machine Code

Programs had to be e-written for every new machine
There was a need for machine independent languages
Fortran was the first widespread high level language

High Level -> compile -> Assembly -> assemble -> Machine Code

There are lots of different high level programming languages

- Evolution
- Special purposes – e.g. Prolog is good for representing logical relationships
- Personal preference
- Economics – e.g. IBM supported Cobol

Declarative:

- Functional – Lisp, ML, Haskell
- Dataflow – Id, Val
- Logic, Constraint – Prolog

Imperative

- Von Neumann – FORTRAN, C
- Object Oriented – Java, C#

Variables, IO and Number Systems

07 October 2019 15:28

Points to consider:

1. Precondition
2. Input
3. Calculation
4. Output
5. Postcondition

Taking Input

Using Scanner (java.util.scanner):

```
Scanner sinput = new Scanner(System.in);  
Int myNumber = sinput.nextInt();
```

Output in Java

```
System.out.print("output");  
System.out.println("hello");
```

When using +, remember that if both variables are integers, then the system will add.
If one or more are strings then the variables will be concatenated.

IEEE 754

The **IEEE Standard for Floating-Point Arithmetic (IEEE 754)** is a technical standard for floating-point arithmetic established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE).

$(-1)^{\text{signbit}} * 1.x * 2^y$

Where x is the fraction and y is the exponent

Special Values

Exponent = 0000 0000

Fraction is -

Value of the floating point is + or - 0

Doubles have a 52 bit mantissa and an 11 bit exponent

Be careful about mixing between int and float as the int will cause the float to lose precision
This can be useful, for example for getting the whole part of a fractional number

To force change a variable type you can use explicit casting

IMPLICIT CASTING (Widening) (target type larger than the source type)

EXPLICIT CASTING (Narrowing) (target type narrower than source type)

Byte->short->int->long->float->double

Char & Boolean

14 October 2019 11:12

There are 8 primitive types in Java. 6 of them are numeric types.
The remaining two are char and Boolean

Char is used to store characters, and is equivalent to a short in terms of bits, but is unsigned. Values range from 0 to 2^{16}

Boolean operators: `&`/`&` `|`/`|` `^` `!`

Lazy and Strict Operators

For logical AND and logical OR there are two variants, `&&` and `||`

Two means lazy and one means strict

This distinction only matters if you're changing something on the right side of the operation, e.g.

```
int a = 5;
boolean b = false;
if (b && (a++ == 5)) { ... }
```

B is false, so the whole statement is false, so the code in the braces never gets executed. After execution, A is STILL 5

If we use a single `&` symbol (Strict operator) then despite the statement being false, the whole code still gets executed.

With lazy operators, java knows the statement is false and so does not bother evaluating the rest of the argument, and so a is never incremented.


With strict operators, java is forced to evaluate both sides of the argument before deciding the truth value

Pre and Post-Increment

`l++` adds the value of l and then increments it. `++l` will increment first and then add the new value.

Operator Precedence

PUMARESLA

- Postfix (expr++ expr--)
- Unary (++expr --expr)
- Multiplicative (* /%) 
- Additive (+ -)
- Shift (<< >> >>>)
- Relational (< > <= >=)
- Equality (== !=)
- Strict Operators (& | ^)
- Lazy Operators (&& ||)
- Assignment (= += -= *= /=)

Encapsulation and Modularisation

04 November 2019 11:10

Public – can be accessed outside of the class

Private – can only be accessed inside the class

The constructor is public so that you can create instances of the class from outside, rather than having to be inside the class to construct objects.

Encapsulation

Forces users to use a class by its external interface, allowing for the private variables to be controlled carefully

E.g. the `BankAccount.Balance` variable wouldn't be publicly accessible, else it could be manipulated easily. It is instead only able to be changed from within the class, meaning the relevant rules must be followed

Setters and Getters

When using data hiding we will want to access instance variables

Use setters and getters

`SetX(int x)` and `getX()` methods

These are also known as mutator and accessor methods

If you want a variable to be shared between instances of a class then use the `static` keyword

It does not need an instance of a class to be used

E.g. when using `Math.round` you don't need a `math` object first. Its methods are static

Whereas `Random` doesn't have static methods and so you need to create a new `Random` object before using any of its methods

Variables that never change should be `final`

Make everything private unless there is good reason for it not to be

Enumerated types allow us to give string values numerical backing. The `enum` gives us a keyword that is human readable

Polymorphism

11 November 2019 11:52

The ability to process objects through a single uniform interface

E.g. we treat every class that extends our class as the original class, so that they can all be stored in an array of the same objects

Static Polymorphism

When we use method overloading, java decides on compile time which method to call based on the parameters provided

Dynamic Polymorphism

Comes into play with objects

You do not want to try and access a subclass and cast it the wrong class

Say we have a class like Cat

And two subclasses Lion and Tiger

If we called the .purr method

Both variables are just declared as cat objects

But java resolves their types at run time, realising one is a tiger and one is a lion subclass, and so calls the correct specific methods

Abstract Classes

15 November 2019 09:30

Guarantees that a class will exist when it is extended

We define an abstract class with various empty functions. This class can be extended and the subclasses will define the function.

Each subclass becomes more specific than its parent class

- Each superclass is therefore less specific
- Eventually they become so general that they become abstract
- We call these most general classes abstract classes
- Abstract classes are special classes that cannot be instantiated
- They allow us to capture common properties and behaviours at an abstract level

Abstract classes are just like normal classes but with a few caveats

- ▶ They cannot be initialised
- ▶ They can contain a mix of abstract methods and concrete methods
- To define one, you use the abstract keyword in the class declaration

Example

We could define an abstract class geometric shape – we would never initialise this class alone but we would extend it with subclasses like square

Interface

Eventually, as we abstract further, we will end in a situation where abstract classes contain only abstract methods

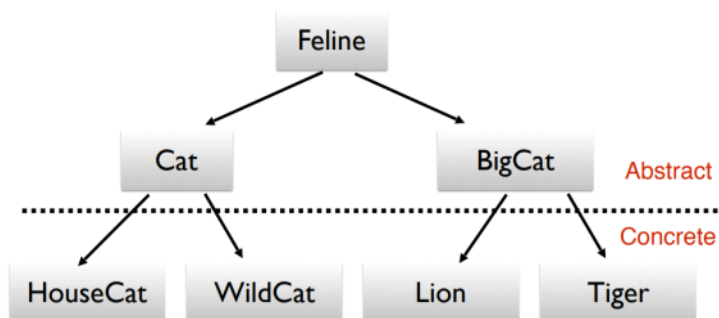
The class is now simply describing the interface each implementing class should use.

For these kind of class, we write interface

Interfaces are defined like a class but use the interface keyword

They only contain methods and the methods cannot be implemented.

Public class testImplementation implements Test { ... }



Inheritance

11 November 2019 11:20

We have base classes that are called superclasses, which contains methods needed by every subclass that may occur

Subclasses are derived classes that take the basic features of the superclass and add their own.

Inheritance is one of the biggest motivators for object oriented programming.

For example, there are many different types of bank account – ISA, savings, current. These share many of the same features but have a few that are unique. Therefore it would be useful to inherit the basic functions from the superclass.

With class inheritance we use method overriding, where the subclass version of the method overrides the version in the superclass

This – represents the current instance of the class

Super – refers to the instance of the superclass

So we can call the superclass constructor and relay constructor parameters

Super() must be the first thing you do in the subclass constructor
If you don't do it, java will call the default no argument constructor
If the superclass doesn't have a no argument constructor, compile error.

Error Handling

26 November 2019 16:33

When taking user input, use a try-catch to avoid unexpected crashes.
These errors are known as exceptions

```
Try {  
    // Exception code  
} catch (exception type) {  
    // exception handling code  
}
```

At the end of the try-catch we can use "finally" to clean things up e.g. close streams
Finally is always executed regardless of whether a return statement is used.
This is the only instance in which code can be executed after a return statement

There are 2 types of Exception:

Checked Exception	Unchecked Exception
Must be caught or rethrown	Extends error or runtime
Rethrown means the exception is passed up	Can be caught
	An example is a null pointer – this can't be caught at compile time. This is an area of active research – trying to catch runtime errors at compile time.
	Big errors in logic exist if these errors occur. The code will still compile

Unchecked exceptions are error or runtime exceptions
Shows that there is something more fundamentally wrong in the program
Something is in violation of the rules that have been setup

ArrayIndexOutOfBoundsException **NullPointerException**

Exceptions are just classes that extend the Exception class
Choose a specific exception class that exists that encapsulates your exception and extend that class

Throws tells us a method may throw a keyword
Throw actually throws the exception
The throw keyword requires that it is followed by and object that is an instance of the throwable class

e.g. throw new RobotTrappedException;

AddSuppressed(Throwable exception) - appends exception to the list of suppressed exceptions
FillInStackTrace() fills in the execution stack trace
GetCause() - gets cause if that constructor was used or null if not
GetSuppressed()
GetStackTrace()
GetMessage()

Generics

25 November 2019 11:17

Allow programmers to write generic code that enforces some stronger type checks at compile time.

Java uses angle brackets to declare what type each object is for

E.g. `Stack myStack = new Stack<String>();`

Generics in java allow for:

Stronger type checks at compile time

Elimination of type casts

Implementation of generic algorithms tailored to different types

Generic classes look like normal classes except they have a list of generic types in their class definition. These type placeholders are then used in the code to refer to the generic type.

When using integers in generics, we need to use the `Integer` class instead of the `int` primitive type

You cannot use primitive types, so you have to use the object equivalents

`Integer`, `Float`, `Double`, `Long`, `Short`, `Byte`, `Boolean`, `Character`

```
Public class myClass<T>
```

```
Private T myVar
```