University of HUDDERSFIELD

CFS2160: Software Design and Development

Lecture 7: Welcome to Java

In Java, Everything is an Object.

Tony Jenkins
A.Jenkins@hud.ac.uk

Java Contents

This part of CFS2160 provides:

- An introduction to object-oriented programming …
- … with a strong software engineering foundation …
- … aimed at producing and maintaining large, high quality software systems.

The programming language we will use from now on is Java.

The development environment is IntelliJ IDEA.

Goals

At the end of this module you should:

- Have a sound knowledge of programming principles.
- Have a sound knowledge of object-orientation.
- Be able to critically assess the quality of a (small) software system.
- Be able to implement a small software system in Java.

GitHub

It is usually easiest to keep a GitHub repo to one language …

Ergo, the code for the Java section of this module is here:

https://github.com/TonyJenkins/cfs2160-2019-java-public.git

It contains programs from lectures, some from practicals, code for the assignments, etc.

Java

Java has been around since 1995.

It is free and open source, but supported by a commercial entity (currently Oracle).

The latest version is Java 13 (September 2019). We are going to use Java 11, which is well established and widely used.

Download links: https://java.com/en/download/ (JDK and JRE).

Java

Java has been around since 1995.

It is free and open source, but supported by a commercial entity (currently Oracle).

The latest version is Java 13 (Septem
use Java 11, which is well establishe

Download links: https://java.com/en

Note
This year we are moving from Java 8 to the next LTS Release, Java 11. It is just possible Java 8 code or links may slip through.

Novelty

```
public class Hello
{

  public Hello ()
  {
    sayHello ();
  }

  private void sayHello ()
  {
    System.out.println ("hello, world");
  }

  public static void main (String args[])
  {
    new Hello ();
  }
}
```

Novelty

```
public class Hello
{

  public Hello ()
  {
    sayHello ();
  }

  private void sayHello ()
  {
    System.out.println ("hello, world");
  }

  public static void main (String args[])
  {
    new Hello ();
  }
}
```

```
public class Hello
{

  public Hello ()
  {
    sayHello ();
  }

  private void sayHello ()
  {
    System.out.println ("hello, world"):
  }

  public static void main (String args[])
  {
    new Hello ();
  }
}
```

Fundamental Concepts

We have met all of these before.

They are now absolutely fundamental:

- Class.
- Object.
- Attribute.
- Method.
- Parameter.
- Data Type.

Fundamental Concepts

We have met all of these before.

They are now absolutely fundamental:

- Class.
- Object.
- Attribute.
- Method.
- Parameter.
- Data Type.

If you are not confident in what *any* of these mean, sort it out this week. **Ask** if necessary!

Objects and Classes

Objects:

- Represent "things" from the real world, or from some problem domain.

Classes:

- Represent all objects of a particular kind (type).

Objects and Classes

Objects:

- Represent "things" from the real world, or from some problem domain.

Classes:

- Represent all objects of a particu

So, in a cinema booking system, there are classes "Film", "Customer", "Ticket", "Booking".

Objects and Classes

Objects:

- Represent "things" from the real world, or from some problem domain.

Classes:

- Represent all objects of a particu

Objects in the class "Film" include *Paddington 2, Justice League, Thor: Ragnarok, My Little Pony: The Movie*.

Attributes

An attribute:

- is something of interest in an object.
- has a *value* and a *type*.

All objects of a class have the same attributes.

The values of the attributes define an object's *state*.

Objects of a class will generally have different states.

Attributes

An attribute:

- ➢ is something of interest in an object.
- ➢ has a *value* and a *type*.

All objects of a class have the same a

The values of the attributes define a

Objects of a class will generally have

Objects in the class "Film" have attributes such as *title, certificate, duration.*

Attributes

An attribute:

- ➢ is something of interest in an object.
- ➢ has a *value* and a *type*.

All objects of a class have the same a

The values of the attributes define a

Objects of a class will generally have

"Film Title" is a String.
"Film Certificate" is a String.
"Film Duration" is an integer.

Attributes

An attribute:

- ➢ is something of interest in an object.
- ➢ has a *value* and a *type*.

All objects of a class have the same a

The values of the attributes define a

Objects of a class will generally have

"Film Title" is a String.
"Film Certificate" is a String.
"Film Duration" is an integer.

All Films Have These.

Methods and Parameters

Objects also have operations which can be invoked.

- Java calls them *methods*.

Methods may have parameters to pass additional information.

- Parameters have a *type* and a *value*.
- Parameters can be other objects.

Methods and Parameters

Objects also have operations which can be invoked.

- Java calls them *methods*.

Methods may have parameters to pass additional information.

- Parameters have a *type* and a
- Parameters can be other obje

A "Bank Account" object has *attributes* such as "account holder", "account number", "sort code", and "balance".

Methods and Parameters

Objects also have operations which can be invoked.

- Java calls them *methods*.

Methods may have parameters to pass additional information.

- Parameters have a *type* and a
- Parameters can be other obje

A "Bank Account" object has *methods* such as "Open Account", "Close Account", "Deposit", "Withdraw", "Print Balance", and probably more.

Methods and Parameters

Objects also have operations which can be invoked.

- Java calls them *methods*.

Methods may have parameters to pass additional information.

- Parameters have a *type* and a
- Parameters can be other obje

The "Deposit" method has a parameter: the amount to be deposited.

Return Values

Some methods are simply a behaviour:

- Print the balance, open the account.

Some methods *return* a value:

- Give the balance, indicate whether the balance is sufficient.

Methods that do not return a value are declared with a `void` return type.

Object Orientation

A class defines the fields (attributes) an object has.

Many instances (objects) can be created from a single class.

Each instance has its own set of values for its attributes, which together define its state.

An object usually represents one "real world" thing.

Why classes?

Correctly, this is called *object-oriented programming*.

It is claimed that classes give a more natural way to program.

We develop classes that represent things in our problem domain, and we can write programs using these.

Ideally, we can reuse the classes in multiple programs.

And we can reuse classes developed by others.

Public and Private

Java classes provide two views: *public* and *private*.

The **public** view (correctly called the *interface*) describes how the class can be used by programs, or by other classes.

The **private** view is concerned with how the class itself works, internally.

Public and Private

Java classes provide two views: *public* and *private*.

The **public** view (correctly called the *interface*) describes how the class can be used by programs, or by other classes.

The **private** view is concerned with h
internally.

This is much the same as using a module in Python.
We need to know *how* to use the functions in the module. We care not how they work.

Public and Private

Java classes provide two views: *public* and *private*.

The **public** view (correctly called the *interface*) describes how the class can be used by programs, or by other classes.

The **private** view is concerned with h
internally.

The usual Java convention is:

- Attributes are private.
- Methods are public.

Java Types

Java supports all the expected data types.

In Java, variables are *statically typed* and must be *declared* before (or at) their first use.

Java Types

Java supports all the expected data types.

In Java, variables are *statically typed* and must be *declared* before (or at) their first use.

Compare with Python, where variables are *dynamically typed*, and are given their type by their first use.

Source Code

Each class has *source code* (Java code).

This code:

- Defines its fields (attributes) and their data types.
- Defines its methods, and provides Java code to implement the methods.
- Is *Compiled* and can then be *Executed*.

A class called "SomeName" *must* be in a file `SomeName.java`.

A First Java Class

We will create a Java class to store details of some employees, and a simple program to use it.

Employees have an id number, a name, and a rating.

The program needs to manage the rating, by adding or subtracting points.

The class will be called `Employee`, and therefore *must* be created in a file called `Employee.java`.

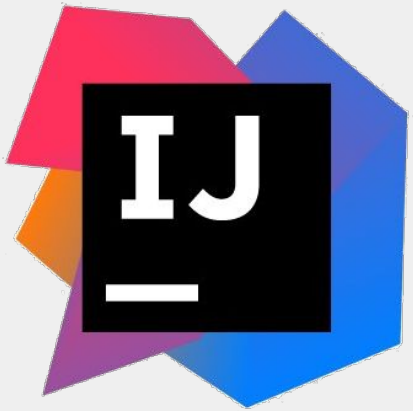
IntelliJ

Java is much more verbose than Python - there is a lot of typing to do.

IntelliJ is *really good* at reducing the typing.

IntelliJ will automatically generate almost all the "boilerplate" code that you find in any Java class.

Learn to use it!



Basic Class Structure

The main wrapper for a class looks like this.

Note the opening and closing curly brackets (or "braces").

```
public class Employee {



}
```

Basic Class Structure

Usually, the first thing to define is the attributes.

These are given a type, and a name.
They are *private* to the class.

```
public class Employee {



}
```

Basic Class Structure

Usually, the first thing to define is the attributes.

These are given a type, and a name.
They are *private* to the class.

```
public class Employee {

  private String name;
  private int id;
  private int rating;


}
```

Conventions

Like Python, Java has conventions for naming things:

- Class names start with a capital letter.
- Variable and field names start with a lowercase letter.
- If more than one word is needed in the name, `lowerCamelCase` is used.
- Constant names are all in capitals.

If you ignore these, you will confuse other programmers.

Conventions

Like Python, Java has conventions for naming things:

- Class names start with a capital letter.
- Variable and field names start with a lowercase letter.
- If more than one word is needed in the name, `lowerCamelCase` is used.
- Constant names are all in capitals.

If you ignore these, **we will take marks off you until you stop.**

The Constructor

Every class must have a "constructor".

This special method is responsible for creating an instance of the class (an object, if you will).

A constructor may be provided with initial values for the fields.

Or it may set some defaults, or it can even do a mixture.

IntelliJ will write the constructor for you.

Getters and Setters

Attributes are private.

If other programs or classes are to use them, they do so via special methods called *accessors* and *mutators*.

These are more usually called "getters" (to get the value of an attribute) and "setters" (to set the value).

IntelliJ will write these methods for you.

The Main Method

When a program runs, it starts by running a *main* method.

This has a distinctive signature (header):

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

And any code inside it is executed as the program.

IntelliJ will write the signature for you.

IntelliJ Demo Time