Multi-Paradigm Programming - Java

Dominic Carr

Galway-Mayo Institute of Technology

dominic.carr@gmit.ie

What We Will Cover

Goals of this Session

Goals

- To Understand...
    - The Java Programming Language
    - The Java Virtual Machine
    - How to write programs in Java
    - How Inheritance works in Java

The Java Language

Java I

"Write once, Run Anywhere"

– Java Language Tagline

Java II

- General-purpose programming language
- Class-based and object-oriented
  - Not a pure object-oriented language, as it contains primitive types
- Intended to let application developers write once, run anywhere (WORA)
  - compiled Java code can run on all platforms that support Java without the need for recompilation.
  - Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM)
- Syntax similar to C and C++
- but it has fewer low-level facilities than either of them
  - For example Java does automatic memory management

Java III

Listing 1: Hello Java World!

```
public class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Prints the string
            to the console.
    }
}
```

Java IV

- So much to learn from this one snippet!
- the “.java” files must be named after the public class they contain so the above code must be in a file called “HelloWorldApp.java”.
    - Don’t forget this! Common source of errors!
- Java files are compiled into “bytecode”, so when compiled this will result int “HelloWorldApp.class”
- TThe bytecode can be run by the JVM, we will talk in greater depth about the JVM later on.
- The keyword public denotes that a method can be called from code in other classes
- The keyword static in front of a method indicates a static method, which is associated only with the class and not with any specific instance of that class
    - Most methods we write will be instance methods and we will see the difference later on.

Java V

- The keyword void indicates that the main method does not return any value to the caller. If a Java program is to exit with an error code, it must call System.exit() explicitly.
  - This is quite similar to the main method in C, and there are many commonalities between C and Java syntactically.
  - Like in C the main method is the entry point into the program, though it may not be the first thing to execute.
- The main method accepts an array of strings as arguments to the program
  - this can be used to capture command line flags and other user input
- Printing is part of a Java standard library
  - The class System has a static variable “out” which represents the output stream, and you invoke the method println() on it to print your desired message to the screen
  - This is an OOP version of the printf function we seen in C.

Java VI

- Inheritance represents an “is-a” relationship between two classes for example all students are humans
  - so it makes sense to model students as a sub class of human
  - A student will have everything a human has but they will have some specific state of their own such as their grades!
- In Java the parent class is termed **super class** and the inherited class is the **sub class**
  - The keyword “extends” is used by the sub class to inherit the features of super class
- Inheritance is important since it leads to reusability of code

```
class subClass extends superClass
{
   //methods and fields
}
```

Java VII

Listing 2: Person Object in Java

```
public class Person {

        private int age;
        private String name;

        public Person(String name, int age){
                this.name = name;
                this.age = age;
        }

        public void setAge(int age){
                if (!(age <= 0 || age >= 110)){
                        this.age = age;
                }
        }
```

Java VIII

```
        @Override
        public String toString(){
                return name + " is " + age;
        }

        public static void main(String[] args){
                Person d = new Person("John", 25);
                d.setAge(1000);
                System.out.println(d);
        }


}
```

Java IX

Compiling and Running

```
$ javac Person.java

$ java Person
```

- When the variables are private they cannot be accessed by callers
  - Modify the code to see what happens if you try to call any of the instance variables (Age and Name)
- We have access control on those variables and there is a method to modify the state of age
  - But it has custom logic so we cannot enter a silly value like "-1" for an age
  - OOP makes such "encapsulation" very easy.
- All objects in Java extend from Object, and from there we inherit some methods
  - toString is one! but why did I write it then?
  - We can override to provide our own implementation

Java X

```
public class Student extends Person {

        private String[] classes;

        public Student(String name, int age, String[] classes){
                super(name,age);
                this.classes = classes;
        }

        public static void main(String [] args){
                String[] classes = {"Java", "Data Analytics", "
                    Research Methods"};

                Student s = new Student("Paul", 25, classes);
        }
}
```

Java XI

- A student “is-a” Person
- So a Student has a name and age like a Person
  - We pass these up to the superclass (Person) and set up the new instance variable ourselves
- What will happen if we try to print out the student?
  - Lets Try!

Java XII

```
>> Paul is 25
```

- It looks like the student prints out just like a Person as it calls the superclass toString method
  - Let's modify it

```
@Override
public String toString(){
        String val = super.toString() + "\nTaking:\n";
        for(int i = 0; i<classes.length; i++){
                val += classes[i] + "\n";
        }
        return val;
}
```

- We used the superclass method and added to it to print out the students classes
- Now the toString() is better suited

Java XIII

- but did we ever actually call toString?
    - Doesn't look like it in the code right?

Java XIV

- We were using an array in the Student class
- This is much like you would be used to in Python
- You can only add items of the same type
    - actually it is okay as long as they have a “is-a” relationship

Comparison with Python

Python OOP I

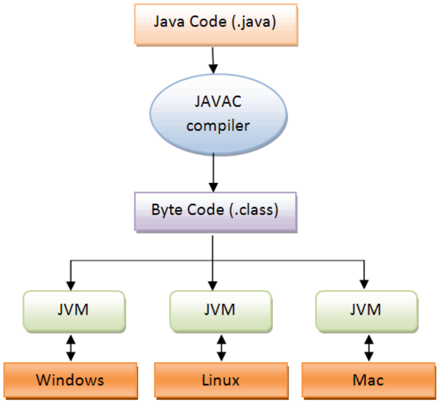
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

Python OOP II

- Java is compiled, python is interpreted
- Java is more verbose than Python
- They both support OOP
- They have different syntax
- Java is statically typed, Python is dynamically typed
- Java is more rigid on structure, Python more flexible

Java Virtual Machine

JVM I



Java Code (.java)
JAVAC compiler
Byte Code (.class)
JVM
JVM
JVM
Windows
Linux
Mac

JVM II

- You write the code in Java
- The Java compiler produces bytecode
- The byte code is executed by the JVM
- Therefore Java code can run on any machine which supports the JVM
    - The same code can run on any OS supported by JVM
- This is a form of Abstraction
    - The JVM hides the differences between for example Windows, Linux, and MacOS.
    - With some caveats this works really well
- Java is the language in which most Android applications are written.

Sources

Sources

- https://www.computerhope.com/jargon/i/imp-programming.htm
- https://en.wikipedia.org/wiki/Abstraction_(computer_science)
- https://www.guru99.com/java-class-inheritance.html
- https://www.w3schools.com/python/python_classes.asp

The End