Platform, App Structure, and Basic IO

Java Developer

StayAhead Training; April, 2022

Characteristics of the Java Language

Platform, App Structure, and Basic IO

- Object-oriented
- Multi-threaded
- Platform independent
- Portable
- Robust
- And more, e.g. simple, secure etc. (which are, IMO, subjective)

Bytecode

Platform, App Structure, and Basic IO

- Source code (*.java file) is manually compiled into bytecode (*.class file)
- The OS does not understand bytecode
- Bytecode must be converted into machine code before it can be executed
- The OS understands machine code
- Windows machine code is not the same as OSX machine code
- Bytecode is portable but needs software to convert it into machine code

JVM

Platform, App Structure, and Basic IO

- JVM = Java Virtual Machine
- Software that converts bytecode into machine code
- Each different OS needs its own peculiar JVM to execute Java bytecode
- The JVM does not produce a machine code file; rather the machine code produced is executed by the OS immediately
- AKA the Java Interpreter
  (technically the Java Interpreter is an implementation of the JVM)

The Java Standard Library

Platform, App Structure, and Basic IO

- Java Standard Library = existing Java code that does common things
- Lots of Java classes organised into modules and packages
- The java.lang package has classes for writing to stdout
- The java.utils package has classes for creating collections
- The java.io package has classes for reading from & writing to files
- The java.sql package has classes for reading from & writing to databases

JRE

Platform, App Structure, and Basic IO

- JRE = Java Runtime Environment
- Comprises everything you need to run Java applications
- Comprises a JVM and the Java Standard Library
- Does **not** include a compiler

JDK

Platform, App Structure, and Basic IO

- JDK = Java Development Kit
- Comprises everything you need to run **and build** Java applications
- Comprises a JVM, the Java Standard Library, and a compiler

Java App Structure

Platform, App Structure, and Basic IO

- A Java application is typically made up of instructions grouped into methods grouped into classes grouped into packages
- A method is a named group of instructions
- A class is a named group of fields and/or methods
- A class is also a template for creating objects
- A source code file typically contains one class
- A package is a tree of directories; a means of grouping classes

Java App Structure

Platform, App Structure, and Basic IO

- The fully qualified name of a class includes its package
- Reversed domain names ensure globally unique names
- E.g. com.stayahead.myapp.App
- The App class exists inside the myapp directory which exists inside the stayahead directory etc.
- By default the compiler and interpreter only look for classes in java.lang
- Classes in other packages must be imported

Java App Structure

Platform, App Structure, and Basic IO

```
package com.stayahead.myapp;

import java.util.*; // look in java.util for classes

public class App {

    // fields and methods

}
```

The main Method

Platform, App Structure, and Basic IO

- Most Java applications comprise many classes
- One class must be designated as containing the method that contains the first instruction to be executed by the JVM
- The method named main is assumed to contain the first instruction
- The JVM will look for a method named main in the given class and begin by executing the instructions in that method

The main Method

Platform, App Structure, and Basic IO

```
public class App {
    public static void main(String[] args) {
        // first instruction
        // second instruction
        // ...
    }
}
```

Java Syntax*

Platform, App Structure, and Basic IO

- Java is case sensitive
- Java keywords are always lowercase
- Variable and method names should be camelCase
- Class names should be PascalCase

Java Syntax*

Platform, App Structure, and Basic IO

- Each statement must be terminated with a semicolon ;
- Each code block should be surrounded by braces { }
- Code inside a block should be indented, e.g. 1 x tab or 4 x spaces
- Single line comment //
- Multi-line comment /* */

Reading from stdin & Writing to stdout

Platform, App Structure, and Basic IO

```
// reading from stdin

var scanner = new Scanner(System.in);

var input = scanner.nextLine();

scanner.close();
```

Reading from stdin & Writing to stdout

Platform, App Structure, and Basic IO

```
// writing to stdout

System.out.println("Hello world");

// writing a variable (myName) with a label

System.out.printf("My name is %s\n", myName);
```

Reading from stdin & Writing to stdout

Platform, App Structure, and Basic IO

```
// combining the two

var scanner = new Scanner(System.in);

System.out.printf("Enter your name: ");

var name = scanner.nextLine();

System.out.printf("Your name is %s\n", name);

scanner.close();
```

Reading from & Writing to a Text File

Platform, App Structure, and Basic IO

```
var contents = Files.readString(Path.of("file1.txt"));

Files.writeString(Path.of("file2.txt"), contents);
```

- Note that both readString and writeString methods may throw exceptions and consequently require exception handling (of which, more later)

Compilation and Execution

Platform, App Structure, and Basic IO

- Compilation = convert source code into bytecode
- javac = Java compiler executable
- Example usage: `$> javac com/stayahead/myapp/App.java`
- If the compiler finds one or more errors in the code it will:
  - write an error message to stdout
  - **not** generate bytecode

Compilation and Execution

Platform, App Structure, and Basic IO

- Execution = convert bytecode into machine code and have the OS execute it
- The execution of machine code as it is converted from bytecode is referred to as interpretation
- java = Java interpreter executable
- Example usage (omit the .class): `$> java com.stayahead.myapp.App`
- If an unexpected error occurs at runtime then the interpreter will:
  - write an error message to stdout
  - terminate execution of the application

The classpath*

Platform, App Structure, and Basic IO

- Classpath = the directory/directories in which both the compiler and interpreter look for classes (.class files)
- By default the classpath is . (current working directory)
- Set the classpath with `-classpath` or `-cp`
- Example usage: `$> javac -cp dir1:dir2 App.java`
- NB: the directory delimiter is : (colon) on UNIX and ; (semicolon) on Windows