JAVA
Fall 2022

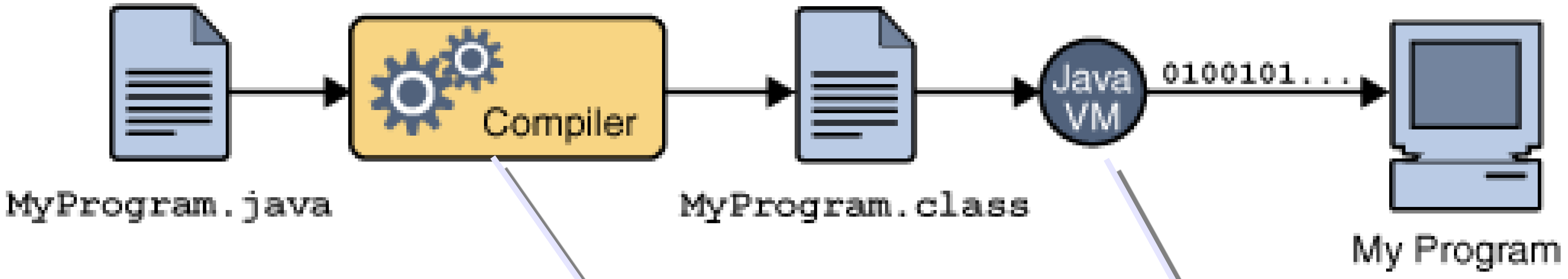
Reading

Java Virtual Machine (JVM)

The HelloWorldApp.java **program**

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

Developing a Java program



javac MyProgram.java

java MyProgram

- An overview of the software development process

4

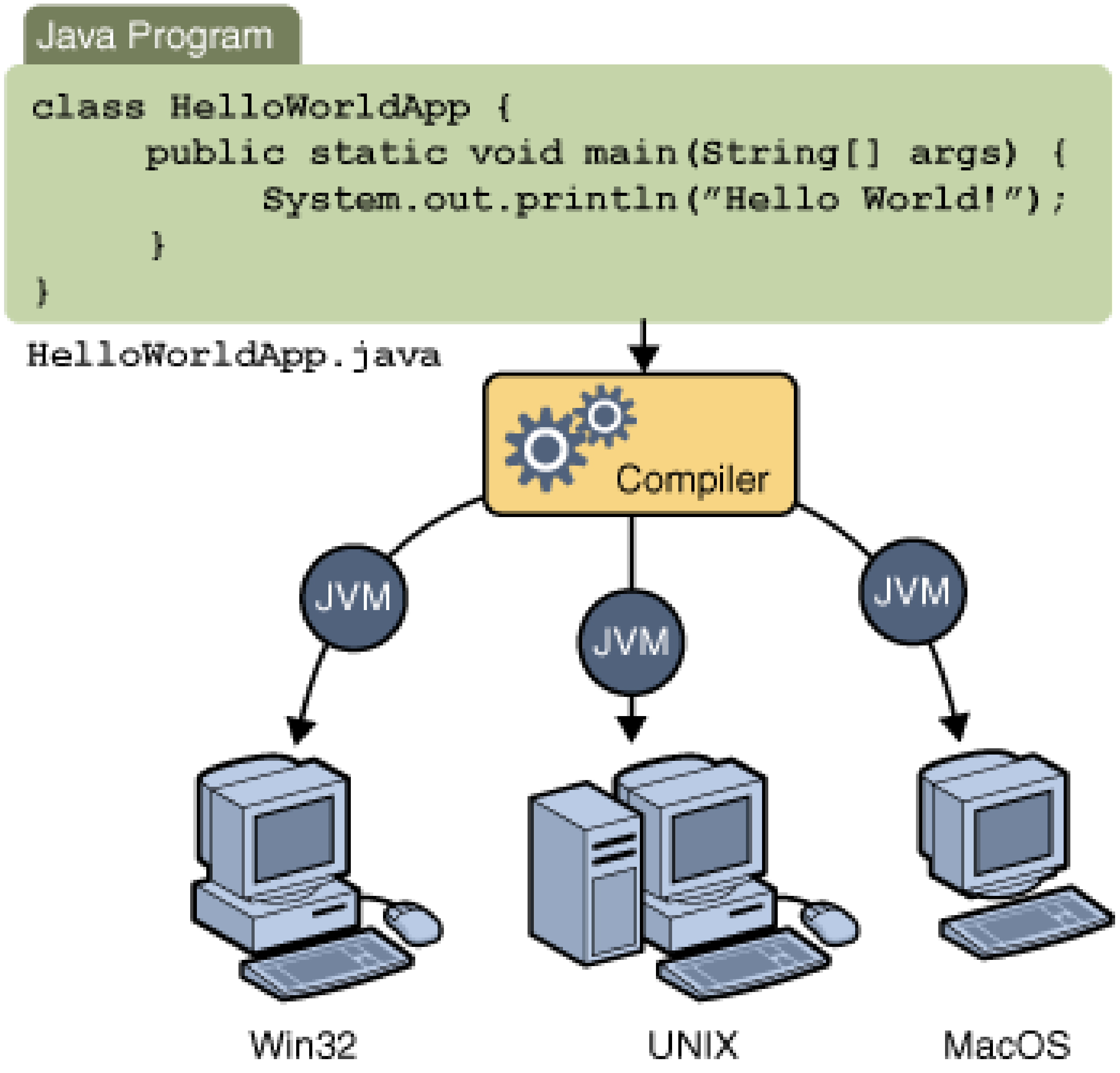
Java virtual machine

- Overview and features
- Program execution cycle

Overview (1)

- The JVM is the base for the Java platform
- Makes Java programs platform independent:
  - “write once, run anywhere”
- Different versions exist for different hardware-based platforms

Overview (2)



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

Features

- An abstract computing machine with:
  - instruction set
  - memory management
- Emulates the host machines
  - to ensure platform-independent byte codes
- Does not require Java programming language
  - supports any language that follows the **class file format**

Program execution cycle

- Virtual machine start up
- Loading
- Linking
- Initialisation
- Creation of new class instances
- Finalisation of class instances
- Unloading
- Virtual machine exit

Virtual machine start up

- The method main is invoked with argument String[]:
  - header: public static void main (String[])
  - argument is a nullable String array
- Invocation is typically from the command line:

```
java HelloWorldApp say hello world!
```

HelloWorldApp with arguments

```
public class HelloWorldApp {
  public static void main(String[] args) {
    for (int i = 0; i < args.length; i++) {
      System.out.println(args[i]);
    }
  }
}
```

Loading

- Class HelloWorldApp is loaded by ClassLoader
- The loaded class is an object of class Class
  - cached for subsequent use
- Loading may fail due to:
  - incorrect class file format
  - incorrect version of the class file format
  - not found

Linking

- Combines the loaded class into the runtime state of the JVM
- Three steps:
  - **verification**: check the class structure
  - **preparation**: creating and initialising class fields to default values
  - **resolution**: resolve references to other classes

Initialisation

- Execute initialisers:
  - class (static) initialisers
  - initialisers for static fields
- Also causes any super class(es) to be loaded, linked and initialised:
  - if not already

Creation of new class instances

- Objects are created if required
- Object creation involves:
  - allocating enough memory space for all variables (declared in the class and super class)
  - initialising the variables
  - executing a constructor method

HelloWorldApp with objects

```
public class HelloWorldApp2 {
  public static void main(String[] args) {
    String msg = "Hello world!";
    // or String msg = new String(“Hello world!”);
    System.out.println(msg);
  }
}
```

Finalisation of class instances

- Remove objects that are no longer in use
- Java garbage collector automatically remove these objects
- Finalizers are used to prepare objects for removals
- Overrides Object.finalize

Unloading

- Unload unused classes to reduce memory use
- A class is unloaded when its associated ClassLoader is removed
- System classes may never be unloaded

Virtual machine exit

- When one of two things happens:
  - all non-daemon processes (threads) finish execution
  - invokes System.exit or RunTime.exit