ESS 201: Programming II
Java

Term 1, 2019-20

Lecture: Introduction to Java

T K Srikanth
IIIT-B

Books and References

Java: How to Program, Paul Deitel and Harvey Deitel

2. Thinking in Java, Bruce Eckel. (3rd edition available online for free download. Covers a good part of the course material)

3. Java Programming: Wikibooks, https://en.wikibooks.org/wiki/Java_Programming

Useful for a quick reference

4. The Java Tutorials: Oracle Java Documentation

https://docs.oracle.com/javase/tutorial/tutorialLearningPaths.html

Getting Started

Install Java Development environment (JDK) in your laptops

- “Java SE downloads” at Oracle
  - Download Java 8. Currently at Java SE 8u221
  - Install. Will set up the Java compiler, run-time environment and default libraries
- A good multi-file code editor
- IDE such as Eclipse - though this will not be available for the exams/tests .

Hello World!

File: HelloWorld.java

```
public class HelloWorld{
        public static void main(String[] args) {
                System.out.println(“Greetings!”);
        }
}
```

Only **one public class per file.** Class name and file name should match

We have main in multiple classes/files? Why would we do that?

Compilation and running

```
> javac HelloWorld.java
```

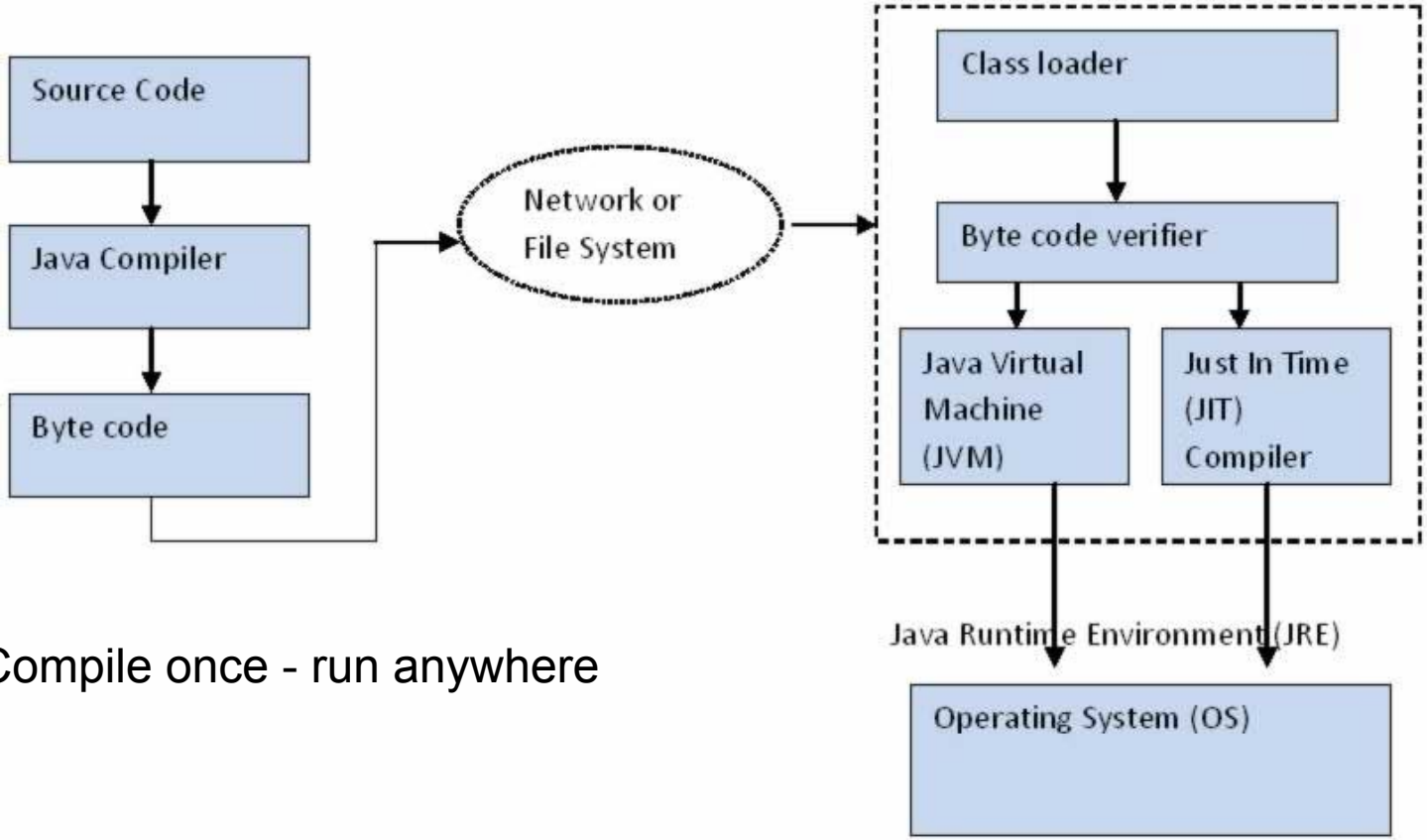
produces `.class` file (one for each class referenced, if it does not already exist)

```
> java HelloWorld
```

Runs the `main` in this class

- Clear distinction between “compiling” and “link/run” phases
- Automatic compilation of referenced classes
- Automatic linking of referenced classes (assuming the `.class` files exist)

Byte code and compilation process



Compile once - run anywhere

Classes

Encapsulation of a concept: a Car, a Printer, a Tree, a Student etc

Class embodies **state** and **behaviour**

(a **struct** in C/C++ has state but no behaviour)

Instances of a class can have different values for their state - we refer to these as “objects”.

A class defines behaviour - all objects of the class share this (also referred to as “objects responding to messages”)

Objects and references

- Everything* is an “object” - an instance of a “class”
- Objects have to be explicitly constructed (using **new**)
- You manipulate objects through their references
- Objects (state) may change, but references do not
- Methods are passed references to objects (or primitives), so always “call by value”

** except primitive types as we will see soon*

Types in Java

- primitives
- reference
- Array
- (a special type) void

Primitive types

Fixed size of basic types

boolean true, false (not equivalent to 0 or 1)
char 16 bits (unicode)
byte 8 bits
short 16 bits
int 32 bits
long 64 bits
float 32 bits
double 64 bits

all numeric types are “signed”. No “unsigned”
Note: primitives **are NOT** objects

Initialization and defaults

Any variable **must** be initialized before its value is used

- Compiler error if not

All data members of classes are initialized to their “default” value at construct time

- equivalent of “zero”/null/false as appropriate

Class definition

```
class IceCreamBar {

  String getName() {   }

  String getFlavour() {   }

  float getTemp() {   }

  float getPrice() {   }

  ….

}
```

Methods

Class definition

```
class IceCreamBar {

  String getName() {   }

  String getFlavour() {   }

  float getTemp() {   }

  float getPrice() {   }

  String name, flavour;

  float temp, price;

}
```

Methods

Data members

Class definition

```
class IceCreamBar {

  String getName() {   }

  String getFlavour() {   }

  float getTemp() {   }

  float getPrice() {   }

  String name, flavour;

  float temp, price;

}
```

Methods

Data members

Design guideline:
Decide on the methods (behaviour) before figuring out the data members (state)
Improves encapsulation

Class definition

```
class IceCreamBar {

  IceCreamBar(String iname, String iflavour, float itemp,
float iprice) { }

  String getName() {  }

  String getFlavour() {  }

  float getTemp() {  }

  float getPrice() {  }

  String name, flavour;

  float temp, price;

}
```

Constructor

Methods

Data members

Constructors

Every class should have at least one constructor. These are methods that have the same name as the class and do not have an explicit return value

Constructor that takes no arguments is called the *default constructor*

Compiler defines a default constructor if no constructors have been defined for the class. This default version sets all data members to their default initial value. If any constructor is defined for the class, then the constructor does not define the default constructor

All instance data members are initialized to default values, if not explicitly initialized in a constructor

Note: no notion of "*destructors*". Facility for cleanup

Objects and references

- Everything* is an “object” - an instance of a “class”
- You manipulate objects through their references
- Objects (state) may change, but references do not
- Methods are passed references to objects (or primitives), so always “call by value”
- Objects are allocated on the heap, references are on the program stack or heap (depending on whether they are local variables or members of objects)
- Memory of objects are “garbage collected” when no longer referenced (How?)

* *except primitive types as we will see soon*

Class definition

```
class IceCreamBar {

  IceCreamBar(String iname, String iflavour, float itemp,
float iprice) { }

  String getName() { return name; }

  String getFlavour() { return flavour; }

  float getTemp() { return temp; }

  float getPrice() { return price; }

  …

}
```

Class definition

```
class IceCreamBar {
  IceCreamBar(String iname, String iflavour, float itemp,
float iprice) {
    name = iname; flavour = iflavour;
        temp = itemp;
        price = iprice;
  }
  String getName() { return name; }
  String getFlavour() { return flavour; }
  float getTemp() { return temp; }
  float getPrice() { return price; }
…
}
```

Methods are implemented where they are defined, as part of the class body. No notion of header files or equivalent

Class definition

```
class IceCreamBar {
  IceCreamBar(String iname, String iflavour, float itemp,
float iprice) {
    name = iname; flavour = iflavour;
        temp = itemp;
        price = iprice;
  }
  String getName() { return name; }
  String getFlavour() { return flavour; }
  float getTemp() { return temp; }
  float getPrice() { return price; }
  void setTemp(float t) { temp = t;}
  void setPrice(float p) { price = p;}
  ...
}
```

Invoking class methods

```
IceCreamBar ic1 = new IceCreamBar("CH", "Chocolate", 8.0,
90.0);

….

System.out.println("Flavour " + ic1.getFlavour() +
                                " from " + ic1.getName() +
                                "costs " + ic1.getPrice() +
                                " and is stored at " +
ic1.getTemp() + " degrees.");
```

Should print:

```
Flavour Chocolate from CH costs 90.0 and is stored at 8.0
degrees.
```

Java operators and control statements

Operators:

- Generally follow the syntax of C/C++
- No operator overloading

Control statements:

- If-else, while, do-while, for, switch: similar to C/C++
- for: variations that allow simple iteration over arrays, lists etc

Java Class Library

- Provide the programmer with a well-known set of functions to perform common tasks, such as maintaining lists of items or performing complex string parsing.
- Provide an abstract interface to tasks that would normally depend heavily on the hardware and operating system.
- Some underlying platforms may not support all of the features a Java application expects. In these cases, the class libraries can either emulate those features using whatever is available, or provide a consistent way to check for the presence of a specific feature.

String and Array

Both are **classes**

Specific methods, constructors, operations

String: immutable in size and content - can only be queried

Operators such as concatenation (+) defined.

Array:

Size defined at construct time

Can be queried for length (arr.length)

Writing to Standard output

System.out - a pref-defined instance of PrintStream, defined in class System

```
String s1 = “Hello”;

int year = 2019;

System.out.println("myString is: " + s1);

System.out.println("myInt is: " + year);
```

Reading from Standard Input

System.in - a pref-defined instance of InputStream, defined in class System

```
Scanner scanner = new Scanner(System.in); // or new Scanner(“filename”);

String myString = scanner.next();

int myInt = scanner.nextInt();

while (sc.hasNextInt()) {

          int nextVal = sc.nextInt();

      }
scanner.close();
```