Java as Object Oriented Programming Language

pm_jat@daiict.ac.in

References:
1. Chapter 1, Object Oriented Design and Patterns, Cay S Horstmann, John Wiley & Sons
2. Java and Object Orientation, John Hunt, Springler
Why Java

• Here, we are not talking about java because it is portable language and provides concurrency and safety features in programming

• We are having look at Java, because it is one of the (probably most) popular Object Oriented Programming languages

• I want to use this as language to revisit OOP concept in more generic sense
Why Java

• You should find Java more object oriented than C++, as there is no concept of global functions, global variables, and other lexical scoping rules

• Everything is wrapped within class

• Still, there are few things, java compromising concept of “pure” Object Oriented programming language -
  – Dealing primitive data types and objects differently; primitive data are not objects in Java,
  – Methods are implemented in imperative way
  – Probably Smalltalk, Eiffel, Ruby are closer to so called “pure” OOP language
Java is C based language

- Syntax and basic constructs in Java are based on C.
- Code below is a valid C, C++, and Java code
- So you already know much of Java

```java
void sort(int a[], int size) {
    for (int i=1; i < size-1; i++) {
        // bubble up max of a[0] to a[n-i] at a[n-i]
        for(int j=0; j < size-i; j++)
            if ( a[j] > a[j+1] ){
                int temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
    } // Result: a[size-i] to a[size-1] is sorted
}
```
Java is Simpler than C++!

- Removes problematic features of C++
  - No memory allocation or de-allocation
  - No Pointers and Pointer Arithmetic
  - No struct, union
  - No global functions, not even main
  - No typedefs
  - No multiple inheritance
  - No gotos
  - No operator overloading
  - No pre-processor (#define or so)
  - Automatic garbage collection
  - Only one way of constructing objects
  - Nothing like object slicing, ….probably you can add more
Classes are building block in Java

• When you program in Java, it is all about creating classes

• Therefore, it would be appropriate (and more precise) to call “writing a class” rather than saying “writing a program” in Java
  – Developing a java application is writing a number of classes

• Let us write our first Java class (or first Java Program)–
public class Greeter {
    public Greeter(String aName) {
        name = aName;
    }
    public String sayHello() {
        return "Hello, " + name + "!";
    }
    private String name;
}
Dissection of the Greeter class

• Classes are saved on disk with the same name is of class, in this case, you would save Greeter class in file Greeter.java

• The class has one constructor, one other method, and one field

• Methods are public and field is private

• Compare with C++: the way access control is specified
Few more observations

• The Greeter class is using another class String from java library classes

• Access control is specified differently in Java
  – Each method or field is tagged with public or private,
  – I guess, you know what it mean !

• Class itself has been declared as public; We shall talk about this little later.
Constructing objects of class

- To construct an object, you use the `new` operator, followed by a call to the constructor:
  ```java
  new Greeter("World")
  ```
- The `new` operator returns the constructed object, or, more precisely, a reference to that object - we will discuss this in a moment.
- More commonly, you store the value that the `new` operator returns in an `object variable`
  ```java
  Greeter worldGreeter = new Greeter("World");
  ```
- You always use new keyword for object construction in Java and objects are always created in heap memory
Compare with C++: object construction
• Once object has been constructed, you can invoke a method as-
  
  ```java
  String greeting = worldGreeter.sayHello();
  ```

• Let us now create another class that can use and test Greeter class-

public class GreeterTest {

    public static void main(String[] args) {

        Greeter worldGreeter = new Greeter("World");
        String greeting = worldGreeter.sayHello();

        System.out.println(greeting);
    }
}

Dissection

- The class has one static method: main with two *command line arguments*

- Note that command line arguments are different than in C++, where you have argc and argv;

- You know what is a *static method*!

- Like in C++, main method, has a specific meaning in Java too
public static void main(String[] args)

• main() method is used to start a Java application, Java Virtual Machine automatically executes main method of “application class”, when it is requested to load the class

• I call Application class the one, which is first to be loaded to start the execution
Some more observations

- `System.out` is a static object, and `println` is its method to output on console and feed newline after printing; equivalent to
  
  ```java
  cout << greeting << endl;
  ```

- Since everything in Java is a class and methods, you will always be finding large number of static methods being used like this.

- `println` has full name as-
  
  ```java
  java.lang.System.out.println
  ```
Compiling and running Java classes

• C++, programs compile to native machine code (for example exe file on windows), while

• Java programs compile to byte code (class file), which is run on Java Virtual Machine (JVM)

• As a result you get portability of compile code across the platforms

• … code compiled on one platform runs on any other platform that has JVM is running
Compiling and running Java classes

• Here is how you compile your classes
  javac Greeter.java
  javac GreeterTest.java

• Compilation would produce
  Greeter.class and GreeterTest.class

• Run test class
  java GreeterTest
## Primitive Data types in Java

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>4 bytes</td>
<td>−2,147,483,648 … 2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>8 bytes</td>
<td>−9,223,372,036,854,775,808L … 9,223,372,036,854,775,807L</td>
</tr>
<tr>
<td>short</td>
<td>2 bytes</td>
<td>−32768 … 32767</td>
</tr>
<tr>
<td>byte</td>
<td>1 byte</td>
<td>−128 … 127</td>
</tr>
<tr>
<td>char</td>
<td>2 bytes</td>
<td>'\u0000' - '\uFFFF'</td>
</tr>
<tr>
<td>boolean</td>
<td>1 byte</td>
<td>false, true</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes</td>
<td>approximately ±1.79769313486231570E+308</td>
</tr>
<tr>
<td>float</td>
<td>4 bytes</td>
<td>approximately ±3.40282347E+38F</td>
</tr>
</tbody>
</table>
Object References

• In Java, value of an object is always a *reference* to an object.
• Figure illustrate meaning of following object construction and saving a reference to it

```
Greeter worldGreeter = new Greeter("World");
```

![Diagram showing object reference](image)
Object References

- You can have multiple references to the same object.
- For example, after the assignment
  ```java
  Greeter anotherGreeter = worldGreeter;
  ```
  the two object variables both share a single object
Object References

- Both references point to same object. You can use other to change state of the object,

```
anotherGreeter.setName("Dave");
```

- Break the link

```
worldGreeter = null;
```
the reference now does not point to any object, and called as *null reference*.

- If you invoke a method on a null reference, an *exception* is thrown. Unless you supply a handler for the exception, the program terminates.
Compare with C++ pointers
Parameter Passing Mechanism in Java

Parameter Passing in Java is

**Always by value**

- You can not write a swap function in Java !!

- In fact, you do not need such function when you program in correct object oriented way ..
Parameter Passing Mechanism in Java

• Consider this, suppose there is a method defined to our Greeter class
  
  ```java
  public void setName(Greeter other) {
    other.name = name;
  }
  ```

• And use this method as shown below-
  ```java
  Greeter worldGreeter = new Greeter("World");
  Greeter daveGreeter = new Greeter("Dave");
  worldGreeter.setName(daveGreeter);
  ```

• What should be the name of daveGreeter object now?
Parameter Passing Mechanism in Java

- You got what you wanted, even if it is by value only!!

Figure 8

Accessing an Object through a Parameter Variable
Parameter Passing Mechanism in Java

• Following methods are failing in what they possibly might be expected to do-

```java
public void fun1(int n) {
    //this assignment has no effect outside the method
    n = name.length();
}

public void fun2(Greeter other) {
    //this assignment has no effect outside the method
    other = new Greeter(name);
}
```
Compare this with parameter passing mechanism in C/C++
Thanks