Lecture 1: Object Oriented Programming

Muhammad Hafeez Javed
www.rmhjaved.com

Procedural vs. Object-Oriented Programming

- The unit in procedural programming is *function*, and unit in object-oriented programming is *class*
- Procedural programming concentrates on creating functions, while object-oriented programming starts from isolating the classes, and then look for the methods inside them.
- Procedural programming separates the data of the program from the operations that manipulate the data, while object-oriented programming focus on both of them

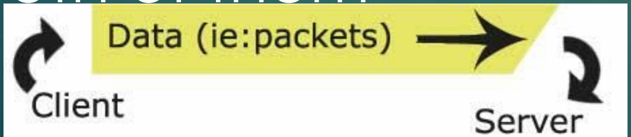


figure1: procedural

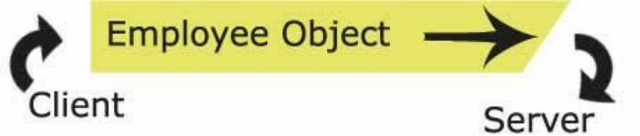


figure2: object-oriented

Concept of Class and Object

- “Class” refers to a blueprint. It defines the variables and methods the objects support

- “Object” is an instance of a class. Each object has a class which defines its data and behavior

Class Members

- **A class can have three kinds of members:**
  - ***fields***: data variables which determine the status of the class or an object
  - ***methods***: executable code of the class built from statements. It allows us to manipulate/change the status of an object or access the value of the data member
  - ***nested classes and nested interfaces***

Sample class

```
class Pencil {
 public String color = “red”;
 public int length;
 public float diameter;

    public static long nextID = 0;

    public void setColor (String newColor) {
          color = newColor;
    }
 }
```

Fields – Declaration

- **Field Declaration**
  - a type name followed by the field name, and optionally an initialization clause
  - primitive data type vs. Object reference
    - boolean, char, byte, short, int, long, float, double

More about field modifiers

- Access control modifiers
  - *private*: private members are accessible only in the class itself
  - *package*: package members are accessible in classes in the same package and the class itself
  - *protected*: protected members are accessible in classes in the same package, in subclasses of the class, and in the class itself
  - *public*: public members are accessible anywhere the class is accessible

Pencil.java

```
public class Pencil {
 public String color = “red”;
 public int length;
 public float diameter;
 private float price;

     public static long nextID = 0;

     public void setPrice (float newPrice) {
            price = newPrice;
     }
 }
```

CreatePencil.java

```
public class CreatePencil {
 public static void main (String args[]){
     Pencil p1 = new Pencil();
     p1.price = 0.5f;
     }
}
```

```
%> javac Pencil.java
%> javac CreatePencil.java
CreatePencil.java:4: price has private access in Pencil
    p1.price = 0.5f;
          ^
```

More about field modifiers

- static
  - only one copy of the static field exists, shared by all objects of this class
  - can be accessed directly in the class itself
  - access from outside the class must be preceded by the class name as follows

    ```
    System.out.println(Pencil.nextID);
    ```

    or via an object belonging to the class
  - from outside the class, non-static fields must be accessed through an object reference

```
public class CreatePencil {
 public static void main (String args[]){
    Pencil p1 = new Pencil();
    Pencil.nextID++;
    System.out.println(p1.nextID);
    //Result?       1

    Pencil p2 = new Pencil();
    Pencil.nextID++;
    System.out.println(p2.nextID);
    //Result?       2

    System.out.println(p1.nextID);
    //Result?        still 2!
 }
}
```

Note: this code is only for the purpose of showing the usage of static fields. It has POOR design!

More about field modifiers (3)

- final
  - once initialized, the value cannot be changed
  - often be used to define named constants
  - static final fields must be initialized when the class is initialized
  - non-static final fields must be initialized when an object of the class is constructed

Fields –Initialization

- **Field initialization**
  - not necessary to be constants, as long as with the right type
  - If no initialization, then a default initial value is assigned depending on its type

Type	Initial Value
boolean	false
char	'\u0000'
byte, short, int, long	0
float	+0.0f
double	+0.0
object reference	null

Methods – Declaration

- Method declaration: two parts

1. method header
   - consists of modifiers (optional), return type, method name, parameter list and a throws clause (optional)
   - types of modifiers
     - *access control modifiers*
     - *abstract*
       - the method body is empty. E.g.

         ```
         abstract void sampleMethod( );
         ```
     - *static*
       - represent the whole class, no a specific object
       - can only access static fields and other static methods of the same class
     - *final*
       - cannot be overridden in subclasses
2. method body

Methods – Invocation

- Method invocations
  - invoked as operations on objects/classes using the dot (.) operator

    ```
    reference.method(arguments)
    ```

  - static method:
    - Outside of the class: “`reference`” can either be the class name or an object reference belonging to the class
    - Inside the class: “`reference`” can be ommitted
  - non-static method:
    - “`reference`” must be an object reference

Method - Overloading

- A class can have more than one method with the same name as long as they have different parameter list.

```
public class Pencil {
      . . .
      public void setPrice (float newPrice) {
          price = newPrice;
      }

      public void setPrice (Pencil p) {
          price = p.getPrice();              }
   }
```

- How does the compiler know which method you’re invoking? — compares the number and type of the parameters and uses the matched one

Methods – Parameter Values

- Parameters are always passed by value.

```
public void method1 (int a) {
    a = 6;
}

public void method2 ( ) {
    int b = 3;
    method1(b);    // now b = ?
         // b = 3
}
```

- When the parameter is an object reference, it is the object reference, not the object itself, getting passed.

☹ Haven't you said it's past by value, not reference ?

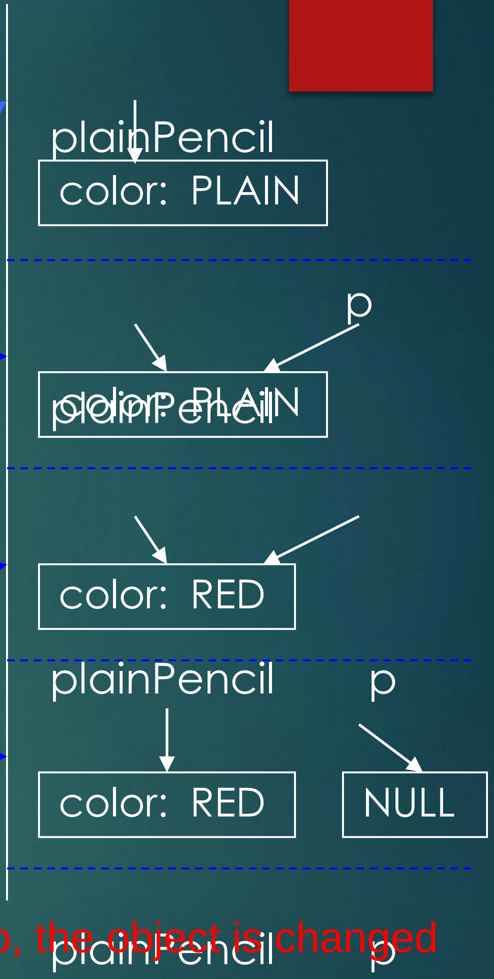
another example: (parameter is an object reference)

```
class PassRef{
   public static void main(String[] args) {
      Pencil plainPencil = new Pencil("PLAIN");
      System.out.println("original color: " +
   plainPencil.color);

      paintRed(plainPencil);

      System.out.println("new color: " +
   plainPencil.color);
   }

   public static void paintRed(Pencil p) {
      p.color = "RED";
      p = null;
   }
}
```



- If you change any field of the object which the parameter refers to, the object is changed for every variable which holds a reference to this object

- You can change which object a parameter refers to inside a method without affecting the original reference which is passed

- What is passed is the object reference, and it's passed in the manner of "PASSING BY VALUE"!

The Main Method - Concept

- **main** method
  - the system locates and runs the main method for a class when you run a program
  - other methods get execution when called by the main method explicitly or implicitly
  - must be public, static and void

The Main Method - Getting Input from the Command Line

- When running a program through the `java` command, you can provide a list of strings as the real arguments for the `main` method. In the `main` method, you can use `args[index]` to fetch the corresponding argument

```
class Greetings {
  public static void main (String args[]){
    String name1 = args[0];
    String name2 = args[1];
    System.out.println("Hello " + name1 + “&“ +name2);
  }
}
```

```
➢ java Greetings Jacky Mary

Hello Jacky & Mary
```

- Note: What you get are strings! You have to convert them into other types when needed.

Modifiers of the classes

- A class can also has modifiers
  - public
    - publicly accessible
    - without this modifier, a class is only accessible within its own package
  - abstract
    - no objects of abstract classes can be created
    - all of its abstract methods must be implemented by its subclass; otherwise that subclass must be declared `abstract` also
  - final
    - can not be subclassed
- Normally, a file can contain multiple classes, but only one public one. The file name and the public class name should be the same