Classes and Objects in Java

Parameter Passing, Delegation, Visibility Control, and Object Cleanup

Static Method in Circle class

- Like class variables, can have class methods too:

```java
public class Circle {
    // A class method for the same
    public static Circle bigger(Circle a, Circle b) {
        if (a.r > b.r) return a; else return b;
    }
}
```

- Accessed with class name

```java
Circle c1 = new Circle();
Circle c2 = new Circle();
Circle c3 = Circle.bigger(c1, c2);
```

Static Methods in Circle Class

- Class methods can only access static variables and methods.

```java
public class Circle {
    // class variable, one for the Circle class, how many circles
    private static int numCircles = 0;
    public double x, y, r;
    public static void printNumCircles() {
        System.out.println("Number of Circles = " + numCircles);
    }
    // This is not VALID
    public static void printRadius() {
        System.out.println("Radius = " + r);
    }
}
```

Back to HelloWorld
[System invokes static main method]

```java
// HelloWorld.java: Hello World program
class HelloWorld {
    public static void main(String args[]) {
        System.out.println("Hello World");
    }
}
```

Back to Constants
[final can also be made as static]

```java
class SquaredNumbers {
    static final int MAX_NUMBER = 25;
    public static void main(String args[]) {
        final int MAX_NUMBER = 25;
        int lo = 1;
        int squared = 0;
        while (squared <= MAX_NUMBER) {
            lo = lo + 1; // Calculate square
            System.out.println(squared);
            lo = lo + 1; // Compute the new lo value */
        }
    }
}
```
Parameter passing

- Method parameters which are objects are passed by reference.

- Copy of the reference to the object is passed into method, original value unchanged.

Parameter passing - Example

```java
public class ReferenceTest {
    public static void main(String[] args) {
        Circle c1 = new Circle(5, 5, 20);
        Circle c2 = new Circle(1, 1, 10);
        System.out.println("c1 Radius = " + c1.getRadius());
        System.out.println("c2 Radius = " + c2.getRadius());
        parameterTester(c1, c2);
        System.out.println("c1 Radius = " + c1.getRadius());
        System.out.println("c2 Radius = " + c2.getRadius());
    }
}
```

Parameter passing - Example

```java
public static void parameterTester(Circle circleA, Circle circleB) {
    circleA.setRadius(15);
    circleB = new Circle(0, 0, 100);
    System.out.println("circleA Radius = " + circleA.getRadius());
    System.out.println("circleB Radius = " + circleB.getRadius());
}
```

Parameter passing - Example

Output –

- c1 Radius = 20.0
- c2 Radius = 10.0
- circleA Radius = 15.0
- circleB Radius = 100.0
- c1 Radius = 15.0
- c2 Radius = 10.0

Parameter passing - Example

**STEP 1 – Before calling parameterTester()**

- circleA
  - (5,5,20)
- circleB
  - (1,1,10)

**STEP 2 – parameterTester(c1, c2)**

- circleA
  - (5,5,20)
- circleB
  - (1,1,10)

**STEP 3 – circleA.setRadius(15)**

- circleA
  - (5,5,15)
- circleB
  - (1,1,10)

**STEP 4 – circleB = new Circle(0,0,100)**

- circleA
  - (5,5,15)
- circleB
  - (0,0,100)

**STEP 5 – After Returning from parameterTester**

- circleA
  - (5,5,15)
- circleB
  - (1,1,10)
Delegation

- Ability for a class to delegate its responsibilities to another class.

- A way of making an object invoking services of other objects through container ship.

Delegation - Example

```
public class Point {
    private double xCoord;
    private double yCoord;
    // Constructor
    ................
    public double getXCoord() {
        return xCoord;
    }
    public double getYCoord() {
        return yCoord;
    }
}
```

Visibility Control: Data Hiding and Encapsulation

- Java provides control over the visibility of variables and methods, encapsulation, safely sealing data within the capsule of the class
- Prevents programmers from relying on details of class implementation, so you can update without worry
- Helps in protecting against accidental or wrong usage.
- Keeps code elegant and clean (easier to maintain)

Visibility Modifiers: Public, Private, Protected

- **Public** keyword applied to a class, makes it available/visible everywhere. Applied to a method or variable, completely visible.
  - Default (No visibility modifier is specified): it behaves like public in its package and private in other packages.
- **Default Public** keyword applied to a class, makes it available/visible everywhere. Applied to a method or variable, completely visible.
- **Private** fields or methods for a class only visible within that class. Private members are *not* visible within subclasses, and are *not* inherited.
- **Protected** members of a class are visible within the class, subclasses and also within all classes that are in the same package as that class.

Visibility

```
public class Circle {
    private double x, y, r;
    // Constructor
    public Circle (double x, double y, double r) {
        this.x = x;
        this.y = y;
        this.r = r;
    }
    //Methods to return circumference and area
    public double circumference() { return 2*3.14*r; }
    public double area() { return 3.14 * r * r; }
}
```
Visibility

Circle

Accessors – “Getters/Setters”

```java
public class Circle {
    private double x, y, r;

    //Methods to return circumference and area
    public double getX() { return x; }
    public double getY() { return y; }
    public double getR() { return r; }
    public double setX(double x) { this.x = x; }
    public double setY(double y) { this.y = y; }
    public double setR(double r) { this.r = r; }
}
```

Objects Cleanup/Destructor

- Unlike c and c++, memory deallocation is automatic in java, don’t worry about it
- no dangling pointers and no memory leak problem.
- Java allows you to define `finalize` method, which is **invoked** (if defined) just before the object destruction.
- In way, this presents an opportunity to perform record-maintenance operation or cleanup any special allocations made by the user.

```java
// done with this circle
protected void finalize() throws IOException {
    Circle.numCircles = Circle.numCircles--;
    System.out.println("number of circles:" + Circle.num_circles);
}
```

Summary

- Static members play a role similar to the global members of classes. They can only access other static members in the class.
- Objects can be passed as parameters and they can be used for exchanging messages (data).
- Delegation enables an object to pass responsibilities to other objects.
- Encapsulation/Data hiding helps in protecting data from accidental or wrong usage and also offers better security for data.
- Java clean-ups object resources automatically, however, users can provide “finalize()” method to do any user-level related clean-up activities.