Static, Final & Memory Management

The static keyword

What if you want to have only one piece of storage regardless of how many objects are created or even no objects are created? What if you need a method that isn’t associated with any particular object of the class? That is, you need a method that you can call even no objects are created.

The static keyword comes into play for solving the above stated problem. When you use static with something (data or method), the data or method is not tied to any particular object.

Class Variables

The declaration of a data member inside the class using the static keyword results in a class variable. There is only one copy of the class variables exist in the memory. One of the significant things about the class variables is that all instances (objects) of the class share the class variables.

It is not necessary to instantiate an object of a class to access class variables as it happens with the case of instance variables (To access an instance variable, you need an object). Class variables are accessed using the name of the class and name of the class variable joined by the dot operator (.) However you can also access class variables using objects but this practice is generally avoided.

Remember that since all objects of the class share the class variable, if any object modifies it, it is modified for all the objects of the class.

Class Methods

The declaration of a member method inside the class using the static keyword results in a class method. This resulted in one copy of the static method for class. Same as with the class variables, class methods can be invoked (call) without a requirement to instantiate an object of the class.

Class methods similarly to class variables are accesses by class name or object of the class but the later approach is avoided generally. Dot operator is used for accessing class methods on the same pattern of class variables.
Example

We have a student class with static variable countStudents and static method getCountStudents. The non-static members include name, roll no, getter/setter and toString method.

Suppose that we create two objects of Student class than memory picture look like one given below. Note that there is only one copy of static members.

Points to Remember

- Occurs as a single copy in the class regardless of how many objects are created
- Static (class) variables & methods are not associated with the object, they belong to class
- They exist prior the creation of any object
- A static method have no this reference as they belong to class and this belongs to objects
- The static method can only access static members (variables + methods) and cannot access non-static members(instance variables + member methods) of a class
Memory Management

Java manages the memory itself and performs the garbage collection and eliminates the need to free objects explicitly as it happened with C++.

When an object has no references to it anywhere except in other objects that are also unreferenced, its space can be reclaimed. Before the object is destroyed, it might be necessary for the object to perform some actions. For example, to close an opened file. How the programmer can make sure that necessary tasks would be performed before its (object) memory reclaimed by the garbage collector.

**finalize() method**

The finalize method comes into picture for solving the above mentioned problem. Define a finalize method inside a class and write any clean up code like closing a connection inside it. You can never call it directly.

Java ensures that finalize method must be called before any object’s occupied memory is reclaimed. A garbage collector reclaims objects in any order and we can not predict when the garbage collector will free the memory.

**System.gc()**

However, we can request to JVM to run the garbage collector to reclaim unreferenced memory. But remember, it’s a request, it does not guaranteed that garbage collector will run.

**Finalize is Not Destructor**

C++ programmers should take into an account that finalize method in java is not a destructor.

The destructor in C++, if exists, is always invoked when the object goes out of scope (as it is done by using the delete operator to return the memory back to the system). Freeing the memory is the programmer responsibility.

Even though, the finalize method is always invoked before the garbage collector reclaims memory in java, but we doesn’t specify/use any such delete operation as it is the responsibility of garbage collector. We only can write any clean up code into it. The job of freeing the memory is taken away from the programmer.
Example Code

We make some modifications to our Student class defined in the last handout. This example code demonstrates the usage of static and finalizes method. The changes made in Student class are highlighted as bold.

A static variable countStudents is added to the class with its getter method, to keep track of how many objects currently reside in the memory. In this class, toString and finalize methods are overridden as they are present in the Object class and in java every class by default inherits from Object class. Inheritance and overriding are going to be discussed in greater detail in coming handouts.

Student Class Code

```java
// File Student.java
/* Demonstrates the most basic features of a class. A student is defined by their name and rollNo. There are standard getter/setters for name and rollNo.

NOTE A well documented class should include an introductory comment like this. Don't get into all the details - just introduce the landscape.
*/

public class Student {
    private String name;
    private int rollNo;

    private static int countStudents = 0;

    public static int getCountStudents() {
        return countStudents;
    }

    // Standard Setters
    public void setName(String name) {
        this.name = name;
    }

    // Note the masking of class level variable rollNo
    public void setRollNo(int rollNo) {
        if (rollNo > 0) {
            this.rollNo = rollNo;
        } else {
            this.rollNo = 100;
        }
    }

    // Standard Getters
```
public String getName () {
    return name;
}
public int getRollNo () {
    return rollNo;
}

// Default Constructor
public Student() {
    name = "not set";
    rollNo = 100;
    countStudnets += 1;
}

// parameterized Constructor for a new student
public Student(String name, int rollNo) {
    setName(name); //call to setter of name
    setRollNo(rollNo); //call to setter of rollNo
    countStudnets += 1;
}

// Copy Constructor for a new student
public Student(Student s) {
    name = s.name;
    rollNo = s.rollNo;
    countStudnets += 1;
}

// method used to display method on console
public void print () {
    System.out.println("Student name:" +name+);
    System.out.println("Roll no:" +rollNo);
}

// Overridden methods

// Overriding toString method of class java.lang.Object
public String toString () {
    return ("name: "+name + "RollNo: " + rollNo);
}

//Overriding finalize method of Object class
protected void finalize () {
    countStudents -= 1;
}
} // end of class
Client Code or Driver Program

```java
// File Test.java

public class Test{

    public static void main (String args[]){
        int numObjs;

        numObjs = Student.getCountStudents();
        System.out.println("Students Objects:"+numObjs);

        Student s1 = new Student("ali", 15);
        System.out.println("Student:" + s1.toString() );

        numObjs = Student.getCountStudents();
        System.out.println("Students Objects:"+numObjs);

        Student s2 = new Student("usman", 49);

        //implicit call to toString()
        System.out.println("Student:" +s2);

        numObjs = Student.getCountStudents();
        System.out.println("Students Objects:"+numObjs);

        s1 = null;

        // request the JVM to run the garbage collector But there is
        // no guarrantee that garbage collector will run
        System.gc();

        numObjs = Student.getCountStudents();
        System.out.println("Students Objects:"+numObjs);

    } //end of main
    } //end of class
```

Compile & Execute

Compile both classes using `javac` command. Run Test class more than once using `java` command to figure out the difference in outputs because we never know, garbage collector will run or not in this turn.
The final keyword

Some instance variables need to be modified and some do not. Such variables are known as constants (Constants variables, once declared and initialized, their values cannot be modified later in any part of the program). In java, keyword **final** is used to achieve the above stated effects.

The programmer may use the keyword **final** to specify that a variable is not modifiable (constant) and that any attempt made to modify final variable results in compile time error. The final variables must be initialized at the time of their declaration or in every constructor of the class.

**Declaring final Variable**

The following snippet of code shows the both approaches of declaring and initializing final variables.

```java
// File FinalTest.java
public class FinalTest {

    public final int INCREMENT = 5;

    public final int DECREMENT;

    // Default Constructor
    public FinalTest() {
        DECREMENT = 3;
    }

    public useFinal() {
        int a = 2;
        a = a + INCREMENT;

        compile time error, can’t modify constant
        //DECREMENT++;  
    }

} // end of class
```

**Note:** It is strongly recommended to follow naming conventions.
References:

- Sun java tutorial: http://java.sun.com/docs/books/tutorial/java
- Java tutorial by Dick Baldwin: http://www.dickbaldwin.com/java
- Thinking in java by Bruce Eckle