Objects and Classes

Part 5 – Final Variables & Static Variables and Methods

Chapter 4, Core Java, Volume I

Contents

- Final Instance Variables
- Static Variables
- Static Methods
- Summary on Method Calls

Final Instance Variables

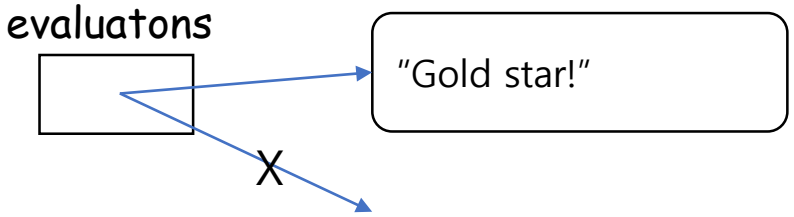
- A final instance variable cannot change:
  - useful for fields whose type is primitive or an immutable class
  - e.g. hireDay in Employee class
- Final instance variables must be initialized when the object constructed.
  - at the declaration
  - at the constructor

```
public class X
{
   private final int a;
   private final int b = 10;

   public X(int val)
   {
      a = val;
   }
}
```

- Caution: A final object can still be mutated:

```
// in Employee class
private final StringBuilder evaluations;
public Employee() { evaluations = new StringBuilder(); . . . }
public void giveGoldStar() { evaluations.append("Gold star!\n"); }
```



Static Fields

- A static field exists once per class (also called class variables):
  - useful for storing class-wide information
  - e.g. how many objects are constructed from the class
    sequential number for the objects of the class

```
class Employee {
   private static int nextId=1;       // one field per class
   private int id;                    // one field per object
   public void setId() { id = nextId; nextId++; }
}
```

Static Constants

- Private static final fields defined in a class is shared constants within the class

```
public class CardDeck
{
   private static final int MAX = 52;
      // Accessible anywhere inside CardDeck
   public void dealCards {  … MAX … }
   public void shuffle {  … MAX … }
   . . .
}
```

- A public static final field is a shared constant within alll classes: Math.PI, System.out, etc.

```
public class Math
{
   public static final double PI = 3.14159265358979323846;
      // Accessible anywhere as Math.PI
   . . .
}
…
MATH.PI
…
```

```
public System {
    public static final PrintStream out = …'
    …
}


…
System.out.println(…);
…
```

Static Methods

- A static method doesn't operate on objects.
- Supply class name when calling the method:

```
int n = Employee.getNextId(); // without a Employee Object
Math.pow(a, b)  // with out a Math object
```

- A static method uses no this.
- Static methods can only access static fields:

```
public static int getNextId()
{
   return nextId; // returns static field
}
```

- Static methods cannot call non-static methods directly.
- Non-static methods can access static fields and static methods.
- The main method is static because no objects have been constructed when the program starts.

Static Test

```
public class StaticTest
{
   public static void main(String[] args)
   {
     Employee[] staff = new Employee[3];
      staff[0] = new Employee("Tom", 40000);
      staff[1] = new Employee("Dick", 60000);
      staff[2] = new Employee("Harry", 65000);
      // print out information about all Employee objects
      for (Employee e : staff)
      {
         e.setId();
         System.out.println("name=" + e.getName() + ",id=" + e.getId() + ",salary="
               + e.getSalary());
      }
      int n = Employee.getNextId(); // calls static method
      System.out.println("Next available id=" + n);
   }
}
```

Static Test

```
class Employee
{
  private static int nextId = 1;
  private String name;
  private double salary;
  private int id;
  public Employee(String n, double s)
  {
    name = n;
    salary = s;
  }

  public String getName()
  {   return name;  }
  public double getSalary()
  {
    return salary;
  }
```

```
public int getId()
  {
    return id;
  }

  public void setId()
  {
    id = nextId; // set id to next available id
    nextId++;
  }

  public static int getNextId()
  {
    return nextId; // returns static field
  }

}
```

Summary on Method Calls

- Two types of methods
  - non-static methods (object methods)
  - static methods (class methods)
- Method calls

```
m() // non-static method
{
   f();          // non-static method in the same object; this.f()
   o.g();        // non-static method in other object
   s();          // static method in the same class; not this.s()
   C.s();         // static method in other class
}
s() // static method
{
   s2();         // static method in the same class
   C.s();        // static method in other class
   o.g();        // non-static method in an object
   m();          // non-static method in the same class;  not allowed!!
}
```