Sunday, December 30, 2018 2:16 PM

The static keyword

Until we started dealing with objects, all the methods we wrote were "static" and the only variables we defined outside of static methods were class constants, who's values never varied.

```
public class MyClass{
   public static final int MAX = 100;

   public static void myMethod() {
}
```

Since we started writing our own methods we stopped using the "static" in front of methods and fields, so what does static really do, and how is it useful? Static simply means "not part of an object", or "not dynamically created" as part of an object when the object is created.

Static Fields

```
public class MyClass{
   private String field1;
   private static int field2;
}
```

When a field is static it is shared by all objects in a class. Sometimes it's referred to as "a class variable". This means each individual instance of that object can all read the field's value and update it. There is only one instance of this field across all instances of the object so the static field's current value is the last value set by any object.

This can be really handy for:

- accumulating statistics or totals for all members of class: total sales for all vendor objects, the number of Bicycle objects created, etc.

```
public class Bicycle {
   private static int bikeCount;

   public Bicycle() {
      bikeCount++; // stores the count of how many Bicycle objects have been
```

created
  }
}

- shared values that all objects of a class use: current speed limit, value for gravity, and other values that might change but need to affect all objects.

```
public class DriversLicense {
  private static int minDrivingAge = 21;

  public void newLaw(int newDrivingAge) {
    minDrivingAge = newDrivingAge;
  }
}
```

Static fields belong to the class as a whole, not to a specific object of the class. So, if you reference it outside the class, you use the class name to qualify it, not an object name: for example, Bicycle.numberOfBikes, Vendor.totalSales, etc.

```
Bicycle bike1 = new Bicycle();
Bicycle bike2 = new Bicycle();
int bikeCount = Bicycle.bikeCount; // should store 2
```

The class constants we saw before are simply class variables that are also final and so can never change. When they are public, they are usually "universal constants" like Math.PI or Color.BLUE.

Static fields behave differently from the typical instance fields.

static fields VS instance fields

shared by all objects of the class	part of each object (instance)
Class.name	objectVariable.name
one value only	may have different values in each object
stored in only one place	stored in memory of each object
"statically" allocated before any objects are created	"dynamically" allocated each time an object is created

Static Methods

Like static fields are shared across all instances of an object, so are static methods

Like static fields are shared across all instances of an object, so are static methods. This means that in any class, a static method is part of the class as a whole:

- The main method that starts a Java program is always static. It usually creates the first objects. There is only one main method for each execution of a Java program.
- From within a static method you cannot call a non-static method without specifying which instance that method belongs to; for example, bike1.getBrand(), or vendor1.addSale().
- When a static method exists within an object declaration it's often a "public service" method. This is often something that doesn't deal with any objects at all, but just acts on data provided as parameters or static variables. For example, whenever we use methods from the Math class: Math.pow(2,2).