Sunday, December 30, 2018 2:02 PM

Constructors

Recall that a client class creates objects by using the new operator with a statement similar to this one:

```
Bicycle bike1 = new Bicycle("Giant","OCR-1",899.99,28);
```

The right hand side of the declaration looks a bit like a method call that takes in parameters. This is because the keyword "new" does call a method, a special method called a "constructor".

Writing Constructors

In your object class you can specify what happens when your client calls the "new" keyword by adding a special type of method, the constructor. Here is what the constructor looks like for our Bicycle class:

```
public Bicycle (String myBrand, String myModel, double myPrice) {
   this.brand = myBrand;
   this.model = myModel;
   this.price = myPrice;
}
```

As you can see there are a couple things in the constructor header that are different from typical methods:

0.The name of the method is the class name "Bicycle"

0.There is no return type (void, int, etc.) specified for the method, because a constructor will always return an object of its own class. If you specify a return type by mistake, even void, you will have a standard method, not a constructor and the client call above won't work.

The general format for a constructor header looks like this:

```
public ClassName (parameters) {
```

Since the constructor is called when creating a new instance of an object you will want to do any work to set up the object for use by the client. Often this means initializing the fields. If we want the user to be able to specify the initial values for the fields, we need parameters that match the fields.

```
public class Bicycle {
   String brand;
   String model;
   double price;
   boolean isSold;
   double salePrice;

   public Bicycle (String myBrand, String myModel, double myPrice) {
      brand = myBrand;
      model = myModel;
      price = myPrice;
      isSold = false;
      salePrice = 0.0;
   }

}
```

Notice:

- This constructor only accepts the value of three fields from the client; the other fields are set to defaults.
- Because we set all the fields values within the constructor we no long need to set their values at their declaration. This approach of splitting declaration and initialization is the preferred style.
- We used different names for the parameters than the fields, but because they store the same type of data it’s really tempting to write:

```
public Bicycle (String brand, String model, double price) {
   brand = brand;
   model = model;
   price = price;
   ...
```

But, this will NOT set the values of the fields! Remember that parameters create local variables of the same name in the scope of a method, and that variable will "mask" the field of the same name in this method, so brand = brand uselessly sets itself to its own value. Oops!

Java does permit you to explicitly qualify that you want the object to refer to its own field by using the keyword this. You could write the constructor using the field names as parameter names as long as you distinguish the difference with the "this" keyword:

```
public Bicycle (String brand, String model, double price) {
   this.brand = brand;
   this.model = model;
   this.price = listPrice;
   ...
```

Default Constructors

You do not have to provide a constructor for an object at all. If you don't, Java provides a default for you known as the zero argument constructor. In the case of our Bicycle class it would look like this:

```
public Bicycle() {

}
```

If you do not specify a constructor yourself, then when the client class creates a new object it cannot pass in any arguments. This means that the fields will initially store their default values:

```
Bicycle bike1 = new Bicycle();
```

Overloading Constructors

You can in fact have as many constructors as you like through overloading. In this way you can allow users to decide what amount of data they want to specify when creating and object, and what amount they want to rely on default values.
To do this properly you should first create a constructor that initializes all fields inside the constructor:

```
public class Bicycle {
   String brand;
   String model;
   double price;
   int tireSize;
   boolean isSold;
   double salePrice;

   public Bicycle (String brand, String model, double price, boolean isSold, double
salePrice) {
      this.brand = brand;
      this.model = model;
      this.price = price;
      this.isSold = isSold;
      this.salePrice = salePrice;
   }
}
```

As you can see we have moved all the field declarations within the constructor, giving the user complete control over the initial state of the object. Now that this constructor exists you can add other constructors that include default values so your user doesn't have to set everything. You can call the most specific constructor from the less specific constructors to reduce redundancy. You call one constructor from the other with the use of the "this" keyword in the place a method name from the other with the use of the "this" keyword in the place a method name would normally be.

If some bikes don’t have a model, for example, you could provide:

```
// this uses the default values of 28 for the tireSize
// false for isSold and set the salePrice to the listPrice
public Bicycle(String brand, String model, double listPrice) {
   this(brand, model, listPrice, 28, false, listPrice);
}

// this uses all default values for a new Bicycle
public Bicycle() {
   this("", "", 0.0, 28, false, 0.0);
}
```

Now we can create new Bicycles in our client class with differing levels of initial information like so:

```
Bicycle specificBike = new Bicycle("Giant","OCR-1",899.99,28,false,599.99);
Bicycle genericBike = new Bicycle("Schwin","Cruiser",250.00);
Bicycle defaultBike = new Bicycle();
```