**Objects as Parameters**

**Learn about best practices for sending an object as a constructor parameter.**

We know that we can use constructor parameters to assign state to an object. Did you also know that we can send an object as a parameter to another object? In this article, we’ll discuss how an object can be sent as a parameter as well as the best practices for accomplishing that.

**Using an Object as a Parameter**

First, let’s create a class called **Cupcake**:

class Cupcake {  
  String flavor;  
  boolean sprinkles;  
   
  public Cupcake(String type, boolean hasSprinkles) {  
    flavor = type;  
    sprinkles = hasSprinkles;  
  }  
}

Now, we’ll create a class called **Bakery** whose constructor accepts an object of **Cupcake** as a parameter:

class Bakery {  
  Cupcake bakeryCupcake;  
  double price;  
  String giveTotal;  
   
  public Bakery(Cupcake cupcakeType, double priceOf) {  
    bakeryCupcake = cupcakeType;  
    price = priceOf;  
    giveTotal = "The " + bakeryCupcake.flavor + " cupcake is €" + price;  
  }  
}



Inside the **main()** method of the **Bakery** class, we’ll create a new **Cupcake** object called **chocolateSprinkle**. Then, we’ll create a new **Bakery** object, called **myBakery**, that takes the **chocolateSprinkle** object as a parameter:

public static void main(String[] args) {  
  Cupcake chocolateSprinkle = new Cupcake("chocolate", true);  
  // Add your code below  
  Bakery myBakery = new Bakery(chocolateSprinkle, 3.25);  
  System.out.println(myBakery.giveTotal);   
}

When we run this program, we get the following output:

The chocolate cupcake is €3.25

**Creating a Copy of an Object**

If we use an object as a parameter for a constructor method, it’s a good programming practice to not modify said object unless the situation requires it. If we do find ourselves needing to modify an object sent as a parameter, we should set the instance variable to hold a *copy* of the referenced object instead of the original object.

Let’s say we added the line **bakeryCupcake.flavor = "vanilla";**to our **Bakery** constructor that changes the **flavor** value of the **chocolateSprinkle** object to **"vanilla".**

public Bakery(Cupcake cupcakeType, double priceOf) {  
  bakeryCupcake = cupcakeType;  
  // change cupcake object flavor to vanilla  
  bakeryCupcake.flavor = "vanilla";  
  price = priceOf;  
  giveTotal = "The " + bakeryCupcake.flavor + " cupcake is €" + price;    
}

What happens to the state of our original **chocolateSprinkle** object after we call the constructor of our **Bakery** object?

**Code Challenge**

Run the following code. Then, uncomment line 21. Feel free to change the current value on line 21. After, run the code again.

What happened to the state **flavor** for the object **chocolateSprinkle**? What about the state **flavor** for the parameter object **bakeryCupcake**?

class Cupcake {

  String flavor;

  boolean sprinkles;

  public Cupcake(String type, boolean hasSprinkles) {

    flavor = type;

    sprinkles = hasSprinkles;

  }

}

class Bakery {

  Cupcake bakeryCupcake;

  double price;

  String giveTotal;

  public Bakery(Cupcake cupcakeType, double priceOf) {

    bakeryCupcake = cupcakeType;

    price = priceOf;

    // Uncomment line below

    // bakeryCupcake.flavor = "vanilla";

    giveTotal = "The " + bakeryCupcake.flavor + " cupcake is €" + price;

  }

  public static void main(String[] args) {

    Cupcake chocolateSprinkle = new Cupcake("chocolate", true);

    // Create an object that takes in an object as a parameter

    Bakery myBakery = new Bakery(chocolateSprinkle, 3.25);

    // Output a value of parameter object

    System.out.println("Our object sent as a parameter has a flavor value of "  + myBakery.bakeryCupcake.flavor);

    // Output a value of original object

    System.out.println("Our original object has a flavor value of " + chocolateSprinkle.flavor);

  }

}

See how the **Bakery** constructor is impacting the **flavor** instance variable of our **chocolateSprinkle** object instead of only changing **bakeryCupcake**? Both of their **flavor** instance variables now have a value of **"vanilla"**.

Why did this happen? When an object is sent as an argument, the value sent to the parameter is a reference to the original object and not a copy of the original object; therefore, whenever we make a change to our parameter object, we are also making a change to our original object. Note that this doesn’t happen with primitive values because the parameter value is initialized with a copy of the argument value.

To only change the value of the parameter object and not the original object, we’ll discuss how to create a copy of an object.

One way to create a copy of an object is to make a copy constructor. The *copy constructor* is an additional class constructor; remember, a class can have multiple constructors as long as each constructor has its own unique signature. Copy constructors create a copy of an object by taking in an object and setting the values of the new object to the same values of the parameter object.

For example, we’ll add a copy constructor to the **Cupcake** class:

class Cupcake {  
  String flavor;  
  boolean sprinkles;  
  // original constructor  
  public Cupcake(String type, boolean hasSprinkles) {  
    flavor = type;  
    sprinkles = hasSprinkles;  
  }  
  // copy constructor  
  public Cupcake(Cupcake copy) {  
    flavor = copy.flavor;  
    sprinkles = copy.sprinkles;  
  }  
}

We now have a copy constructor, but in order to call it, we need to make a minor change to the **Bakery** constructor:

public Bakery(Cupcake cupcakeType, double priceOf) {  
  // Create a copy Cupcake object  
  Cupcake copy = new Cupcake(cupcakeType);  
  // set bakeryCupcake to hold value of object copy  
  bakeryCupcake = copy;  
  price = priceOf;  
  bakeryCupcake.flavor = "vanila";  
  giveTotal = "The " + bakeryCupcake.flavor + " cupcake is €" + price;  
}

In our constructor, we created a new object, **copy**, that calls the copy constructor in order to create and store a copy of our parameter object **cupcakeType**. Then, we set our instance variable, **bakeryCupcake** to store the value of **copy**.

Now that we have a copy of the object, we don’t have to worry about an object sent as a parameter to the **Bakery** class changing the state of our **Cupcake** objects, like **chocolateSprinkle**. By adding a copy constructor to our code, our output changes.

**Code Challenge**

Take a look at the complete code. Feel free to play around with the program to get a better understanding of copy constructors and using objects as parameters.

class Cupcake {

  String flavor;

  boolean sprinkles;

  public Cupcake(String type, boolean hasSprinkles) {

    flavor = type;

    sprinkles = hasSprinkles;

  }

  // copy constructor

  public Cupcake(Cupcake copy) {

    flavor = copy.flavor;

    sprinkles = copy.sprinkles;

  }

}

class Bakery {

  Cupcake bakeryCupcake;

  double price;

  String giveTotal;

  public Bakery(Cupcake cupcakeType, double priceOf) {

    // Create a copy Cupcake object

    Cupcake copy = new Cupcake(cupcakeType);

    // set bakeryCupcake to hold value of object copy

    bakeryCupcake = copy;

    price = priceOf;

    bakeryCupcake.flavor = "vanila";

    giveTotal = "The " + bakeryCupcake.flavor + " cupcake is €" + price;

  }

  public static void main(String[] args) {

    Cupcake chocolateSprinkle = new Cupcake("chocolate", true);

    Bakery myBakery = new Bakery(chocolateSprinkle, 3.25);

    // Output a value of parameter object

    System.out.println("Our object sent as a parameter has a flavor of "  + myBakery.bakeryCupcake.flavor);

    // Output a value of original object

    System.out.println("Our original object has a flavor of " + chocolateSprinkle.flavor);

  }

}

We now see an output that reflects that even though we changed the value of an object sent as an argument value, the state of our original object **chocolateSprinkle** remains unchanged.

**Conclusion**

Nice job reaching the end of this article. Let’s recap what we learned:

* Objects can be sent as a parameter to a constructor.
* In order to avoid changing the state of the original object, it’s best practice to create a copy of the object sent as a parameter.
* One method of copying an object is creating a copy constructor that copies the attributes of an object into a new object.