Constructors

by Sophia

WHAT'S COVERED

In this lesson, you will explore initializing objects from classes and deleting objects when they are no longer needed. Specifically, this lesson covers:

1. Constructors
2. Creating an Object

1. Constructors

Each time an object is created from a class, through class instantiation, a Java constructor is invoked. The **constructor's** sole purpose is to initialize, or assign values to, the object's attributes. It is only used within a class. The Java constructor is similar to constructors in other programming languages like C# and C++.

In Java, there are two kinds of constructors. They are default and parameterized. The **default constructor** does not have any parameters. The default, or parameterless, constructor does the internal bookkeeping. This means that it allocates memory for an object and initializes the attributes to default values. An example of this is the use of 0 for an int or an empty String for a String attribute. If the default values for the attributes need to be changed, the code for the class will need to provide the methods to do so. This means that if a class uses a default constructor, the values in the attributes can be changed after the object has been created.

A **parameterized constructor**, as the name indicates, includes method parameters that are used to initialize the values of some or all of the attributes. If a class has a parameterized constructor, the default constructor without parameters is no longer automatically available, unless it is explicitly declared.

KEY CONCEPT

A parameterized constructor is not required, but it is a good practice to use. To set attributes to specific values and not depend on system-defined default values, you would use this method.

Java has very specific requirements for the declaration of a constructor. As you have seen, the name of the constructor must always match the name and capitalization of the class name. Since a class name should always begin with a capital letter, the name of the constructor also should begin with a capital letter.

The second distinctive factor in the declaration of a constructor is that the constructor has no return type. This includes `void`.

KEY CONCEPT

Before moving on, it is important to acknowledge another kind of constructor besides a parameterized constructor. The simplest form of a constructor is the default constructor that takes no parameters. In fact, if there is no parameterized constructor in a class, the default constructor does not need to be declared or defined. The compiler just generates the code needed to allocate memory and set up the class.

Here is a sample class (`PeopleCounter`) that does not have an explicitly defined constructor. It relies on the default constructor.

```
public class PeopleCounter {
  private long count = 0;

  public void anotherOne() {
    count++;
    System.out.println("So far " + count);
  }
}
```

To start the `PeopleCounter` with a count value other than 0, you could create a parameterized constructor like the one shown in this version of the class:

```
public class PeopleCounter {
  private long count = 0;

  public PeopleCounter(long count) {
      this.count = count;
  }

  public void anotherOne() {
    count++;
    System.out.println("So far " + count);
  }
}
```

When defining a parameterized constructor like this, though the default (parameterless) constructor is no longer available, include it along with the parameterized constructor.

KEY CONCEPT

There can be more than one constructor in a class, as long as they take different types or numbers of parameters (as with any overloaded method):

```
public class PeopleCounter {
  private long count = 0;

  // Default parameterless constructor
  // Attribute count will have the initial value declared above
  public PeopleCounter() {}

  // Parameterized constructor
  // The count passed in will override initial value of count
  public PeopleCounter(long count) {
      this.count = count;
  }

  public void anotherOne() {
    count++;
    System.out.println("So far " + count);
  }
}
```

You may recognize the syntax used to define constructors as it's somewhat similar to the creation of any method. First, though, it bears repeating that a constructor has no return type. Then comes the name of the method, which must match the name of the class in spelling and capitalization. Since the class name should start with the capital letter, the constructor's name must start with a capital letter. This is then followed by parentheses that may or may not contain parameters.

```
public class PeopleCounter {
  private long count = 0;

  public void anotherOne() {
    count++;
    System.out.println("So far " + count);
  }
}
```

Here, the attribute is assigned a value of x to 0. This means that it will always start with the value 0. To have a different initial value for x would require using a constructor with an int parameter to pass in a different default value. Having a class with attributes with default value(s) is only unique to that class. So, for our example PeopleCounter class, the variable `x` is only available to the PeopleCounter class. Let's create a new class called "User" that can be used for an application/program that creates user login information.

First, you would create a class named `User`. This class contains private `String` attributes named `userName` and `password`. The values of these attributes are set via the `User()` constructor, which has two `String` parameters. In this case, a parameterless constructor (the default constructor) would not be very helpful because there are not likely to be good default values for the attributes in the `User` class.

```
public class User {
  private String userName;
  private String password;

  public User(String userName, String password) {
    this.userName = userName;
    this.password = password;
  }

  // Allow read-only access to user name
  public String getUserName() {
    return userName;
  }

  // There is no direct access to the password.
  // In real code there would be a method check
  // for a match with the stored password.
}
```

This code can't do a lot, but it can create a `User` object and also has a method to return the user name, `getUserName()`.

The method has been started; however, it does not define what comes next. At this point, if the code was executed, the constructor, `User()`, executes and it would set the values for the `userName` and `password` attributes. The constructor's parameters would hold whatever argument data that is being passed.

HINT

Remember that this is meant to be a `User` class about a user and looks to have attributes that reflect that kind of data. If you wanted each user to have a `userName` attribute that contains the user's username along with a password, the attribute would store the password.

You can define and populate those attributes by doing the following.

```
public class User {
  private String userName;
  private String password;

  public User(String userName, String password) {
    this.userName = userName;
    this.password = password;
  }
```

```
  // Allow read-only access to user name
  public String getUserName() {
    return userName;
  }

  // There is no direct access to the password.
  // In real code there would be a method check
  // for a match with the stored password.
}
```

The first indented line of the constructor sets the attribute named `userName` for the new instance to the value passed via the `userName` parameter. Note how the attribute `userName` is accessed via `this.userName` to distinguish it from the `userName` parameter. The keyword `this` refers to the current object being constructed or used. The second line in the body of the constructor (inside the curly brackets that mark the code executed when the constructor is called) sets the value of `this.password` to the `String` passed via the `password` parameter. A couple of conventions were used when setting up the new class. They include:

- Class names should be capitalized.
- Attributes must be declared private and use the same standard as used for creating variable names.

You will now focus on creating an instance of a class, using what you know regarding how a constructor is used to create an object.

TERMS TO KNOW

Constructor

The constructor's sole purpose is to initialize an object and, if it is a parameterized constructor, to set the object's attributes to specific (non-default) values. A constructor only exists inside a class and is only used to set up a new object based on the class's template.

Default Constructor

The default constructor does not have any parameters. It allocates memory for an object and initializes the attributes to default values.

Parameterized Constructor

Includes method parameters that are used to initialize the values of some or all of the attributes.

2. Creating an Object

Now that the class is created, you will learn how to create instances of it. The format would look something like this in relation to the defined `User` class:

EXAMPLE

```
User userObject = new User("userNameString", "passwordString");;
```

You can use the class in an application with the class name `UserExample` (typed into the Main.java file renamed as UserExample.java). You would need to replace the instanceName with the variable name that you choose and the `usernameValue` and `passwordValue` with their respective argument values. Let's say that you want to create a `User` object called "account" with the username "sophia" and the password as "myPass":

```
class UserExample {
  public static void main(String[] args) {
    User account = new User("sophia", "mypass");
  }
}
```

If the code is run as is, it will create an instance of the `User` class passing the arguments "sophia" and "mypass" as argument values. It will also assign that returned object to the variable "account". It may help to see more details by printing the username out:

```
class UserExample {
  public static void main(String[] args) {
    User account = new User("sophia", "mypass");
    System.out.println("Account created for " + account.getUserName());
  }
}
```

The output screen should look like this:

```
Account created for sophia
```

BIG IDEA

It is important to note that you will see the term "object" and "instance" used through these tutorials. They mean the same thing. It's important to note that an instance of an object is just a way to hold information about an element that's similar to other elements of the same class.

The terms "properties" and "attributes" mean the same thing as well. The element's attributes may be unique to other instances. For example, in an application, it would make sense for each of the users to have unique usernames. However, some users may have the same value for their password.

TRY IT

Directions: Try creating an instance/object from a class. Start with our example above and see if you can modify it to something unique for yourself.

BRAINSTORM

What other sorts of data might the `User` class contain about the user? Which data types would you use for these new attributes? (When working in Java, it's always important to be mindful of data types.)

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source py4e.com/html3/

TERMS TO KNOW

Constructor

The constructor's sole purpose is to initialize an object and, if it is a parameterized constructor, to set the object’s attributes to specific (non-default) values. A constructor only exists inside a class and is only used to set up a new object based on the class's template.

Default Constructor

The default constructor does not have any parameters. It allocates memory for an object and initializes the attributes to default values.

Parameterized Constructor

Includes method parameters that are used to initialize the values of some or all of the attributes.