Constructors

In this lesson, an explanation of what are constructors in classes and the different types of constructors that can be created is provided.

We'll cover the following

Introduction

A *constructor* is automatically called when an *object* of the *class* is declared.

- A *constructor* is a *member* method that is usually `public`.
- A *constructor* can be used to initialize *member* variables when an *object* is declared.

> **Note:** A *constructor's* **name** must be the **same** as the *name* of the *class* it is declared in.

A constructor cannot *return* a value.

> **Note:** No *return* type, not even `void` can be used while declaring a *constructor*

```
class Pet {
    private int petAge;
    private String petType;
    private String petName;

    //This is the constructor without any paramters
    public Pet() {
        petAge = 0;
        petName = "";
        petType = "";
    }


    //This is the constructor with parameters
    public Pet(String name, String type, int age) {
        petAge = age;
        petType = type;
        petName = name;
    }

    //This is the copy constructor
    public Pet(Pet copyThisPet) {
        petName = copyThisPet.petName;
        petType = copyThisPet.petType;
        petAge = copyThisPet.petAge;
    }
}
```

Understanding constructors

As you can see the constructor syntax is pretty close to that of declaring a method. There are **three** types of constructors.

- Line 7: `Default constructor` takes no parameters and is simply used to create an object of class `Pet` without any value assigned to the variables within it.
- Line 14: `Constructor Overloading` - this constructor takes in *parameters* which are then stored in the respective variables of the newly created object.
- Line 21: `Copy constructor` - this has an object of the same class as a parameter. It then assigns the values of new objects variables to those of the input object. The purpose is to create a copy of an existing object into a new one.

> **Note:** In copy constructor, we “usually” assign all members of one object to the other. It isn’t necessary that all members are copied. It depends on the situation.

```
class Pet {
    private int petAge;
    private String petType;
    private String petName;

    //This is the constructor without any paramters
    public Pet() {
        petAge = 0;
        petName = "";
        petType = "";
    }
    //This is the constructor with parameters
    public Pet(String name, String type, int age) {
        petAge = age;

        petType = type;
        petName = name;
    }

    //This is the copy constructor
    public Pet(Pet copyThisPet) {
        petName = copyThisPet.petName;
        petType = copyThisPet.petType;
        petAge = copyThisPet.petAge;
    }

    public void print() {
        System.out.println("Pet Name: " + petName);
        System.out.println("Pet Type: " + petType);
        System.out.println("Pet Age: " + petAge);
    }

}
class pet_list {
    public static void main(String[] args) {
        Pet dog = new Pet();
        dog.print();

        Pet cat = new Pet("Princess", "cat", 45);
        cat.print();

        Pet cat_copy = new Pet(cat);
        cat_copy.print();

    }
}
```

Invoking a constructor

As you can see above in **lines 36, 39, and 42**, the way to call a *constructor* is not like a normal *member* function.

- It is called in *object* declaration.
- It creates a `Pet` object.
- Then *calls* the *constructor* to initialize *variables*, this is preceded by the keyword `new`.

In the example, we also *declare* and use the **default constructor** which takes no parameters and just *initializes* `petAge` to **0** and `petName` and `petType` to empty **Strings**. As you can see in **line 36** a *default constructor* is automatically called when you create an *object*, no *parameters* are needed.

> **Note:** It’s a good practice to use default constructors even if you don’t want to initialize any variables.

Similarly, on **line 39** an **overloaded constructor** is called, such that the constructor is given parameters which are then stored in the objects respective variables. This can be seen when we print the `cat` object.

Lastly, the third implementation, the **copy constructor** can be seen on **line 42**. This shows that an object of type `Pet` is passed as a parameter and the variable values of that object are copied into the new object that we create called `cat_copy`.

Now that we have a basic understanding of constructors in Java, the next lesson will be about member methods in Java.