

Forum:[Hack Forums](#)

Thread: [\[P1\] Java and the OOP paradigm - The basics of objects](#)

magicparty - September 2, 2019, 11:02 pm

Seeing as my last programming tutorial (In the Python section) received both good feedback and some anticipation on me providing more I now bring to you a series of tutorials covering a language and paradigm that I consider myself to be somewhat of an expert with. --- At the end of this series I expect you to feel the same way!

I would also like to add that all my tutorials are off the top of my head entirely so if you see a typo or miscalculation please raise it to my attention immediately and it will be reviewed and fixed as fast as possible.

The basics

(0) Pre-requisites

Here I will list all of the knowledge you should acquire before attempting to follow through with this series.

Understanding of UML diagrams [Ideal Modeling & Diagramming Tool for Agile Team Collaboration](#)

Basic Java programming knowledge [Java Tutorial](#)

Ability to compile and run Java programs (I will be using the Terminal and Sublime Text/vim for the entire series)

Why use Sublime Text and not a robust IDE? --- Unless you are designing a giant collaborative project that requires serious organization there is really no reason to use more than a simple text editor with great features.

(1) Defining what an object is

What helped me understand what an Object is immediately was replacing the word class with the word object when discussing code.

Object oriented programming or OOP is a paradigm (or style) of programming that the Java language is based upon. Other languages such as C have been given an upgrade to support this paradigm which is what the ++ represents in the C++ programming language name!

Let's create a class or...object named "Orange" --- I will also give it a color and shape

```
Code class Orange {  
    public String color = "Orange";  
    public String shape = "Round";  
}
```

This can be saved as a separate file called "Orange.java" and compiled to create "Orange.class".

Code `javac Orange.java`

We have created our first object!

(2) Instantiation

I will cover this twice --- for now just the basics, but in a later part in this series I will go over how your objects are managed via Java's garbage collector.

What is Instantiation you ask? If you have ever seen some basic Java code in the passed you may have seen something similar to this example with the Orange object we have created.

```
Code Orange fruitOrange = new Orange();
```

The key statement is the new statement.

When you call this statement upon your Object "new Orange();" an instance of your object and all of its parameters are moved into memory.

The first part of the statement "Orange fruitOrange =" sets the variable "fruitOrange" as a pointer to the memory address where the object is located.

(3) Construction

There is ALWAYS a constructor even if you do not assign one!

Constructors are used to execute instructions upon instantiation and can either accept argument parameters, or no arguments at all (no-arg).

(a) The No-Arg constructor

The no-arg constructor is assigned by default when the code is compiled if no other constructor is provided. This constructor executes no code instructions and assigns no variables in memory but is necessary when running the Java byte code in the JVM.

(b) Arguments constructor

This is a section of code as explained before which can execute instructions upon object instantiation. The constructor is either Public or Protected (For now we will worry only about Public) which means it can be accessed from anywhere which is necessary for when we instantiate our object from other objects.

You may absolutely have multiple constructors and this is called "Overloading" which is a term that can also be applied to methods. It is good practice to overload an objects constructor with a custom no-arg constructor (Unless the object absolutely needs to be instantiated with certain values). This custom no-arg constructor usually assigns default values to critical variables in your object.

Let's switch it up and make a "Fruit" object and give it some constructors and variables.

"this" refers to the higher scope of the object --- You will see this in my example code, it is also good practice to give argument names the same name as the variable they are to assign.

```
Code class Fruit {
```

```
    public String type;  
    public String color;  
    public String shape;
```

```
    public Fruit(String type, String shape, String color) {
```

```
        this.type = type;  
        this.color = color;  
        this.shape = shape;
```

```
    }
```

```
    public Fruit() {
```

```

    this.type = "Orange";
    this.color = "Orange";
    this.shape = "Round";
}
}

```

We can now instantiate our object based upon our new constructor architecture.

```

Code Fruit orange = new Fruit();
Fruit apple = new Fruit("Apple", "Red", "Semi-Round");
(4) Wrapping up the basics

```

Let's make a program using our new knowledge of Objects!

```

Code class Fruit {

    public String type;
    public String color;
    public String shape;

    public Fruit(String type, String shape, String color) {

        this.type = type;
        this.color = color;
        this.shape = shape;

    }

    public Fruit() {

        this.type = "Orange";
        this.color = "Orange";
        this.shape = "Round";

    }
}

```

```

class HF {

    public static void main(String[] args) {
        Fruit orange = new Fruit();
        Fruit apple = new Fruit("Apple", "Red", "Semi-Round");

        System.out.println(orange.type);
        System.out.println(apple.type);
    }
}
compile
Code javac HF.java
run
Code Orange
Apple
A UML wrapping up what we have thus far:

```

Stay tuned for part two! --- This will be a fairly large series of tutorials.

MavenGradle - September 3, 2019, 10:47 am

You are writing a tutorial about object oriented programming in which you declare variables in a class but fail to implement encapsulation. Encapsulation is fundamental in OOP. Please fix this.

Your classes are also declared using package-private visibility. You might want to improve this tutorial by adding a step about class visibility. You could have declared your Fruit class as abstract and go even further by declaring your orange as a kind of fruit, for which I once again refer to one of the basic principles of OOP (see image above) although this is probably something you could explain in a part 2.

magicparty - September 3, 2019, 3:58 pm

Thank you for the input, I was saving those concepts on polymorphism and how public/private works for part 2.

Here is an outline of how I saw this series going:

Part 2 - Polymorphism, Heap & Stack

Part 3 - Methods & toString()

Part 4 - Inheritance, Abstraction, and Interfaces

This Part 1 was just to get the groundwork down for the basic syntax of an object.

You are 100% correct and the best possible way to make the code up to par with all OOP rules would look something like this:

Code public abstract class Fruit {

String shape;

String color;

protected Fruit(String shape, String color) {

 this.shape = shape;

 this.color = color;

}

protected Fruit() {

 this.shape = "Round";

 this.color = "Red";

}

public String getColor() {

 return this.color;

```

    }

    public String getShape() {

        return this.shape;

    }

    public String toString() {

        return "Your fruit has a shape: " + this.shape + " and a color: " + this.color;

    }

    public abstract float getArea();

}

public class Orange extends Fruit {

    private float radius;

    public Orange(String shape, String color, float radius) {

        super(shape, color);
        this.radius = radius;

    }

    public Orange() {

        super();
        this.radius = 10.0;

    }

    @Override
    public float getArea() {

        return (4 * Math.pow((3.14 * this.radius), 2));

    }
}
Code Fruit orange = new Orange("Round", "Orange", 20.0);

```

MavenGradle - September 3, 2019, 5:31 pm

Very well. Carry on! Seems like this will be an excellent addition to the Java subforum to get people interested.

newQuery - September 4, 2019, 5:43 am

Glad to see this happening! Keep up the good work, long life to Java!