Definition of OOPs Concepts in Java

OOPs, concepts in Java are the main ideas behind Java’s Object Oriented Programming. They are an [**abstraction**](http://stackify.com/oop-concept-abstraction/)**,**[**encapsulation**](http://stackify.com/oop-concept-for-beginners-what-is-encapsulation/), [**inheritance**](http://stackify.com/oop-concept-inheritance/), and[**polymorphism**](http://stackify.com/oop-concept-polymorphism/). Grasping them is key to understanding how Java works. Basically, Java OOPs concepts let us create working methods and variables, then re-use all or part of them without compromising security.

List of OOPs Concepts in Java

There are four main OOPs concepts in Java. These are:

* **Abstraction.** Abstraction means using simple things to represent complexity. We all know how to turn the TV on, but we don’t need to know how it works in order to enjoy it. In Java, abstraction means simple things like **objects**, **classes**, and **variables** represent more complex underlying code and data. This is important because it lets avoid repeating the same work multiple times.
* **Encapsulation.**This is the practice of keeping fields within a class private, then providing access to them via public methods. It’s a protective barrier that keeps the data and code safe within the class itself. This way, we can re-use objects like code components or variables without allowing open access to the data system-wide.
* **Inheritance.**This is a special feature of Object Oriented Programming in Java. It lets programmers create new classes that share some of the attributes of existing classes. This lets us build on previous work without reinventing the wheel.
* **Polymorphism.**This Java OOPs concept lets programmers use the same word to mean different things in different contexts. One form of polymorphism in Java is **method overloading**. That’s when different meanings are implied by the code itself. The other form is **method overriding**. That’s when the different meanings are implied by the values of the supplied variables. See more on this below.

How OOPs Concepts in Java Work

OOPs, concepts in Java work by letting programmers create components that can be re-used in different ways, but still maintain security.

How Abstraction Works

Abstraction as an OOPs concept in Java works by letting programmers create useful, reusable tools. For example, a programmer can create several different types of **objects**. These can be variables, functions, or data structures. Programmers can also create different **classes** of objects. These are ways to define the objects.

For instance, a class of variable might be an address. The class might specify that each address object shall have a name, street, city, and zip code. The objects, in this case, might be employee addresses, customer addresses, or supplier addresses.

How Encapsulation Works

Encapsulation lets us re-use functionality without jeopardizing security. It’s a powerful OOPs concept in Java because it helps us save a lot of time. For example, we may create a piece of code that calls specific data from a database. It may be useful to reuse that code with other databases or processes. Encapsulation lets us do that while keeping our original data private. It also lets us alter our original code without breaking it for others who have adopted it in the meantime.

How Inheritance Works

Inheritance is another labor-saving Java OOPs concept. It works by letting a new class adopt the properties of another. We call the inheriting class a **subclass** or a **child class**. The original class is often called the **parent**. We use the keyword **extends** to define a new class that inherits properties from an old class.

How Polymorphism Works

Polymorphism in Java works by using a reference to a parent class to affect an object in the child class. We might create a class called “horse” by extending the “animal” class. That class might also implement the “professional racing” class. The “horse” class is “polymorphic,” since it inherits attributes of both the “animal” and “professional racing” class.

Two more examples of polymorphism in Java are method overriding and method overloading.

In **method** **overriding**, the child class can use the OOP polymorphism concept to override a method of its parent class. That allows a programmer to use one method in different ways depending on whether it’s invoked by an object of the parent class or an object of the child class.

In **method overloading,**a single method may perform different functions depending on the context in which it’s called. That is, a single method name might work in different ways depending on what arguments are passed to it.

Examples of OOPs Concepts in Java

Let’s look at a few common examples of OOPs concepts in Java.

Short Encapsulation Example in Java

In the [example below](http://www.javatpoint.com/encapsulation), encapsulation is demonstrated as an OOPs concept in Java. Here, the variable “name” is kept private or “encapsulated.”

*//save as Student.java*

package com.javatpoint;

public class Student {

private String name;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name

}

}

*//save as Test.java*

package com.javatpoint;

class Test {

public static void main(String[] args) {

Student s = new Student();

s.setName(“vijay”);

System.out.println(s.getName());

}

}

Compile By: javac -d . Test.java

Run By: java com.javatpoint.Test

Output: vijay

Example of Inheritance in Java

It’s quite simple to achieve inheritance as an OOPs concept in Java. Inheritance can be as easy as using the **extends** keyword:

class Mammal {

}

class Aardvark extends Mammal {

}

For a full tutorial on the different ways to use inheritance in java, [see this blog post](https://www.tutorialspoint.com/java/java_inheritance.htm).

Short Example of Polymorphism in Java

In the example below of polymorphism as an OOPs concept in Java, we have two classes: Person and Employee. The Employee class inherits from the Person class by using the keyword **extends**. Here, the child class overrides the parent class. For the full example, [see this blog post](http://www.sitesbay.com/java/java-polymorphism).

class Person {

void walk() {

System.out.println(“Can Run….”);

}

}

class Employee extends Person {

void walk() {

System.out.println(“Running Fast…”);

}

public static void main(String arg[]) {

Person p = new Employee(); *//upcasting*

p.walk();

}

}

Best Practices for OOPs Concepts in Java

Since the aim of OOPs concepts in Java is to save time without sacrificing security and ease of use, the best practices are all oriented toward advancing that main goal.

* **DRY (Don’t Repeat Yourself).**This is the core concept in Java. You should never have two blocks of identical code in two different places. Instead, have one method you use for different applications.
* If you expect your Java code to change in the future, encapsulate it by making all variables and methods private at the outset. As the code changes, increase access to **“protected”**as needed, but not too **public.**
* **Single Responsibility.**Another best practice for OOPs concepts in Java is the Single Responsibility Principle. Simply put, a class should always have only one functionality. That way, it can be called and/or extended on its own when new uses arise for it, without causing **coupling** between different functionalities.
* **Open Closed Design.**Make all methods and classes Closed for modification but Open for an extension. That way, tried and tested code can remain static but can be modified to perform new tasks as needed.

For a good, full list of best practices for OOPs concepts in Java, [see this blog post](https://javarevisited.blogspot.com/2012/03/10-object-oriented-design-principles.html). You can also check out our article about [OOPs concepts in C#](https://stackify.com/oop-concepts-c-sharp/).