Java OOPs Concepts – Object Oriented Principles

By Lokesh Gupta | Filed Under: Object Oriented Principles

In this **Java OOPs concepts** tutorial, we will learn four major **object oriented principles – abstraction**, **encapsulation**, **inheritance**, and **polymorphism**. They are also known as **four pillars of the object oriented programming paradigm**.

1. ***Abstraction*** is the process of exposing the essential details of an entity, while ignoring the irrelevant details, to reduce the complexity for the users.
2. ***Encapsulation*** is the process of bundling data and operations on the data together in an entity.
3. ***Inheritance*** is used to derive a new type from an existing type, thereby establishing a parent-child relationship.
4. ***Polymorphism*** lets an entity take on different meanings in different contexts.

Collections in Java
Java Concurrency
Spring Boot Tutorial
Spring AOP Tutorial
Spring MVC Tutorial
Spring Security Tutorial
Hibernate Tutorial
Python Tutorial
Jersey Tutorial
Maven Tutorial
Log4j Tutorial
Regex Tutorial

Table of Contents

1. Abstraction

Abstraction **in OOPs** is very easy to understand when you relate it to the real time example. For example, when you drive your car you do not have to be concerned with the exact internal working of your car. What you are concerned with is interacting with your car via its interfaces like steering wheel, brake pedal, accelerator pedal etc. Here the knowledge you have of your car is abstract.

In computer science, abstraction is the process by which data and programs are defined with a representation similar in form to its meaning (*semantics*) while hiding away the implementation details.

In more simple terms, abstraction is to **hide information that is not relevant to context** or rather show only relevant information and to simplify it by comparing it to something similar in the real world.

Abstraction captures only those details about an object that is relevant to the current perspective.

Typically abstraction can be seen in two ways:

1. **Data abstraction**

Data abstraction is the way to create complex data types from multiple smaller data types – which is more close to real life entity. e.g. An `Employee` class can be a complex object of having various small associations.

```
public class Employee
{
    private Department department;
    private Address address;
    private Education education;
    //So on...
}
```

So, if you want to fetch information of a employee, you ask it from `Employee` object – as you do in real life, ask the person itself.

2. Control abstraction

Control abstraction is achieved by hiding the sequence of actions for a complex task – inside a simple method call, so logic to perform the task can be hidden from the client and could be changed in future without impacting the client code.

```
public class EmployeeManager
{
    public Address getPrefferedAddress(Employee e)
    {
        //Get all addresses from database
        //Apply logic to determine which address is preferred
        //Return address
    }
}
```

In above example, tomorrow if you want to change the logic so that everytime domestic address is always the preferred address, you will change the logic inside `getPrefferedAddress()` method, and client will be unaffected.

Read More : Understanding abstraction in java

2. Encapsulation

Wrapping data and methods within classes in combination with implementation hiding (through access control) is often called encapsulation in OOPs. The result is a data type with characteristics and behaviors. Encapsulation essentially has both i.e. information hiding and implementation hiding.

"**Whatever changes, encapsulate it**" – A famous design principle

Information hiding is done through using access control modifiers (public, private, protected) and `implementation hiding` is achieved through creation of interface for a class.

Implementation hiding gives the designer the freedom to modify how the responsibility is fulfilled by an object. This is especially valuable at points where the design (or even the requirements) are likely to change.

Let's take an example to make it more clear.

2.1. Information hiding

```
class InformationHiding
{
    //Restrict direct access to inward data
    private ArrayList items = new ArrayList();

    //Provide a way to access data - internal logic can safely be cha
    public ArrayList getItems(){
        return items;
    }
}
```

2.2. Implementation hiding

```
interface ImplemenatationHiding {
    Integer sumAllItems(ArrayList items);
}
class InformationHiding implements ImplemenatationHiding
{
    //Restrict direct access to inward data
    private ArrayList items = new ArrayList();

    //Provide a way to access data - internal logic can safely be cha
    public ArrayList getItems(){
        return items;
    }

    public Integer sumAllItems(ArrayList items) {
        //Here you may do N number of things in any sequence
        //Which you do not want your clients to know
        //You can change the sequence or even whole logic
        //without affecting the client
    }
}
```

Read More : Encapsulation In Java

3. Inheritance

Inheritance is another important concept in object oriented programming. **Inheritance in Java** is a mechanism by which one object acquires the properties and behaviors of the parent object. It's essentially creating a parent-child relationship between classes. In Java, you will use inheritance mainly for code re-usability and maintainability.

Keyword "`extends`" is used to **inherit a class in java**. The "`extends`" keyword indicates that you are making a new class that derives from an existing class. In the terminology of Java, a class that is inherited is called a **super** class. The new class is called a **subclass**.

A subclass inherits all the non-private members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.

e.g.

```
class Employee
{
    private Department department;
    private Address address;
    private Education education;
    //So on...
}
class Manager extends Employee {
    private List<Employee> reportees;
}
```

In above example, `Manager` is specialized version of `Employee` and reuses department, address and education from `Employee` class as well as define it’s own reportees list.

4. Polymorphism

Polymorphism is the ability by which, we can create functions or reference variables which behaves differently in different programmatic context.

In java language, polymorphism is essentially considered into two versions:

- Compile time polymorphism (static binding or **method overloading**)
- Runtime polymorphism (dynamic binding or **method overriding**)

Read More : Polymorphism in java

Above are four Java OOPs concepts, and I will suggest you to develop a good understanding of each one of it.

Happy Learning !!

References:

https://docs.oracle.com/javase/tutorial/java/concepts/
http://c2.com/cgi/wiki?InformationHiding

About Lokesh Gupta

A family guy with fun loving nature. Love computers, programming and solving everyday problems. Find me on Facebook and Twitter.

13

Leave a Reply

Join the discussion...

8 5 0 11

Subscribe newest oldest most voted

Burhan Ameen

Hi Lokesh,

Manager can not reuse department, address and education from Employee because these attributes are private.
Manager can reuse them if they are protected.

+ 1 − Reply 3 months ago

Priyak

Yes, he can resuse the. For brevity Lokesh has omitted the getters and setters I believe. You can access those data using the getters and setters of the Parent Class and initialize them using setters/chaining of constructors.

```
Child class {
@Override
public getSuperClassProperty() {
super.getProperty(); // This is defined in the parent class.
}
```

+ 0 − Reply 2 months ago

nisha

Wonderful points you have mentioned here, Its actually a great and helpful piece of information. I am satisfied that you simply shared this helpful info with us. Please stay us informed like this.
Thank you for sharing.

+ 0 − Reply 1 year ago

Ads

Control Abstraction and Implementation Hiding seems similar. Can you please elaborate those.

+ 4 − Reply 2 years ago

mayur

Informative and Insightful blog!
All i was looking for the OOPs concept in java in detail and i have found it very well here, OOPs concepts you have defined clearly and precise which is understandable for us. As beginner i have enjoyed your article and will look forward to read more from you. Thanks A lot for this meaningful article!

+ 0 − Reply 2 years ago

Pavan Raghani

Hi Lokesh,
Can you please point out some differences between inheritance and

composition. And as they say we should prefer composition over inheritance, but what is the reason behind it.

+ 0 − Reply 3 years ago

Lokesh Gupta

I will list out differences between inheritance and composition some other time. Main reason to prefer composition over inheritance is “loose coupling”. Inheritance introduces tight coupling which shall be avoided in most of the cases.

+ 0 − Reply 3 years ago

Arpit Agrawal

```
public class Person {

//composition has-a relationship
private Job job;

public Person(){
this.job=new Job();
job.setSalary(1000L);
}
public long getSalary() {
return job.getSalary();
}

}
```

The above example is for Composition in which we are creating object of dependent object inside constructor so i think it is tight coupling? right ?

+ 0 − Reply 2 years ago

Pradeep T

Hi Lokesh,

please clarify my doubt in spring.

Can we inject Service Class into Controller in spring framework?. this is one of the interview question i was faced in CTS Pune.

please provide me the answer with simple example. thanks in advance.

if any tutorials are there please send it to my mail: pradepp.tanugula@gmail.com

+ 0 − Reply 3 years ago

Lokesh Gupta

Yes. and that's how it should be done all the time... answering is.. using autowiring... refer how manager class in injected into controller class in https://howtodoinjava.com/spring/spring-mvc/spring-mvc-hello-world-example/

+ 0 − Reply 3 years ago

Alok Shukla

Sir, by mistake chenage word is written in place of change inside Control abstraction section of blog.

+ 0 − Reply 3 years ago

Lokesh Gupta

It's embarrassing. Thanks for pointing it out.

+ 0 − Reply 3 years ago

Binh Nguyen

Thanks, nice summary

+ 0 − Reply 4 years ago

Meta Links

Advertise
Contact Us
Privacy policy
About Me

Recommended Reading

10 Life Lessons
Secure Hash Algorithms
How Web Servers work?
How Java I/O Works Internally?
Best Way to Learn Java
Java Best Practices Guide
Microservices Tutorial
REST API Tutorial
How to Start New Blog