HowToDoInJava

Guide to Abstraction

Last Updated: January 30, 2022 | By: Lokesh Gupta | Java Object Oriented Programming | Abstraction, Java OOP

In simplest words, you can define abstraction as which captures only those details about a Java object that are relevant to the current perspective.

For example, a `HashMap` stores key-value pairs. It provides you two methods `get()` and `put()` methods to store and retrieve key-value pairs from map. It is, in fact, the only information you will need if you want to use the map in your application. How it works inside, you are not required to know it to use it. This is very much **example of abstraction in Java**.

Take a more **real-life example of abstraction** which can be a TV remote. You know that when you press any button in remote, some function is applied on television e.g. change the channel, change the volume level etc. You are not required to know how internally remote works, to use it properly. It is an abstraction example.

Table of Contents

1. What is abstraction in oops?

In object-oriented programming theory, abstraction involves the facility to **define objects that represent abstract "actors" that can perform work**, report on and change their state, and "communicate" with other objects in the system.

> 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. – WikiPedia

Abstraction in any programming language works in many ways. It can be seen from creating subroutines to defining interfaces for making low-level language calls.

Some abstractions try to limit the breadth of concepts a programmer needs, by completely hiding the abstractions they in turn are built on, e.g. design patterns.

2. Types of abstraction

Typically abstraction can be seen in two ways:

1. **Data abstraction**

   Data abstraction is the way to create complex data types and exposing only meaningful operations to interact with the data type, whereas hiding all the implementation details from outside works.

   The benefit of this approach involves capability of improving the implementation over time e.g. solving performance issues is any. The idea is that such changes are not supposed to have any impact on client code since they involve no difference in the abstract behavior.

2. **Control abstraction**

   A software is essentially a collection of numerous statements written in any programming language. Most of the times, statement are similar and repeated over places multiple times.

   Control abstraction is the process of identifying all such statements and expose them as a unit of work. We normally use this feature when we create a function to perform any work.

3. How to achieve abstraction in java?

As abstraction is one of the core principles of Object-oriented programming practices and Java following all OOPs principles, abstraction is one of the major building blocks of java language.

In java, abstraction is achieved by `interfaces` and `abstract classes`. Interfaces allows you to abstract the implemetation completely while abstract classes allow partial abstraction as well.

Data abstraction spans from creating simple data objects to complex collection implementations such as HashMap or `HashSet`.

Similarly, ***control abstraction*** can be seen from defining simple function calls to complete open source frameworks. control abstraction is main force behind structured programming.

3.1. Java abstraction example

Let's see one more **example of abstraction in Java using interfaces**. In this example, I am creating various reports which can be run on demand at any time during application lifetime. As a consumer of the report, a class needs not to know the internal of report's `run()`, it only should execute this method and report will be executed.

Report.java

```
import java.util.List;

public interface Report
{
    List&amp;lt;Object&amp;gt; run(ReportContext reportContext);
}
```

ReportContext.java

```
public class ReportContext {
  //fields
}
```

EmployeeReport.java

```
import java.util.List;

public class EmployeeReport implements Report
{
    @Override
    public List&amp;lt;Object&amp;gt; run(ReportContext reportContext) {
        //Custom Logic
        System.out.println("Executing employee report");
        return null;
    }
}
```

SalaryReport.java

```
import java.util.List;
```

```
public class SalaryReport implements Report
{
    @Override
    public List&amp;lt;Object&amp;gt; run(ReportContext reportContext) {
        //Custom logic
        System.out.println("Executing salary report");
        return null;
    }
}
```

Now execute the reports with `run()` method.

```
Main.java

package com.howtodoinjava.abstraction;

public class Main {
    public static void main(String[] args) {

        ReportContext reportContext = new ReportContext();
        //Populate context

        Report eReport = new EmployeeReport();
        eReport.run(reportContext);

        Report sReport = new EmployeeReport();
        sReport.run(reportContext);
    }
}
```

Program output.

```
Console

Executing employee report
Executing employee report
```

4. Encapsulation vs Abstraction

> `Encapsulation` is realization of your desired `abstraction`.

Abstraction is more about hiding the implementation details. In Java abstraction is achieved through abstract classes and interfaces.

Encapsulation is about wrapping the implementation (code) and the data it manipulates (variables) within the same class. A Java class, where all instance variables are private and only the methods within the class can manipulate those variables, is an example of an encapsulated class.

If you want to read more about abstract classes and interfaces in Java, follow my next post Exploring interfaces and abstract classes in java.

Happy Learning !!

Recommended Reading:

1. Encapsulation vs Abstraction in Java

2. Guide to Polymorphism
3. Guide to Inheritance
4. Interface vs Abstract Class in Java
5. Overloading vs Overriding in Java
6. Object Oriented Programming
7. Java Access Modifiers
8. Association, Aggregation and Composition
9. Constructors in Java
0. Java Instance Initializer Blocks

** We do not spam !!*

7 thoughts on “Guide to Abstraction”

Lord of Abstraction

November 27, 2016 at 7:49 pm

Abstraction simply means separating responsibilities between different modules.

In software, abstraction reduces dependency among your modules. A specific module will see only the required functionality of the other modules it is dependent on. Hence, implementation changes in the dependent modules will have minimal effect on the other. Less changes because of this “ripple effect” will result in less effort during any modification. This will make the maintenance easier.

Almost always, there will be multiple modules in an application. If changes done in one module does not require any change in its dependent module, we say that the coherence in your application is high. Abstraction helps us achieve this coherence.

Reply

swotsolutions

October 28, 2015 at 7:25 am

JAVA ABSTRACTION:

1.Is one which make a class abstract in object orient programming. It means it provides only essential features for a certain program when needed.
2. Abstract class cannot be instantiated.
3.Abstract methods will be declared in parent class.
4.If a class has abstract method .Its class should also been declared as abstract.

```
public abstract class Employee
{
private String name;
private String address;
private int number;
public Employee(String name, String address, int number)
{
System.out.println(&amp;quot;Constructing an Employee&amp;quot;);
this.name = name;
this.address = address;
this.number = number;
}
public double computePay()
{
System.out.println(&amp;quot;Inside Employee computePay&amp;quot;);
return 0.0;
}
public void mailCheck()
{
System.out.println(&amp;quot;Mailing a check to &amp;quot; + this.name
+ &amp;quot; &amp;quot; + this.address);
}
public String toString()
{
return name + &amp;quot; &amp;quot; + address + &amp;quot; &amp;quot; + number;
}
public String getName()
{
return name;
}
public String getAddress()
```

```
{
return address;
}
public void setAddress(String newAddress)
{
address = newAddress;
}
public int getNumber()
{
return number;
}
}
```

This above code is an example to abstract class,where abstract keyword appears before class keyword in class declaration.

5. Abstract Class can be inherited.
http://8subjects.com/abstraction-java/

Reply

Irshad Kashmiri

April 17, 2015 at 4:38 pm

i got lot of inspiration from this blog
thanks mr lokesh

Reply

hasini

June 6, 2014 at 7:21 am

good explanation.it is better if there was explanation with examples.

Reply

Saikumar

January 4, 2014 at 7:46 pm

Excellent ..!!

Reply

Bob

July 15, 2013 at 11:39 am

Simple and best explanation. Many thanks

Reply

Brijesh

July 7, 2013 at 12:47 am

Brilliantly explained. Loved it man. Abstraction is now clear. Next step encapsulation.

Reply

Leave a Comment

Name *

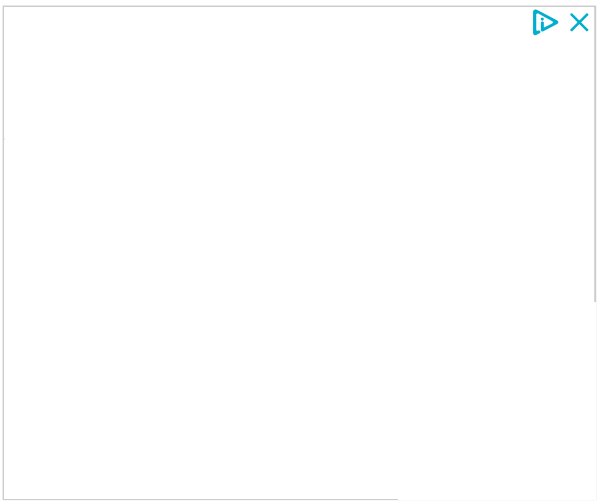
Email *

Website

Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search ...



HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

Meta Links

- About Me
- Contact Us
- Privacy policy
- Advertise
- Guest Posts

Blogs

REST API Tutorial