

Inheritance in Java OOPs with Example

<https://www.guru99.com/java-class-inheritance.html>

What is Inheritance?

Inheritance is a mechanism in which one class acquires the property of another class. For example, a child inherits the traits of his/her parents. With inheritance, we can reuse the fields and methods of the existing class. Hence, inheritance facilitates Reusability and is an important concept of OOPs.

In this tutorial, you will learn-

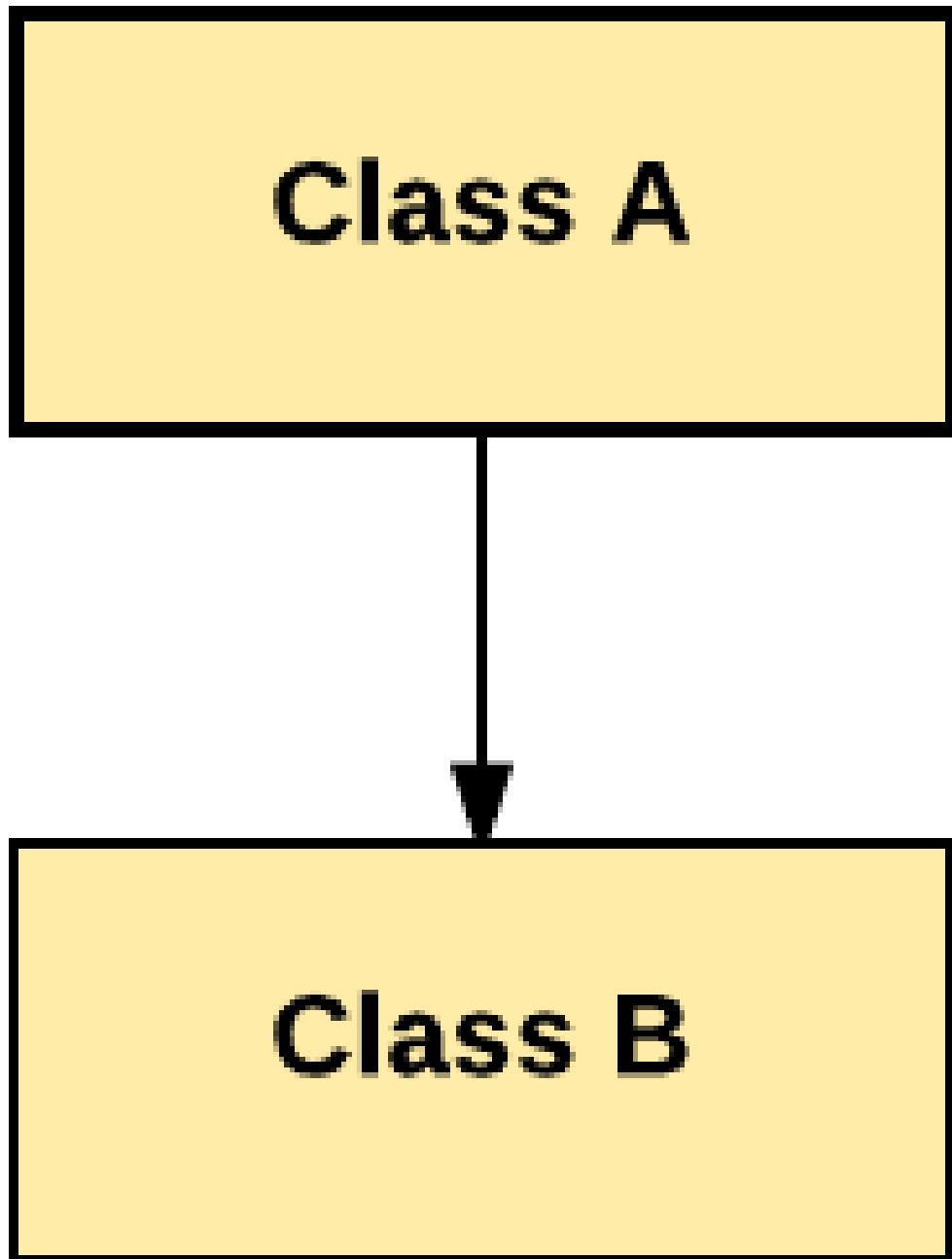
- [Types of Inheritance](#)
- [Inheritance in Java](#)
- [Java Inheritance Example](#)
- [Super Keyword](#)
- [Learn Inheritance in OOP's with Example](#)

Types of Inheritance

There are Various types of inheritance in Java:

Single Inheritance:

In Single Inheritance one class extends another class (one class only).



Single Inheritance

In above diagram, Class B extends only Class A. Class A is a super class and Class B is a Sub-class.

Multiple Inheritance:

In Multiple Inheritance, one class extending more than one class. Java does not support multiple inheritance.

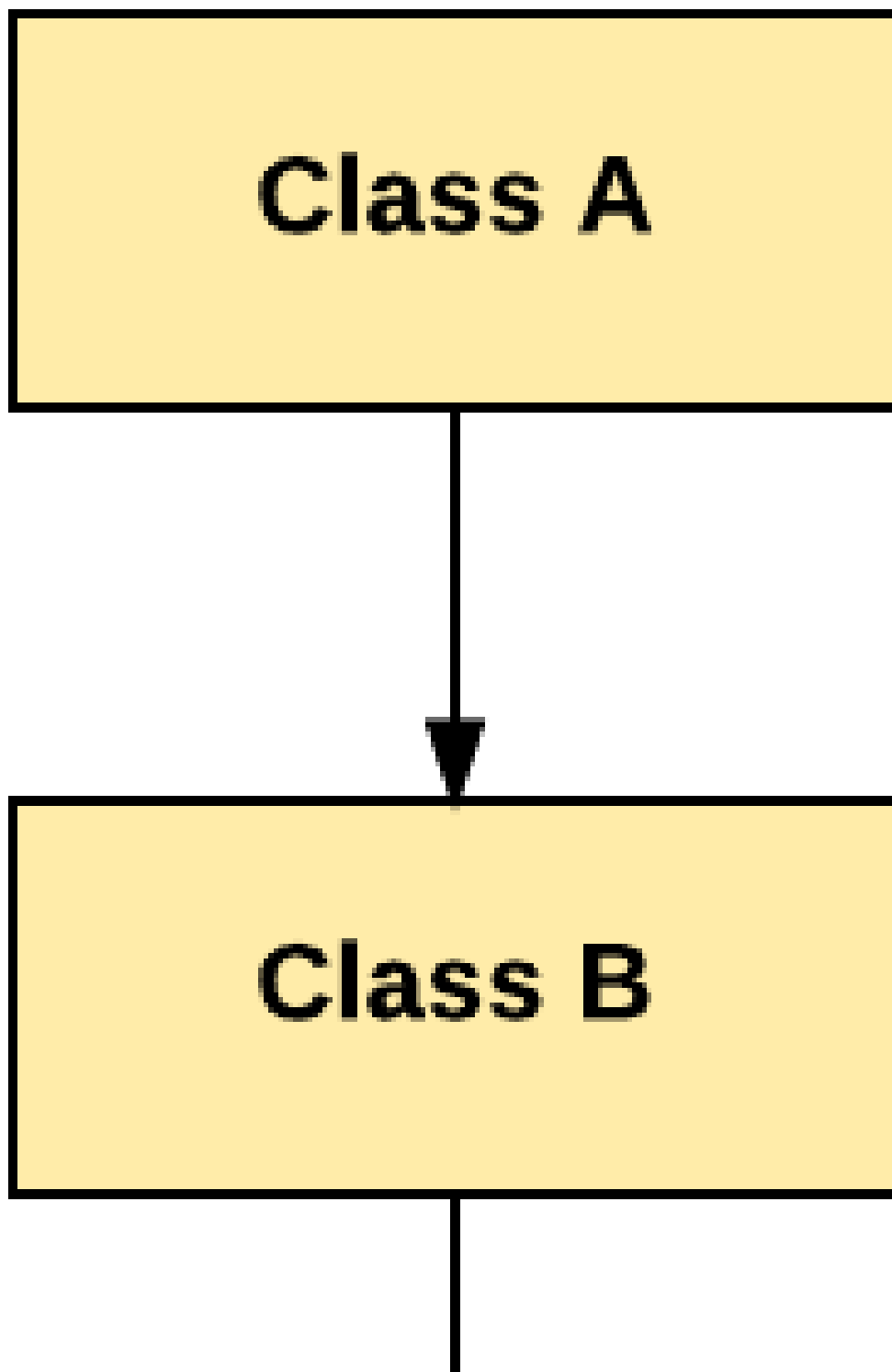
Types of Inheritance

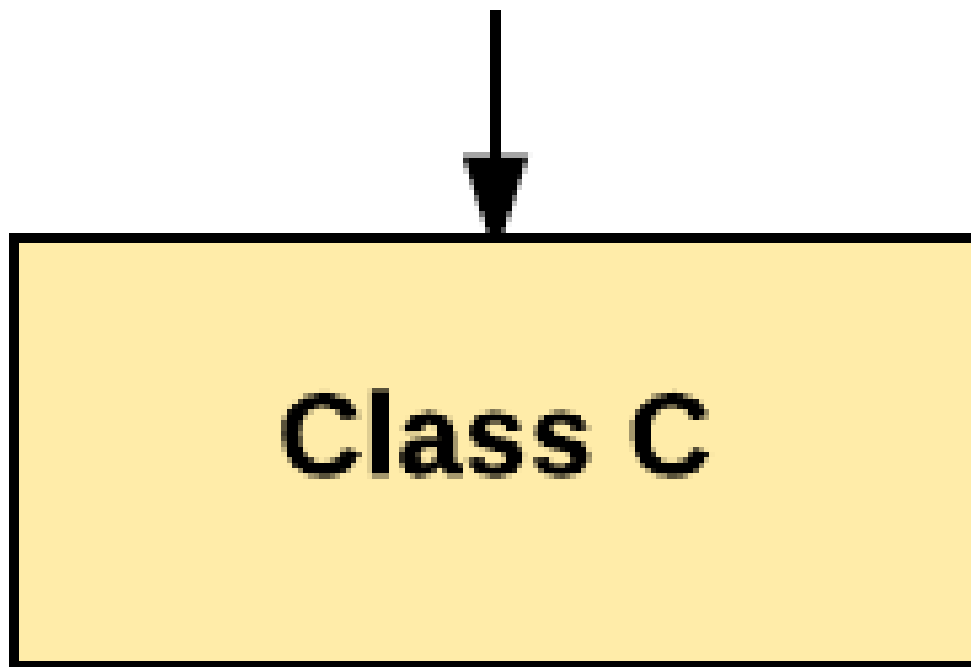
Multiple Inheritance

As per above diagram, Class C extends Class A and Class B both.

Multilevel Inheritance:

In Multilevel Inheritance, one class can inherit from a derived class. Hence, the derived class becomes the base class for the new class.



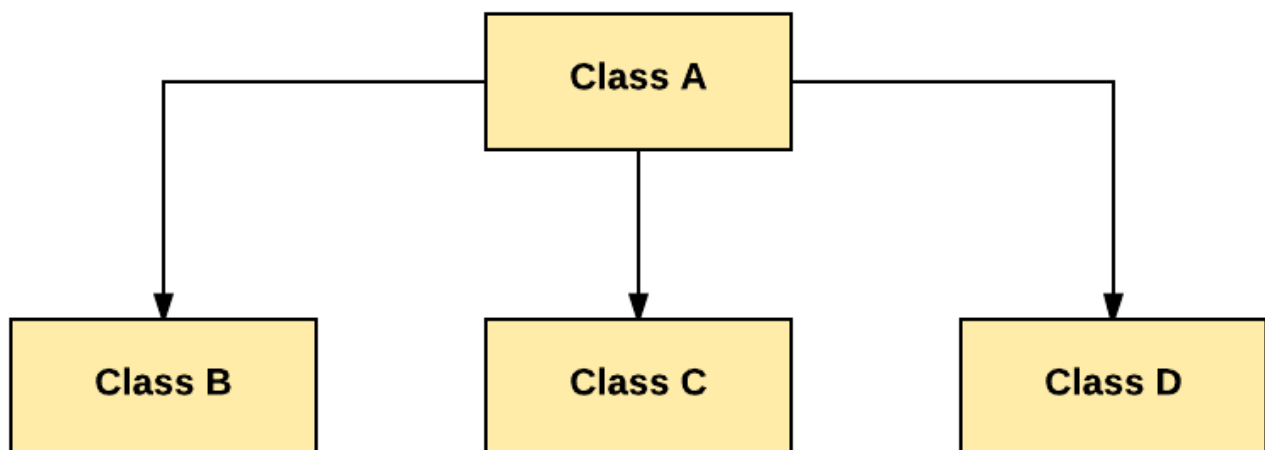


Multilevel Inheritance

As per shown in diagram Class C is subclass of B and B is a of subclass Class A.

Hierarchical Inheritance:

In Hierarchical Inheritance, one class is inherited by many sub classes.

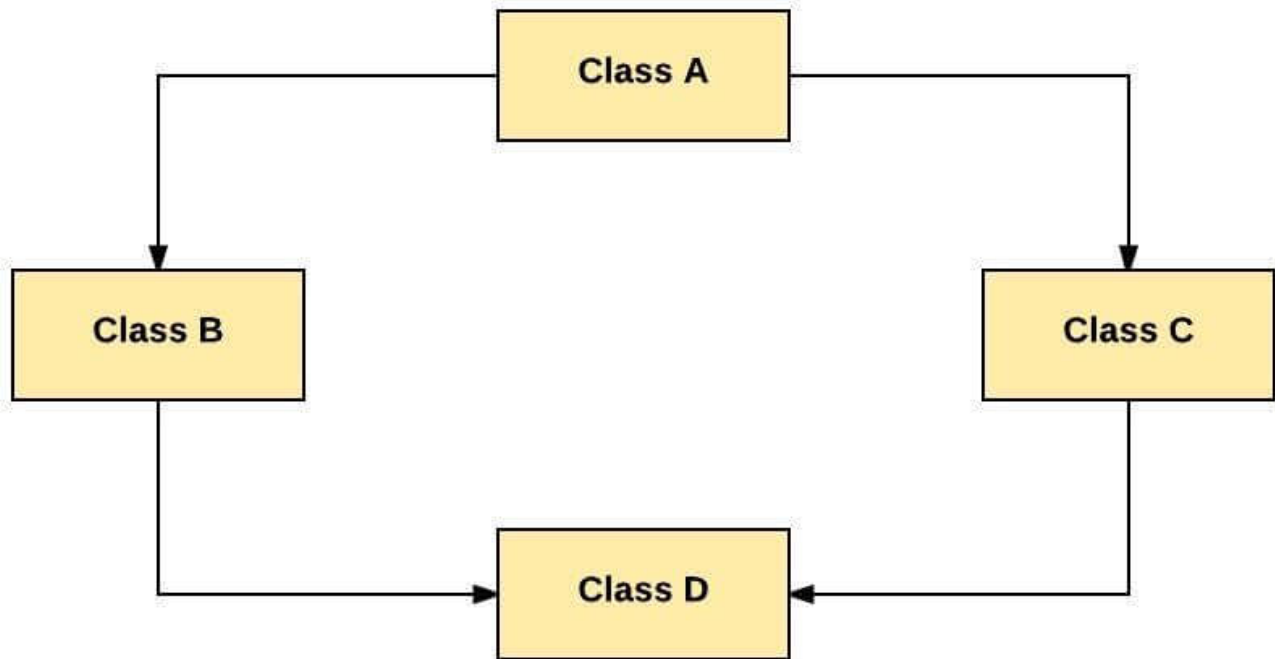


Hierarchical Inheritance

As per above example, Class B, C, and D inherit the same class A.

Hybrid Inheritance:

Hybrid inheritance is a combination of Single and Multiple inheritance.



Hybrid Inheritance

As per above example, all the public and protected members of Class A are inherited into Class D, first via Class B and secondly via Class C.

Note: Java doesn't support hybrid/Multiple inheritance

Inheritance In Java

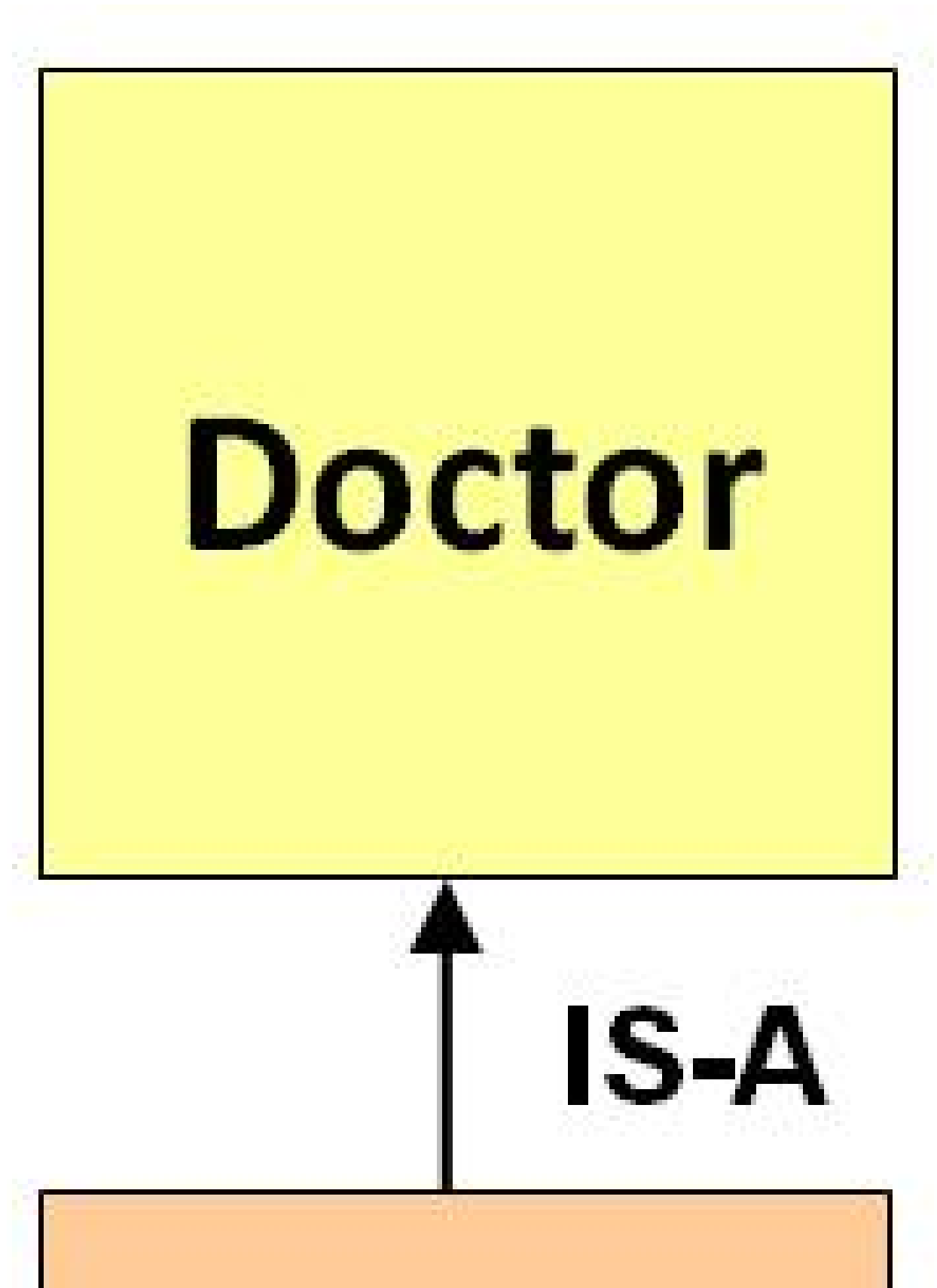
JAVA INHERITANCE is a mechanism in which one class acquires the property of another class. In Java, when an "Is-A" relationship exists between two classes, we use Inheritance. The parent class is called a super class and the inherited class is called a subclass. The keyword `extends` is used by the sub class to inherit the features of super class.

Inheritance is important since it leads to the reusability of code.

Java Inheritance Syntax:

```
class subClass extends superClass
{
    //methods and fields
}
```

Java Inheritance Example



Surgeon

```
class Doctor {  
    void Doctor_Details() {  
        System.out.println("Doctor Details...");  
    }  
}  
  
class Surgeon extends Doctor {  
    void Surgeon_Details() {  
        System.out.println("Surgen Detail...");  
    }  
}  
  
public class Hospital {  
    public static void main(String args[]) {  
        Surgeon s = new Surgeon();  
        s.Doctor_Details();  
        s.Surgeon_Details();  
    }  
}
```

Super Keyword

The super keyword is similar to "this" keyword.

The keyword super can be used to access any data member or methods of the parent class.

Super keyword can be used at variable, method and constructor level.

Syntax:

```
super.<method-name>();
```

Learn Inheritance in OOP's with Example

Consider the same banking application from the [previous example](#).

We are supposed to open two different account types, one for saving and another for checking (also known as current).

- 
- A rectangular box with a light blue gradient background and a subtle drop shadow. Inside the box, there is a handwritten list in red ink. The first item is '1) saving' and the second item is '2) Current / Checking'.
- 1) saving
 - 2) Current / Checking

Let's compare and study how we can approach coding from a **structured and object-oriented programming perspective**. **Structural approach:** In structured programming, we will create two functions –

1. One to withdraw
2. And the other for deposit action.

Since the working of these functions remains same across the accounts.

Structural Approach

```
public withdraw(){  
  //code to withdraw  
}
```

1

```
public deposit(){  
  //code to deposit  
}
```

2

OOP's approach: While using the OOPs programming approach. We would create two classes.

- Each having implementation of the deposit and withdraw functions.
- This will redundant extra work.

oop's Approach



Change Request in Software

Now there is a change in the requirement specification for something that is so common in the software industry. You are supposed to add functionality privileged Banking Account with Overdraft Facility. For a background, overdraft is a facility where you can withdraw an amount more than available the balance in your account.

- 1) saving
- 2) Current / Checking
- 3) Privileged

including a privileged
function (overdraft
facility)

Structural approach: Using functional approach, I have to modify my withdraw function, which is already tested and baselined. And add a method like below will take care of new requirements.

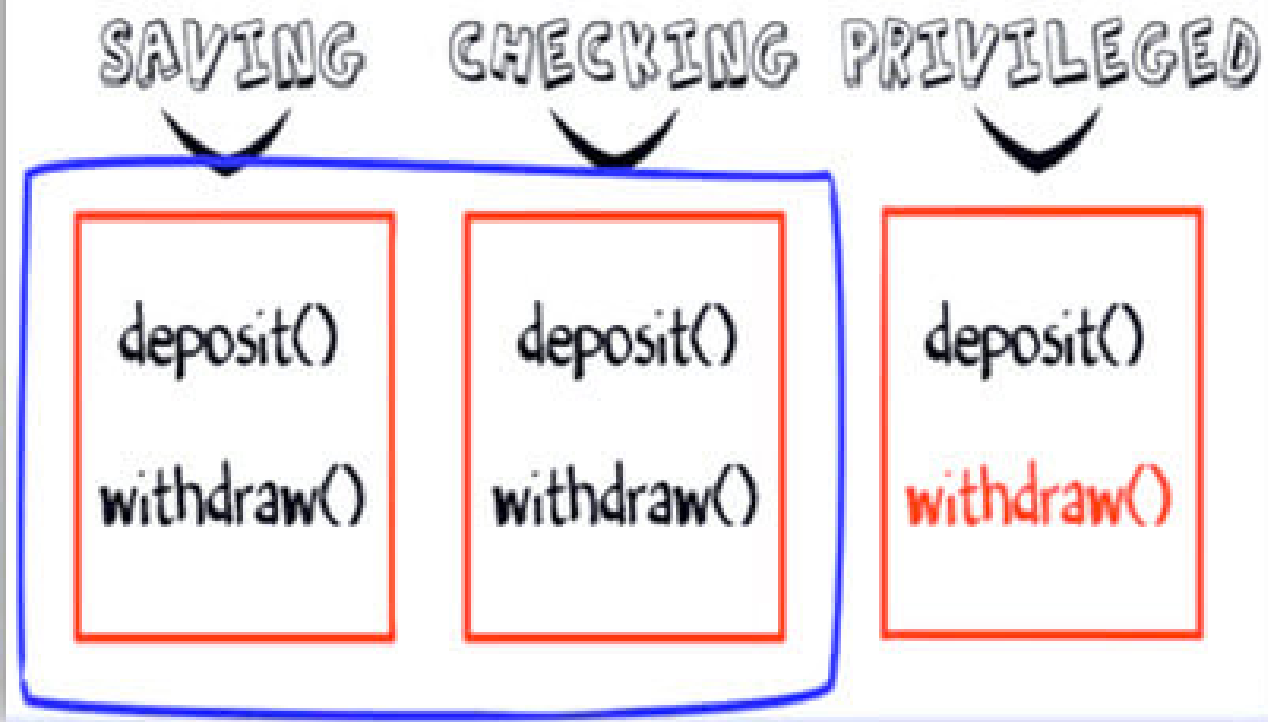
Structural Approach

```
public withdraw(int Account Type){  
    if Account Type = Privileged  
        //code to withdraw  
    }  
    else {  
        //code to withdraw for other accounts  
    }  
}
```

modifying the
code for
"withdraw"
function through
structural
programming
approach

OOP's approach: Using OOP's approach, you just need to write a new class with unique implementation of withdraw function. We never touched the tested piece of code.

oop's Approach

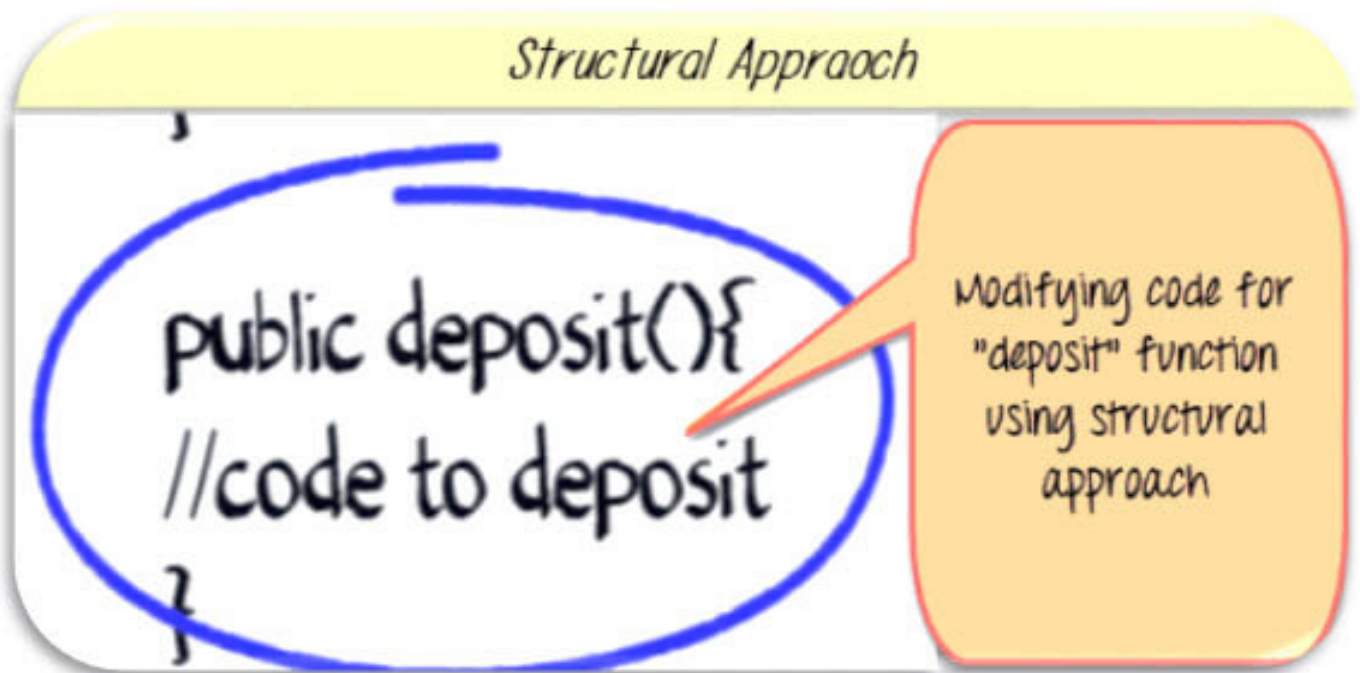


Another Change Request

What if the requirement changes further? Like to add credit card account with its own unique requirement of deposits.

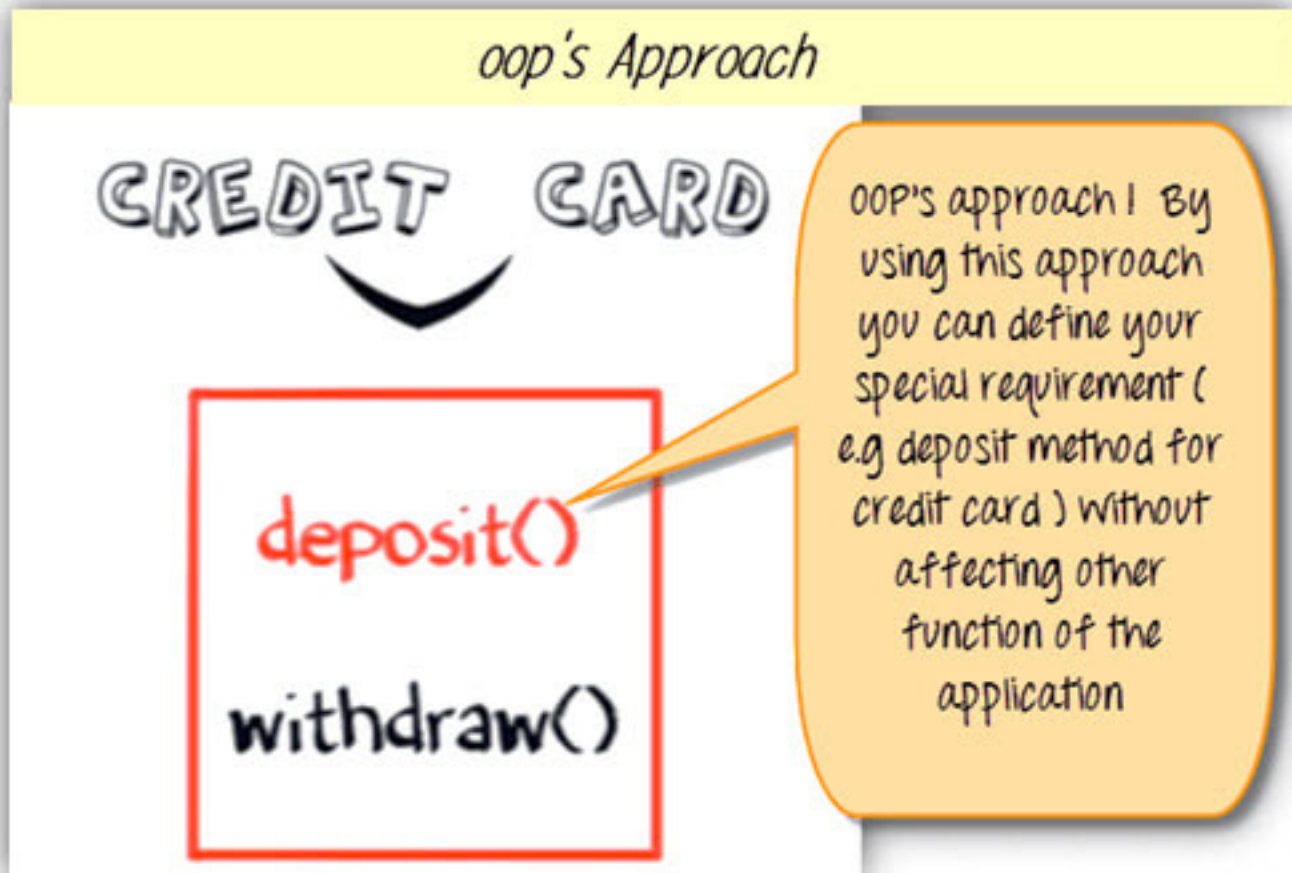
- 1) saving
- 2) Current / Checking
- 3) Privileged
- 4) Credit Card

Structural approach: Using structural approach you have to change tested piece of deposit code again.



OOP's approach: But using object-oriented approach, you will just create a new class with its unique implementation of deposit method (highlighted red in the image below).

So even though the structural programming seems like an easy approach initially, OOP's wins in a long term.



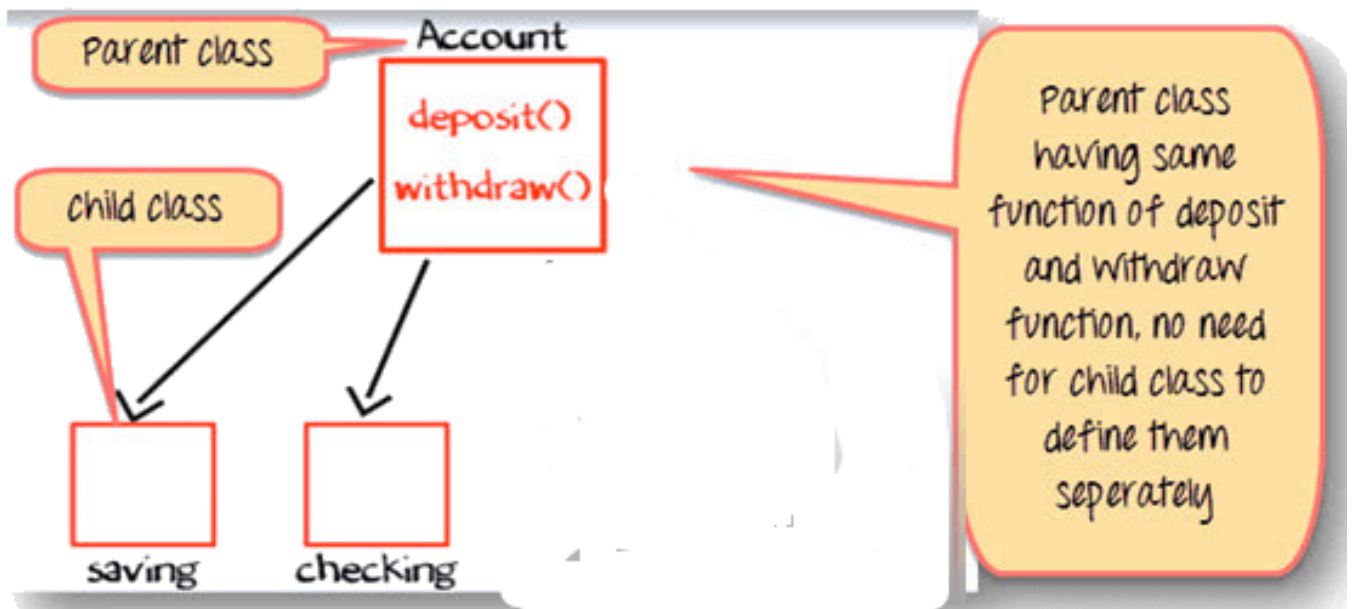
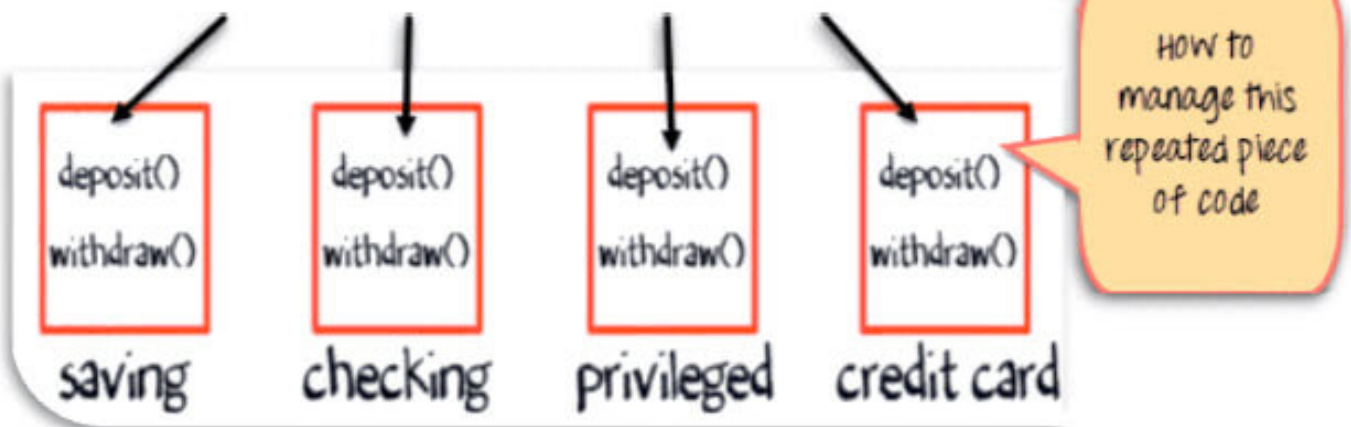
Advantage of Inheritance in OOPs

But one may argue that across all classes, you have a repeated pieces of code.

To overcome this, you create a parent class, say "account" and implement the same function of deposit and withdraw. And make child classes inherited "account" class. So that they will have access to withdraw and deposit functions in account class.

The functions are not required to be implemented individually. This is **Inheritance in java. .**

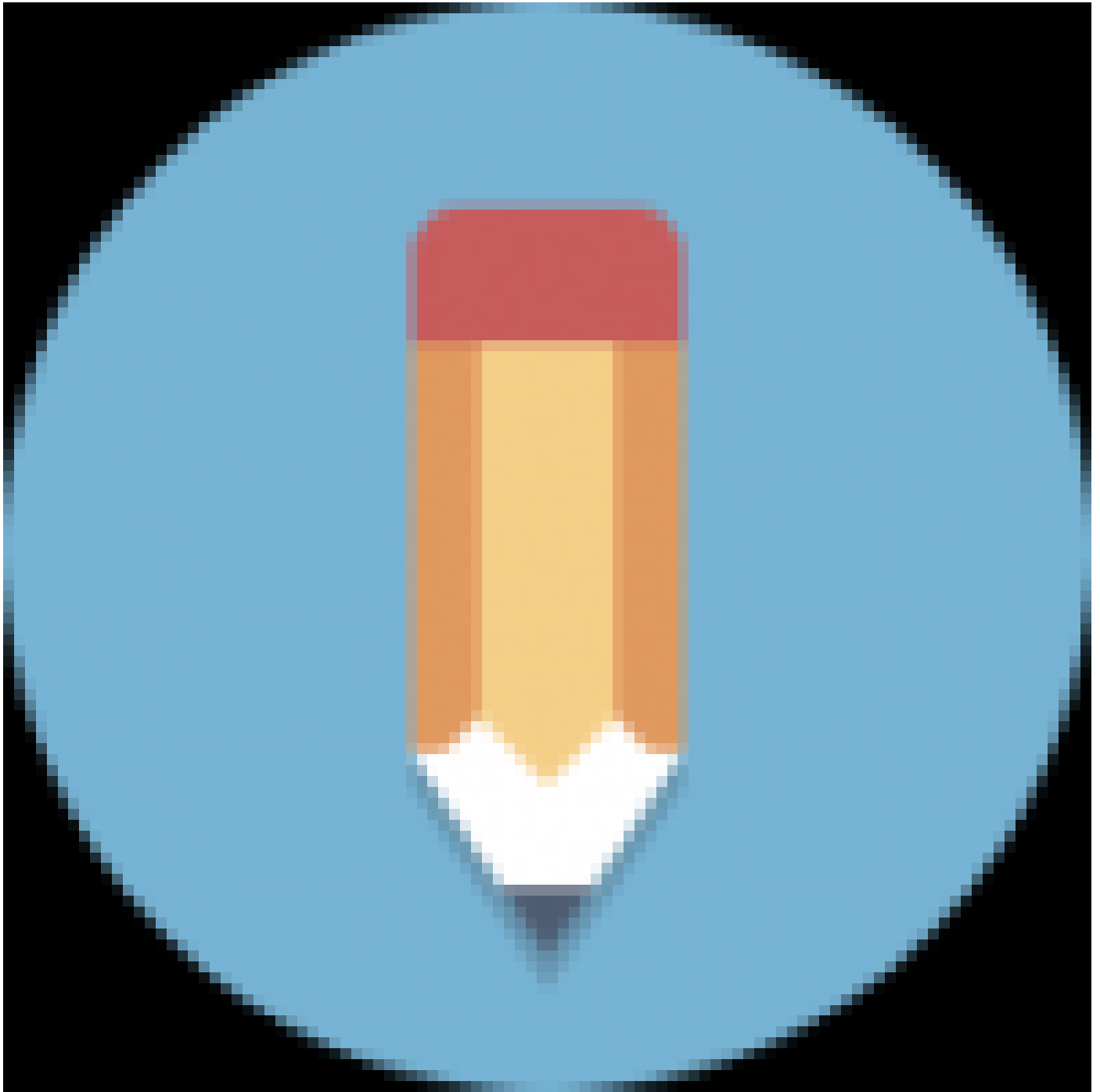
Repeated piece of code



- [Prev](#)
- [Report a Bug](#)
- [Next](#)

YOU MIGHT LIKE:

JavaScript



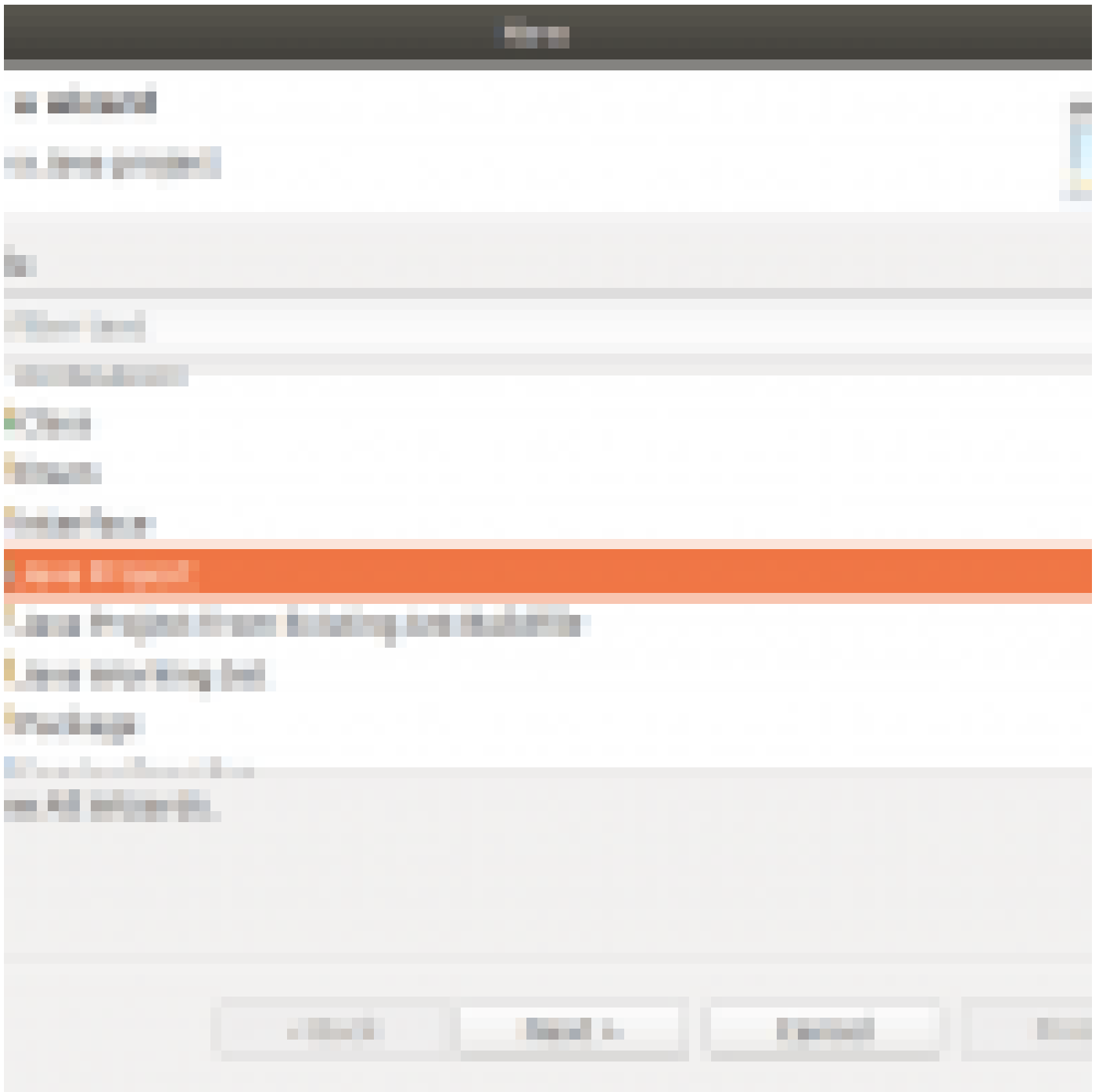
You can use JavaScript code in two ways. You can either include the JavaScript code internally within...

Java Tutorials



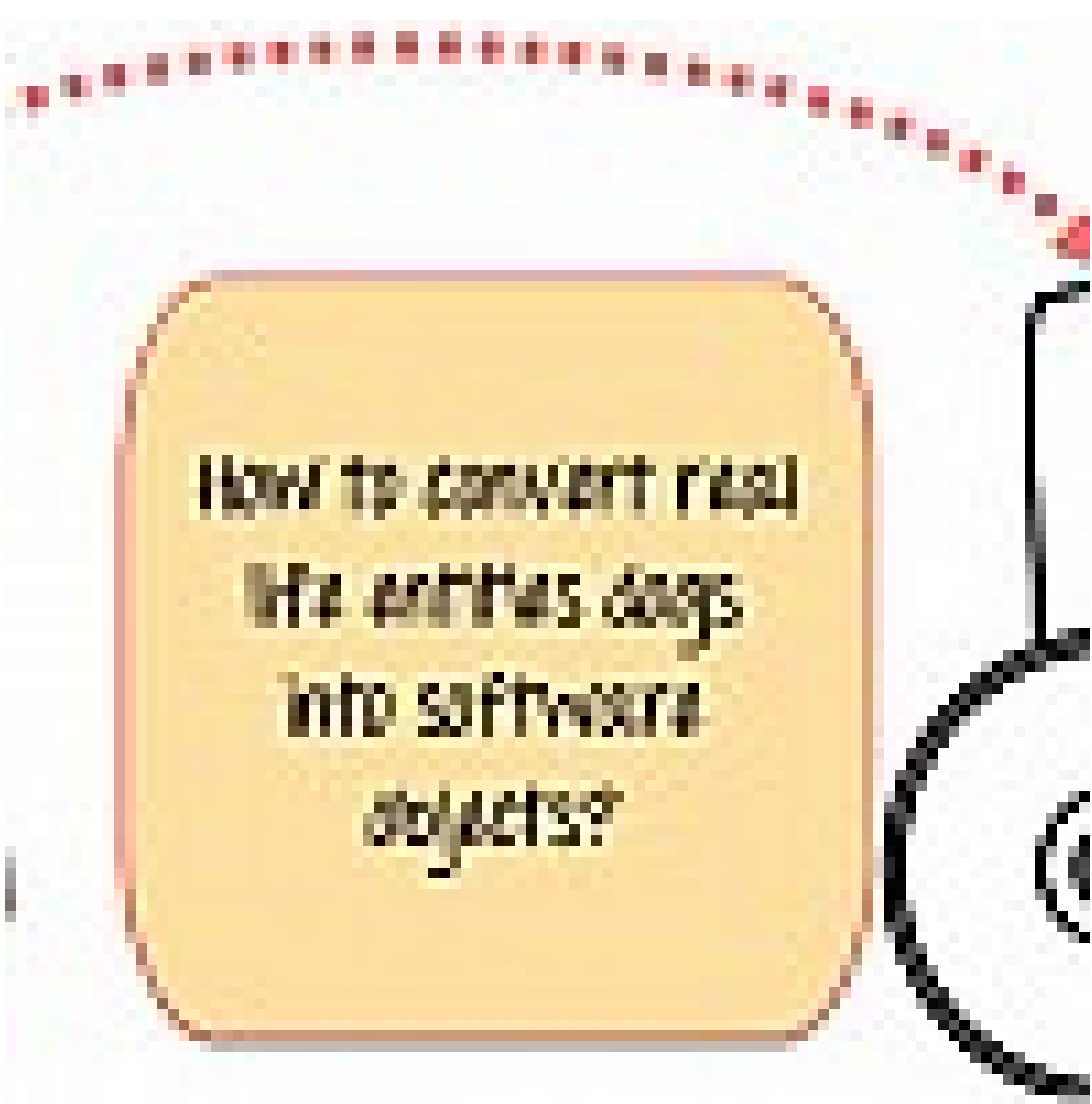
What is Command Line Argument in Java? Command Line Argument in Java is the information that is...

Java Tutorials



What is JSON? JSON is an abbreviation for Javascript Object Notation, which is a form of data that...

Java Tutorials



How to convert real
life entities into
software
objects?

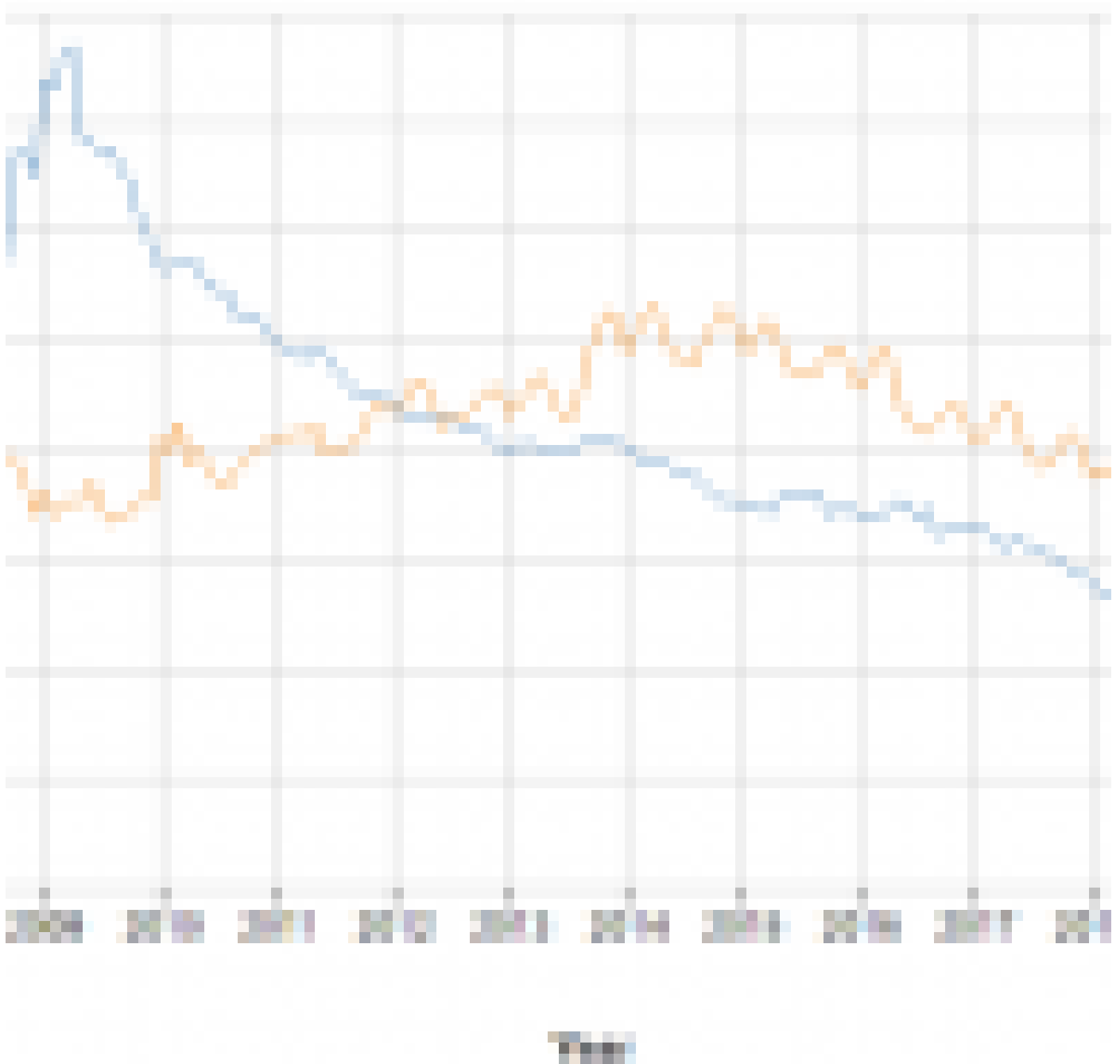
Classes and Objects in Java are the fundamental components of OOP's.
Often there is a confusion...

Java Tutorials

Java Collections Interview Questions

Here are Java Collections Interview Questions for fresher as well as experienced candidates to get...

Java Tutorials



What is Java? Java was released by Sun Microsystem in 1995. It was developed by James Gosling. It is a...