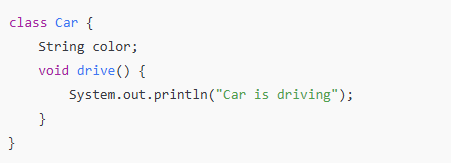
**Java OOPs – Object oriented programming system:**

Java oops are the pillar or the foundation of the language. Following are the main object-oriented concepts in JAVA.

They help in designing programs using real world entities like objects and class.

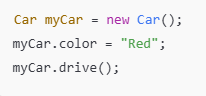
1. **Class**

Class is blueprint for creating an object, it defines the properties and behaviours.



1. **Object**

Object is an instance of a class, represent the real-world entity. In below examples **myCar** is an object of class car. It is accessing the properties of a class i.e. **variable** colour and behaviour i.e. drive method.

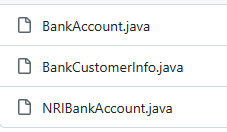


1. **Inheritance**

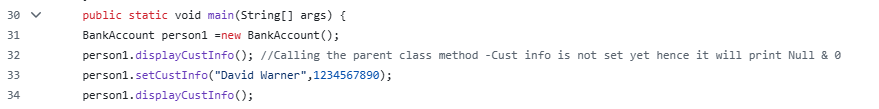
Inheritance allows us to inherit the properties and method from one class to another class, It’s support code reusability.

How? Using extends keyword one class can extends a propertied of another class

[Examples:](https://github.com/TechGuyy1/basicjava/tree/9d31273ba76a016e8ac2237ea2be8f290f69b109/abcBank) Here in an example BanAccount.java extends the parent class BankCustomerInfo,



Hence using the object of BankAccount class we are able to access the BankCustomerInfo class method, displayCustInfo(). As shown in bellow example.

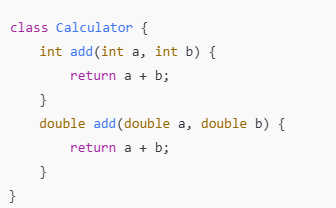


1. **Polymorphism**

In java polymorphism is core concept which allows us to create methods with the same name but different implementation.

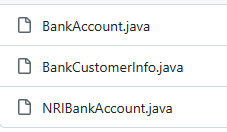
**Types of polymorphism:**

1. **Compile time polymorphism/**Overloading [Method name same & signature different]
2. All the method name should be same and implemented in same class, it should differ with its return type and input parameter(signature).
3. Only the method name is same, as show in below **example**.
4. During the compile time only we will get to know which method is being called.
5. If we write everything same i.e. method name, return type and argument in same class then the compiler will give us an error.

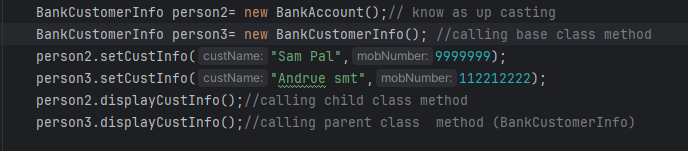


1. **Run time polymorphism** /Overriding / Dynamic Binding
2. Overriding is done when child or subclass has **same method** name, **return type** and **arguments** as the parent class method.
3. The method to be called is determined based on the object which is being referred to by the reference variable(object).

[Examples:](https://github.com/TechGuyy1/basicjava/tree/9d31273ba76a016e8ac2237ea2be8f290f69b109/abcBank) Here in an example BanAccount.java extends the parent class BankCustomerInfo,



displayCustInfo() method is overridden in BankAccount.java.



**Output**

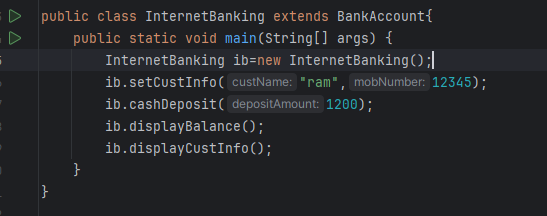


1. **Encapsulation**
2. Encapsulation is wrapping or bundling a data and methods withing single unit, example class.
3. Access modifiers help us to encapsulate the data.
4. Here in bank class example, we have declared a variable bankBalance as private, hence it is not accessible outside the class, but we can access it in extended class using public getter or setter method. (We need to create a public method to set the bankBalance and similarly public get method).

[Examples:](https://github.com/TechGuyy1/basicjava/tree/9d31273ba76a016e8ac2237ea2be8f290f69b109/abcBank)

bankBalance is private in BankAccount directly we cannot access in InternetBanking Class.

With the help of public method setCustInfo and cashDeposit we are able to retrieve it in InternetBanking class.



1. **Abstraction**

**Abstraction is a process of hiding implements details from the user, only the functionality is provided to the user. In other words, the user will have the information on what object dose instead of how its dose.**

Abstraction in Java is performed in two ways

1. **Abstract Class** 2. **Interface**

**Abstract Class**

An abstract class is a class that has been declared as abstract. It might include both abstract and non-abstract methods.

It needs to be expanded and its method implemented.

Final methods can prevent subclasses from changing their methods' contents.

**Key Points:**

An abstract class must be declared with the **abstract** keyword.

It can have **abstract** and **non-abstract** methods.

It cannot be instantiated.

It can have final methods.

It can also have constructors and static methods.

**Abstract Method**

* In Java, an abstract method is declared but not implemented. It is specified within an abstract class and must be overridden by its subclasses.
* An abstract method is intended to be a placeholder that enforces specific behaviour in subclasses.
* It serves a useful purpose when the desired behaviour of the method should remain consistent while allowing different implementations within different classes.

**Interface**

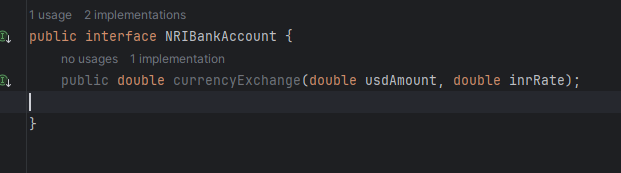
**By using interface, we can achieve 100% abstraction.**

**Java dose not permit multiple inheritance for classes, utilizing interface allows for multiple inheritance.**

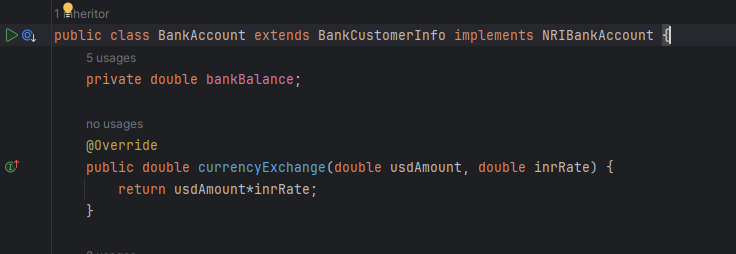
* Using **interface** keyword, we can create an **interface.** All method should be abstract method that means we just need to declare method.
* In **interface** it mandatory to use the public access specifier, as we need to implement those abstract methods in child class.

**Example:** In NRIBankingAccount interface currencyExchange method is declared. Its implementation is done in Bankaccount class.

<https://github.com/TechGuyy1/basicjava/tree/9d31273ba76a016e8ac2237ea2be8f290f69b109/abcBank>



**Implementation**



|  |  |
| --- | --- |
| **Interfaces** | **Abstraction** |
| * All Methods are abstracts | * If class have implemented method and non-implemented method that class called as abstract class. |
| * Access specifier must be a **public**, variable defined must be **public**, **static** and **final** | * **Private** access is not allowed in the abstraction. * Except private variable we can have any access specifier. |
| * To implement an interface, we use **implements** Keywords. | * **extends** keyword is used to implement |