Abstraction

Table Of Content

Table Of Content

- Abstraction
- Abstraction & Polymorphism (it is 100% abstraction)
- Rules In Abstraction
- Abstraction Example
- Note

Abstraction

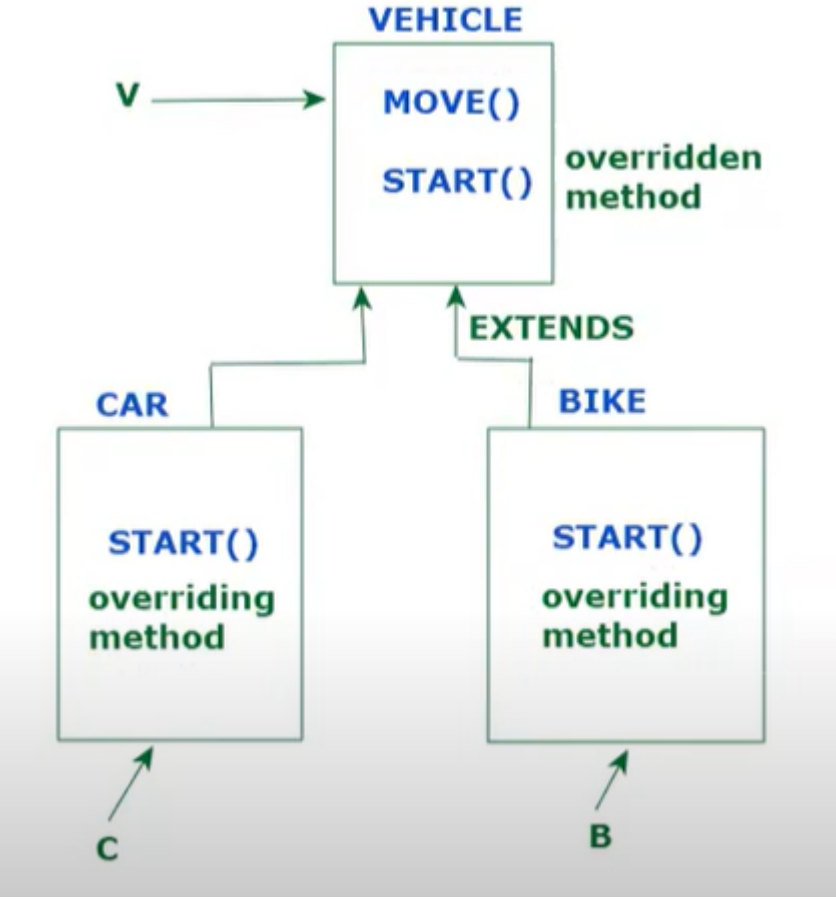
- Abstraction is one of the important object oriented features in java. **It is process of hiding the internal implementation and sharing only the related functionality to the user.**
  ABSTRACTION : It is all about hiding the implementation(body) of a overridden method.
  ENCAPSULATION : It is all about hiding the data members(variable and methods).

▼ Simple Example For **Abstraction**

```
package _1Java_Codes_From_Basics._22Abstraction;

import java.util.Scanner;

abstract class remoteFunction
{
    abstract void buttonON();
    abstract void buttonOFF();
}

class TV_Remote extends remoteFunction
{
    void operateRemote()
    {
        Scanner scanner= new Scanner(System.in);
        System.out.print("Do you want to watch 'TV', if yes press remote(on or of
f): ");
        String input = scanner.next();
        //if(input == "On") or like below
        if(input.equalsIgnoreCase("on") )
        {
```

```
            buttonON();
        }
        else
        {
            buttonOFF();
        }
    }
    void buttonON()
    {
        System.out.println("TV is turned ON.....!");
    }
    void buttonOFF()
    {
        System.out.println("TV is turned OFF.....!");
    }
}



public class _5abstractionSimpleApplication
{
    public static void main(String[] args)
    {
        TV_Remote tvr = new TV_Remote();
        tvr.operateRemote();
    }
}
//output
Do you want to watch 'TV', if yes press remote(on or off): on
TV is turned ON.....!

Process finished with exit code 0

Do you want to watch 'TV', if yes press remote(on or off): off
TV is turned OFF.....!

Process finished with exit code 0
```

- In java abstraction can be achieved in two ways:
  1. Abstract class [0-100%]
  2. Interface



- Here **vehicle is a parent class** where as **car and bike are child class**
- As we see in a pitcher **start()** is a method which is **overridden for both child classes**.
- As we know that **car's & bike's start()** has different properties
- So in order to extend/derive **start()** method for different child classes we use abstraction

▼ **Program Vehicle extends car & bike without Abstraction**

```
package _1Java_Codes_From_Basics._22Abstraction;

class vehicle
{
    void move()
    {
        System.out.println("Each vehicle has different power so it moves in differ
ent speed");
    }
    void start()//overridden method
    {
        System.out.println("Each vehicle has different engine so starting the vehi
cle varies from vehicle to vehicle");
    }
```

```
}
class bike extends vehicle
{
    void start()//overriding method
    {
        System.out.println("Bike can be start using self-start or kick-start");
    }
}
class car extends vehicle
{
    void start()//overriding method
    {
        System.out.println("Car can be start using only key");
    }
}

public class _1withoutAbstractionEx1
{
    public static void main(String[] args)
    {
        vehicle v = new vehicle();
        bike b = new bike();
        car c = new car();

        v.move();
        v.start();
        System.out.println();
        b.move();
        b.start();
        System.out.println();
        c.move();
        c.start();
    }
}

//output
Each vehicle has different power so it moves in different speed
Each vehicle has different engine so starting the vehicle varies from vehicle to v
ehicle

Each vehicle has different power so it moves in different speed
Bike can be start using self-start or kick-start

Each vehicle has different power so it moves in different speed
Car can be start using only key

Process finished with exit code 0
```

▼ **Program Vehicle extends car & bike with Abstraction**

- Here void start() method is overriding method
- where its child classes is altering its property according to their requirement
- anyway if we wrote ant thing in overridden method it won't get executed

- so inorder to hide the implementation we use Abstraction

```
package _1Java_Codes_From_Basics._22Abstraction;

abstract class vehicleClass
{
    void move()
    {
        System.out.println("Each vehicle has different power so it moves in differ
ent speed"); //concrete method
    }
//    void start()//overridden method
//    {
//        System.out.println("Each vehicle has different engine so starting the ve
hicle varies from vehicle to vehicle");
//    }

    /**
     * @Abstraction-Explanation
     * here void start() method is overriding method
     * where its child classes is altering its property according to their require
ment
     * anyway if we wrote ant thing in overridden method it won't get executed
     * so inorder to hide the implementation we use Abstraction
     */

    abstract void start(); //abstract method

}
class bikeClass extends vehicleClass
{
    void start()//overriding method
    {
        System.out.println("Bike can be start using self-start or kick-start");
    }
}
class carClass extends vehicleClass
{
    void start()//overriding method
    {
        System.out.println("Car can be start using only key");
    }
}

public class _2withAbstractionEx2
{
    public static void main(String[] args)
    {
        //vehicleClass vehicleClass =  new vehicleClass()
        bikeClass bikeClass = new bikeClass();
        bikeClass.move();
        bikeClass.start();

        carClass carClass = new carClass();
        carClass.move();
        carClass.start();
```

```
    }
}

//output
Each vehicle has different power so it moves in different speed
Bike can be start using self-start or kick-start

Each vehicle has different power so it moves in different speed
Car can be start using only key

Process finished with exit code 0
```

NOTE:

- For normal method we won't give semicolon at the end of method declaration but **for abstraction method we should give semicolon at the end.**
- Abstract methods should be present only in **abstract class.**

> **Abstract methods** are the methods which doesn't having implementation part

> **Abstract class** are the class which is having at-least one abstract method

> For **Abstract class** we should not create a object

> All the abstract methods **should end with the semicolon**

> The inherited method from the parent class is also called as **'concrete method'.**

▼ Simple Example For **Abstraction**

```
package _1Java_Codes_From_Basics._22Abstraction;

abstract class bank
{
    public void data()//concrete class
    {
        System.out.println("Each bank has different rate of interest");
        System.out.println("Rate Of Interest Follows Like This");
    }

    abstract void rateOfInterest(); //abstract method
```

```
}

class hdfc extends bank
{
    void rateOfInterest()
    {
        System.out.println("HDFC Bank's Rate Of Interest Is 2.50% to 5.50%");
    }
}

class sbi extends bank
{
    void rateOfInterest()
    {
        System.out.println("SBI Bank's Rate Of Interest Is 2.90% to 5.40%");
    }
}

class icici extends bank
{
    void rateOfInterest()
    {
        System.out.println("ICICI Bank's Rate Of Interest Is 2.50% to 5.50%");
    }
}


public class _3withAbstractionEx3
{
    public static void main(String[] args)
    {

        hdfc b1 = new hdfc();
        b1.data();
        System.out.println();
        b1.rateOfInterest();
        System.out.println();

        sbi b2 = new sbi();
        b2.rateOfInterest();
        System.out.println();

        icici b3 = new icici();
        b3.rateOfInterest();
    }
}

//output
Each bank has different rate of interest
Rate Of Interest Follows Like This

HDFC Bank's Rate Of Interest Is 2.50% to 5.50%

SBI Bank's Rate Of Interest Is 2.90% to 5.40%

ICICI Bank's Rate Of Interest Is 2.50% to 5.50%

Process finished with exit code 0
```

Abstraction & Polymorphism (it is 100% abstraction)

> 1. For abstract class we **can not create object but we can have refence.**

> 2. by parent reference we can perform **upcasting and polymorphism** can be achieved.

> 3. abstract class which consist of **only abstract methods is considered as 100% abstraction.**

▼ Simple Example For **Abstraction & Polymorphism**

```
package _1Java_Codes_From_Basics._22Abstraction;

abstract class Bank
{
    abstract void rateOfInterest(); //abstract method
}

class HDFC extends Bank
{
    void rateOfInterest()
    {
        System.out.println("HDFC Bank's Rate Of Interest Is 2.50% to 5.50%");
    }
}

class SBI extends Bank
{
    void rateOfInterest()
    {
        System.out.println("SBI Bank's Rate Of Interest Is 2.90% to 5.40%");
    }
}

class ICICI extends Bank
{
    void rateOfInterest()
    {
        System.out.println("ICICI Bank's Rate Of Interest Is 2.50% to 5.50%");
    }
}


public class _4abstractionAndPolymorphismEx4
```

```
{
    public static void main(String[] args)
    {
        System.out.println("Parent Reference Child Object(Polymorphism) & Abstract
ion");
        System.out.println();

        //parent class reference
        Bank banks;

        //child object with parent reference - 1
        banks = new HDFC();
        banks.rateOfInterest();
        System.out.println();

        //child object with parent reference - 2
        banks = new SBI();
        banks.rateOfInterest();
        System.out.println();

        //child object with parent reference - 3
        banks = new ICICI();
        banks.rateOfInterest();
    }
}

//output
Parent Reference Child Object(Polymorphism) & Abstraction

HDFC Bank's Rate Of Interest Is 2.50% to 5.50%

SBI Bank's Rate Of Interest Is 2.90% to 5.40%

ICICI Bank's Rate Of Interest Is 2.50% to 5.50%

Process finished with exit code 0
```

Rules In Abstraction

1. If a method is made as abstract then compulsory the child class should override the method.
2. If the child class doesn't override the abstract method then error is displaced during compilation.
3. If a **child class is not in a position to override the abstract method** then in the **child class the method can be re -declared as abstract** and corresponding child class should also be **made as abstract**.

▼ **Abstraction In Child/Sub class**

```
package _1Java_Codes_From_Basics._22Abstraction;

abstract class Bird
{
    abstract void fly();
    abstract void eat();
}
class Pigeon extends Bird
{
    void fly()
    {
        System.out.println("pigeon fly very low");
    }
    void eat()
    {
        System.out.println("pigeon eats grains");
    }
}
abstract class Eagle extends Bird
{
    void fly()
    {
        System.out.println("eagle fly very high");
    }
    abstract void eat();
}
class MountainEagle extends Eagle
{
    void eat()
    {
        System.out.println("Mountain eagles hunt and eat over mountains");
    }
}
class GoldenEagle extends Eagle
{
    void eat()
    {
        System.out.println("Golden eagles hunt and eat over oceans");
    }
}
class sky
{
    void allow(Bird b)
    {
        b.fly();
        b.eat();
    }
}


public class _6abstractionInChildClass
{
    public static void main(String[] args)
    {
        Pigeon pigeon =new Pigeon();
        GoldenEagle goldenEagle = new GoldenEagle();
```

```
        MountainEagle mountainEagle = new MountainEagle();
        sky sky = new sky();

        sky.allow(pigeon);
        System.out.println();
        sky.allow(goldenEagle);
        System.out.println();
        sky.allow(mountainEagle);
    }
}

//output
pigeon fly very low
pigeon eats grains

eagle fly very high
Golden eagles hunt and eat over oceans

eagle fly very high
Mountain eagles hunt and eat over mountains

Process finished with exit code 0
```

4. An abstract class **can have constructor and this gets called during the object creation** of its child class.

▼ Abstraction With Constructors

```
package _1Java_Codes_From_Basics._22Abstraction;

abstract class demoParent
{
    public demoParent()
    {
        System.out.println("Inside abstract class constructor");
    }
    abstract void func();
}

class childDemo extends demoParent
{

    public childDemo()
    {
        /**
         * @super() to call the constructor of other class
           @this() to call the constructor of same class
         */
        super();
    }
    void func()
    {
        System.out.println("Inside the child class user defined method");
    }
```

```
}

public class _7abstractClassWithConstructor
{
    public static void main(String[] args)
    {
        demoParent demoParent;
        demoParent = new childDemo();
        demoParent.func();
    }
}

//output
Inside abstract class constructor
Inside the child class user defined method

Process finished with exit code 0
```

5. An abstract class can also **have the final methods**

▼ Abstraction With Final Methods

```
package _1Java_Codes_From_Basics._22Abstraction;

abstract class testParent
{
    final void func1()
    {
        System.out.println("Inside the parent class final method");
    }
    abstract void func2();
}

class testChild extends testParent
{
    void func2()
    {
        System.out.println("Inside the child class method");
    }
}


public class _8abstractWithFinalMethods
{
    public static void main(String[] args)
    {
        testParent testParent;
        testParent = new testChild();
        testParent.func1();
        testParent.func2();
    }
}

//output
Inside the parent class final method
```

```
Inside the child class method

Process finished with exit code 0
```

Abstraction Example

▼ **area of square and rectangle using abstraction**

```
package _1Java_Codes_From_Basics._22Abstraction;

import java.util.Scanner;

abstract class shape
{
    float area;
    float length;
    abstract void input();
    abstract void compute();
    abstract void display();
}

class square extends shape
{
    public void input()
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the length of the square: ");
        length = scanner.nextInt();
    }
    public void compute()
    {
        area = length * length;
    }
    public void display()
    {
        System.out.println("The area of square is: "+area);
    }
}
class rectangle extends shape
{
    float breadth;
    public void input()
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the length of the rectangle: ");
        length = scanner.nextInt();
        System.out.print("Enter the breadth of the rectangle: ");
        breadth = scanner.nextInt();
    }
    public void compute()
    {
        area = length * breadth;
    }
    public void display()
```

```
    {
        System.out.println("The area of rectangle is: "+area);
    }
}
/**
 * @Important
 * Below type of class minimizes the length of program by passing reference of a p
arent class to a method of some other class
 * inorder to access the members
 */
class solve
{
    void allow(shape shapes)
    {
        shapes.input();
        shapes.compute();
        shapes.display();
    }
}
public class _9abstractionRealExample
{
    public static void main(String[] args)
    {
        square square = new square();
        rectangle rectangle = new rectangle();
        solve s = new solve();

        s.allow(square);
        System.out.println();
        s.allow(rectangle);
    }
}

//output
Enter the length of the square: 2
The area of square is: 4.0

Enter the length of the rectangle: 4
Enter the breadth of the rectangle: 4
The area of rectangle is: 16.0

Process finished with exit code 0
```

Note

- **NOTE: Abstract keyword can be used with following members:**
    1. class
    2. methods
    3. Interface
    4. Inner class

- **NOTE: Illegal combination of using abstraction**
    1. static
    2. final
    3. private
- **legal combinations are:**
    1. Strictfp
    2. Synchronized
    3. native