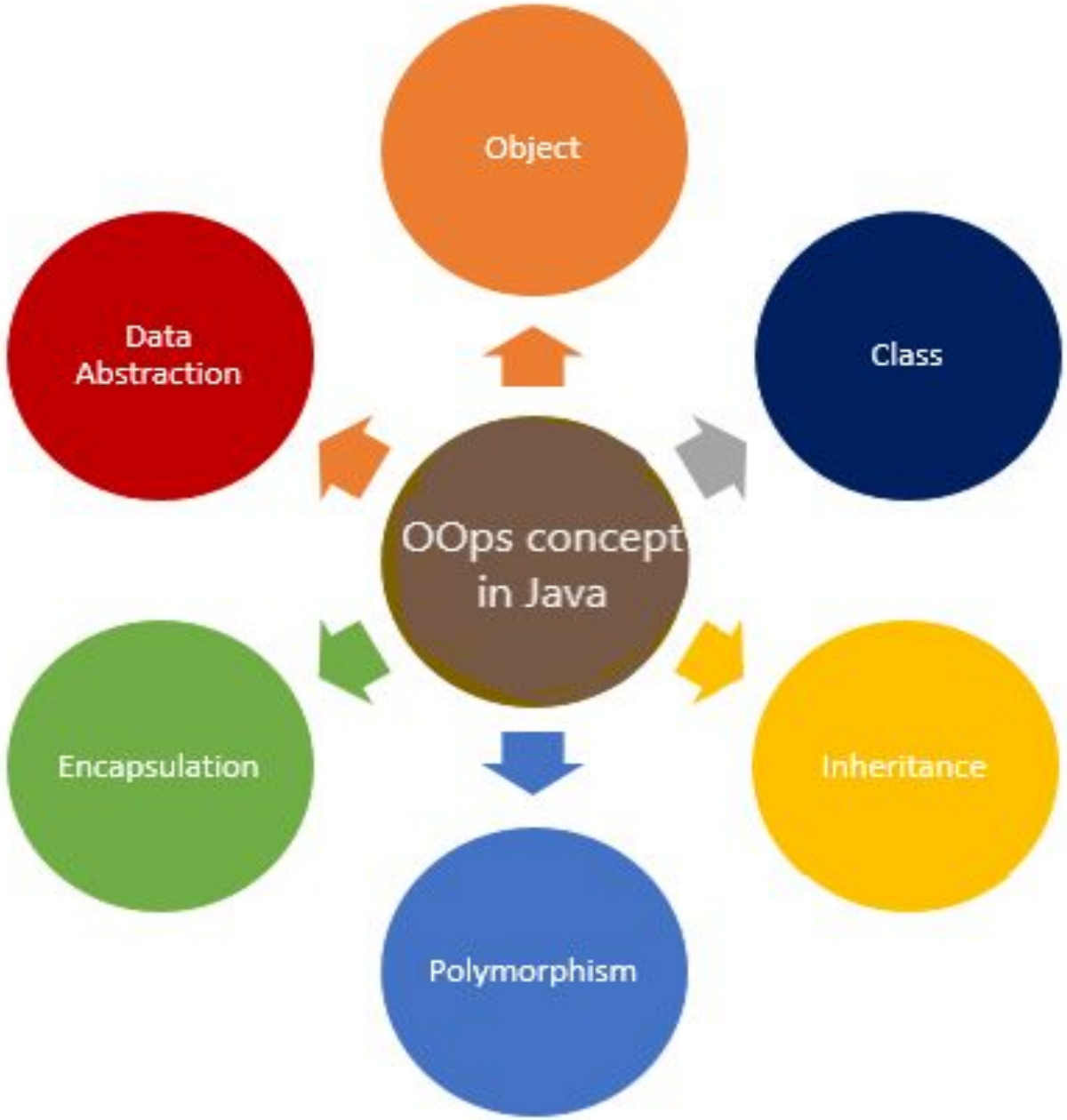
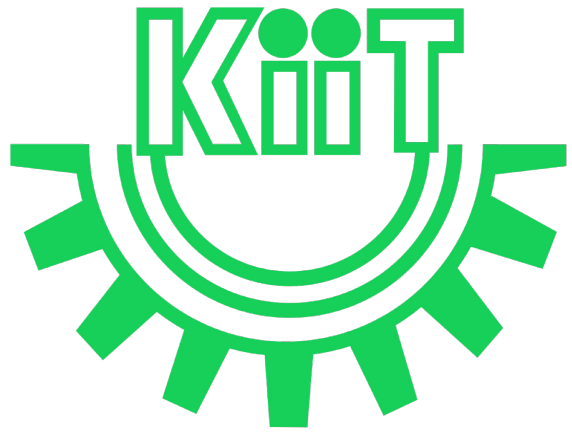


CS20004: Object Oriented Programming using Java

Lec-12

In this Discussion . . .

- Method overriding
  - super and overriding
  - overriding Vs. overloading
- Abstract class
- References



Method Overriding

- If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

  OR

- When a method of a subclass has the same name and type as a method of the super-class, we say that this method is overridden.

  OR

- If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Method Overriding usages

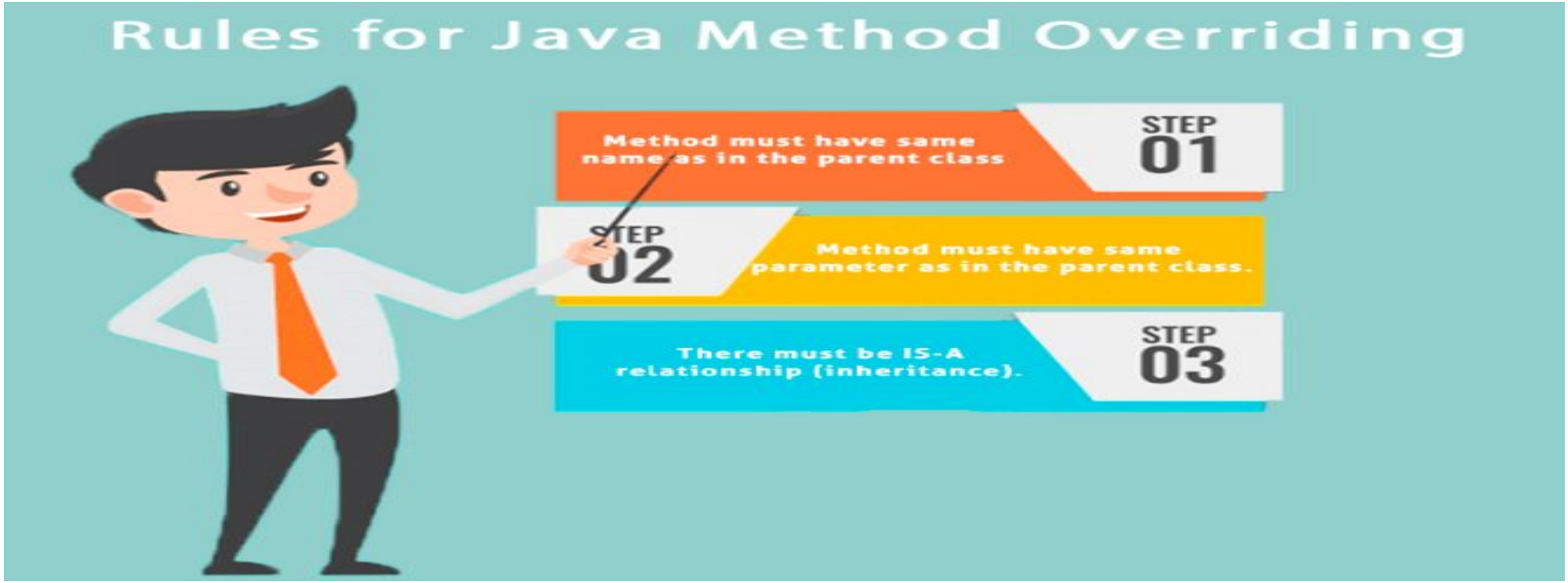
- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Note: When an overridden method is called from within the sub-class:

it will always refer to the sub-class method

super-class method is hidden

Rules for Method Overriding



IS-A Relationship Means Inheritance of some type must have been implemented.

Method Overriding Example

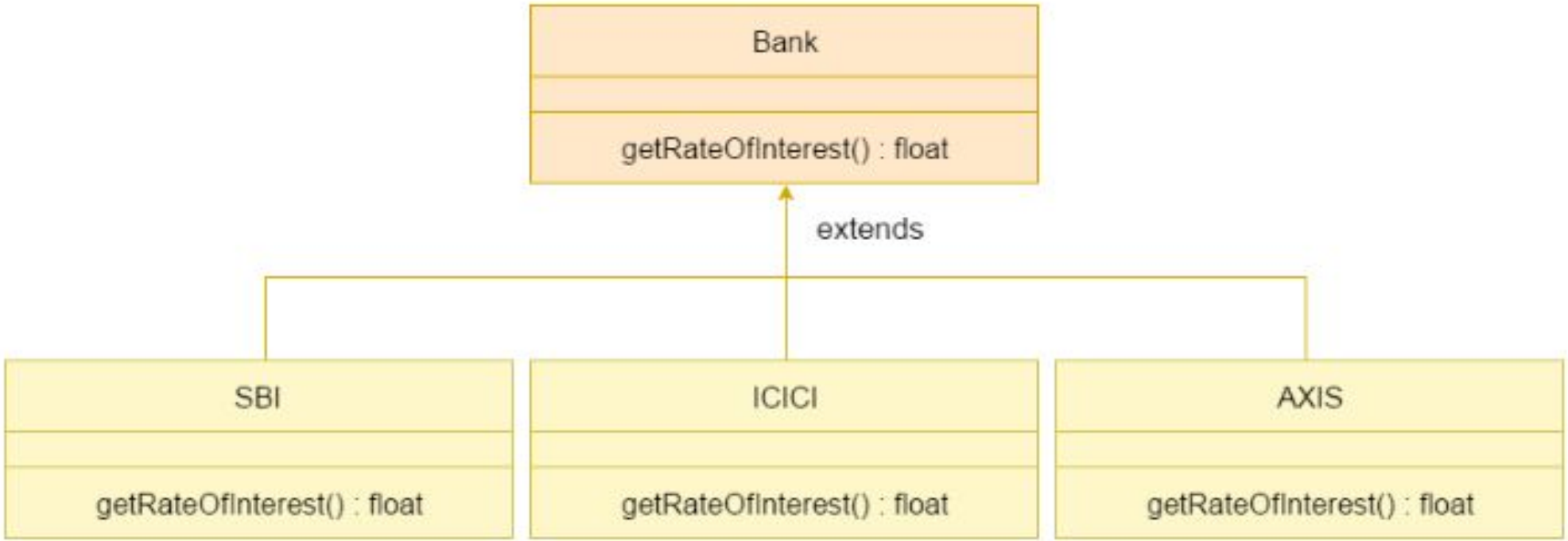
```
//Java Program to illustrate the use of Java Method Overriding
//Creating a parent class.
class Vehicle
{
        //defining a method
        void run()
        {
                System.out.println("Vehicle is running");
        }
}
//Creating a child class
class Bike2 extends Vehicle
{
        //defining the same method as in the parent class
        void run()
        {
                System.out.println("Bike is running safely");
        }

        public static void main(String args[])
        {
                Bike2 obj = new Bike2();//creating object
                obj.run();//calling method
        } }
```

- In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation.
- The name and parameter of the method are the same, and there is IS-A relationship between the classes, so there is method overriding.

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ javac Bik
e2.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ java Bike
2
Bike is running safely
```

Method overriding Real Life Example

- Consider a scenario where Bank is a class that provides functionality to get the rate of interest. However, the rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7%, and 9% rate of interest.



Method overriding Real Life Example

```
class Bank{
int getRateOfInterest(){return 0;}
}
//Creating child classes.
class SBI extends Bank{
int getRateOfInterest(){return 8;}
}

class ICICI extends Bank{
int getRateOfInterest(){return 7;}
}
class AXIS extends Bank{
int getRateOfInterest(){return 9;}
}
//Test class to create objects and call the methods
class TestOverriding{
public static void main(String args[]){
SBI s=new SBI();
ICICI i=new ICICI();
AXIS a=new AXIS();
System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
}
}
```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ javac Tes
tOverriding.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ java Test
Overriding
SBI Rate of Interest: 8
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9
```

super and Method Overriding

- The hidden super-class method may be invoked using super

```
class Vehicle
{
        void run()
        {
                System.out.println("Vehicle is running");
        }
}
class Bike extends Vehicle
{
        void run()
        {
                System.out.println("Calling the run() in child class");
        }
        public static void main(String args[])
        {
                //creating an instance of child class
                Bike obj = new Bike();
                //calling the method with child class instance
                obj.run();
        }
}
```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ javac Bike.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ java Bike
Calling the run() in child class
```

super and Method Overriding

- The hidden super-class method may be invoked using super

```
class Vehicle
{
        void run()
        {
                System.out.println("Vehicle is running");
        }
}
class Bike extends Vehicle
{
        void run()
        {
                super.run();
                System.out.println("Calling the run() in child class");
        }
        public static void main(String args[])
        {
                //creating an instance of child class
                Bike obj = new Bike();
                //calling the method with child class instance
                obj.run();
        } }
```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ javac Bike.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ java Bike
Vehicle is running
Calling the run() in child class
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$
```

Some general concepts regarding Method Overriding

- Can we override static method?
  - No, a static method cannot be overridden. It can be proved by runtime polymorphism, which we will deal with in upcoming classes.
- Why can we not override static method?
  - It is because the static method is bound with class whereas instance method is bound with an object. Static belongs to the class area, and an instance belongs to the heap area.
- Can we override java main method?
  - No, because the main is a static method.

Method Overriding Vs. Method Overloading

Method Overloading	Method Overriding
• Method overloading is used to *increase the readability* of the program.	• Method overriding is used to *provide the specific implementation of the* method that is already provided by its super class.
• Method overloading is performed *within class*.	• Method overriding occurs *in two classes* that have IS-A (inheritance) relationship.
• In case of method overloading, *parameter must be different*.	• In case of method overriding, *parameter must be same*.
• Method overloading is the example of *compile time polymorphism*.	• Method overriding is the example of *run time polymorphism*.
• In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter.	• *Return type must be same or covariant* in method overriding.

Method Overriding Vs. Method Overloading

- Overriding occurs only when the names and types of the two methods (super and subclass methods) are identical.
- If not identical, the two methods are simply overloaded.

```
class A {
        int i, j;
        A(int a, int b) {    i = a; j = b;        }
        void show() {        System.out.println("i and j: " + i + " " + j);  }
        }
class B extends A {
        int k;
        B(int a, int b, int c) {        super(a, b); k = c;          }
        void show(String msg) {     System.out.println(msg + k);        }
        }
class Override {
        public static void main(String args[]) {
                B subOb = new B(1, 2, 3);
                subOb.show("This is k: ");
                subOb.show();
                }
        }
```

Abstract class

- A class which is declared with the **abstract** keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

Abstraction in Java:

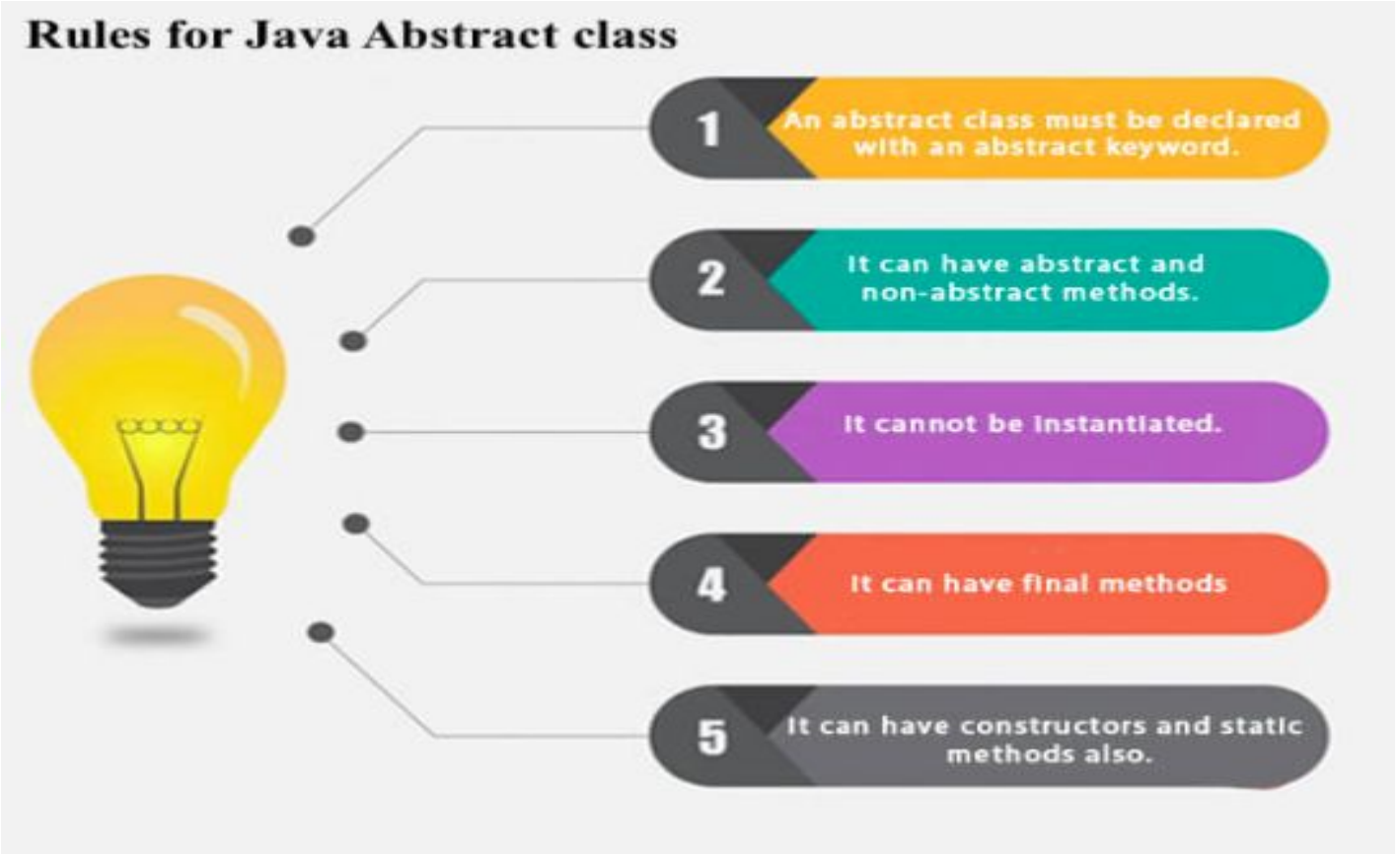
- **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.
- In other terms, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.
- Abstraction lets to focus on what the object does instead of how it does it.

Ways to achieve abstraction in Java

- There are two ways to achieve abstraction in java:
  - Abstract class (0 to 100%)
  - Interface (100%)

Abstract class

- A class which is declared as **abstract** is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.



Abstract class

- A class that contains an abstract method must be itself declared **abstract**
- An abstract class has no instances - it is illegal to use the new operator
- It is legal to define variables of the abstract class type
- A subclass of an abstract class:
  - implements all abstract methods of its super-class, or
  - is also declared as an abstract class
- Abstract super-class, concrete subclass

Abstract class and Abstract Method Syntax

- Abstract class

Syntax

```
abstract class class_name
{

}
```

- Abstract Method: A method which is declared as abstract and does not have implementation is known as an abstract method.

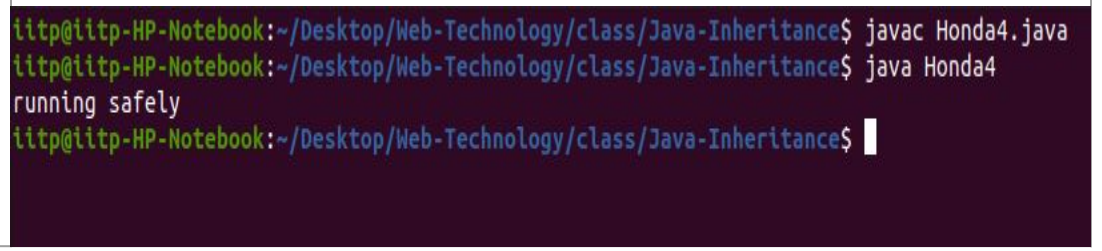
Syntax

```
abstract void method_name(); //no method body and abstract
```

Abstract class with Abstract Method Example

```
abstract class Bike{
  abstract void run();
}
class Honda4 extends Bike
{
        void run()
        {
                System.out.println("running safely");
        }
        public static void main(String args[])
        {
                Bike obj = new Honda4();
                obj.run();
        }
}
```

In this example, Bike is an abstract class that contains only one abstract method run. Its implementation is provided by the Honda class.



Abstract class with Abstract Method Example-I

```
abstract class Bank{
abstract int getRateOfInterest();
}
class SBI extends Bank{
int getRateOfInterest(){return 7;}
}
class PNB extends Bank{
int getRateOfInterest(){return 8;}
}

class TestBank{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("Rate of Interest is:
"+b.getRateOfInterest()+" %");
b=new PNB();
System.out.println("Rate of Interest is:
"+b.getRateOfInterest()+" %");
}}
```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ javac TestBank.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ java TestBank
Rate of Interest is: 7 %
Rate of Interest is: 8 %
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$
```

Abstract class having constructor, data member and methods

- An abstract class can have a data member, abstract method, method body (non-abstract method), constructor, and even main() method.

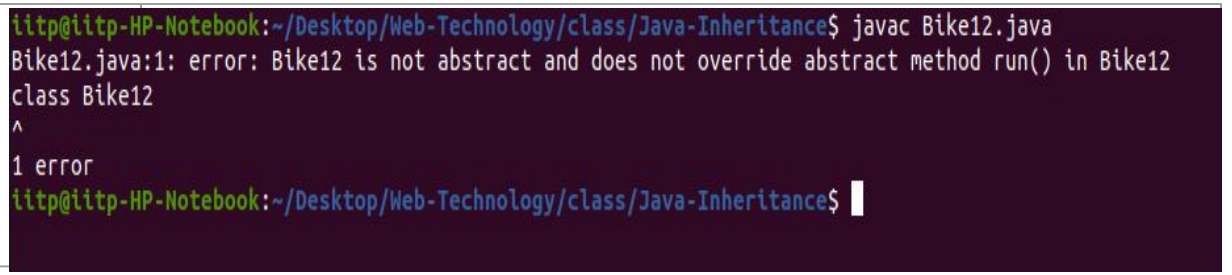
Rule: If there is an abstract method in a class, that class must be abstract.

```
//Example of an abstract class that has abstract and
non-abstract methods
 abstract class Bike
 {
        Bike()
        {System.out.println("bike is created");}
        abstract void run();
        void changeGear()
        {System.out.println("gear changed");}
 }
//Creating a Child class which inherits Abstract class
class Honda extends Bike
{
        void run(){System.out.println("running safely..");}
}
//Creating a Test class which calls abstract and non-abstract
methods
class TestAbstraction2
{
        public static void main(String args[])
        {
                Bike obj = new Honda();
                obj.run();
                obj.changeGear();
 } }
```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ javac TestAbstraction2.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$ java TestAbstraction2
bike is created
running safely..
gear changed
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Inheritance$
```

Abstract Class: Compilation Error

```
class Bike12
{
        abstract void run();
}
```



Rule: If we are extending an abstract class that has an abstract method, we must either provide the implementation of the method or make this class abstract.

References

1. https://www.javatpoint.com/method-overloading-vs-method-overriding-in-java
2. https://www.javatpoint.com/method-overriding-in-java
3.
4.