Unit-1 Object Oriented Programming with Java

OOPs (Object-Oriented Programming System)

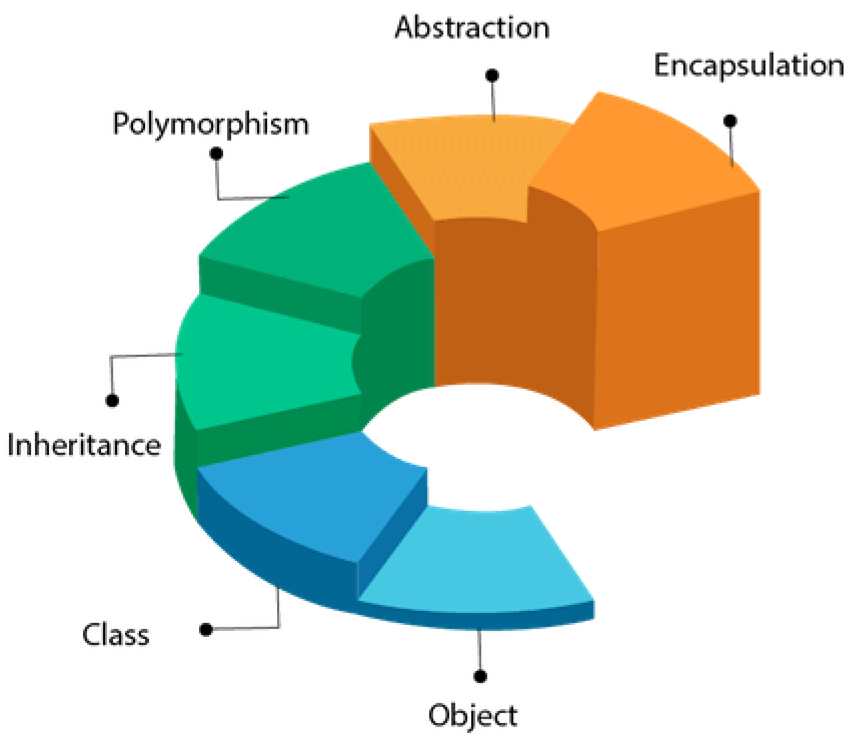
Object means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

Unit-1 Object Oriented Programming with Java

OOPs (Object-Oriented Programming System)



Object

Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

Class

Collection of objects is called class. It is a logical entity.

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

If *one task is performed in different ways*, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

Encapsulation

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

.

What is a method in Java?

A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the **reusability** of code.

Types of Method

There are two types of methods in Java:

- Predefined Method
- User-defined Method

Predefined Method

In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods.

Let's see an example of the predefined method.

Demo.java

```
public class Demo
{
public static void main(String[] args)
{
// using the max() method of Math class
System.out.print("The maximum number is : " + Math.max(9,7));
}
}
```

Output:

```
The maximum number is: 9
```

User-defined Method

The method written by the user or programmer is known as **a user-defined** method.

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

How to Create a User-defined Method

```
import java.util.*;
 class Add
 {

 int first_no=2;
 int second_no =2;
 int total;
 void details()
 {
 total=(first_no+second_no);
  System.out.println("srk");

 }
 void show()
 {
 System.out.println("add of two no "+total);
 }

public static void main(String args[])
 {
```

```
 Add ob = new Add();
 ob.details();
 ob.show();
 }
}
```

Method Overloading in Java

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**

```
class TestOverloading1

{

public static void main(String[] args)

{

  TestOverloading1 ob=new TestOverloading1();

  ob.add(2,3);

  ob.add(1,2,3);

}




void add(int a,int b)

{
  int  add=a+b;

  System.out.println(add);


}

void add(int a,int b,int c)

{

   int add= a+b+c;

   System.out.println(add);

}

}
```

Constructors in Java

a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor,

.

Rules for creating Java constructor

There are two rules defined for the constructor.

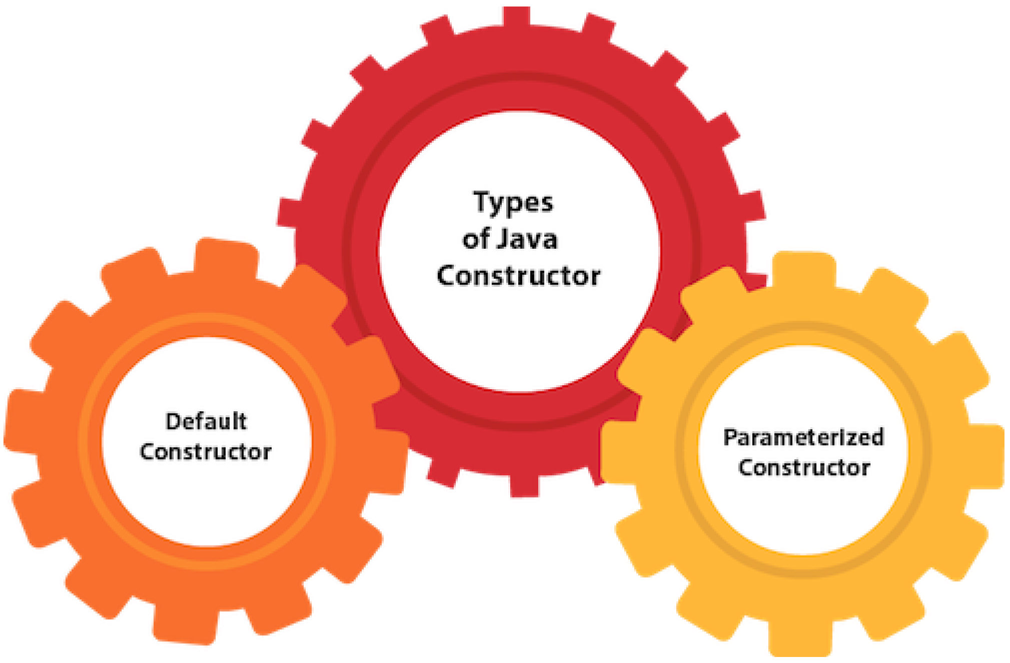
Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor



Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Example of default constructor

```
//Java Program to create and call a default constructor
class Bike1{
//creating a default constructor
Bike1()
{
System.out.println("Bike is created");
}
//main method
public static void main(String args[])
  {
  //calling a default constructor
  Bike1 b=new Bike1();
  }
  }
```

Test it Now

Output:

```
Bike is created
```

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Example of parameterized constructor

```
class Student4
{
    int id;
    String name;
    //creating a parameterized constructor
    Student4(int i, String n)
{
    id = i;
    name = n;
        }
        //method to display the values
        void display()
    {
    System.out.println(id+" "+name)
;}

    public static void main(String args[]){
    //creating objects and passing values
    Student4 s1 = new Student4(111,"Karan");
    S1.display();
   }
}
```

Test it Now

Output:

```
111 Karan
```

Constructor Overloading in Java

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task.

Example of Constructor Overloading

```
//Java program to overload constructors
class Student5{
    int id;
    String name;
    int age;
    //creating two arg constructor
    Student5(int i,String n)
{
    id = i;
        name = n;
        }
        //creating three arg constructor
        Student5(int i,String n,int a)
    {
        id = i;
        name = n;
        age=a;
        }
        void display(){System.out.println(id+" "+name+" "+age);}

        public static void main(String args[]){
        Student5 s1 = new Student5(111,"Karan");
        Student5 s2 = new Student5(222,"Aryan",25);
        s1.display();
        s2.display();
      }
  }
```

Test it Now

Output:

```
111 Karan 0
222 Aryan 25
```

Difference between constructor and method in Java

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

Next →← Prev

Unit-1 Object Oriented Programming with Java

Inheritance in Java

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

Terms used in Inheritance

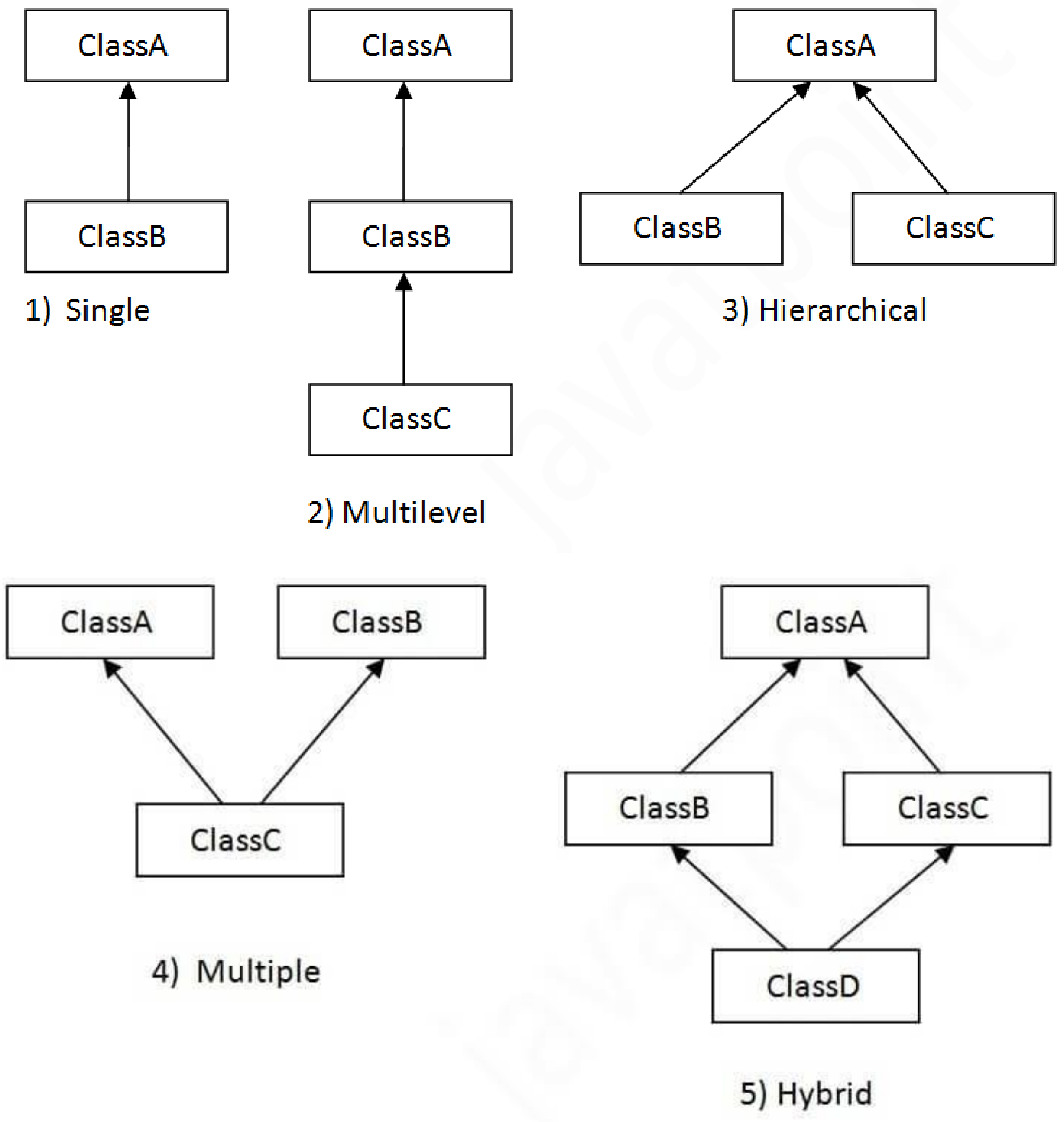
- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

The syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
   //methods and fields
}
```

types of inheritance in java

Unit-1 Object Oriented Programming with Java



Single Inheritance Example

When a class inherits another class, it is known as a *single inheritance*. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

```
class Animal
```

```
        {
        void eat()
        {
        System.out.println("eating...");
        }
    }
        class Dog extends Animal
        {
void bark()
        {
        System.out.println("barking...");
        }
        }
        class TestInheritance
        {
        public static void main(String args[])
        {
        Dog d=new Dog();
        d.bark();
        d.eat();
        }

        }
```

Output:

```
barking...
eating...
```

Multilevel Inheritance Example

When there is a chain of inheritance, it is known as *multilevel inheritance*. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

File: TestInheritance2.java

```
        class Animal
```

Unit-1 Object Oriented Programming with Java

```
{
void eat()
{
System.out.println("eating...");
}
}
class Dog extends Animal
{
void bark()
{
System.out.println("barking...");
}
}
class BabyDog extends Dog
{
void weep()
{
System.out.println("weeping...");
}
}
class TestInheritance2
{
public static void main(String args[])
{
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}

}
```

Output:

```
weeping...
barking...
eating...
```

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

Hierarchical Inheritance Example

When two or more classes inherits a single class, it is known as *hierarchical inheritance*. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

File: TestInheritance3.java

```
class Animal
        {
        void eat()
        {
        System.out.println("eating...");
        }
        }
        class Dog extends Animal
        {
        void bark()
        {
        System.out.println("barking...");
        }
        }
        class Cat extends Animal
        {
        void meow()
        {
        System.out.println("meowing...");
        }
        }
        class TestInheritance3
        {
        public static void main(String args[])
        {
        Cat c=new Cat();
        c.meow();
        c.eat();
        //c.bark();//C.T.Error
        }

        }
```

Output:

```
meowing...
eating...
```

Q) Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

```
class A
{
void msg(){System.out.println("Hello");
}
}
class B
{
void msg()
{
System.out.println("Welcome");
}
}
class C extends A,B
{
//suppose if it were
```

Unit-1 Object Oriented Programming with Java

```
 public static void main(String args[])
{
  C obj=new C();
  obj.msg();//Now which msg() method would be invoked?
}
}
```

Test it Now

```
Compile Time Error
```

Polymorphism in Java

Polymorphism in Java is a concept by which we can perform a *single action in different ways*. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

If you overload a static method in Java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

Example of Java Runtime Polymorphism

In this example, we are creating two classes Bike and Splendor. Splendor class extends Bike class and overrides its run() method. We are calling the run method by the reference variable of Parent class. Since it refers to the subclass object and subclass method overrides the Parent class method, the subclass method is invoked at runtime.

Since method invocation is determined by the JVM not compiler, it is known as runtime polymorphism.

```
class Bike{
 void run()
{
System.out.println("running");}
}
class Splendor extends Bike{
 void run(){System.out.println("running safely with 60km");}

 public static void main(String args[]){
  Bike b = new Splendor();//upcasting
  b.run();
 }
}
```

Test it Now

Output:

```
running safely with 60km.
```

Abstraction in Java

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, 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 you focus on what the object does instead of how it does it.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

Abstract class (0 to 100%)

Interface (100%)

Unit-1 Object Oriented Programming with Java

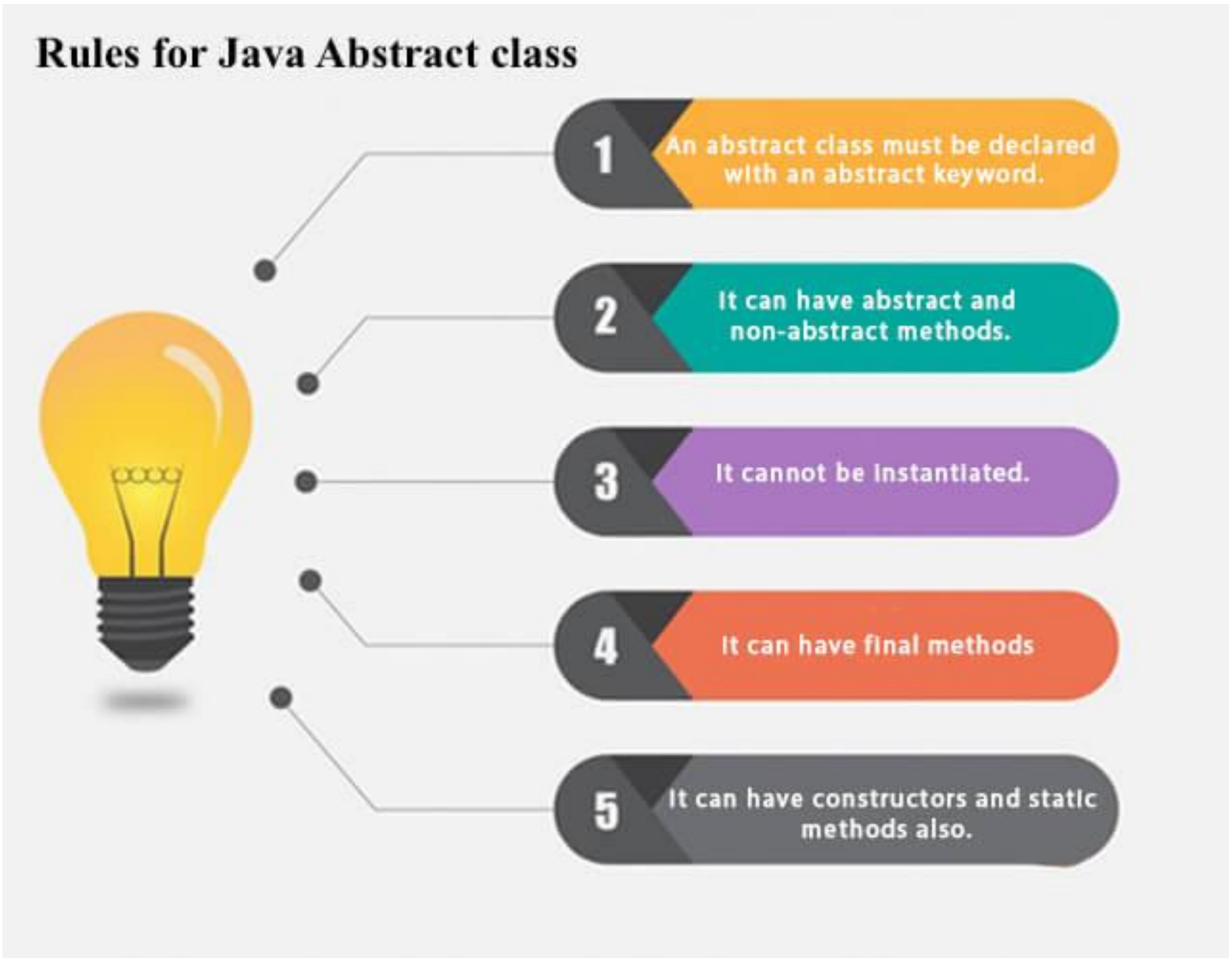
Abstract class in Java

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.

Points to Remember

ADVERTISEMENT

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.


Rules for Java Abstract class
1
An abstract class must be declared with an abstract keyword.
2
It can have abstract and non-abstract methods.
3
It cannot be instantiated.
4
It can have final methods
5
It can have constructors and static methods also.

Unit-1 Object Oriented Programming with Java

Example of abstract class

```
1. abstract class A{}
```

Abstract Method in Java

A method which is declared as abstract and does not have implementation is known as an abstract method.

Example of abstract method

```
1. abstract void printStatus();//no method body and abstract
```

Example of Abstract class that has an abstract method

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 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();
}
}
```

Test it Now

```
running safely
```

Unit-1 Object Oriented Programming with Java