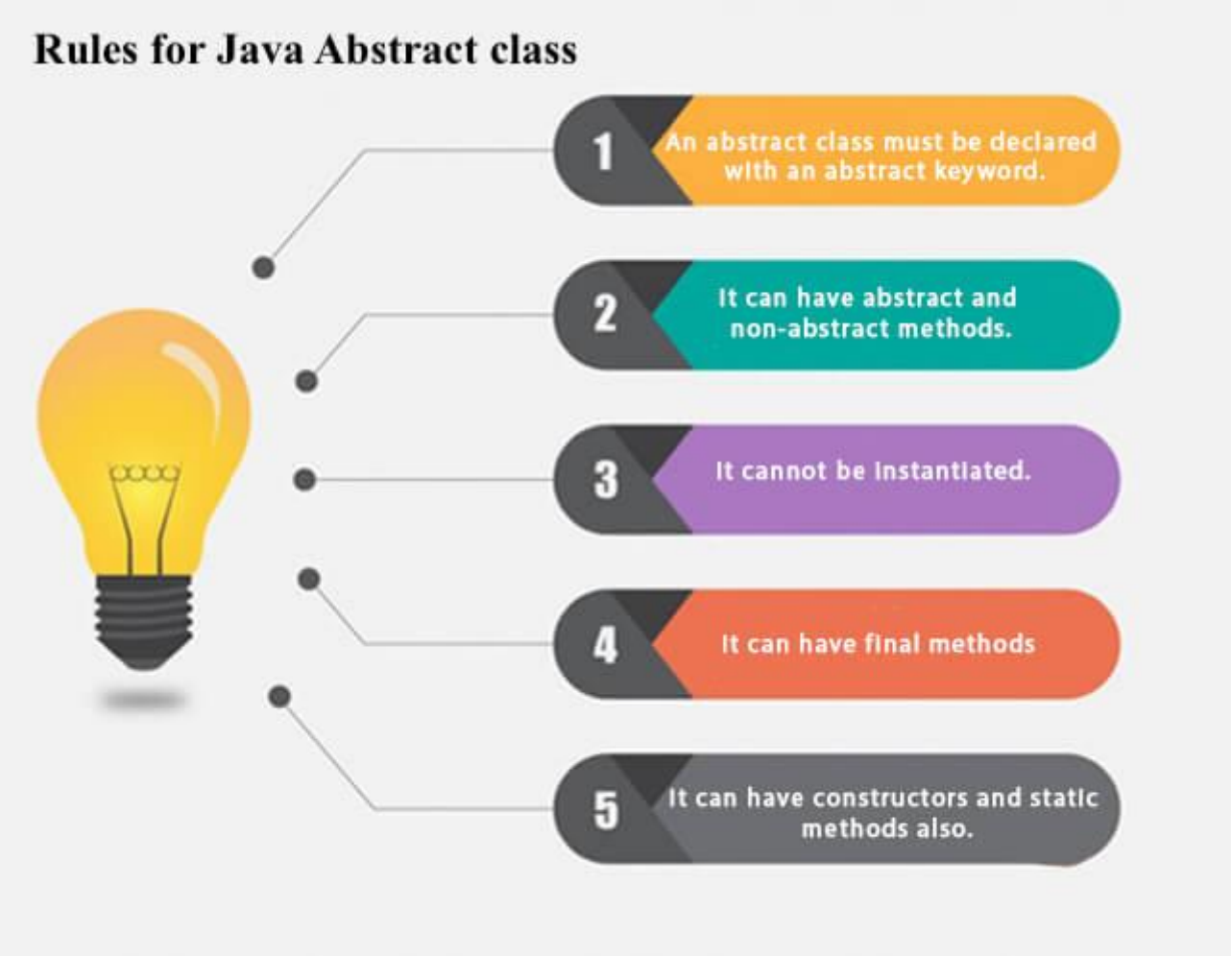
Abstract class in Java

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



Elements abstract class can have

- data member
- abstract method
- method body (non-abstract method)
- constructor
- main() method.

Why And When To Use Abstract Classes and Methods?

To achieve security - hide certain details and only show the important details of an object.

Note: Abstraction can also be achieved with Interfaces,

```
abstract class Shape{
abstract void draw();
}
//In real scenario, implementation is provided by others i.e. unknown by end user
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");}
}
class Circle1 extends Shape{
void draw(){System.out.println("drawing circle");}
}
//In real scenario, method is called by programmer or user
class TestAbstraction1{
public static void main(String args[]){
Shape s=new Circle1();//In a real scenario, object is provided through method, e.g., getS
hape() method
s.draw();
}
}
```

Test it Now

```
drawing circle
```

```
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()+" %");
```

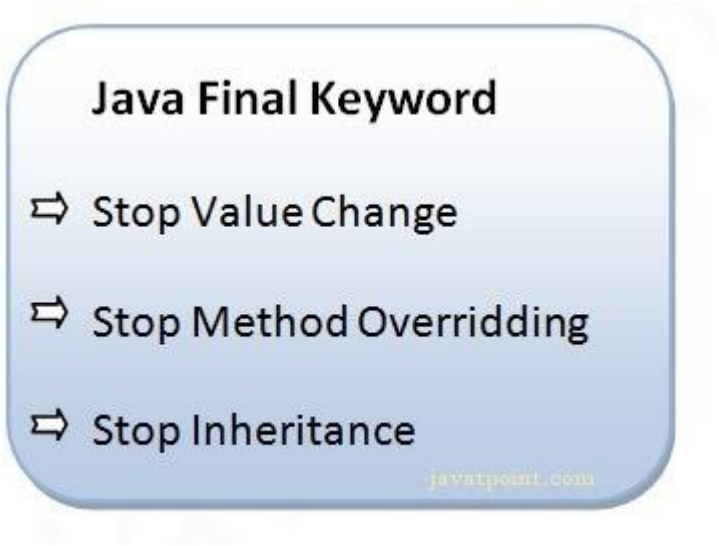
```
b=new PNB();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
}}
```

Test it Now

```
Rate of Interest is: 7 %
Rate of Interest is: 8 %
```

Using final with Inheritance in Java

Inheritance allows a new class to inherit the members (fields and methods) of an existing class. The 'final' keyword can restrict this inheritance. When a class is declared as 'final', it can't be extended. Similarly, a 'final' method can't be overridden by subclasses



Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

Example of final variable

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
class Bike9{
```

```
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[]){
 Bike9 obj=new  Bike9();
 obj.run();
 }
}//end of class
```

Test it Now

```
Output:Compile Time Error
```

---

3) Java final class

If you make any class as final, you cannot extend it.

Example of final class

```
final class Bike{}

class Honda1 extends Bike{
  void run(){System.out.println("running safely with 100kmph");}

  public static void main(String args[]){
  Honda1 honda= new Honda1();
  honda.run();
  }
}
```

Test it Now

```
Output:Compile Time Error
```

Q) Is final method inherited?

Ans) Yes, final method is inherited but you cannot override it. For Example:

```
class Bike{
  final void run(){System.out.println("running...");}
}
```

```
class Honda2 extends Bike{
  public static void main(String args[]){
   new Honda2().run();
  }
}
```

Test it Now

```
Output:running...
```

There are two types of binding

➢Static binding (also known as early binding) – At Compile Time

Java Package

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

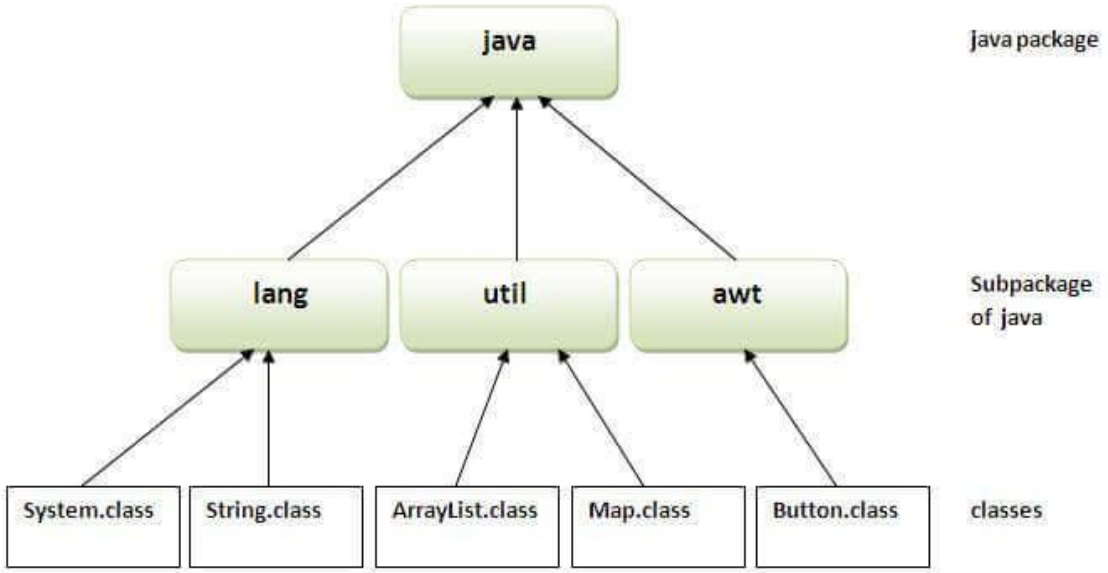
Here, we will have the detailed learning of creating and using user-defined package

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.



How to access package from another package?

There are three ways to access the package from outside the package.

1. import package.*;
2. import package.classname;
3. fully qualified name.

Example of package that import the packagename.*

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.*;

class B{
  public static void main(String args[]){
   A obj = new A();
```

```
  obj.msg();
 }
}
Output:Hello
```

2) Using packagename.classname

If you import package.classname then only declared class of this package will be accessible.

Example of package by import package.classname

```
//save by A.java

package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.A;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
Output:Hello
```

3) Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

Example of package by import fully qualified name

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
class B{
  public static void main(String args[]){
   pack.A obj = new pack.A();//using fully qualified name
   obj.msg();
  }
}
Output:Hello
```

Interface in Java

In Java, an interface specifies the behavior of a class by providing an abstract type. As one of Java's core concepts, abstraction, polymorphism, and multiple inheritance are supported through this technology. Interfaces are used in Java to achieve abstraction. By using the implements keyword, a Java class can implement an interface.

Java interfaces define method signatures without implementations, giving classes a template to follow. They encourage code flexibility, which facilitates scalability and easier maintenance.

Need for Interface in Java

So we need an Interface in Java for the following reasons:

- Total Abstraction
- Multiple Inheritance
- Loose-Coupling

Total Abstraction

Abstraction is the critical concept of Object-Oriented programming techniques. An interface only stores the method signature and not the method definition. Method Signatures make an Interface achieve complete Abstraction by hiding the method implementation from the user.

Multiple Inheritance

Without Interface, the process of multiple inheritances is impossible as the conventional way of inheriting multiple parent classes results in profound ambiguity. This type of ambiguity is known as the Diamond problem. Interface resolves this issue.

Loose Coupling

The term Coupling describes the dependency of one class for the other. So, while using an interface, we define the method separately and the signature separately. This way, all the methods, and classes are entirely independent and archives Loose Coupling.

Without Interface, the process of multiple inheritances is impossible as the conventional way of inheriting multiple parent classes results in profound ambiguity. This type of ambiguity is known as the Diamond problem. Interface resolves this issue.

The term Coupling describes the dependency of one class for the other. So, while using an interface, we define the method separately and the signature separately. This way, all the methods, and classes are entirely independent and archives Loose Coupling.

Java Interface Syntax

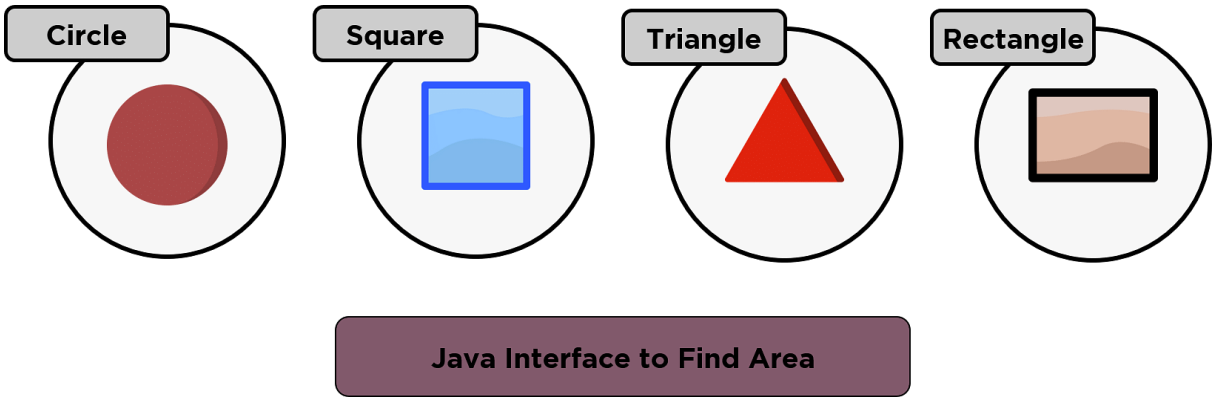
So the Syntax of an Interface in Java is written as shown below.

```
Interface <Interface Name> {

//Declare Constant Fields;

//Declare Methods;

//Default Methods;

 }
```

With the syntax explained, let us now move ahead onto the next part, where we go through an example.

Java Interface Example

Following is an ideal example of Interface in Java. Here we try to calculate the area of geometrical shapes, and for each shape, we have different methods. And all the methods are defined, independent of each other. Only method signatures are written in the Interface.




```
//Interface

package simplilearn;

public interface Area {

public void Square();

public void Circle();

public void Rectangle();

public void Triangle();

}

//Class
```

```
package simplilearn;

import java.util.Scanner;

public class shapeArea implements Area {

public void Circle() {

Scanner kb = new Scanner(System.in);

System.out.println("Enter the radius of the circle");

double r = kb.nextInt();

double areaOfCircle = 3.142 * r * r;

System.out.println("Area of the circle is " + areaOfCircle);

}

@Override

public void Square() {

// TODO Auto-generated method stub

Scanner kb2 = new Scanner(System.in);

System.out.println("Enter the length of the side of the square");

double s = kb2.nextInt();

double areaOfSquare = s * s;

System.out.println("Area of the square is " + areaOfSquare);

}

@Override

public void Rectangle() {
```

```
// TODO Auto-generated method stub
Scanner kb3 = new Scanner(System.in);
System.out.println("Enter the length of the Rectangle");
double l = kb3.nextInt();
System.out.println("Enter the breadth of the Rectangle");
double b = kb3.nextInt();
double areaOfRectangle = l * b;
System.out.println("Area of the Rectangle is " + areaOfRectangle);
}
@Override
public void Triangle() {
// TODO Auto-generated method stub
Scanner kb4 = new Scanner(System.in);
System.out.println("Enter the base of the Triangle");
double base = kb4.nextInt();
System.out.println("Enter the height of the Triangle");
double h = kb4.nextInt();
double areaOfTriangle = 0.5 * base * h;
System.out.println("Area of the Triangle is " + areaOfTriangle);
}
public static void main(String[] args) {
```

```
shapeArea geometry = new shapeArea();

geometry.Circle();

geometry.Square();

geometry.Rectangle();

geometry.Triangle();

}

}

//ExpectedOutput:

Enter the radius of the circle

15

Area of the circle is 706.9499999999999

Enter the length of the side of the square

12

Area of the square is 144.0

Enter the length of the Rectangle

10

Enter the breadth of the Rectangle

25

Area of the Rectangle is 250.0

Enter the base of the Triangle

25
```

Enter the height of the Triangle

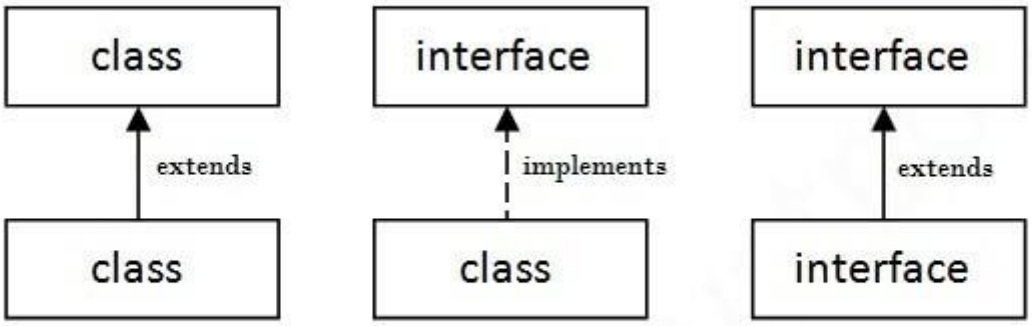
30

Area of the Triangle is 375.0

Why use Java interface?

The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.



Differences between a Class and an Interface

The following table lists all the major differences between an interface and a class in Java language:

Class	Interface
The keyword used to create a class is “class”	The keyword used to create an interface is “interface”
A class can be instantiated i.e., objects of a class can be created.	An Interface cannot be instantiated i.e. objects cannot be created.
Classes do not support multiple inheritance.	The interface supports multiple inheritance.
It can be inherited from another class.	It cannot inherit a class.
It can be inherited by another class using the keyword ‘extends’.	It can be inherited by a class by using the keyword ‘implements’ and it can be inherited by an interface using the keyword ‘extends’.
It can contain constructors.	It cannot contain constructors.
It cannot contain abstract methods.	It contains abstract methods only.

Class	Interface
Variables and methods in a class can be declared using any access specifier(public, private, default, protected).	All variables and methods in an interface are declared as public.
Variables in a class can be static, final, or neither.	All variables are static and final.