Web Technology
(KCS-602)
Unit 1

Prepared By

Abhishek Kesharwani

Assistant Professor,UCER Naini,Allahabad

Lecture 8

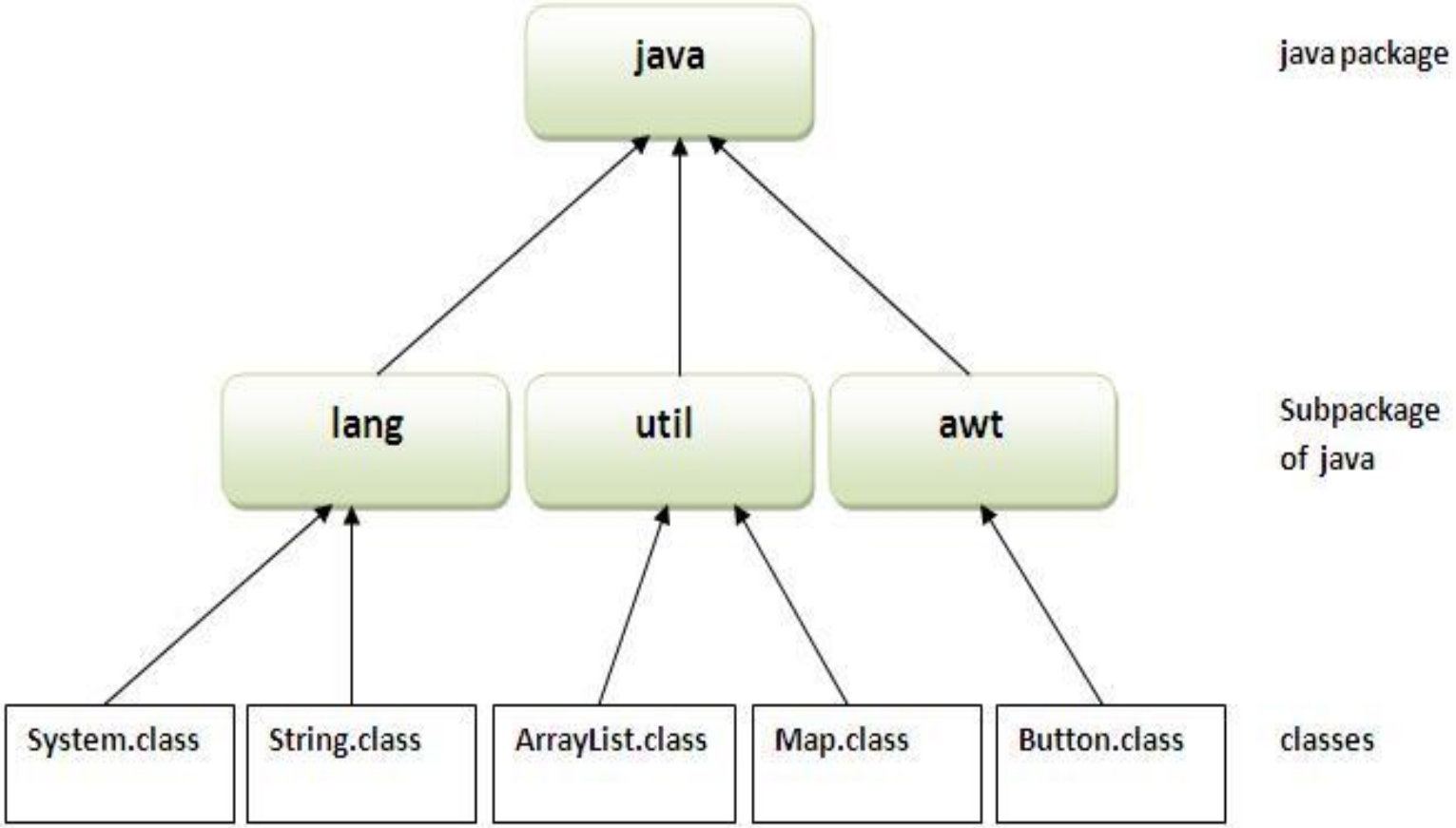
- Package and Interface

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.

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.


java
java package
lang
util
awt
Subpackage
of java
System.class
String.class
ArrayList.class
Map.class
Button.class
classes

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

- import package.*;
- import package.classname;
- fully qualified name.

1) Using packagename.*

If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

```
//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();
  }  }
```

2) Using packagename.classname

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

```
//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();
  }
}
```

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.

```
//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();
  }
}
```

Abstract class in Java

- A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).

Abstraction in Java

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

Ways to achieve Abstaction

There are two ways to achieve abstraction in java

- Abstract class (0 to 100%)
- Interface (100%)

Abstract class in Java

- A class that is declared as abstract is known as **abstract class**. It needs to be extended and its method implemented. It cannot be instantiated.

Example

abstract class A{}

Abstract method

- A method that is declared as abstract and does not have implementation is known as abstract method.

Example

abstract void printStatus();//no body and abstract

Example of abstract class that has abstract method

In this example, Bike the abstract class that contains only one abstract method run. It 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[]){
 Honda4 obj = new Honda4();
 obj.run();
}
}
```

```
abstract class Bank{
abstract double getRateOfInterest();
}

class SBI extends Bank{
double getRateOfInterest(){return 7.5;}
}
class PNB extends Bank{
double getRateOfInterest(){return 7;}
}

class TestBank{
public static void main(String args[]){
SBI b=new SBI();//if object is PNB, method of PNB will be invoked
int interest=b.getRateOfInterest();
System.out.println("Rate of Interest is: "+interest+" %");
}}
```

- Rate of Interest is: 7.5 %

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

File: TestAbstraction2.java

```
//example of abstract class that have method body
 abstract class Bike{
   Bike(){System.out.println("bike is created");}
   abstract void run();
   void changeGear(){System.out.println("gear changed");}
 }
 class Honda extends Bike{
 void run(){System.out.println("running safely..");}
 }
 class TestAbstraction2{
 public static void main(String args[]){
  Bike obj = new Honda();
  obj.run();
  obj.changeGear();
 } }
```

- **Rule: If you are extending any abstract class that have abstract method, you must either provide the implementation of the method or make this class abstract.**
- **Rule: If there is any abstract method in a class, that class must be abstract.**

The abstract class can also be used to provide some implementation of the interface. In such case, the end user may not be forced to override all the methods of the interface.

```
interface A{
void a();
void b();
void c();
void d();
}

abstract class B implements A{
public void c(){System.out.println("I am C");}
}
```

```
class M extends B{
public void a(){System.out.println("I am a");}
public void b(){System.out.println("I am b");}
public void d(){System.out.println("I am d");}
}

class Test5{
public static void main(String args[]){
M a=new M();
a.a();
a.b();
a.c();
a.d();
}}
```

Interface in Java

- An **interface in java** is a blueprint of a class. It has static constants and abstract methods only.
- The interface in java is **a mechanism to achieve fully abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java.
- Java Interface also **represents IS-A relationship**.
- It cannot be instantiated just like abstract class.

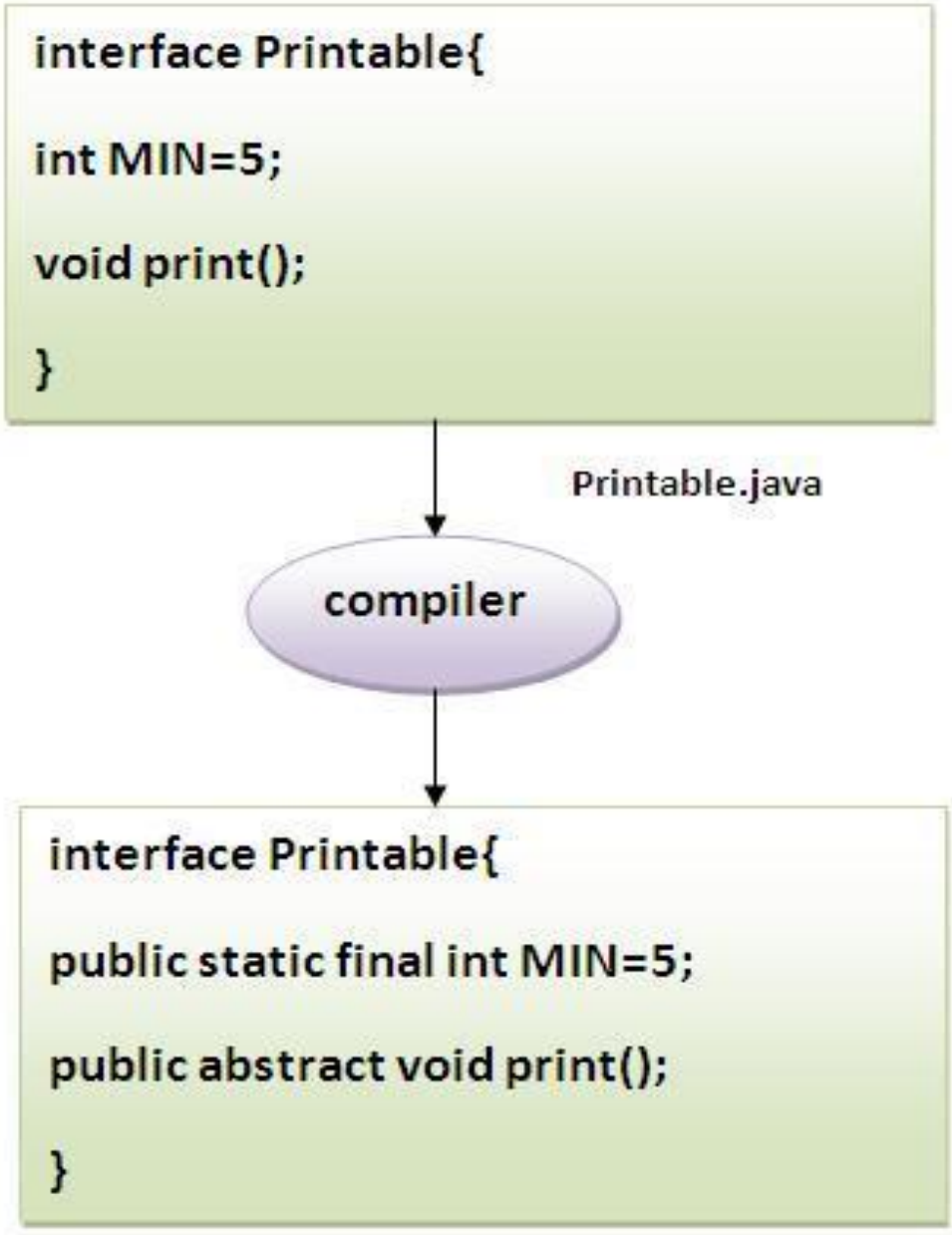
Why use Java interface?

There are mainly three reasons to use interface. They are given below.

- It is used to achieve fully abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.

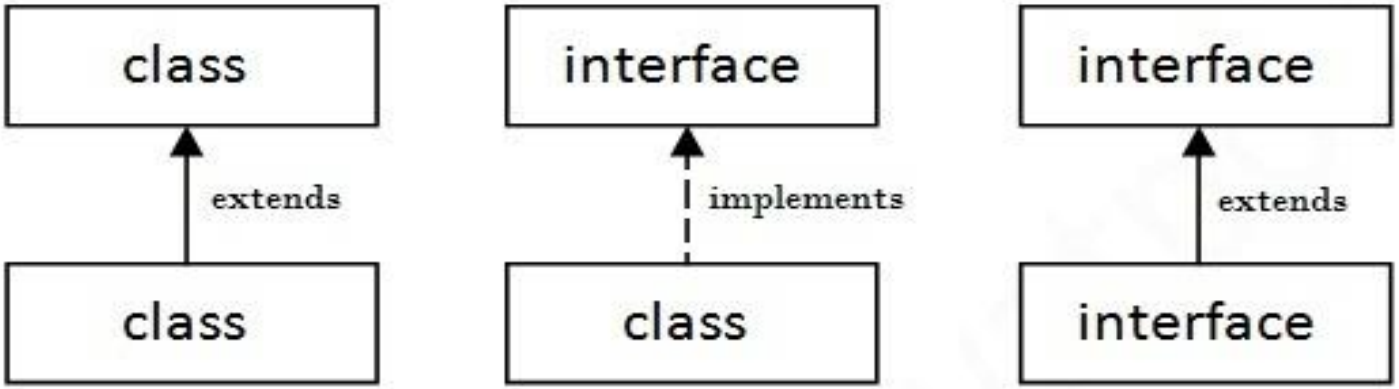
In other words, Interface fields are public, static and final by default, and methods are public and abstract.

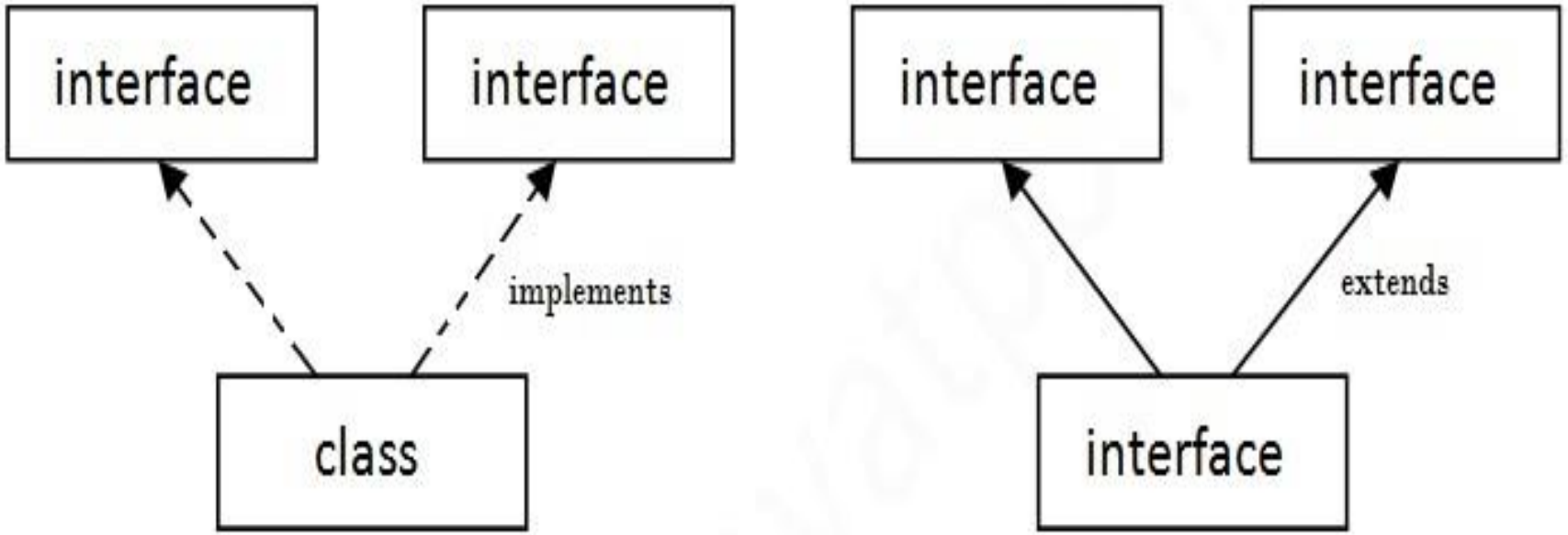

interface Printable{
int MIN=5;
void print();
}
Printable.java
compiler
interface Printable{
public static final int MIN=5;
public abstract void print();
}
Printable.class

```
interface printable{
void print();
}

class A6 implements printable{
public void print(){System.out.println("Hello");}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }
}
```

Multiple inheritance in Java by interface

- If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.





Multiple Inheritance in Java

```
interface Printable{
void print();
}

interface Showable{
void show();
}

class A7 implements Printable,Showable{

public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
 }
}
```

Interface inheritance

A class implements interface but one interface extends another interface .

```
interface Printable{
void print();
}
interface Showable extends Printable{
void show();
}
class Testinterface2 implements Showable{

public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
Testinterface2 obj = new Testinterface2();
obj.print();
obj.show();
 } }
```

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract**methods.	Interface can have **only abstract** methods.
2) Abstract class **doesn't support multiple inheritance.**	Interface **supports multiple inheritance.**
3) Abstract class **can have final, non-final, static and non-static variables.**	Interface has **only static and final variables.**
4) Abstract class **can have static methods, main method and constructor.**	Interface **can't have static methods, main method or constructor.**
5) Abstract class **can provide the implementation of interface.**	Interface **can't provide the implementation of abstract class.**
6) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.

) **Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Can we define private and protected modifiers for data members (fields) in interfaces?

- Ans: No, we cannot define private and protected modifiers for variables in interface because the fields (data members) declared in an interface are by default public, static, and final.

Which modifiers are allowed for methods in an Interface?

- Ans: Only abstract and public modifiers are allowed for methods in interfaces.

Suppose A is an interface. Can we create an object using new A()?

- Ans: No, we cannot create an object of interface using new operator. But we can create a reference of interface type and interface reference refers to objects of its implementation classes.