Programming Language II
CSE-215

Prof. Dr. Mohammad Abu Yousuf

yousuf@juniv.edu

Java interface

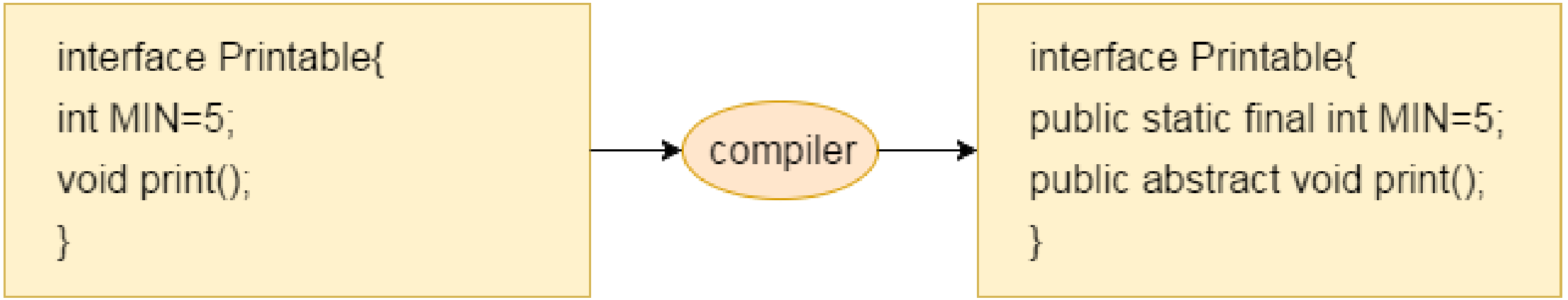
- A **Java interface** is a bit like a class, except a **Java interface** can only contain method signatures and fields. An **Java interface** cannot contain an implementation of the methods, only the signature (name, parameters and exceptions) of the method. You can use **interfaces** in **Java** as a way to achieve polymorphism.
- Since Java 8, we can have method body in interface. But we need to make it **default** method.
- Since Java 8, we can have **static method** in interface.
- 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 abstraction.
  - By interface, we can support the functionality of multiple inheritance.
  - It can be used to achieve loose coupling.

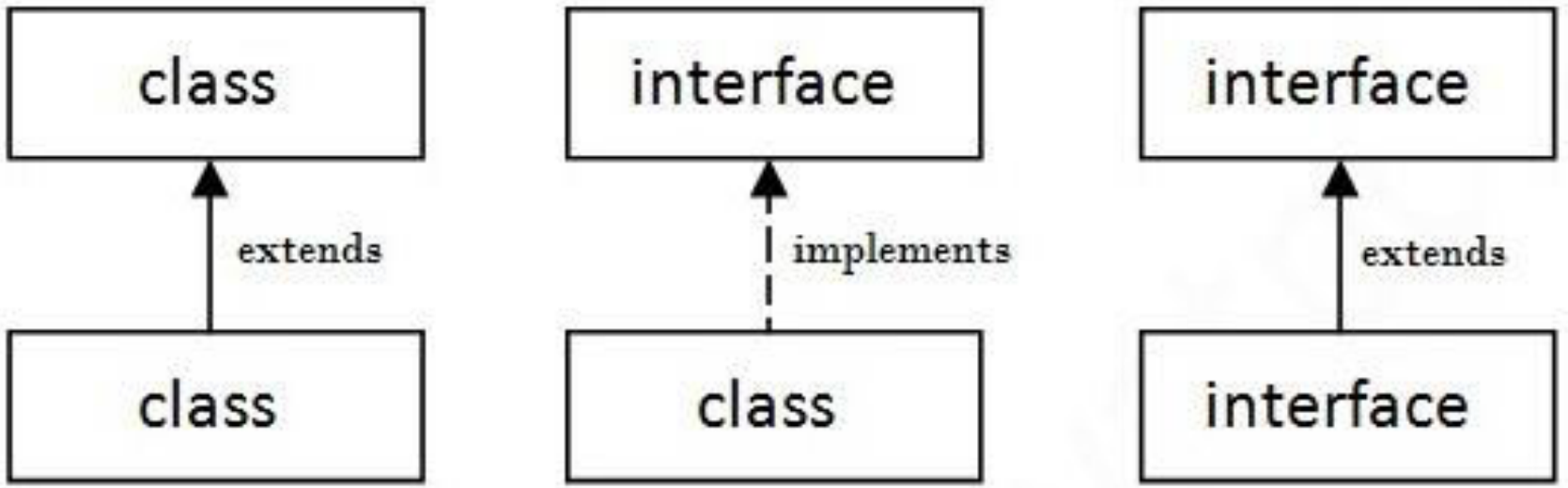
Internal addition by compiler in interface

- Interface fields are public, static and final by default, and methods are public and abstract.



Understanding 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**.



Java Interface Example 1

```
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();
 }
}
```

Output:
Hello

In this example, Printable interface has only one method, its implementation is provided in the A class.

Java Interface Example 2

```
/Interface declaration: by first user
interface Drawable{
void draw();
}
//Implementation: by second user
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
class Circle implements Drawable{
public void draw(){System.out.println("drawing circle");}
}
//Using interface: by third user
class TestInterface1{
public static void main(String args[]){
Drawable d=new Circle();//In real scenario, object is provided
 by method e.g. getDrawable()
d.draw();
}}
```

Output:

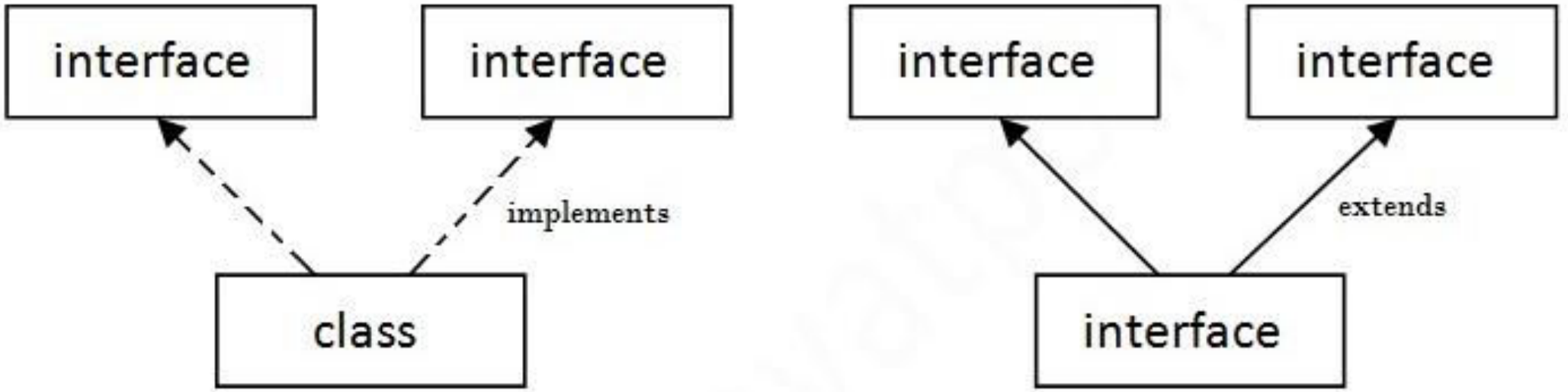
drawing circle

Here, Drawable interface has only one method. Its implementation is provided by Rectangle and Circle classes.

In real scenario, interface is defined by someone but implementation is provided by different implementation providers.

And, it is used by someone else. The implementation part is hidden by the user which uses the interface.

Multiple inheritance in Java by interface



- Multiple inheritance is not supported in case of class because of ambiguity. **But it is supported in case of interface.**
- If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.

Multiple inheritance in Java by interface

```
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();
 }
}
```

Output: Hello Welcome

Multiple inheritance in Java by interface

- Question: Multiple inheritance is not supported through class in java but it is possible by interface, why?
- Answer: As we have explained in the inheritance chapter, multiple inheritance is not supported in case of class because of ambiguity. But it is supported in case of interface because there is no ambiguity as implementation is provided by the implementation class. Example : Next slide.

Multiple inheritance in Java by interface

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

class TestInterface3 implements Printable, Showable{
public void print(){System.out.println("Hello");}
public static void main(String args[]){
TestInterface3 obj = new TestInterface3();
obj.print();
 }
}
```

Output:
Hello

As you can see in the above example, Printable and Showable interface have same methods but its implementation is provided by class TestTnterface1, so there is no ambiguity.

Interface inheritance

```
interface Printable{
void print();
}
interface Showable extends Printable{
void show();
}
class TestInterface4 implements Showable {
public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
TestInterface4 obj = new TestInterface4();
obj.print();
obj.show();
 }
}
```

Output:
Hello
Welcome

A class implements interface but one interface extends another interface .

Java 8 Default Method in Interface

```
interface Drawable{
void draw();
default void msg(){System.out.println("default method");}
}
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
class TestInterfaceDefault{
public static void main(String args[]){
Drawable d=new Rectangle();
d.draw();
d.msg();
}}
```

Output:

drawing rectangle

default method

Since Java 8, we can have method body in interface. But we need to make it default method.

Java 8 Static Method in Interface

```
interface Drawable{
void draw();
static int cube(int y){return y*y*y;}
}
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}

class TestInterfaceStatic{
public static void main(String args[]){
Drawable d=new Rectangle();
d.draw();
System.out.println(Drawable.cube(3));
}}
```

Output:

drawing rectangle
27

Nested Interface in Java

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

Thank you