GANPAT UNIVERSITY

U. V. PATEL COLLEGE OF ENGINEERING

2CEIT302

OBJECT ORIENTED PROGRAMMING

UNIT 7

ABSTRACTION

Prepared by: Prof. Y. J. Prajapati (Asst. Prof in IT Dept. , UVPCE)

Outline

- Abstract Class and method
- Interfaces (Defining Interface, Implementing an Interface)

Abstract Classes and Methods

Abstraction

- **Abstraction** is the process of hiding certain details and showing only essential information to the user.
- For example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery

There are two ways to achieve abstraction in java

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

Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

Abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

Abstract Class

- An abstract class is a class that cannot be instantiated (we cannot create objects of an abstract class).
- In Java, we use the abstract keyword to declare an abstract class.

```
abstract class Animal
{
//attributes and methods
}
```

- If we try to create objects of an abstract class, we will get a compilation error. For example,

```
Animal a1 = new Animal()
```

It will generate a compilation error:

```
Animal is abstract; cannot be instantiated
```

Though abstract classes cannot be instantiated, we can create subclasses from it. We can create objects of subclasses to access members of the abstract class.

Abstract Method

- We use the same keyword abstract to create abstract methods. An abstract method is declared without an implementation. For example,
  - abstract void makeSound();
- Here, makeSound() is an abstract method. The body of makeSound() is replaced by ;.
- It's important to note that, only an abstract class can contain abstract methods. If we include abstract methods inside a class that is not abstract, we will get an error.

- An abstract class can contain both abstract and non-abstract methods. Here's an example.

```
abstract class Animal
{
        public void displayInfo()
        {
                System.out.println("I am an animal.");
        }
abstract void makeSound();
}
```

- In the above example, we have created an abstract class Animal. It contains an abstract method makeSound() and a non-abstract method displayInfo().

Abstract Classes and Methods

- **Inheritance of Abstract Class**

  An abstract class cannot be instantiated. To access the members of an abstract class, we must inherit it. For example

```
abstract class Animal
{
    public void displayInfo()
    {
        System.out.println("I am an animal.");
    }
}

class Dog extends Animal
{
}
```

```
class Main
{
    public static void main(String[] args)
    {
        Dog d1 = new Dog();
        d1.displayInfo();
    }
}
```

Output:
I am an animal.

In the above example, we have created an abstract class Animal. We cannot create objects of Animal. To access the displayInfo() of Animal, we have inherited a subclass Dog of Animal.

Abstract Classes and Methods

- **Overriding of Abstract Methods**

```
abstract class Animal
    {
        abstract void makeSound();
        public void eat()
        {
            System.out.println("I can eat.");
        }
}
class Dog extends Animal
{
    public void makeSound()
    {
        System.out.println("Bark bark");
    }
}
```

```
class Main
{
    public static void main(String[] args)
    {
        Dog d1 = new Dog();
        d1.makeSound(); d1.eat();
    }
}
```

Output:
Bark bark.
I can eat

Points to Remember

- We use the abstract keyword to create abstract classes and methods.
- An abstract method doesn't have any implementation (method body).
- A class containing abstract methods should also be abstract.
- We cannot create objects of an abstract class.
- To implement features of an abstract class, we inherit subclasses from it and create objects of the subclass.
- A subclass must override all abstract methods of an abstract class. However, if the subclass is declared abstract, it's not mandatory to override abstract methods.
- We can access the static attributes and methods of an abstract class using the reference of the abstract class.

Interfaces

- An interface is a reference type in Java. It is similar to class.
- It is a collection of abstract methods.
- A class implements an interface, thereby inheriting the abstract methods of the interface.
- Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types.
- Method bodies exist only for default methods and static methods.
- It is used to achieve abstraction and multiple inheritance in Java.
- Java Interface also **represents the IS-A relationship**.
- It cannot be instantiated just like the abstract class.
- Since Java 8, we can have **default and static methods** in an interface.
- Since Java 9, we can have **private methods** in an interface.

Interfaces

An interface is similar to a class in the following ways :

- An interface can contain any number of methods.
- An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
- The byte code of an interface appears in a **.class** file.

Interface is different from a class in several ways

- All of the methods in an interface are abstract.
- An interface does not contain any constructors.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

Interfaces

Declaring Interfaces

- An interface is declared by using the interface keyword.
- It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.
- A class that implements an interface must implement all the methods declared in the interface.

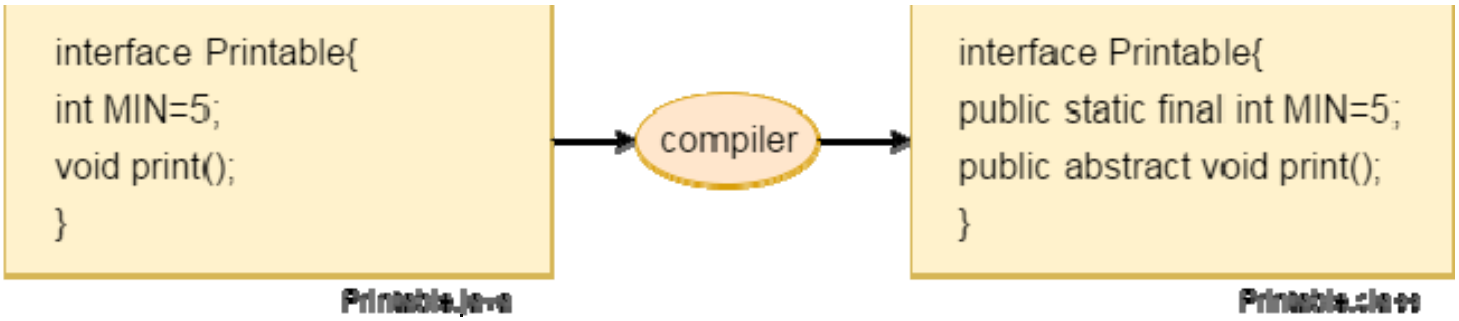
- The **interface** keyword is used to declare an interface.

```
public interface NameOfInterface
{
// Any number of final, static fields
// Any number of abstract method declarations
 }
```

```
interface Animal
{
public void eat();
public void travel();
}
```

Interfaces

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



Interfaces

- Implementing Interfaces

```
interface Animal
{
        public void eat();
        public void travel();
}
public class MammalInt implements Animal
{
        public void eat()
        {
                System.out.println("Mammal eats");
        }
        public void travel()
        {
                System.out.println("Mammal travels");
        }
public static void main(String args[])
{
        MammalInt m = new MammalInt(); m.eat();
        m.travel();
}
}
```

Output

```
Mammal eats
Mammal travels
```

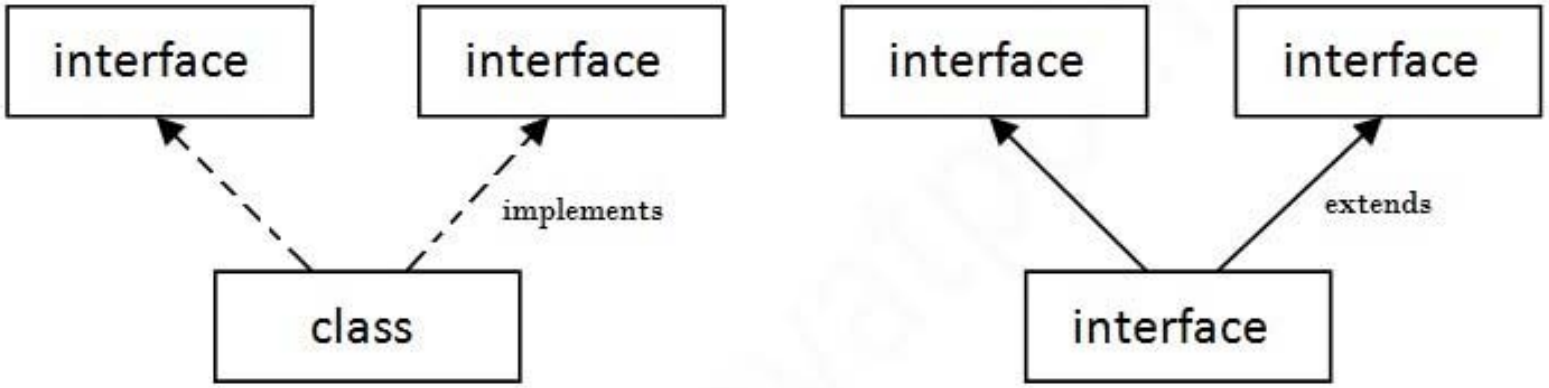
Examples

```
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

Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



Multiple Inheritance in Java

Multiple inheritance

```
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

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

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

Static Method in Interface

```
interface Drawable
{
    void draw();
    static int cube(int x)
    {return x*x*x;}
}
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

Difference between abstract class and interface

Abstract class	Interface
Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
An abstract class can be extended using keyword "extends".	An interface can be implemented using keyword "implement
A Java abstract class can have class members like private, protected, etc.	Members of a Java interface are public by default.


THANK YOU!
ANY QUESTIONS?