Programming Language II
CSE-215

Prof. Dr. Mohammad Abu Yousuf

yousuf@juniv.edu

Abstraction in Java

Abstract class 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.

- There are two ways to achieve abstraction in java
- Abstract class (0 to 100%)
- Interface (100%)

Abstract class in Java

- A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

- 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.

Abstract class in Java

- Points to Remember
  - 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.

Abstract class in Java

- **Example of abstract class**

  abstract class A{}

- **Example of abstract method**

  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.

```java
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();
}
}
```

```
Output:
running safely
```

- In this example, Shape is the abstract class, and its implementation is provided by the Rectangle and Circle classes.
- if you create the instance of Rectangle class, draw() method of Rectangle class will be invoked.

```java
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();
s.draw();
}
}
```

Output:

```
drawing circle
```

Abstract class having constructor, data member and methods

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

```java
/Example of an abstract class that has abstract and non-abstract methods
abstract class Bike{
  Bike(){System.out.println("bike is created");}
  abstract void run();
  void changeGear(){System.out.println("gear changed");}
}
//Creating a Child class which inherits Abstract class
class Honda extends Bike{
void run(){System.out.println("running safely..");}
}
//Creating a Test class which calls abstract and non-abstract methods
class TestAbstraction2{
public static void main(String args[]){
 Bike obj = new Honda();
 obj.run();
 obj.changeGear();
}
}
```

Output:

```
bike is created
running safely..
gear changed
```

- Rule 1: If there is an abstract method in a class, that class must be abstract.

```
class Bike12{
abstract void run();  // CT Error
}
```

- Rule 2: If you are extending an abstract class that has an abstract method, you must either provide the implementation of the method or make this class abstract.

Another real scenario of abstract class

- 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.

```
Output:
I am a
I am b
I am c
I am d
```

```java
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[]){
A a=new M();
a.a();
a.b();
a.c();
a.d();
}}
```

Difference between abstract class and interface

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.
But there are many differences between abstract class and interface.

Abstract class	Interface
1) 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.
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 provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.

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

```
//Creating interface that has 4 methods
interface A{
void a();//bydefault, public and abstract
void b();
void c();
void d();  }
//Creating abstract class that provides the implementation of one method of A interface
abstract class B implements A{
public void c(){System.out.println("I am C");}
}
//Creating subclass of abstract class, now we need to provide the implementation of rest of the methods
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");}
}
//Creating a test class that calls the methods of A interface
class Test5{
public static void main(String args[]){
A a=new M();
a.a();
a.b();
a.c();
a.d();
}}
```

Example of abstract class and interface in Java

```
Output:
I am a
I am b
I am c
I am d
```

Thank you