Object Oriented Programming

Instructor Name:

Lecture-15

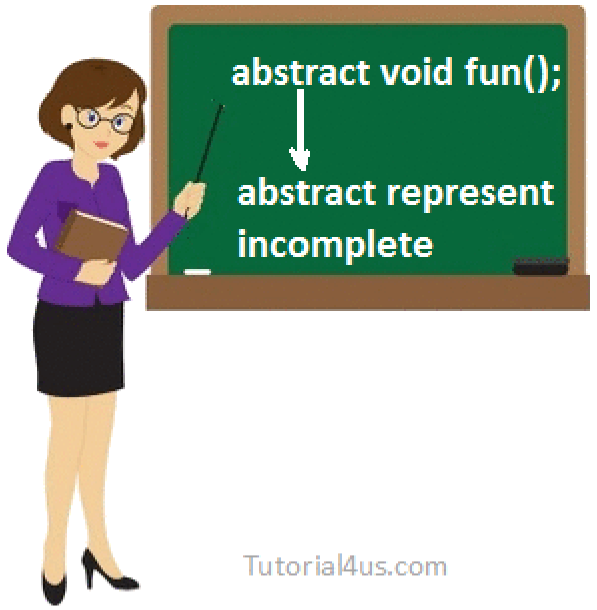
Today's Lecture

- **Abstract Methods**
- **Abstract Classes**
- **Interfaces**

Abstract Classes

What is Abstract Class?

- A class that is declared with abstract keyword, is known as abstract class.
- An abstract class is one which is containing some defined method and some undefined method.
- In java programming undefined methods are known as un-Implemented or abstract method.



Abstract Classes

Syntax of Abstract Class

```
abstract class className
{
  ......
}


abstract class A
{
  .....
}
```

Abstract Methods

What is an Abstract Method?

- An abstract method is one which contains only declaration or prototype but it never contains body or definition.
- In order to make any undefined method as abstract, the declaration must be predefined by abstract keyword.

Syntax

```
abstract  ReturnType methodName(List of formal parameter)
```

Examples

```
abstract void sum();

abstract void diff(int, int);
```

Abstract Class & Methods

Example Abstract Class & Methods

```
abstract class Vachile {
  abstract void speed(); // abstract method
}
class Bike extends Vachile {
  void speed() {
   System.out.println("Speed limit is 40 km/hr..");
  }
  public static void main(String args[]) {
   Vachile obj = new Bike();
   obj.speed();
  }
}
```

Abstract Class & Methods

Important Points About Abstract Classes

Abstract class of java always contains common features.

- **Every abstract class participate in inheritance.**
- **Abstract classes definitions should not be made as final because abstract classes always participate in inheritance classes.**
- **An object of abstract class can not be created directly but it can be created indirectly.**
- **All the abstract classes of java makes use of polymorphism along with method overriding for business logic development and makes use of dynamic binding for execution logic.**

Abstract Class & Methods

Advantage of Abstract Classes

- **Less memory space for the application**
- **Less execution time**
- **More performance**

Abstract Class & Methods

When to Use Abstract Classes & Methods?

- Abstract methods are usually declared where two or more subclasses are expected to fulfill a similar role in different ways through different implementations
- These subclasses extend the same Abstract class and provide different implementations for the abstract methods
- Use abstract classes to define broad types of behaviors at the top of an object-oriented programming class hierarchy, and use its subclasses to provide implementation details of the abstract class.

Interface

What is an Interface?

- Interface is similar to class which is collection of public static final variables (constants) and abstract methods.
- The interface is a mechanism to achieve fully abstraction in java.
- There can be only abstract methods in the interface.
- It is used to achieve fully abstraction and multiple inheritance in Java.

Properties of Interface

- It is implicitly abstract. So no need to use the abstract keyword
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.
- All the data members of interface are implicitly public static final.

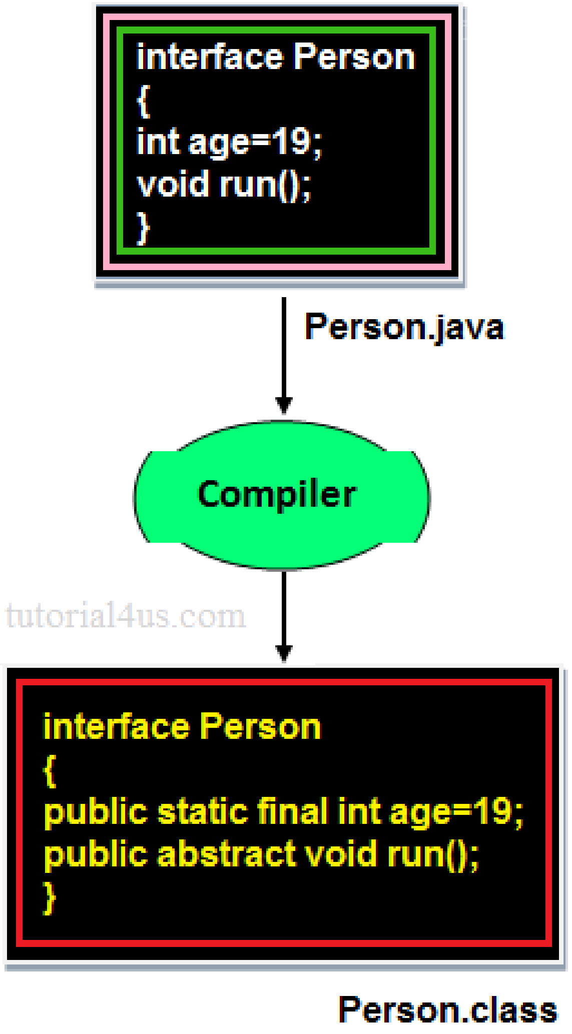
Interface

How Interface different from Class?

- **You can not instantiate an interface.**
- **It does not contain any constructors.**
- **All methods in an interface are abstract.**
- **Interface can not contain instance fields. Interface only contains public static final variables.**
- **Interface is can not extended by a class; it is implemented by a class.**
- **Interface can extend multiple interfaces. It means interface support multiple inheritance**

Interface

Behaviour of Compiler with Interface Program



```
interface Person
{
int age=19;
void run();
}
```

```
interface Person
{
public static final int age=19;
public abstract void run();
}
```

Abstract vs Interface

When use Abstract & when Interface

- **If we do not know about any things about implementation just we have requirement specification then we should be go for Interface**
- **If we are talking about implementation but not completely (partially implemented) then we should be go for abstract**

Why do we use Interface?

Reason 1

- To reveal an object's programming interface (functionality of the object) without revealing its implementation

  – This is the concept of encapsulation

  – The implementation can change without affecting the caller of the interface

- The caller does not need the implementation at the compile time. It needs only the interface at the compile time

- During runtime, actual object instance is associated with the interface type.

Why do we use Interface?

Reason 2

- Interfaces are used in unrelated classes but have implement similar methods (behaviors)
  - – One class is to a sub-class of another
- Example:
- – Class Line and class MyInteger
- They are not related through inheritance
- You want both to implement comparison methods
  - `checkIsGreater(Object x, Object y)`
  - `checkIsLess(Object x, Object y)`
  - `checkIsEqual(Object x, Object y)`

Why do we use Interface?

Reason 3

- To model multiple inheritance
- A class can implement multiple interfaces while it can extend only one class

Interface

Interface as Type

- When you define a new interface, you are defining a new reference type.
- You can use interface names anywhere you can use any other type name.
- If you define a reference variable whose type is an interface, any object you assign to it must be an instance of a class that implements the interface
- Let's say Person class implements PersonInterface interface
- You can do

```
Person p1 = new Person();

PersonInterface pi1 = p1;

PersonInterface pi2 = new Person();
```

Problem Rewriting an Existing Interface

- Consider an interface that you have developed called DoIt:

```
public interface DoIt {
   void doSomething(int i, double x);
   int doSomethingElse(String s);
}
```

- Suppose that, at a later time, you want to add a third method to DoIt

```
public interface DoIt {
   void doSomething(int i, double x);
   int doSomethingElse(String s);
   boolean didItWork(int i, double x, String s);
}
```

Solution of Rewriting an Existing Interface

- If you make this change, all classes that implement the old DoItinterface will break because they don't implement all methods of the the interface anymore
- Solution:
- Create more interfaces later● For example, you could create a DoItPlus interface thatbextends DoIt:

```
public interface DoItPlus extends DoIt {
    boolean didItWork(int i, double x, String s);
}
```

- Now users of your code can choose to continue to use the old interface or to upgrade to the new interface

R@CAP