Agenda

- What is an Abstract method and an Abstract class?
- What is Interface?
- Why Interface?
- Interface as a Type
- Interface vs. Class
- Defining an Interface
- Implementing an Interface
- Implementing multiple Interface's
- Inheritance among Interface's
- Interface and Polymorphism
- Rewriting an Interface

Abstract Methods

- Methods that do not have implementation (body)
- To create an abstract method, just write the method declaration without the body and use the abstract keyword
  - No { }
- For example

```
// Note that there is no body
public abstract void someMethod();
```

Abstract Class

- An abstract class is a class that contains one or more abstract methods
- An abstract class cannot instantiated

  ```
  // You will get a compile error on the following code
  MyAbstractClass a1 = new MyAbstractClass();
  ```
- Another class (Concrete class) has to provide implementation of abstract methods
  - Concrete class has to implement all abstract methods of the abstract class in order to be used for instantiation
  - Concrete class uses extends keyword

Sample Abstract Class

```
public abstract class LivingThing {
    public void breath(){
    System.out.println("Living Thing breathing...");
    }
    public void eat(){
    System.out.println("Living Thing eating...");
    }
    /**
    * Abstract method walk()
    * We want this method to be implemented by a
    * Concrete class.
    */
    public abstract void walk();
}
```

Extending an Abstract Class

- When a concrete class extends the LivingThing abstract class, it must implement the abstract method walk(), or else, that subclass will also become an abstract class, and therefore cannot be instantiated.
- For example,

```
public class Human extends LivingThing {
    public void walk(){
    System.out.println("Human walks...");
    }
}
```

When to use Abstract Methods & Abstract Class?

- 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.

What is the understanding of Abstract Class?

- Classes and methods can be declared as abstract.
- Abstract class can extend only one Class.
- If a Class is declared as abstract , no instance of that class can be created.
- If a method is declared as abstract, the sub class gives the implementation of that class.
- Even if a single method is declared as abstract in a Class , the class itself can be declared as abstract.
- Abstract class have at least one abstract method and others may be concrete.
- In abstract Class the keyword abstract must be used for method.
- Abstract classes have sub classes.
- Combination of modifiers Final and Abstract is illegal in java.

What is an Interface?

- It defines a standard and public way of specifying the behavior of classes
  – Defines a contract
- All methods of an interface are abstract methods
  – Defines the signatures of a set of methods, without the body (implementation of the methods)
- A concrete class must implement the interface (all the abstract methods of the Interface)
- It allows classes, regardless of their locations in the class hierarchy, to implement common behaviors

Example:

Note that Interface contains just set of method signatures without any implementations.

No need to say abstract modifier for each method since it assumed.

```
public interface Relation {
    public boolean isGreater( Object a, Object b);
    public boolean isLess( Object a, Object b);
    public boolean isEqual( Object a, Object b);
}
```

Example 2: OperatorCar Interface

```
public interface OperateCar {
   // constant declarations, if any method signatures
    int turn(Direction direction,
    double radius, double startSpeed, double endSpeed);
    int changeLanes(Direction direction, double startSpeed,
    double endSpeed);
    int signalTurn(Direction direction, boolean signalOn);
    int getRadarFront(double distanceToCar,
    double speedOfCar);
    int getRadarRear(double distanceToCar,double speedOfCar);
    ......
    // more method signatures
}
```

Why do we use Interfaces?
Reason #1

- To reveal an object's programming interface (functionality of the object) without revealing its implementation
  - – 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 Interfaces?
Reason #2

- To have unrelated classes implement similar methods (behaviors)
  - One class is not 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)
  - Define Comparison interface which has the three abstract methods above

Why do we use Interfaces?

Reason #3

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

Defining Interfaces

● To define an interface, we write:

```
public interface [InterfaceName] {
        //some methods without the body
}
```

As an example, let's create an interface that defines relationships between two objects according to the “natural order” of the objects.

```
public interface Relation {
    public boolean isGreater( Object a, Object b);
    public boolean isLess( Object a, Object b);
    public boolean isEqual( Object a, Object b);
}
```

Implementing Interfaces

● To create a concrete class that implements an interface, use the **implements** keyword.

```
/**
* Line class implements Relation interface
*/
public class Line implements Relation {
    private double x1;
    private double x2;
    private double y1;
    private double y2;
    public Line(double x1, double x2, double y1, double y2){
        this.x1 = x1;
        this.x2 = x2;
        this.y1 = y1;
        this.y2 = y2;
    }
// More code follows
public double getLength(){
        double length = Math.sqrt((x2-x1)*(x2-x1) +(y2-y1)* (y2-y1));
        return length;
}
public boolean isGreater( Object a, Object b){
        double aLen = ((Line)a).getLength();
        double bLen = ((Line)b).getLength();
        return (aLen > bLen);
}
public boolean isLess( Object a, Object b){
        double aLen = ((Line)a).getLength();
        double bLen = ((Line)b).getLength();
        return (aLen < bLen);
}
public boolean isEqual( Object a, Object b){
        double aLen = ((Line)a).getLength();
        double bLen = ((Line)b).getLength();
        return (aLen == bLen);
     }
}
```

When your class tries to implement an interface,always make sure that you implement all the methods of that interface, or else, you would

encounter this error,

```
Line.java:4: Line is not abstract and does not override abstract method
  isGreater(java.lang.Object,java.lang.Object) in Relation public class Line
implements Relation
               ^
  1 error
```

Note:

- Implementing class can have its own methods
- Implementing class extend a single super class or abstract class

Some of important points related to Interface

- A class can implement more than one Interface.
- An Interface can extend one or more interfaces, by using the keyword extends.
- All the data members in the interface are public, static and Final by default.
- An Interface method can have only Public, default and Abstract modifiers.
- An Interface is loaded in memory only when it is needed for the first time.
- A Class, which implements an Interface, needs to provide the implementation of all the methods in that Interface.
- If the Implementation for all the methods declared in the Interface are not provided , the class itself has to declare abstract, other wise the Class will not compile.
- If a class Implements two interface and both the Interfaces have identical method declaration, it is totally valid.
- If a class implements tow interfaces both have identical method name and argument list, but different return types, the code will not compile.

- An Interface can’t be instantiated. Intf Are designed to support dynamic method resolution at run time.
- An interface can not be native, static, synchronize, final, protected or private.
- The Interface fields can’t be Private or Protected.
- A Transient variables and Volatile variables can not be members of Interface.
- The extends keyword should not used after the Implements keyword, the Extends must always come before the Implements keyword.
- A top level Interface can not be declared as static or final.
- If an Interface species an exception list for a method, then the class implementing the interface need not declare the method with the exception list.
- If an Interface can’t specify an exception list for a method, the class can’t throw an exception.
- If an Interface does not specify the exception list for a method, he class can not throw any exception list.

- **Implementing Multiple Interfaces**

  A concrete class can only extend one super class, but it can implement multiple Interfaces

  – The Java programming language does not permit multiple inheritance (inheritance is discussed later in this lesson), but interfaces provide an alternative.

- All abstract methods of all interfaces have to be implemented by the concrete class

Example: Implementing Multiple Interfaces

- A concrete class extends one super class but multiple Interfaces:

```
public class ComputerScienceStudent
extends Student implements PersonInterface,
 AnotherInterface,Thirdinterface{
  // All abstract methods of all interfaces
  // need to be implemented.
}
```