JAVA: INTERFACE IN DEPTH.

By Saurav Saxena.

1. What is Interface.
->Interface is something which helps 2 system to interact with each other. without one system has to know the details of other.
->Or In simple term I can say it helps to achieve ABSTRACTION.
2. Interface declaration Consist Of.
->Modifiers.
->"interface" keyword
->Interface Name
->Comma separated list of parent interfaces.
->Body

> *Only PUBLIC and Default Modifiers are allowed [PROTECTED AND PRIVATE ARE NOT ALLOWED]*

```
public interface Bird{
  public void fly();
}
interface Animal{
  public void run();
}
public interface LivingThing extends Bird,Animal{
  public void eat();
}
```

3. Why we need Interface.
3.1. Abstraction[100%]:
->Using Interface, We can achieve full Abstraction means, we can define WHAT class must do, but not How it will do.

```
public class Eagle implements Bird{
   @Override
   public void fly(){
     System.out.println("Eagle Is Flying");
   }
}
```

3.2 . Polymorphism:
->Interface can used to as a Data Type.
->We can not created the Object of an interface, but it can hold the reference of all the classes which implements it , and at runtime, it decided which method needs to be invoked.

```
public class Men implements Bird{
   @Override
   public void fly(){
     System.out.println("Men can not fly");
   }
}
public class Main{
  public static void main(String[] args){
    Bird birdObjEagle = new Eagle();
    Bird birdObjMen = new Men();
    birdObjMen.fly();
    birdObjEagle.fly();
  }
}
output:
Eagle is flying
Men can not fly
```

3.3 . Multiple Inheritance:

->In java Multiple inheritance is possible only through interface only.

3.4. Diamond Problem.

->

```
public class WaterAnimal{
  public boolean canBreathe(){
    return true;
  }
}
public class Animal{
  public boolean canBreath(){
    return true;
  }
}
//compile time error
public class Crocodile extends LandAnimal,WaterAnimal{

}
public class Main{
  public static void main(String[] args){
    Crocodile obj = new Crocodile();
    //obj get's confused which method to call whether it is WaterAnimal or LandAnimal
    obj.canBreath();
  }
}
```

Multiple Inheritance Support through Interfaces.

```
public interface LandAnimal{
  public boolean canBreadth();
}
public interface WaterAnimal{
  public boolean canBreadth();
}
public class Crocodile implements LandAnimal,WaterAnimal{
  @Override
  public boolean canBreadth(){
    return true;
  }
}
public class Main{
  public static void main(String[] args){
    Crocodile obj = new Crocodile();
    Syatem.out.println(obj.canBreath());
  }
}
output:
true
```

4. Methods In Interface.
->All Methods are implicit public only.
->Method can not be declared as final.
->interface contain only signature and declaration of body.

5. Fields In Interfaces.
->Fields are public, static and final implicitly [CONSTANTS]
->You can not make field private or protected.

```
public interface Bird{
  //Both are equal
  int MAX_HEIGHT_IN_FEET = 2000;
  public static final int MAX_HEIGHT_IN_FEET = 2000;
}
```

6. Interfaces Implementation and Rules.
->Overriding method can not have more restrict access modifiers .
->Concrete class must override all the methods declared int the interface.
->Abstract classes are not forced to override all the methods.
->A class can implement from multiple interfaces.

```
public class Eagle implements Bird{
  //Commple time error
  //you cannot restrict the access
  //it has to be public
modifier's
  @Override
  protected void fly(){
    //do something
  }
}
```

EXAMPLE of Abstract class implementation of interface.

```
  public interface{
    public void canFly();
    public void noOfWings();
  }
  //abstract class not forced to be Override //all methods of interface.
  //there is no implementation for //noOfWings()
  public abstract class Eagle implements Bird{
    @Override
    public void canFly(){
      //Implementation goes here.
    }
    public abstract void breakLength();
  }
  //In concrete class if any abstract method
  //which needs implementation, need to
  //implement here forced.
  public class WhiteEagle extends Eagle{
    @Override
    public void noOfWings(){
      //do something here
    }

    @Override
    public void beakLength(){
    }
  }
```

7. Nested Interfaces.
->Nested Interface declared within another interface.
-> Nested Interface declared within a class.
->Generally it's used to group, logical related interfaced. And nested interface.
RULES:
->Anested interface declared within an interface must be public.
->A nested interface declared within a class can have any access modifiers.
->When you implement outer interface, inner interface implementation is not required and vice versa.

```
  //In interface nested interface should be public only.
  //
  public interface Bird{
    public void canFly();
    public interface NonFlyingBird{
      public void canRun();
    }
  }
  //here there is no need for implementation
  //of Nested interface
  public class Eagle implements Bird{
    @Override
    public void canFly(){
    }
  }
  //It's only need to provide implementation
  //for Nested Interface
  public class Eagle implements Bird.NonFlyingBird{
    @Override
    public void canRun(){
    }
  }
  public class Main{
    public static void main(String[] args){
      Bird.NonFlyingBird obj = new Eagle();
      obj.canRun();
    }
  }
  public class Eagle implement Bird, Bird.NonFlyingBird{
    @Override
    public void canRun(){
    }
    @Override
    public void canFly(){
    }
  }
```

Example of nested Interface IN class itself.

->

```
  //class have member's and member's can be
  //public, protected and private
  public class Bird{
    protected interface NonFlyingBird{
      public void canRun();
    }
  }
  public class Eagle implements Bird.NonFlyingBird{
    @Override
    public void canRun(){
    }
  }
```

8. Difference Between Abstract classes and interfaces.

ABSTRACT CLASS:

->Keyword used here is "abstract"
->Child classes needs to use keyword "extend"
->It can have both abstract and non abstract method.
->It can extend from Another class and multiple interfaces.
->Variables can be static , non-static, final, non-final etc.
->Variables and methods can be private, protected, public and default.
->Multiple Inheritance is not supported.
->It can provide the implementation of the interface.
->It can have Constructor
->To declare abstract method , we have to use keyword
"abstract" keyword and it can be protected public and private too.

INTERFACES:
->Keyword used here is "interface"
->Child classes needs to use keyword "implements"
->It can have only abstract method.
    ->[from Java8 onwards it can have default, static, and private methods too, where we can provide implementation.]
->It can only extend from Another Interfaces.
->Variables are by default CONSTANT.
->Variables and methods are by default public.
[in java 9 private method is supported]
->Multiple Inheritance supported with this in java.
->it can not provide implementation of any other interface and abstract class.
->It can not have constructor.
->No need for any keyword to make method abstract, and by defaults it is public.

9. [JAVA 8 AND JAVA 9 Features].
9.1. Default Method[JAVA8].
->Before Java 8, interface can have only Abstract method. And all child classes has to provide abstract method implementation.
->When we are introducing some new method in interface . All child classes which implements that interface needs to provide new method implementation
although it's core functionality will be the same.

```
public interface Bird{
  public void canFly();
  //Introducing new Method their
  //core functionality will be the same.
  public int getMinimumFlyHeight();
}
```

```
public class Eagle implements Bird{
  @Override
  public void canFly(){
  }
  @Override
  public int getMinimumFlyHeight(){
    return 100;
  }
}
public class Sparrow implements Bird{
  @Override
  public void canFly(){
  }
  @Override
  public int getMinimumFlyHeight(){
    return 100;
  }
}
```

SOLUTION:

```
public interface Bird{
  public void canFly();
  default int getMinimumFlyHeight(){
    return 100;
  }
}
```

9.2 Why default method was introduced.
->To add functionality in existing Leagcy interface we need to use default method.
->Example stream() method in Collection interface.

10. Default and Multiple Inheritance, How to handle.
->

```
public interface Bird {
  default boolean canBreath(){
    return true;
  }
}
public interface LivingThing{
  default boolean canBreath(){
    return true;
  }
}
//Multiple Ineritance with same name //default method
//compile time error
public class Eagle implements Bird,LivingThing{
}
public class Eagle implements Bird,LivingThing{
  public boolean canBreathe(){
    return true;
  }
}
```

11 . Default method when an interface extend another interface.

->There are 3 way.

11.1. FIRST WAY.

```
  public interface LivingThing{
    default boolean canBreadth(){
      return true;
    }
  }
  public interface Bird extends LivingThing{

  }
  public class Eagle implements Bird{
  }
  public class Main{
    public static void main(String[] args){
      Eagle eagleObj = new Eagle();
      eagleObj.canBreadth();
    }
  }
```

11.2. Second Way.

->

```
  public interface LivingThing{
    default boolean canBreadth(){
      return true;
    }
  }
  public interface Bird extends LivingThing{
    boolean canBreadth();
  }
  //comile time error
  //we need to provide implementation for //abstract method.
  public class Eagle implements Bird{

  }
  public class Eagle implements Bird{
    @Override
    public boolean canBreadth(){
       return true;
    }
  }
```

11.3 Third Way.

->

```
public interface LivingThing{
  default boolean canBreadth(){
    return true;
  }
}
public interface Bird extends LivingThing{
  @Overriding
  default boolean canBreadth(){
    boolean canBreadthOrNot = LivingThing.super.canBreadth();
    return canBreadthOrNot;
  }
}
```

12. Static Method In [JAVA 8].
->we can provide implementation of the method in interface.
->But it can not be overridden by classes which implement the interface.
->We can access it using Interface name itself.
->It's by default public.

```
public interface Bird{
  static boolean canBreadth(){
    return true;
  }
}
public class Eagle implements Bird{
  public void digestiveSystemTestMethod(){
    if(Bird.canBreadth()){

    }
  }
}
```

> *Default method can be overrides but not STATIC ONE.*

13 . Private method and private static method JAVA 9.
->We can provide the implementation of method but as private access modifiers in interface.
->It brings more readability of the code. For Example it multiple default method share some code, than this can help.
->It can defined as static and not-static.
->From Static method, we can call only private static interface method.
->Private static method, can be called from both static and non-static method.
->Private static method can not be abstract. means we have to provide definition.
->private method or private static method It can be used inside of the particular interface only.

```
public interface Bird{
  //this is equivalent to
  //public abstract void canFly();
  void canFly();
  public default void minimumFlyingHeight(){
    myStaticPublicMethod();//calling static method
    myPrivateMethod();//calling Private
    myPrivateStaticMethod()// calling
  }

  static void myStaticPublicMethod(){
    //from static we can call other static method only.
    myPrivateStaticMethod();
    priavte void myPrivateMethod(){
      //implementation here
    }
    private static void myPrivateStaticMethod(){
       //implementation
    }
  }
}
```