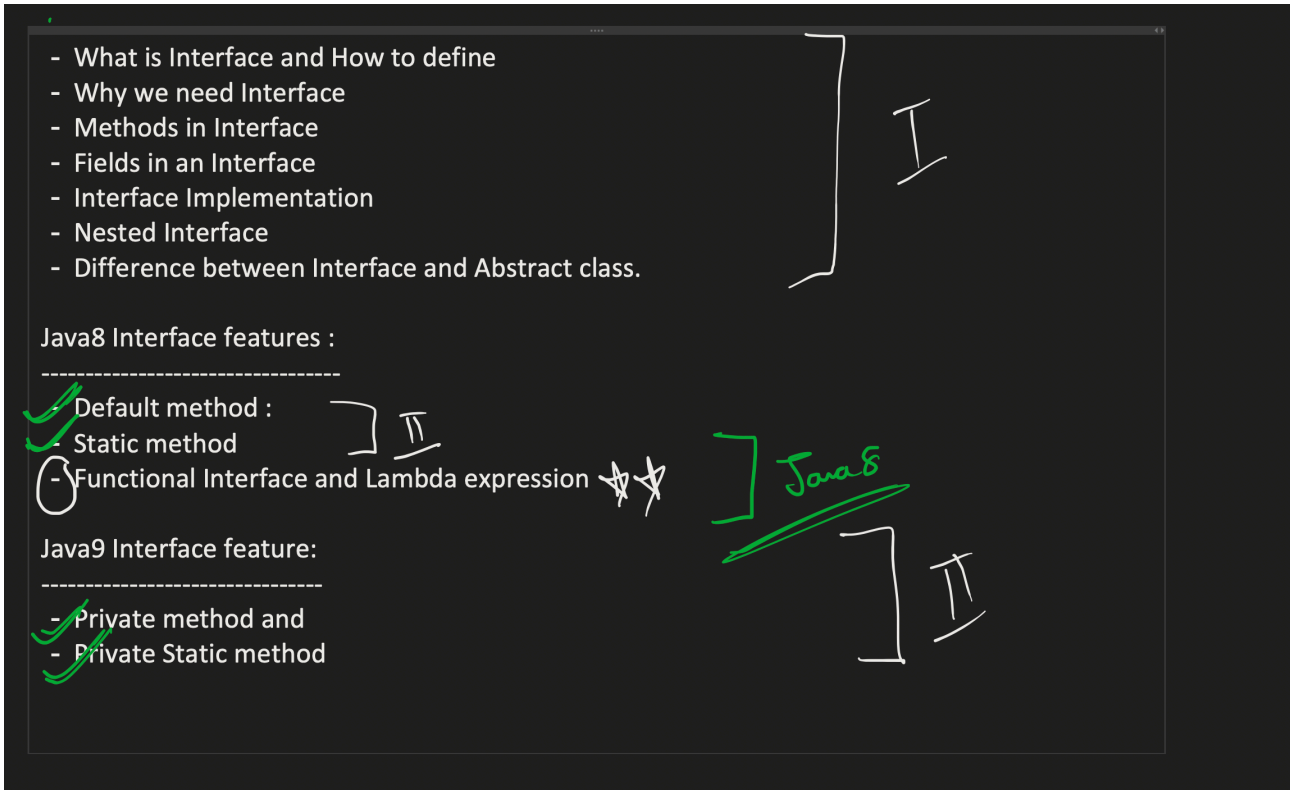
Java: Interface in Depth

"Concept && Coding" YT Video Notes



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.

How to define the interface?

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 {

    no usages
    public void fly();
}
```

```
interface Bird {

    no usages
    public void fly();
}
```

Comma separated list of parent interfaces (it can extend from ***Class***) Example:

```
public interface NonFlyingBirds extends Bird, LivingThings{

    no usages
    public void canRun();
}
```

Why we need Interface?

1. **Abstraction**:

Using interface, we can achieve full Abstraction means, we can define WHAT class must do, but not HOW it will do

```
public interface Bird {

    no usages
    public void fly();
}
```

```
public class Eagle implements Bird{
    no usages
    @Override
    public void fly() {
        //the complex process of flying take place here
    }
}
```

2. Polymorphism:

- Interface can be used as a Data Type.

- we can not create the object of an interface, but it can hold the reference of all the classes which implements it. And at runtime, it decide which method need to be invoked.

```
public interface Bird {

    2 usages  2 implementations
    public void fly();
}
```

```
public class Main {

    public static void main(String args[]){

        Bird birdObject1 = new Eagle();
        Bird birdObject2 = new Hen();

        birdObject1.fly();
        birdObject2.fly();
    }
}
```

```
public class Eagle implements Bird{

    2 usages
    @Override
    public void fly() {
        System.out.println("Eagle Fly Implementation");
    }
}
```

```
public class Hen implements Bird{

    2 usages
    @Override
    public void fly() {
        System.out.println("Hen Fly Implementation");
    }
}
```

3. Multiple Inheritance:

In Java Multiple inheritance is possible only through Interface only.

Diamond problem:

```java
public class Main {

    public static void main(String args[]){

        Crocodile obj = new Crocodile();
        obj.canBreathe();
    }
}
```

```java
public class WaterAnimal {

    no usages
    public boolean canBreathe(){
        return true;
    }
}
```

```java
public class LandAnimal {

    1 usage
    public boolean canBreathe(){
        return true;
    }
}
```

```java
public class Crocodile extends LandAnimal, WaterAnimal{
}
```

```java
public class Main {

    public static void main(String args[]){

        Crocodile obj = new Crocodile();
        obj.canBreathe();
    }
}
```

```java
public interface LandAnimal {

    1 usage  1 implementation
    public boolean canBreathe();
}
```

```java
public interface WaterAnimal {

    1 usage
    public boolean canBreathe();
}
```

```java
public class Crocodile implements LandAnimal, WaterAnimal{

    2 usages
    @Override
    public boolean canBreathe() {
        return true;
    }
}
```

Methods in Interface:

- All methods are implicit public only.
- Method can not be declared as final.

```java
public interface Bird {

    1 usage  1 implementation
    void fly();
    no usages
    public void hasBeak();
}
```

Fields in Interface:

- Fields are public, static and final implicitly (***CONSTANTS***)
- You can not make field private or protected.

```java
public interface Bird {
    no usages
    int MAX_HEIGHT_IN_FEET = 2000;
}
```

==

```java
public interface Bird {
    no usages
    public static final int MAX_HEIGHT_IN_FEET = 2000;
}
```

Interface Implementation:

- Overriding method can not have more restrict access specifiers.
- Concrete class must override all the methods declared in the interface.
- Abstract classes are not forced to override all the methods.
- A class can implement from multiple interfaces.

```java
public interface Bird {

    2 usages   2 implementations
    public void fly();
}
```

```java
public class Eagle implements Bird{

    2 usages
    @Override
    public void fly() {
        System.out.println("Eagle Fly Implementation");
    }
}
```

```java
public class Eagle implements Bird{
    1 usage
    @Override
    protected void fly() {

        //do something
    }
}
```

Example of Abstract class implementation of Interface:

```java
public interface Bird {
    no usages   1 implementation
    public void canFly();
    no usages   1 implementation
    public void noOfLegs();
}
```

```java
public abstract class Eagle implements Bird{
    no usages
    @Override
    public void canFly() {
        //Implementation goes here
    }

    no usages
    public abstract void beakLength();
}
```

```java
public class WhiteEagle extends Eagle{

    no usages
    @Override
    public void noOfLegs() {
        //implement interface method
    }

    no usages
    @Override
    public void beakLength() {
        //implementing abstract class method
    }
}
```

Nested Interface:

- Nested Interface declared withing another Interface.
- Nested Interface declared within a Class.

Generally its is used to group, logical related interfaced. And Nested interface

Rules:

- A nested interface declared withing an interface must be public.
- A nested interface declared within a class can have any access modifier.
- When you implement outer interface, inner interface implementation is not required and vice versa.

```java
public interface Bird {

    no usages   1 implementation
    public void canFly();

    no usages
    public interface NonFlyingBird{

        no usages
        public void canRun();
    }
}
```

```java
public  class Eagle implements Bird{
    no usages
    @Override
    public void canFly() {
        //Implementation goes here
    }
}
```

```java
public  class Eagle implements Bird.NonFlyingBird{

    no usages
    @Override
    public void canRun() {

    }
}
```

```java
public class Main {

    public static void main(String args[]){

        Bird.NonFlyingBird obj = new Eagle();
        obj.canRun();
    }
}
```

```java
public  class Eagle implements Bird, Bird.NonFlyingBird{

    1 usage
    @Override
    public void canRun() {

    }

    no usages
    @Override
    public void canFly() {

    }
}
```

```java
public class Bird {

    1 usage
    protected interface NonFlyingBird{
        no usages
        public void canRun();
    }

}
```

```java
public  class Eagle implements Bird.NonFlyingBird{

    no usages
    @Override
    public void canRun() {

    }
}
```

Interface Vs Abstract Class:

S.No.	Abstract Class	Interface
1.	Keyword used here is "abstract"	Keyword used here is "interface"
2.	Child Classes need to use keyword "extends"	Child Classes need to use keyword "implements"
3.	It can have both abstract and non abstract method	It can have only Abstract method (from Java8 onwards it can have default ,static and private method too, where we can provide implementation)
4.	It can Extend from Another class and multiple interfaces	It can only extends from other interfaces
5.	Variables can be static, non static, final , non final etc.	Variable are by default CONSTANTS
6.	Variables and Methods can be private, protected, public, default.	Variable and Methods are by default public (in Java9 , private method is supported)
7.	Multiple Inheritance is not Supported	Multiple Inheritance supported with this in Java
8.	It can provide the implementation of the interface	It can not provide implementation of any other interface or abstract class.
9.	It can have Constructor	It can not have Constructor
10.	To declare the method abstract, we have to use "abstract" keyword and it can be protected, public, default.	No need for any keyword to make method abstract. And be default its public.

JAVA 8 and 9 FEATURES:

1. **Default Method(Java8):**

- Before Java8, interface can have only Abstract method. And all child classes has to provide abstract method implementation.

```
public interface Bird {

    public void canFly();
}
```

```
public class Eagle implements Bird{

    @Override
    public void canFly() {
        //eagle fly implementation
    }
}
```

```
public class Sparrow implements Bird{

    @Override
    public void canFly() {
        //sparrow fly logic
    }
}
```

Any new method added in interface, means need to change in all its implementation

```
public interface Bird {

    public void canFly();
    public int getMinimumFlyHeight();
}
```

```
public class Eagle implements Bird{

    @Override
    public void canFly() {
        //eagle fly implementation
    }

    @Override
    public int getMinimumFlyHeight() {
        return 100;
    }
}
```

```
public class Sparrow implements Bird{

    @Override
    public void canFly() {
        //sparrow fly logic
    }

    @Override
    public int getMinimumFlyHeight() {
        return 100;
    }
}
```

Using Default Method:

```java
public class Main {

    public static void main(String args[]){

        Eagle eagleObj = new Eagle();
        eagleObj.getMinimumFlyHeight();
    }
}
```

```java
public interface Bird {

    public void canFly();

    default int getMinimumFlyHeight(){
        return 100;
    }
}
```

```java
public class Eagle implements Bird{

    @Override
    public void canFly() {
        //eagle fly implementation
    }
}
```

```java
public class Sparrow implements Bird{

    @Override
    public void canFly() {
        //sparrow fly logic
    }
}
```

Why Default method was introduced:

- To add functionality in existing Legacy Interface we need to use Default method. Example *steam()* method in Collection.

Default and Multiple Inheritance, how to handle:

```java
public interface Bird {

    default boolean canBreathe(){
        return true;
    }
}
```

```java
public interface LivingThing {

    default boolean canBreathe(){
        return true;
    }
}
```

Eagle ob = new Eagle();
ob. canBreathe()

```java
public class Eagle implements Bird, LivingThing{

}
```

```java
public class Eagle implements Bird, LivingThing{

    public boolean canBreathe(){
        return true;
    }
}
```

Interface (default)
1,
Interface.

How to extend interface, that contains Default Method:

1st Way:

```java
public interface LivingThing {

    default boolean canBreathe(){
        return true;
    }
}
```

Parent

```java
public interface Bird extends LivingThing{

}
```

Child

```java
public class Eagle implements Bird{

}
```

```java
public class Main {

    public static void main(String args[]){

        Eagle eagleObj = new Eagle();
        eagleObj.canBreathe();

    }
}
```

2nd Way:

```java
public interface LivingThing {

    default boolean canBreathe(){
        return true;
    }
}
```

Parent

```java
public interface Bird extends LivingThing{

    boolean canBreathe();
}
```

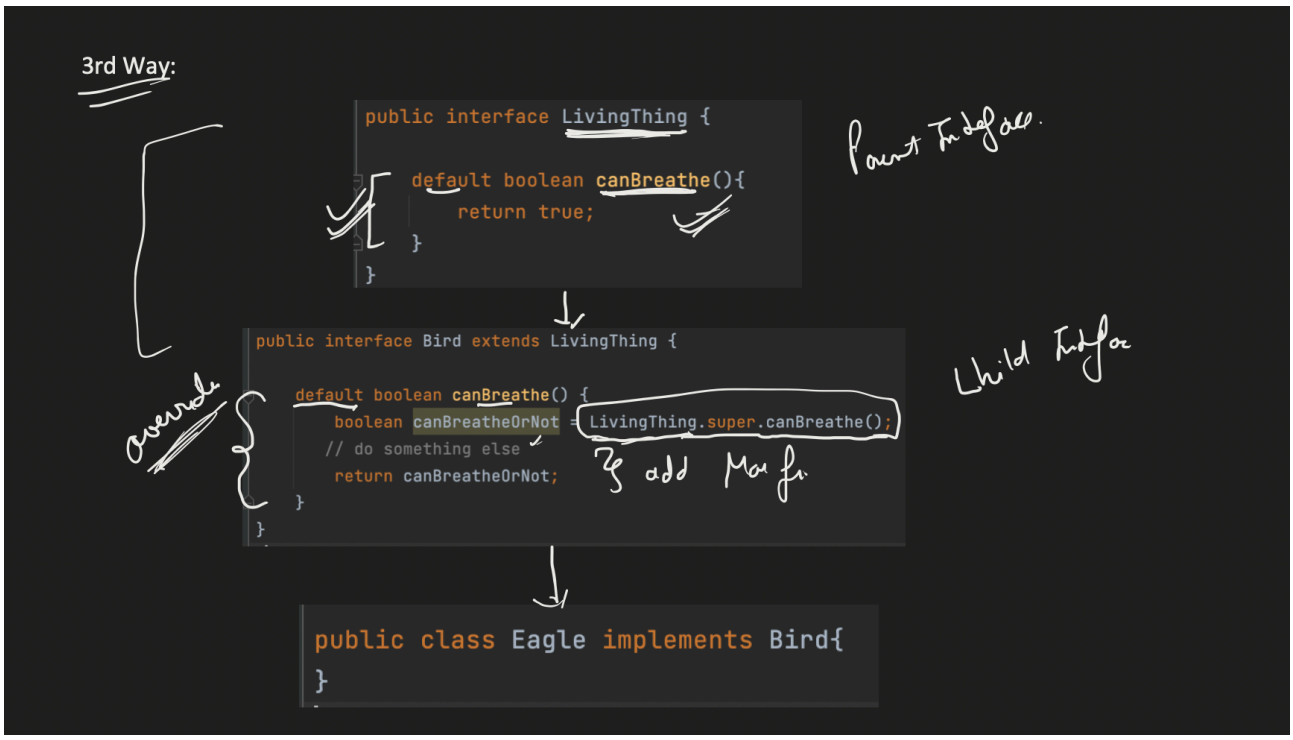
Child

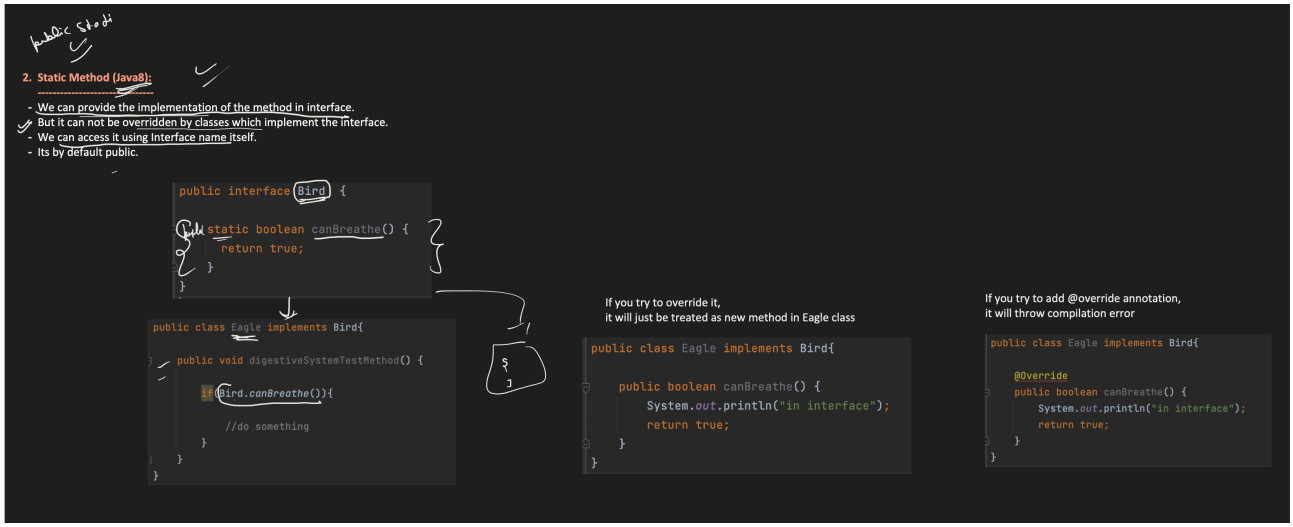
```java
public class Eagle implements Bird{

}
```

```java
public class Eagle implements Bird{

    @Override
    public boolean canBreathe() {
        return true;
    }
}
```


3rd Way:
public interface LivingThing {
default boolean canBreathe(){
return true;
}
}
Parent Interface.
public interface Bird extends LivingThing {
default boolean canBreathe() {
boolean canBreatheOrNot = LivingThing.super.canBreathe();
// do something else
return canBreatheOrNot;
}
}
Child Interface
override
add More
public class Eagle implements Bird{
}


2. Static Method (Java8):
- We can provide the 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.
- Its by default public.
public interface Bird {
static boolean canBreathe() {
return true;
}
}
public class Eagle implements Bird{
public void digestiveSystemTestMethod() {
if(Bird.canBreathe()){
//do something
}
}
}
If you try to override it,
it will just be treated as new method in Eagle class
public class Eagle implements Bird{
public boolean canBreathe() {
System.out.println("in interface");
return true;
}
}
If you try to add @override annotation,
it will throw compilation error
public class Eagle implements Bird{
@Override
public boolean canBreathe() {
System.out.println("in interface");
return true;
}
}

3. Private Method and Private Static method (Java9):

- We can provide the implementation of method but as a private access modifier in interface.
- It brings more readability of the code. For example if multiple default method share some code, that this can help.
- It can be defined as static and non-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 interface method can not be abstract. Means we have to provide the definition.
- It can be used inside of the particular interface only.

```
public interface Bird {

    void canFly(); //this is equivalent to public abstract void canFly();

    public default void minimumFlyingHeight() {
        myStaticPublicMethod();  //calling static method
        myPrivateMethod(); //calling private method
        myPrivateStaticMethod();   //calling private static method
    }

    static void myStaticPublicMethod() {
        myPrivateStaticMethod(); //from static we can call other static method only
    }

    private void myPrivateMethod(){
            // private method implementation
    }

    private static void myPrivateStaticMethod(){
            // private static method implementation

    }
```