

15. Java Default Static & Private Method In Interface.

Q) What was the Problem with Interfaces?

A) All the methods need to be implemented by the concrete class that is a headache.

So as we saw, java 8 solved this problem

```
interface Bird {
```

```
    default void canFly() {
```

```
        sout("Fly");
```

```
    }
```

```
}
```

class Eagle implements Bird {

No need to implement

```
}
```

(Another Case)

```
interface Bird {
```

```
    default void canFly() {
```

```
        sout("Fly");
```

```
    }
```

(default method)

```
}
```

interface AdvancedBird

~~implements~~ Bird {

extends

```
    void canFly();
```

writing it in this

Format will cause it to become abstract again.

Class Eagle implements AdvancedBird {

Here if you don't implement

canFly() method, it will throw

Error,

```
}
```

(Reverse Case)

The opposite is also possible,

Parent Interface is there. Now am creating a child Interface which is implementing the default method of Parent Interface, its way

```
public Interface LivingThing {
```

```
    default boolean canBreathe() {
```

```
        return true;
```

```
    }
```

```
public Interface Bird extends LivingThing {
```

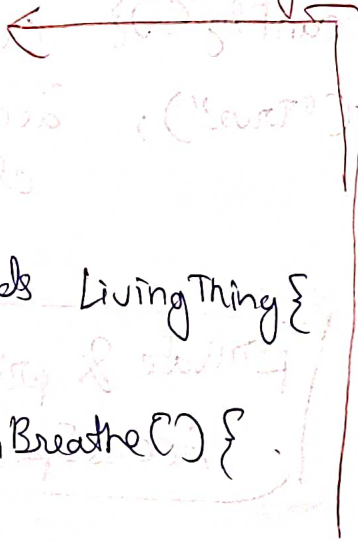
```
    default boolean canBreathe() {
```

```
        return ! LivingThing.super.canBreathe();
```

```
    }
```

```
}
```

Calling this Internally



Static methods In Interface. (Java 8)

By default if you do not provide any access specifier.

All methods are Public in the Interface.

Also all methods are abstract by default

```
interface Bird {
```

```
    static void canFly() {
```

```
        sout("True");
```

```
    }
```

```
}
```

Can be
accessed by
className as
well

Bird.canFly()

private & private Static
Access Specifiers

(Java 9)

Since Java 9 onwards - We can make default and static methods private in Interface. There was a need for this to improve code Reuseability. We can use same code in the default / static blocks which are public and save ourselves from Redundant Code