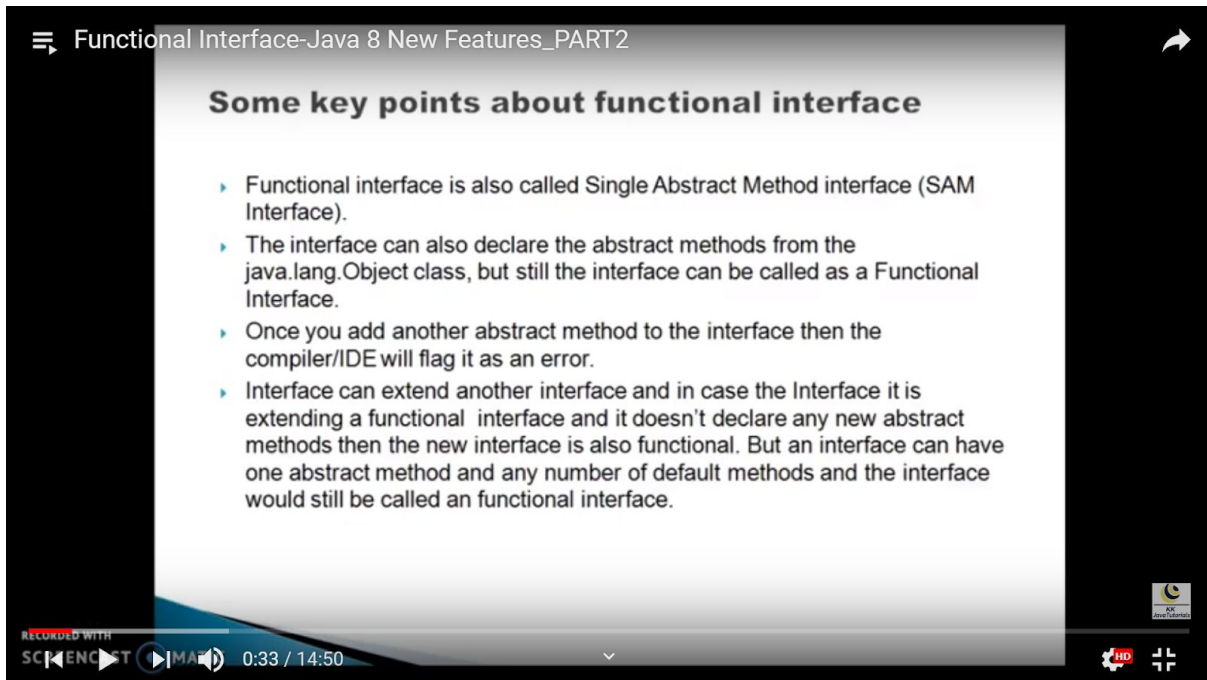


# Functional Interface :



The screenshot shows a video player interface. The title bar at the top reads "Functional Interface-Java 8 New Features\_PART2". The main content area displays a slide titled "Some key points about functional interface". The slide contains four bullet points: 1. Functional interface is also called Single Abstract Method interface (SAM Interface). 2. The interface can also declare the abstract methods from the java.lang.Object class, but still the interface can be called as a Functional Interface. 3. Once you add another abstract method to the interface then the compiler/IDE will flag it as an error. 4. Interface can extend another interface and in case the Interface it is extending a functional interface and it doesn't declare any new abstract methods then the new interface is also functional. But an interface can have one abstract method and any number of default methods and the interface would still be called an functional interface. The video player controls at the bottom show a progress bar at 0:33 / 14:50, a play button, and a volume icon.

Functional Interface-Java 8 New Features\_PART2

### Some key points about functional interface

- ▶ Functional interface is also called Single Abstract Method interface (SAM Interface).
- ▶ The interface can also declare the abstract methods from the java.lang.Object class, but still the interface can be called as a Functional Interface.
- ▶ Once you add another abstract method to the interface then the compiler/IDE will flag it as an error.
- ▶ Interface can extend another interface and in case the Interface it is extending a functional interface and it doesn't declare any new abstract methods then the new interface is also functional. But an interface can have one abstract method and any number of default methods and the interface would still be called an functional interface.

RECORDED WITH SCIENCE T 0:33 / 14:50

## 1. What is functional interface

Functional interfaces are new additions in [java 8](#) which permit exactly one abstract method inside them. These interfaces are also called Single Abstract Method interfaces (SAM Interfaces).

In Java 8, functional interfaces can be represented using lambda expressions, method reference and constructor references as well.

Java 8 introduces an annotation i.e. `@FunctionalInterface` too, which can be used for compiler level errors when the interface you have annotated violates the contracts of exactly one abstract method.

Let's build our first functional interface:

```
@FunctionalInterface
public interface MyFirstFunctionalInterface
{
    public void firstWork();
}
```

Let's try to add another abstract method:

```
@FunctionalInterface
public interface MyFirstFunctionalInterface
{
    public void firstWork();
    public void doSomeMoreWork(); //error
}
```

## 2. Do's and Don't's in functional interfaces

Below is list of things which are allowed and which are not in a functional interface.

- As discussed above, *only one abstract method is allowed* in any functional interface. Second abstract method is not permitted in a functional interface. If we remove `@FunctionalInterface` annotation then we are allowed to add another abstract method, but it will make the interface non-functional interface.
- A functional interface is *valid even if the `@FunctionalInterface` annotation would be omitted*. It is only for informing the compiler to enforce single **abstract method** inside interface.
- Conceptually, a functional interface has exactly one abstract method. Since **default methods** have an implementation, they are not abstract. Since default methods are not abstract you're *free to add default methods to your functional interface as many as you like*.

Below is valid functional interface:

```

@FunctionalInterface
public interface MyFirstFunctionalInterface
{
    public void firstWork();

    default void doSomeMoreWork1(){
        //Method body
    }

    default void doSomeMoreWork2(){
        //Method body
    }
}

```

If an interface declares an *abstract method overriding one of the public methods of `java.lang.Object`*, that also does not count toward the interface's abstract method count since any implementation of the interface will have an implementation from `java.lang.Object` or elsewhere. e.g. [Comparator](#) is a functional interface even though it declared two abstract methods. Why? Because one of these abstract methods "`equals()`" which has signature equal to `public` method in `Object` class.

e.g. Below interface is a valid functional interface.

```

@FunctionalInterface
public interface MyFirstFunctionalInterface
{
    public void firstWork();

    @Override
    public String toString();                //Overridden from Object
class

    @Override
    public boolean equals(Object obj);        //Overridden from Object
class
}

```