Interfaces

In this lesson, another important topic of Java abstraction is covered i.e. Interfaces.

WE'LL COVER THE FOLLOWING

What is an Interface?

> An interface is just like a `class` and specifies the behavior that a class **must** *implement*.

An interface can be used to achieve **100%** abstraction as it contains the method signatures/abstract methods(*what to be done*) and no implementation details (*how to be done*) of these methods. In this way, interfaces satisfy the definition of abstraction. The implementation techniques of the methods declared in an interface are totally up to to the classes implementing that interface.

> An interface can be thought of as a **contract** that a class has to fulfill while implementing an interface. According to this contract, the class that `implements` an interface has to `@Override` all the abstract methods declared in that very interface.

Declaration

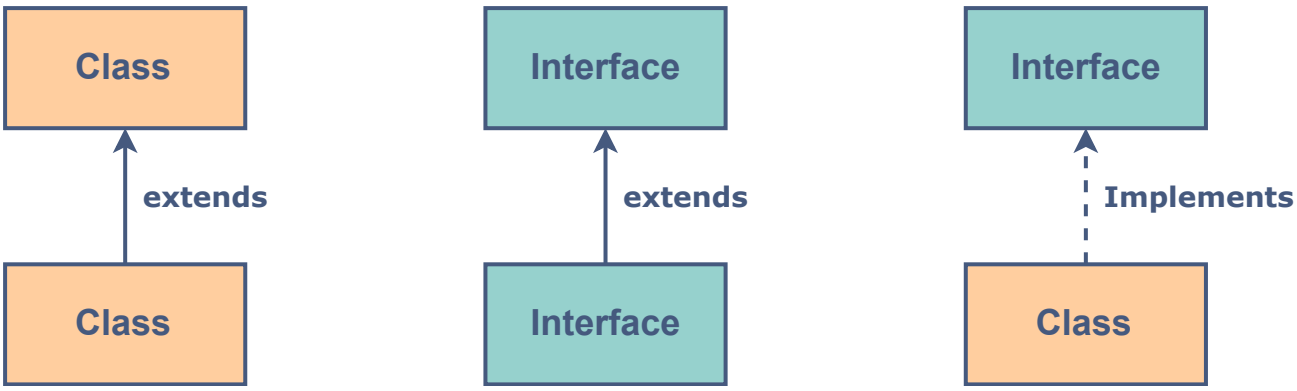
An interface is declared just like a class but using the keyword `interface`:

```
interface interfaceName {

  // Code goes here

}
```

Rules to be Followed

- An interface can have:
  - `abstract` method(s)
  - `default` method(s)
  - `static` method(s)
  - `public static final` variable(s)
- All the methods declared or implemented in an interface are by default `public` and all the variables are by default `public static final`.
- Just like an `abstract` class, an `interface` cannot be instantiated.
- To use an interface, a class **must** `implement` all of the `abstract` method(s) declared in it.
- An interface **cannot** have constructor(s) in it.
- A class cannot extend from more than one class, but it can implement **any number** of interfaces."?
- An interface can `extend` from another interface.
- An interface cannot be declared `private` or `protected`.

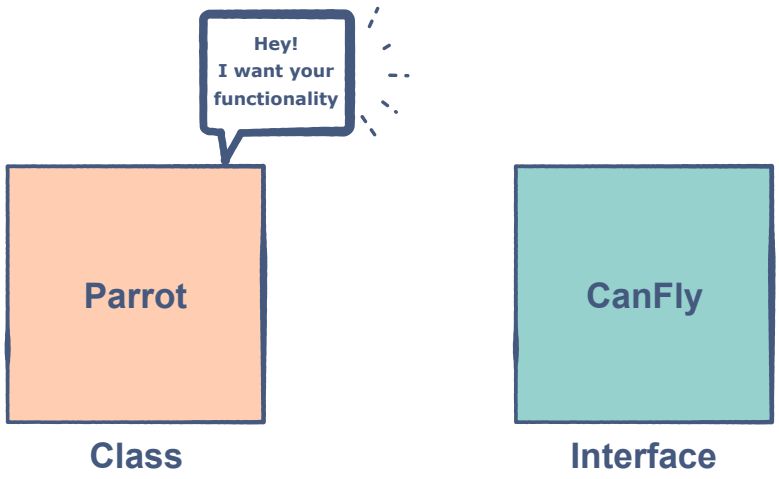
> **Note:** A class uses the keyword `implements` to use an interface but an interface uses the keyword `extends` to use *another* interface.



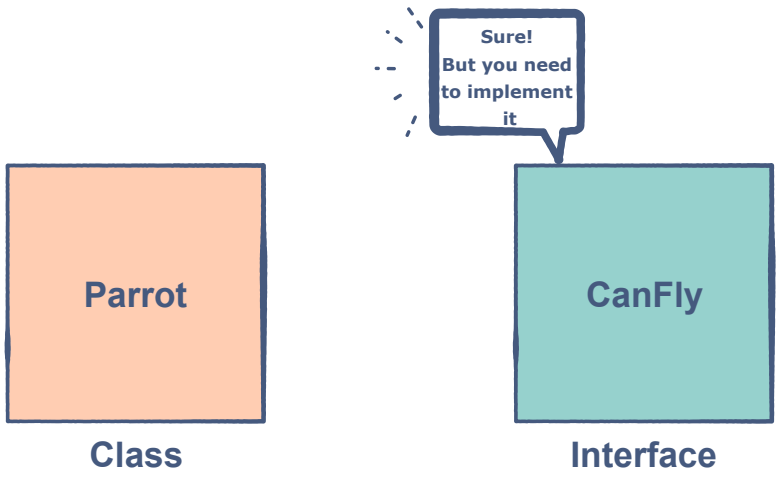
Implementation

Let's see interfaces in action using the below example:

- A base class named `Bird`
- A derived class named `Parrot`
- A derived class named `Penguin`
- An interface named `CanFly`



1 of 3



2 of 3

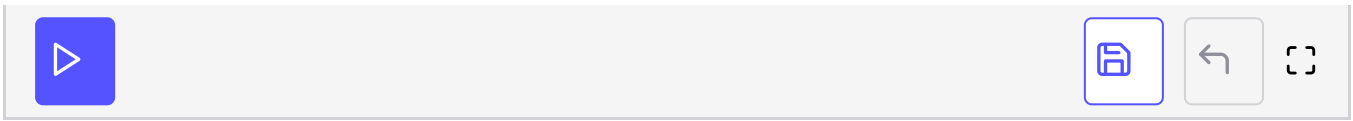
```
class Parrot implements CanFly
```

3 of 3

The code goes below:

```
// Base class Bird
class Bird {

  // Common trait of all birds so implemented in the base class
  public void eat() {
    System.out.println(getClass().getSimpleName() + " is eating!");
  }

}// End of Bird class

interface CanFly {
```

```
interface CanFly {

  // The method is by default abstract and public
  void flying();

}// End of CanFly interface

class Parrot extends Bird implements CanFly { // Parrot is extending from Bird and implementi

  @Override               // If you don't implement the flying() you will get an error!
  public void flying() { // Overriding the method of CanFly interface
    System.out.println("Parrot is Flying!");
  }
} // End of Parrot class

class Penguin extends Bird { // Penguin is a bird so extending from Bird

  // Penguin cannot fly so not implementing CanFly
  public void walk() {
    System.out.println("Penguin is Walking!");
  }

} // End of Penguin class

class Main {

  public static void main(String[] args) {

    Parrot parrot = new Parrot();   // Creating the Parrot object
    Penguin penguin = new Penguin(); // Creating the Penguin object

    parrot.eat();
    parrot.flying();

    System.out.println();    // Just creating a newline on console

    penguin.eat();
    penguin.walk();

  } // End of main()

} // End of Main class
```



> The highlighted line shows how to implement an interface syntactically.

Advantages of Interfaces

- Interfaces allow us to achieve *100% abstraction*.
- Interfaces can be used to achieve *loose coupling* in an application. This means that a change in one class doesn't affect the implementation of the other class.

other class.

- By the use of interfaces, one can break up complex designs and clear the dependencies between objects.
- Interfaces can be used to achieve *multiple inheritance*(discussed in the next lesson).