Another way to achieve abstraction in Java, is with interfaces.

An `interface` is a completely "**abstract class**" that is used to group related methods with empty bodies:

## Example

```
// interface

// interface

interface Animal {

  public void animalSound(); // interface method (does not have a body)

  public void run(); // interface method (does not have a body)

}
```

To access the interface methods, the interface must be "implemented" (kinda like inherited) by another class with the `implements` keyword (instead of `extends`).

```
// Interface

interface Animal {

  public void animalSound(); // interface method (does not have a body)

  public void sleep(); // interface method (does not have a body)

}


// Pig "implements" the Animal interface

class Pig implements Animal {

  public void animalSound() {

    // The body of animalSound() is provided here

    System.out.println("The pig says: wee wee");

  }

  public void sleep() {

    // The body of sleep() is provided here

    System.out.println("Zzz");
  }
}


class MyMainClass {
  public static void main(String[] args) {
    Pig myPig = new Pig();  // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

- Like **abstract classes**, interfaces **cannot** be used to create objects (in the example above, it is not possible to create an "Animal" object in the MyMainClass)

- Interface methods do not have a body - the body is provided by the "implement" class

- An interface cannot contain a constructor (as it cannot be used to create object
-

- Interface provides full abstraction as none of its methods have body. On the other hand abstract class provides partial abstraction as it can have abstract and concrete(methods with body) methods both.

- While providing implementation in class of any method of an interface, it needs to be mentioned as public.

- Class that implements any interface must implement all the methods of that interface, else the class should be declared abstract.

- Interface cannot be declared as private, protected or transient.

- All the interface methods are by default **abstract and public**.

- Variables declared in interface are **public, static and final** by default

## *Why And When To Use Interfaces?*

1) To achieve security - hide certain details and only show the important details of an object (interface).

2) Java does not support "multiple inheritance" (a class can only inherit from one superclass). However, it can be achieved with interfaces, because the class can **implement** multiple interfaces.

## Diff between abstract and interfaces

The reason is, abstract classes may contain non-final variables, whereas variables in interface are final, public and static.