A general requirement for an interface to be a "callback interface" is that the interface provides a way for the callee to invoke the code inside the caller. The main idea is that the caller has a piece of code that needs to be executed when something happens in the code of another component. Callback interfaces provide a way to pass this code to the component being called: the caller implements an interface, and the callee invokes one of its methods.

The callback mechanism may be implemented differently in different languages: C# has delegates and events in addition to callback interfaces, C has functions that can be passed by pointer, Objective C has delegate protocols, and so on. But the main idea is always the same: the caller passes a piece of code to be called upon occurrence of a certain event.

Java callback function

In C and C++ programming language, the process of calling a function from another function is referred to as **callback**. The function's memory address is represented as the **function pointer**.

In C and C++ languages, we achieve the callback bypassing the function pointer to another function.

Unlike C

and C++ language
, Java
doesn't have the concept of the callback function. Java doesn't use the concept of the pointer, due to which callback functions are not supported. However, we can create a callback function by passing an interface that refers to the location of the function instead of passing the function's memory address.

The syntax of the callback function in Java using the interface is given below:

```
1. public interface interfaceA {
2. public String callA() ;
3. }
```

How does the Java callback function work?

The callback function in Java works in the following manner:

1. Create an interface X having a single method A().
2. Create a method method1() with A as a method parameter.

3. Call the A() method inside of the method1().
4. For calling method1(), we pass the instance of X and override the A().
5. Use arrow notation as an alternative to keyword news so that the code is clear.

Let's take an example and understand a scenario in which we can use callback function in Java,

Example:

We implement the tax calculator that calculates the tax(total tax) for the state. For a state, tax can be more than one but for our example, let's assume that we have only two taxes, i.e., state tax and central tax. In both the taxes, the central tax will be the same for all states, but state tax may be different for states.

```
1. static void calculateTax(address of stateTax() function)
2. {
3.     central_tax = 2500.0
4.     state_tax = get based on the address of the interface method
5.     total_tax = central_tax + state_tax;
6. }
```

In the above code, the address or the reference of the stateTax() method is passed for calculating the sum of taxes in the calculateTax() method. The state code varies from state to state, so we declare the stateTax() method as an abstract method in the interface.

```
1. interface StateTax
2. {
3.     double stateTax();
4. }
```

Below is the implementation of the stateTax() method for Haryana and Utter Pradesh state:

```
class Haryana implements StateTax{
    public double stateTax(){
        return 1500.0;
     }
}
class UP implements StateTax{
    public double stateTax(){
        return 2500.0;
     }
}
```

CallbackExample1.java

```
//import required classes and packages
import java.util.Scanner;

//create interface StateTax having abstract method stateTax()
interface StateTax {
    //declare abstract method
    double stateTax();
}

// implement the StateTax interface for Haryana
class Haryana implements StateTax {
    public double stateTax()
    {
        return 1500.0;
    }
```

```
}

// implement the StateTax interface for Utter Pradesh
class UP implements StateTax {
    public double stateTax()
    {
        return 2500.0;
    }
}

// create CallbackExample1 to calculate total tax
class CallbackExample1 {

    //main() method start
    public static void main(String[] args) throws ClassNotFoundException, IllegalAccessException, InstantiationException    //handling exceptions
    {
        //create an instance of the Scanner class to take user input
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the state name");

        //get state name from the user
        String state = sc.next();

        // store the state name in an object obj
        Class obj = Class.forName(state);

        // Create a new object of the obj class that reference by the StateTax
        StateTax refer = (StateTax)obj.newInstance();

        //call the calculateTax() method by passing the reference of the StateTax  to calculate total tax
        calculateTax(refer);
    }

    static void calculateTax(StateTax data)
```

```
    {
        //central tax will be same for all state
        double central_Tax = 3500.0;

        // get the state_Tax for the user entered state and find the total tax
        double state_Tax = data.stateTax();
        double total_Tax = state_Tax + central_Tax;

        // show the result
        System.out.println("The total Tax of the state is =" + total_Tax);
    }
}
```

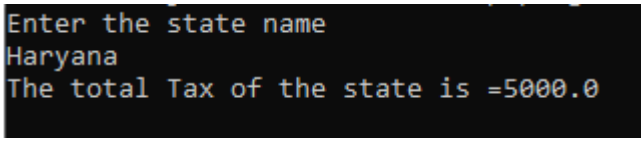
```
Enter the state name
Haryana
The total Tax of the state is =5000.0
```

CallbackExample2.java

```
//Create an interface to handle click event for the callback method
interface ClickHandler {
    //create abstract method, i.e., clickEventHandler() that will act as callback
    public void clickEventHandler();
}

//Create a class that handle the callback and implements the ClickHandler
class Handler implements ClickHandler {
    //define the clickEventHandler()method
    public void clickEventHandler() {
        System.out.println("Button is clicking");
    }
}
//Create a class that generate the event
class ButtonClass {
```

```
    public void onClickEvent(ClickHandler obj)
    {
        obj.clickEventHandler();
    }
}
//create CallbackExample2 class
public class CallbackExample2 {
    //main() method starts
    public static void main(String[] args) {
        //create an object of the ButtonClass
        ButtonClass button = new ButtonClass();

        //create an object of the Handler class
        Handler hndlrObj = new Handler();

        //pass the object of ClickHandler for performing the default operation
        button.onClickEvent(hndlrObj);

        //create another object of ButtonClass
        ButtonClass newButton = new ButtonClass();

        //pass the interface to implement own operation
        newButton.onClickEvent(new ClickHandler() {

            @Override
            //overriding the clickEventHandler() that shows output on clicking
            public void clickEventHandler() {
                System.out.println("A button is clicked");
            }
        });
    }
}
```

Output:



CallbackExample3.java

```
1.  //create class CallbackExample3 for callback() method
2.  public class CallBackExample3 {
3.      //main method start
4.      public static void main(String args[]) {
5.          //function that passes interface name as parameter
6.
7.          //create a method and pass the interface as parameter
8.          callbackMethod(new interfaceX()
9.          {
10.             //create callMethodX()
11.             public String callMethodX()
12.             {
13.                 return  "It is the first callback";
14.             }
15.         });
16.
17.         // create a method and pass the interface as parameter
18.         callbackMethod(new interfaceX()
19.         {
20.             //create callMethodX()
21.             public  String  callMethodX() {
22.                 return  "It is the second callback";
23.             }
24.         });
25.
26.         callbackMethod(() ->
27.         {
```

```
28.             return "It is the third callback";
29.         });
30.     }
31.
32.     //define callbackMethod()
33.     public static void callbackMethod(interfaceX obj)
34.     {
35.         //print callback messages
36.         System.out.println(obj.callMethodX());
37.     }
38.     //create an interface
39.     public interface interfaceX {
40.         public String callMethodX();
41.     }
42. }
```

Output: