<https://pediaa.com/what-is-the-difference-between-synchronous-and-asynchronous-calls-in-java/#:~:text=The%20main%20difference%20between%20synchronous,another%20function%20as%20an%20argument>.

The **main difference** between synchronous and asynchronous calls in Java is that, **in synchronous calls, the code execution waits for the event before continuing while asynchronous calls do not block the program from the code execution.**

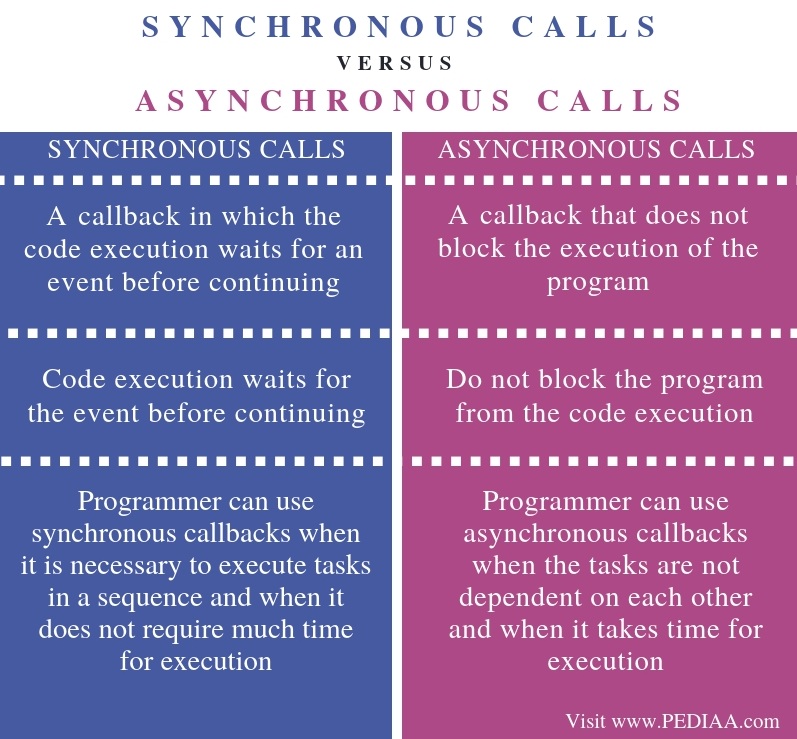
A programmer can pass callback function to another function as an argument. It is executed after an event. We can use them when we have to perform some operations after a button click or fetching data from the internet. [Interfaces](https://pediaa.com/what-is-the-difference-between-abstract-class-and-interface-in-java/#Interface%20in%20Java) help to implement callbacks in [Java](https://pediaa.com/what-is-the-difference-between-c-and-java/#Java). There are two types of callbacks as Synchronous and Asynchronous Calls.

Key Areas Covered

**1.**[**How to Implement a Callback Function**](https://pediaa.com/what-is-the-difference-between-synchronous-and-asynchronous-calls-in-java/#Implement%20a%20Callback%20Function)  
*– Steps*  
**2.**[**What are Synchronous Calls in Java**](https://pediaa.com/what-is-the-difference-between-synchronous-and-asynchronous-calls-in-java/#Synchronous%20Calls%20in%20Java)  
*– Functionality, Program*  
**3.**[**What are Asynchronous Calls in Java**](https://pediaa.com/what-is-the-difference-between-synchronous-and-asynchronous-calls-in-java/#Asynchronous%20Calls%20in%20Java)  
*– Functionality, Program*  
**4.**[**What is the Difference Between Synchronous and Asynchronous Calls in Java**](https://pediaa.com/what-is-the-difference-between-synchronous-and-asynchronous-calls-in-java/#Synchronous%20vs%20Asynchronous%20Calls%20in%20Java%20-%20Comparison%20of%20Key%20Differences)  
*– Comparison of Key Differences*

Key Terms

*Asynchronous Calls, Interface, Java, Synchronous Calls, Thread*

**

How to Implement a Callback Function?

The steps of implementing a callback function are as follows.

* Define the methods in an interface. The method will be invoked after a callback.
* Define a [class](https://pediaa.com/what-is-the-difference-between-class-and-structure/#Class) to implement the callback methods of the interface.
* Then, define the reference in the other class. It helps to register the callback interface.
* Use the defined reference to invoke the callback method.

What are Synchronous Calls in Java

In synchronous calls or callbacks, the code execution waits for an event before continuing. The program will not execute until an event returns a response. Callback performs all the tasks before returning to the call statement. One issue with synchronous callbacks is that it can cause lagging.

public interface MyEventListener {

// Define the methods in an interface.

// The method will be invoked after a callback.

void event1();

}

// Define a class to implement the callback methods of the interface.

public class A implements MyEventListener {

@Override

public void event1() {

System.out.println("Callback after sychronous Task");

}

}

public class B {

private MyEventListener listener;

public void registerEventListener(MyEventListener listener) {

this.listener = listener;

}

public void method1() {

System.out.println("Callback before synchronous Task");

if(this.listener != null) {

// Use the defined reference to invoke the callback method.

listener.event1();

}

}

public static void main(String[] args) {

B obj = new B();

// Then, define the reference in the other class.

// It helps to register the callback interface.

MyEventListener listener = new A();

obj.registerEventListener(listener);

obj.method1();

}

}

In the above program, MyEventListener is an interface. It has an abstract method called event1. In class B, there is a declaration for the listener of type MyEventListener. The registerEventListener sets the obtained value to the listener. The method1 contains synchronous tasks. The “if” condition checks whether the listener is registered. If the condition is true, the callback method of class A is invoked using the listener.

Class A implements MyEventListener. Therefore, it provides the definition of the event1 method. In class B, main method, there is an object of type B and a listener of type MyEventListener. Thus, the programmer can pass the listener to registerEventListener using the object. Finally, the method1 is called on the object.

## What are Asynchronous Calls in Java

An asynchronous call does not block the execution of the program. When the call returns from the event, the call returns back to the callback function. Therefore, the programmer has to create a [thread](https://pediaa.com/difference-between-process-and-thread/#Thread)and invoke the method inside the thread. It is not necessary to invoke a callback from a thread.

public interface MyEventListener {

// Define the methods in an interface.

// The method will be invoked after a callback.

void event1();

}

// Define a class to implement the callback methods of the interface.

public class A implements MyEventListener {

@Override

public void event1() {

System.out.println("Callback after sychronous Task");

}

}

public class B {

private MyEventListener listener;

public void registerEventListener(MyEventListener listener) {

this.listener = listener;

}

public void method1() {

new Thread(new Runnable() {

@Override

public void run() {

System.out.println("Operations in Asynchronous Task");

if(listener != null) {

listener.event1();

}

}

}).start();

}

public static void main(String[] args) {

B obj = new B();

// Then, define the reference in the other class.

// It helps to register the callback interface.

MyEventListener listener = new A();

obj.registerEventListener(listener);

obj.method1();

}

}

In the above program, MyEventListener is an interface. It has an abstract method called event1. In class B, there is a declaration for the listener of type MyEventListener. The registerEventListener sets the obtained value to the listener. The method1 contains asynchronous tasks. The programmer creates a thread inside it. Therefore, the asynchronous threads are always executed inside the new thread.  The “if” condition checks whether the listener is registered. If the condition is true, the callback method of class A is invoked using the listener.

Class A implements MyEventListener. Therefore, it provides the definition of the event1 method. In class B, main method, there is an object of type B and a listener of type MyEventListener. Therefore, the programmer can pass the listener to registerEventListener using the object. Finally, the method1 is called on the object.

## Difference Between Synchronous and Asynchronous Calls in Java

### Definition

Synchronous calls refer to a callback where the code execution waits for an event before continuing. Asynchronous calls, on the other hand, refer to a callback that does not block the execution of the program. Thus, this is the main difference between synchronous and asynchronous calls in Java.

### Functionality

Furthermore, in synchronous calls or callbacks, the code execution waits for the event before continuing while asynchronous calls or callbacks do not block the program from the code execution. Hence, this is another difference between synchronous and asynchronous calls in Java.

### Application

Besides, the programmer can use synchronous callbacks when it is necessary to execute tasks in a sequence and when it does not require much time for execution. In contrast, the programmer can use asynchronous callbacks when the tasks are not dependent on each other and when it takes time for execution. This is also a difference between synchronous and asynchronous calls in Java.

### Conclusion

It is possible to implement synchronous and asynchronous call using Java programming. The main difference between synchronous and asynchronous calls in Java is that, in synchronous calls, the code execution waits for the event before continuing while asynchronous calls do not block the program from the code execution.