**1.Give a note on GUI Development**

**wing in Java** is a Graphical User Interface (GUI) toolkit that includes the GUI components. Swing provides a rich set of widgets and packages to make sophisticated GUI components for Java applications. Swing is a part of Java Foundation Classes(JFC), which is an API for Java GUI programing that provide GUI.

The Java Swing library is built on top of the Java Abstract Widget Toolkit (**AWT**), an older, platform dependent GUI toolkit. You can use the Java simple GUI programming components like button, textbox, etc., from the library and do not have to create the components from scratch.

**GUI (Graphical User Interface) in Java** is an easy-to-use visual experience builder for Java applications. It is mainly made of graphical components like buttons, labels, windows, etc. through which the user can interact with an application. GUI plays an important role to build easy interfaces for Java applications.

2.List out SWING Concept with an Example

**Java Swing class Hierarchy Diagram**

Java Swing Class Hierarchy Diagram

All components in Java Swing are JComponent which can be added to container classes.

2.2M

**What is a Container Class?**

Container classes are classes that can have other components on it. So for creating a Java Swing GUI, we need at least one container object. There are 3 types of Java Swing containers.

1. **Panel**: It is a pure container and is not a window in itself. The sole purpose of a Panel is to organize the components on to a window.
2. **Frame**: It is a fully functioning window with its title and icons.
3. **Dialog**: It can be thought of like a pop-up window that pops out when a message has to be displayed. It is not a fully functioning window like the Frame.

**What is GUI in Java?**

**GUI (Graphical User Interface) in Java** is an easy-to-use visual experience builder for Java applications. It is mainly made of graphical components like buttons, labels, windows, etc. through which the user can interact with an application. GUI plays an important role to build easy interfaces for Java applications.

Simple Java Swing Example

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

*File: FirstSwingExample.java*

1. **import** javax.swing.\*;
2. **public** **class** FirstSwingExample {
3. **public** **static** **void** main(String[] args) {
4. JFrame f=**new** JFrame();//creating instance of JFrame
5.
6. JButton b=**new** JButton("click");//creating instance of JButton
7. b.setBounds(130,100,100, 40);//x axis, y axis, width, height
8.
9. f.add(b);//adding button in JFrame
10.
11. f.setSize(400,500);//400 width and 500 height
12. f.setLayout(**null**);//using no layout managers
13. f.setVisible(**true**);//making the frame visible
14. }
15. }



**I/O Streams**

[Java](https://www.geeksforgeeks.org/java/) brings various Streams with its I/O package that helps the user to perform all the input-output operations. These streams support all the types of objects, data-types, characters, files etc to fully execute the I/O operations.



Before exploring various input and output streams lets look at **3 standard or default streams** that Java has to provide which are also most common in use:



1. [System.in](https://www.geeksforgeeks.org/java-lang-system-class-java/)**:** This is the **standard input stream** that is used to read characters from the keyboard or any other standard input device.
2. [System.out](https://www.geeksforgeeks.org/java-lang-system-class-java/)**:** This is the **standard output stream** that is used to produce the result of a program on an output device like the computer screen.

**3.** [System.err](https://www.geeksforgeeks.org/java-lang-system-class-java/)**:** This is the **standard error stream** that is used to output all the error data that a program might throw, on a computer screen or any standard output device.

**Types of Streams:**

* **Depending on the type of operations**, streams can be divided into two primary classes:
	1. [Input Stream:](https://www.geeksforgeeks.org/java-io-inputstream-class-in-java/) These streams are used to read data that must be taken as an input from a source array or file or any peripheral device. For eg., FileInputStream, BufferedInputStream, ByteArrayInputStream etc.



**2.**[Output Stream:](https://www.geeksforgeeks.org/java-io-outputstream-class-java/%22%20%5Ct%20%22_blank) These streams are used to write data as outputs into an array or file or any output peripheral device. For eg., FileOutputStream, BufferedOutputStream, ByteArrayOutputStream etc.

**Depending on the types of file**, Streams can be divided into two primary classes which can be further divided into other classes as can be seen through the diagram below followed by the explanations. 

**ByteStream:** This is used to process data byte by byte (8 bits). Though it has many classes, the FileInputStream and the FileOutputStream are the most popular ones. The FileInputStream is used to read from the source and FileOutputStream is used to write to the destination. Here is the list of various ByteStream Classes:

| Stream class | Description |
| --- | --- |
| [BufferedInputStream](https://www.geeksforgeeks.org/java-io-bufferedinputstream-class-java/) | It is used for Buffered Input Stream. |
| [DataInputStream](https://www.geeksforgeeks.org/java-io-datainputstream-class-java-set-1/) | It contains method for reading java standard datatypes. |
| [FileInputStream](https://www.geeksforgeeks.org/java-io-fileinputstream-class-java/) | This is used to reads from a file |
| [InputStream](https://www.geeksforgeeks.org/java-io-inputstream-class-in-java/) | This is an abstract class that describes stream input. |
| [PrintStream](https://www.geeksforgeeks.org/java-io-printstream-class-java-set-1/) | This contains the most used print() and println() method |
| [BufferedOutputStream](https://www.geeksforgeeks.org/java-io-bufferedoutputstream-class-java/) | This is used for Buffered Output Stream. |
| [DataOutputStream](https://www.geeksforgeeks.org/dataoutputstream-in-java/) | This contains method for writing java standard data types. |
| [FileOutputStream](https://www.geeksforgeeks.org/creating-a-file-using-fileoutputstream/) | This is used to write to a file. |
| [OutputStream](https://www.geeksforgeeks.org/java-io-outputstream-class-java/) | This is an abstract class that describe stream output. |

 **CharacterStream:** In Java, characters are stored using Unicode conventions (Refer this for details). Character stream automatically allows us to read/write data character by character. Though it has many classes, the FileReader and the FileWriter are the most popular ones. FileReader and FileWriter are character streams used to read from the source and write to the destination respectively. Here is the list of various CharacterStream Classes:

| Stream class | Description |
| --- | --- |
| [BufferedReader](https://www.geeksforgeeks.org/java-io-bufferedreader-class-java/) | It is used to handle buffered input stream. |
| [FileReader](https://www.geeksforgeeks.org/file-handling-java-using-filewriter-filereader/) | This is an input stream that reads from file. |
| [InputStreamReader](https://www.geeksforgeeks.org/java-io-inputstreamreader-class/) | This input stream is used to translate byte to character. |
| OutputStreamReader | This output stream is used to translate character to byte. |
| [Reader](https://www.geeksforgeeks.org/java-io-reader-class-java/) | This is an abstract class that define character stream input. |
| [PrintWriter](https://www.geeksforgeeks.org/java-io-printwriter-class-java-set-1/) | This contains the most used print() and println() method |
| [Writer](https://www.geeksforgeeks.org/java-io-writer-class-java/) | This is an abstract class that define character stream output. |
| [BufferedWriter](https://www.geeksforgeeks.org/io-bufferedwriter-class-methods-java/) | This is used to handle buffered output stream. |
| [FileWriter](https://www.geeksforgeeks.org/file-handling-java-using-filewriter-filereader/) | This is used to output stream that writes to file. |

# Generic Collections in Java

The generic collections are introduced in **Java 5 Version.** The generic collections**disable**the**type-casting** and there is no use of **type-casting** when it is used in generics. The generic collections are**type-safe**and checked at **compile-time**. These generic collections allow the datatypes to pass as parameters to classes. The **Compiler**is responsible for checking the **compatibility**of the types.

## Syntax

class<type>, interface<type>

## Type safety

Generics allows a single type of object.

List list = new ArrayList(); **// before generics**

list.add(10);

list.add("100");

**List<Integer> list1 = new ArrayList<Integer>()**; // **adding generics**

list1.add(10);

list1.add("100"); **// compile-time error.**

## Type Casting

No need for type-casting while using generics.

**List<String> list = new ArrayList<String>();**

list.add("Adithya");

String str = list.get(0); // **no need of type-casting**

## Compile-time

The errors are checked at **compile-time** in generics.

List list = new ArrayList(); **// before generics**

list.add(10);

list.add("100");

**List<Integer> list1 = new ArrayList<Integer>();** //**adding generics**

list1.add(10);

list1.add("100");// **compile-time error**

# Serialization in Java

**Serialization in**[**Java**](https://www.edureka.co/java-j2ee-training-course) is an important concept that deals with the conversion of objects into a byte stream to transport the java objects from one Java Virtual Machine to the other and recreate them to the original form

## ****What is Serialization in Java?****

**Serialization** in Java is the process of converting the Java code Object into a Byte Stream, to transfer the Object Code from one Java Virtual machine to another and recreate it using the process of Deserialization.

## Serialization-in-Java-Edureka-Picture-1

## ****Why do we need Serialization in Java?****

We need Serialization for the following reasons:

* **Communication**: Serialization involves the procedure of object serialization and transmission. This enables multiple computer systems to design, share and execute objects simultaneously.
* **Caching**: The time consumed in building an object is more compared to the time required for de-serializing it. Serialization minimizes time consumption by caching the giant objects.
* **Deep Copy**: Cloning process is made simple by using Serialization. An exact replica of an object is obtained by serializing the object to a byte array, and then de-serializing it.
* **Cross** **JVM Synchronization:** The major advantage of Serialization is that it works across different JVMs that might be running on different architectures or Operating Systems
* **Persistence:** The State of any object can be directly stored by applying Serialization on to it and stored in a database so that it can be retrieved later.



## ****How do we Serialize an Object?****

A **Java object** is **serializable**if and only if its class or any of its parent classes implement either the **java**.**io**.**Serializable**interface or its subinterface, **java.io.Externalizable.**

In the Serialization process, we convert an object’s state into a byte stream so that it could be transferred from one JVM to the other and revert the byte stream back into the original object.

//Interface

|  |  |
| --- | --- |
| 123456789101112 | package Serial1; import java.io.Serializable;public class Employee implements Serializable{       private static final long serialVersionUID = 1L; //Serial Version UID       int id;       String name;       public Employee(int id, String name) {             this.id = id;             this.name = name;        }} |

//Serialize

|  |  |
| --- | --- |
| 12345678910111213141516171819202122 | package Serial1; import java.io.\*;class Persist{       public static void main(String args[]){               try{                      Employee emp1 =new Employee(20110,"John");                      Employee emp2 =new Employee(22110,"Jerry");                      Employee emp3 =new Employee(20120,"Sam");                      FileOutputStream fout=new FileOutputStream("output.txt");                      ObjectOutputStream out=new ObjectOutputStream(fout);                      out.writeObject(emp1);                      out.writeObject(emp2);                      out.writeObject(emp3);                      out.flush();                      out.close();                      System.out.println("Serialization and Deserialization is been successfully executed");               }               catch(Exception e){                      System.out.println(e);}               }} |

Output:Serialization and Deserialization is been successfully executed

**Deserialization**: It is the reverse process of Serialization where the Serialized Byte Stream of an object from the sender is recreated at the receiving end.

//Deserialise

|  |  |
| --- | --- |
| 12345678910111213141516171819 | package Serial1; import java.io.\*;class Depersist{      public static void main(String args[]){            try{                  ObjectInputStream in=new ObjectInputStream(new FileInputStream("output.txt"));                  Employee e1=(Employee)in.readObject();                  Employee e2=(Employee)in.readObject();                  Employee e3=(Employee)in.readObject();                  System.out.println(e1.id+" "+e1.name);                  System.out.println(e2.id+" "+e2.name);                  System.out.println(e3.id+" "+e3.name);                  in.close();            }            catch(Exception e){                  System.out.println(e);}            }} |

Output:

20110 John
22110 Jerry

20120 Sam

 **Advantages and Disadvantages of Serialization in Java**

**Advantages:**

* Serialization process is a *built-in* feature that does not require third-party software to execute Serialization
* The Serialization procedure is proven to be *simple* and *easy* to understand
* Serialization procedure is *universal* and developers from different background are familiar to it
* It is easy to use and*simple to customize*
* Serialized data streams *support Encryption, Compression, Authentication* and*secure Java computing*
* There are many *critical technologies* relying on serialization.

**Disadvantages:**

* Objects while DeSerialization becomes brittle and they are not sure to be DeSerialized effectively.
* The Transient variables declared while Serialization creates memory space, but the constructor is not called which results in the failure in the initialization of transient variables resulting in a variation to the Standard Java Flow.
* The process of serialization is inefficient in terms of memory utilization.
* Serialization is not preferable to be used in the applications which need concurrent access without the requirement of third-party APIs, as Serialization does not offer any transition control mechanism per every SE.
* Serialization procedure fails to offer fine-grained control to access Objects.

**CONCURRENCY IN JAVA**

**Multithreading in**[**Java**](https://www.javatpoint.com/java-tutorial) is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation, etc.

Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

2) You **can perform many operations together, so it saves time**.

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

## What is Thread in java

A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.

Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.



# Lifecycle and States of a Thread in Java

A [thread](https://www.geeksforgeeks.org/multithreading-in-java/) in Java at any point of time exists in any one of the following states. A thread lies only in one of the shown states at any instant:

1. New
2. Runnable
3. Blocked
4. Waiting
5. Timed Waiting
6. Terminated

The diagram shown below represents various states of a thread at any instant in time.



#### **Life Cycle of a thread**

1. **New Thread:** When a new thread is created, it is in the new state. The thread has not yet started to run when the thread is in this state. When a thread lies in the new state, its code is yet to be run and hasn’t started to execute.
2. **Runnable State:** A thread that is ready to run is moved to a runnable state. In this state, a thread might actually be running or it might be ready to run at any instant of time. It is the responsibility of the thread scheduler to give the thread, time to run.
A multi-threaded program allocates a fixed amount of time to each individual thread. Each and every thread runs for a short while and then pauses and relinquishes the CPU to another thread so that other threads can get a chance to run. When this happens, all such threads that are ready to run, waiting for the CPU and the currently running thread lie in a runnable state.
3. **Blocked/Waiting state:** When a thread is temporarily inactive, then it’s in one of the following states:
	* Blocked
	* Waiting
4. **Timed Waiting:** A thread lies in a timed waiting state when it calls a method with a time-out parameter. A thread lies in this state until the timeout is completed or until a notification is received. For example, when a thread calls sleep or a conditional wait, it is moved to a timed waiting state.
5. **Terminated State:** A thread terminates because of either of the following reasons:
	* Because it exits normally. This happens when the code of the thread has been entirely executed by the program.
	* Because there occurred some unusual erroneous event, like segmentation fault or an unhandled exception.

### Implementing the Thread States in Java

In Java, to get the current state of the thread, use **Thread.getState()** method to get the current state of the thread. Java provides **java.lang.Thread.State** class that defines the ENUM constants for the state of a thread, as a summary of which is given below:

#### 1. New

Declaration: public static final Thread.State NEW

**Description:**Thread state for a thread that has not yet started.

#### 2. Runnable

Declaration: public static final Thread.State RUNNABLE

**Description:**Thread state for a runnable thread. A thread in the runnable state is executing in the Java virtual machine but it may be waiting for other resources from the operating system such as a processor.

#### 3. Blocked

Declaration: public static final Thread.State BLOCKED

**Description:**Thread state for a thread blocked waiting for a monitor lock. A thread in the blocked state is waiting for a monitor lock to enter a synchronized block/method or reenter a synchronized block/method after calling Object.wait().

#### 4. Waiting

Declaration: public static final Thread.State WAITING

**Description:**Thread state for a waiting thread. Thread state for a waiting thread. A thread is in the waiting state due to calling one of the following methods:

* Object.wait with no timeout
* [Thread.join](https://www.geeksforgeeks.org/joining-threads-in-java/) with no timeout
* LockSupport.park

#### 5. Timed Waiting

Declaration: public static final Thread.State TIMED\_WAITING

**Description:**Thread state for a waiting thread with a specified waiting time. A thread is in the timed waiting state due to calling one of the following methods with a specified positive waiting time:

* Thread.sleep
* Object.wait with timeout
* Thread.join with timeout
* LockSupport.parkNanos
* LockSupport.parkUntil

#### 6. Terminated

Declaration: public static final Thread.State TERMINATED

**Description:**Thread state for a terminated thread. The thread has completed execution.

* Java

|  |
| --- |
| // Java program to demonstrate thread statesclass thread implements Runnable {    public void run()    {        // moving thread2 to timed waiting state        try {            Thread.sleep(1500);        }        catch (InterruptedException e) {            e.printStackTrace();        }         System.out.println(            "State of thread1 while it called join() method on thread2 -"            + Test.thread1.getState());        try {            Thread.sleep(200);        }        catch (InterruptedException e) {            e.printStackTrace();        }    }} public class Test implements Runnable {    public static Thread thread1;    public static Test obj;     public static void main(String[] args)    {        obj = new Test();        thread1 = new Thread(obj);         // thread1 created and is currently in the NEW        // state.        System.out.println(            "State of thread1 after creating it - "            + thread1.getState());        thread1.start();         // thread1 moved to Runnable state        System.out.println(            "State of thread1 after calling .start() method on it - "            + thread1.getState());    }     public void run()    {        thread myThread = new thread();        Thread thread2 = new Thread(myThread);         // thread1 created and is currently in the NEW        // state.        System.out.println(            "State of thread2 after creating it - "            + thread2.getState());        thread2.start();         // thread2 moved to Runnable state        System.out.println(            "State of thread2 after calling .start() method on it - "            + thread2.getState());         // moving thread1 to timed waiting state        try {            // moving thread1 to timed waiting state            Thread.sleep(200);        }        catch (InterruptedException e) {            e.printStackTrace();        }        System.out.println(            "State of thread2 after calling .sleep() method on it - "            + thread2.getState());         try {            // waiting for thread2 to die            thread2.join();        }        catch (InterruptedException e) {            e.printStackTrace();        }        System.out.println(            "State of thread2 when it has finished it's execution - "            + thread2.getState());    }} |

**Output**

State of thread1 after creating it - NEW

State of thread1 after calling .start() method on it - RUNNABLE

State of thread2 after creating it - NEW

State of thread2 after calling .start() method on it - RUNNABLE

State of thread2 after calling .sleep() method on it - TIMED\_WAITING

State of thread1 while it called join() method on thread2 -WAITING

State of thread2 when it has finished it's execution - TERMINATED

**Explanation:**When a new thread is created, the thread is in the NEW state. When the start() method is called on a thread, the thread scheduler moves it to Runnable state. Whenever the join() method is called on a thread instance, the current thread executing that statement will wait for this thread to move to the Terminated state. So, before the final statement is printed on the console, the program calls join() on thread2 making the thread1 wait while thread2 completes its execution and is moved to the Terminated state. thread1 goes to Waiting state because it is waiting for thread2 to complete its execution as it has called join on thread2.

# **Synchronization in Java**

Synchronization in Java is the capability to control the access of multiple threads to any shared resource.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

### Why use Synchronization?

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

**Types of synchronization:**

There are two types of synchronization that are as follows:

1. Process synchronization
2. Thread synchronization

Here we will be mainly focusing on thread synchronization.

Thread synchronization basically referstoThe concept of one thread execute at a time and the rest of the threads are in [waiting state](https://www.geeksforgeeks.org/lifecycle-and-states-of-a-thread-in-java/). This process is known as thread synchronization. It prevents the thread interference and inconsistency problem.

Synchronization is build using [locks or monitor](https://www.geeksforgeeks.org/difference-between-lock-and-monitor-in-java-concurrency/). In Java, a monitor is an object that is used as a mutually exclusive lock. Only a single thread at a time has the right to own a monitor. When a thread gets a lock then all other threads will get suspended which are trying to acquire the locked monitor. So, other threads are said to be waiting for the monitor, until the first thread exits the monitor. In a simple way, when a thread request a resource then that resource gets locked so that no other thread can work or do any modification until the resource gets released.

**Thread Synchronization are of two types:**

1. **Mutual Exclusive**
2. [**Inter-Thread Communication**](https://www.geeksforgeeks.org/inter-thread-communication-java/)

**A.** **Mutual Exclusive**

While sharing any resource, this will keep the thread interfering with one another i.e. mutual exclusive. We can achieve this via

* Synchronized Method
* Synchronized Block
* Static Synchronization

**Synchronized Method**

We can declare a method as synchronized usingthe [*“synchronized”*](https://www.geeksforgeeks.org/synchronized-in-java/)keyword. This will make the code written inside the method thread-safe so that no other thread will execute while the resource is shared.

**B. Synchronized Block**

If we declare a block as synchronized, only the code which is written inside that block is executed sequentially not the complete code. This is used when we want sequential access to some part of code or to synchronize some part of code.

**Syntax:**

synchronized (object reference)

{

 // Insert code here

}

**C. Static Synchronization**

In this, the synchronized method is declared as**“static”**which means the lock or monitor is applied on the class not on the object so that only one thread will access the class at a time.

# Java Networking

Java Networking is a concept of connecting two or more computing devices together so that we can share resources.

Java socket programming provides facility to share data between different computing devices.

### Advantage of Java Networking

1. Sharing resources
2. Centralize software management

The java.net package supports two protocols,

1. **TCP:** Transmission Control Protocol provides reliable communication between the sender and receiver. TCP is used along with the Internet Protocol referred as TCP/IP.
2. **UDP:** User Datagram Protocol provides a connection-less protocol service by allowing packet of data to be transferred along two or more nodes

Java Networking Terminology

The widely used Java networking terminologies are given below:

## Java Networking Terminology

The widely used Java networking terminologies are given below:

Skip Ad

1. IP Address
2. Protocol
3. Port Number
4. MAC Address
5. Connection-oriented and connection-less protocol
6. Socket

### 1) IP Address

IP address is a unique number assigned to a node of a network e.g. 192.168.0.1 . It is composed of octets that range from 0 to 255.

It is a logical address that can be changed.

### 2) Protocol

A protocol is a set of rules basically that is followed for communication. For example:

* TCP
* FTP
* Telnet
* SMTP
* POP etc.

### 3) Port Number

The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications.

The port number is associated with the IP address for communication between two applications.

### 4) MAC Address

MAC (Media Access Control) address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC address.

For example, an ethernet card may have a **MAC** address of 00:0d:83::b1:c0:8e.

### 5) Connection-oriented and connection-less protocol

In connection-oriented protocol, acknowledgement is sent by the receiver. So it is reliable but slow. The example of connection-oriented protocol is TCP.

But, in connection-less protocol, acknowledgement is not sent by the receiver. So it is not reliable but fast. The example of connection-less protocol is UDP.

### 6) Socket

A socket is an endpoint between two way communications.

Visit next page for Java socket programming.

## java.net package

The java.net package can be divided into two sections:

1. **A Low-Level API:** It deals with the abstractions of addresses i.e. networking identifiers, Sockets i.e. bidirectional data communication mechanism and Interfaces i.e. network interfaces.
2. **A High Level API:** It deals with the abstraction of URIs i.e. Universal Resource Identifier, URLs i.e. Universal Resource Locator, and Connections i.e. connections to the resource pointed by URLs.

The java.net package provides many classes to deal with networking applications in Java. A list of these classes is given below:

* Authenticator
* CacheRequest
* CacheResponse
* ContentHandler
* CookieHandler
* CookieManager
* DatagramPacket
* DatagramSocket
* DatagramSocketImpl
* InterfaceAddress
* JarURLConnection
* MulticastSocket
* InetSocketAddress
* InetAddress
* Inet4Address
* Inet6Address
* IDN
* HttpURLConnection
* HttpCookie
* NetPermission
* NetworkInterface
* PasswordAuthentication
* Proxy
* ProxySelector
* ResponseCache
* SecureCacheResponse
* ServerSocket
* Socket
* SocketAddress
* SocketImpl
* SocketPermission
* StandardSocketOptions
* URI
* URL
* URLClassLoader
* URLConnection
* URLDecoder
* URLEncoder
* URLStreamHandler

**List of interfaces available in java.net package:**

* ContentHandlerFactory
* CookiePolicy
* CookieStore
* DatagramSocketImplFactory
* FileNameMap
* SocketOption<T>
* SocketOptions
* SocketImplFactory
* URLStreamHandlerFactory
* ProtocolFamily

# **Java Socket Programming**

Java Socket programming is used for communication between the applications running on different JRE.

Java Socket programming can be connection-oriented or connection-less.

Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

The client in socket programming must know two information:

1. IP Address of Server, and
2. Port number.

Here, we are going to make one-way client and server communication. In this application, client sends a message to the server, server reads the message and prints it. Here, two classes are being used: Socket and ServerSocket. The Socket class is used to communicate client and server. Through this class, we can read and write message. The ServerSocket class is used at server-side. The accept() method of ServerSocket class blocks the console until the client is connected. After the successful connection of client, it returns the instance of Socket at server-side.



## Socket class

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

### Important methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public InputStream getInputStream() | returns the InputStream attached with this socket. |
| 2) public OutputStream getOutputStream() | returns the OutputStream attached with this socket. |
| 3) public synchronized void close() | closes this socket |

## ServerSocket class

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

### Important methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public Socket accept() | returns the socket and establish a connection between server and client. |
| 2) public synchronized void close() | closes the server socket |