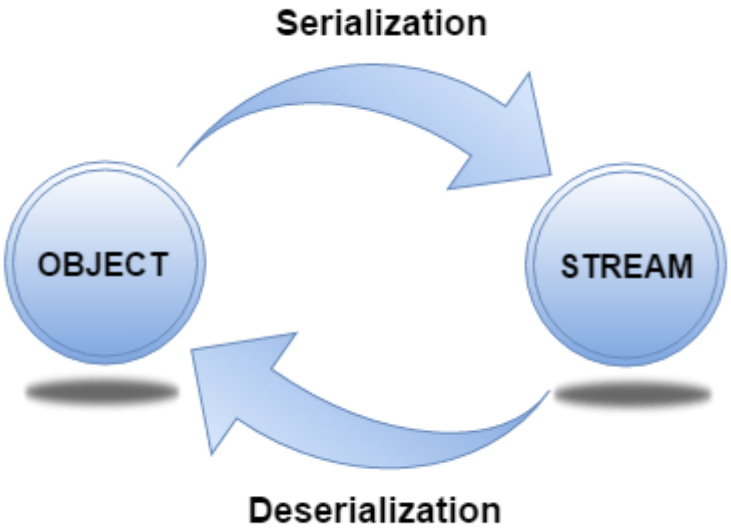
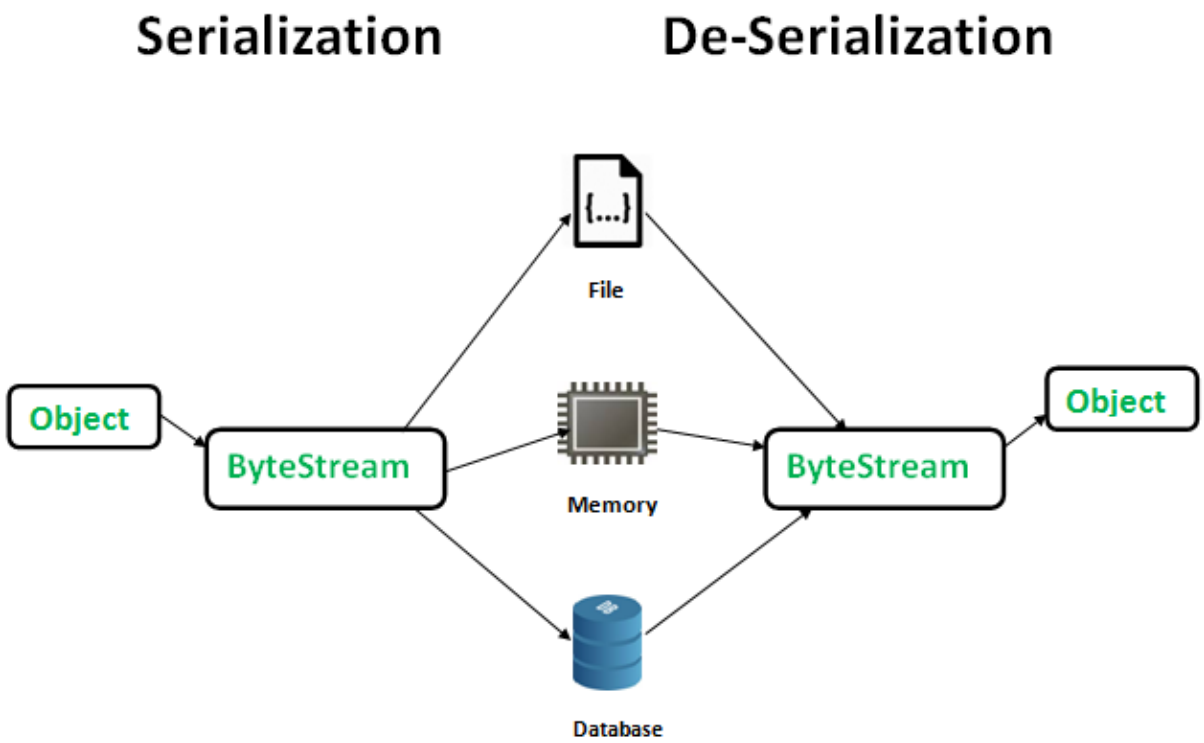
I/O Streams and Object Serialization

Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.

For serializing the object, we call the **writeObject()** method of *ObjectOutputStream* class, and for deserialization we call the **readObject()** method of *ObjectInputStream* class.





Advantages of Serialization

1. To save/persist state of an object.
2. To travel an object across a network.

```
import java.io.Serializable;

public class Student implements Serializable{

 int id;

 String name;

 public Student(int id, String name) {

  this.id = id;

  this.name = name;

 }

}
```

ObjectOutputStream class

The ObjectOutputStream class is used to write primitive data types, and Java objects to an OutputStream. Only objects that support the java.io.Serializable interface can be written to streams.

ObjectInputStream class

An ObjectInputStream deserializes objects and primitive data written using an ObjectOutputStream.

Generics in Java

It makes the code stable by detecting the bugs at compile time.

Before generics, we can store any type of objects in the collection, i.e., non-generic. Now generics force the java programmer to store a specific type of objects.

Advantage of Java Generics

There are mainly 3 advantages of generics. They are as follows:

1) Type-safety: We can hold only a single type of objects in generics. It doesn?t allow to store other objects

2) Type casting is not required: There is no need to typecast the object.

Before Generics, we need to type cast.

3) Compile-Time Checking: It is checked at compile time so problem will not occur at runtime.

Concurrency

Java Concurrency package covers **concurrency, multithreading**, and **parallelism** on the Java platform. Concurrency is the ability to run several or multi programs or applications in parallel. The **backbone** of Java concurrency is threads (a lightweight process, which has its own files and stacks and can access the shared data from other threads in the same process)

Thread States in Java

A thread is a program in execution created to perform a specific task. Life cycle of a Java thread starts with its birth and ends on its death.

The start() method of the Thread class is used to initiate the execution of a thread and it goes into runnable state and the sleep() and wait() methods of the Thread class sends the thread into non runnable state.

After non runnable state, thread again comes into runnable state and starts its execution. The run() method of thread is very much important. After executing the run() method, the lifecycle of thread is completed.

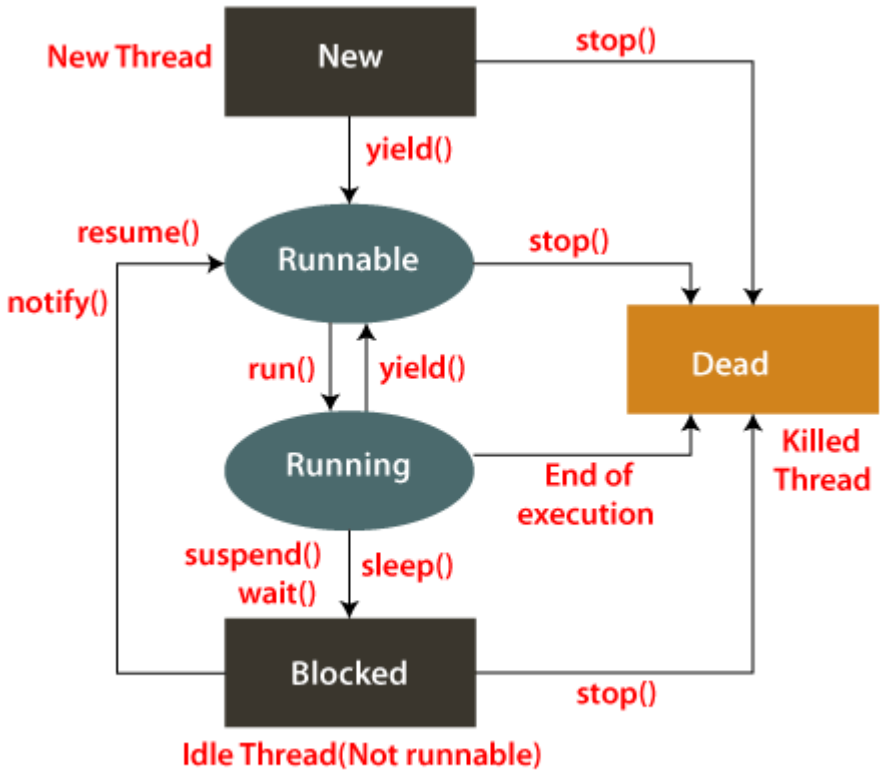


Fig: State Transition Diagram of a Thread

Thread States in Java

A thread is a path of execution in a program that goes through the following states of a thread. The five states are as follows:

1. New

2. Runnable
3. Running
4. Blocked (Non-runnable state)
5. Dead

New (Newborn State)

When an instance of the Thread class is created a new thread is born and is known to be in New-born state.

Runnable State

The second phase of a new-born thread is the execution phase. When the start() method is called on a the new instance of a thread, it enters into a runnable state. In the runnable state, thread is ready for execution and is waiting for availability of the processor (CPU time). There are many threads that are ready for execution, they all are waiting in a queue (line).

Running State

Running means Processor (CPU) has allocated time slot to thread for its execution. When thread scheduler selects a thread from the runnable state for execution, it goes into running state.

A running thread may give up its control in any one of the following situations and can enter into the blocked state.

1. When sleep() method is invoked on a thread to sleep for specified time period, the thread is out of queue during this time period. The thread again reenters into the runnable state as soon as this time period is elapsed.
2. When a thread is suspended using suspend() method for some time in order to satisfy some conditions. A suspended thread can be revived by using resume() method.
3. When wait() method is called on a thread to wait for some time. The thread in wait state can be run again using notify() or notifyAll() method.

Blocked State

A thread is considered to be in the blocked state when it is suspended, sleeping, or waiting for some time in order to satisfy some condition.

Dead State

A thread dies or moves into dead state automatically when its run() method completes the execution of statements. That is, a thread is terminated or dead when a thread comes out of run() method. A thread can also be dead when the stop() method is called.

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.

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

1. Process Synchronization
2. Thread Synchronization

Process Synchronization in Java

Process Synchronization is a technique used to coordinate the execution of multiple processes. It ensures that the shared resources are safe and in order.

2. Thread Synchronization in Java

Thread Synchronization is used to coordinate and ordering of the execution of the threads in a multi-threaded program. There are two types of thread synchronization are mentioned below:

- Mutual Exclusive
- Cooperation (Inter-thread communication in Java)

Mutual Exclusive

Mutual Exclusive helps keep threads from interfering with one another while sharing data. There are three types of Mutual Exclusive mentioned below:

- Synchronized method.
- Synchronized block.
- Static synchronization.

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:

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.