**4. MULTITHREADING fundamentals**

**Thread :** Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

**Multithreading :** It is a process of executing multiple threads simultaneously.

* But we use multithreading than multiprocessing because threads share a common 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.

**Multitasking**

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved by two ways:

1. Process-based Multitasking(Multiprocessing)
2. Thread-based Multitasking(Multithreading)

**1) Process-based Multitasking (Multiprocessing)**

Each process have its own address in memory i.e. each process allocates separate memory area.

Process is heavyweight.

Cost of communication between the process is high.

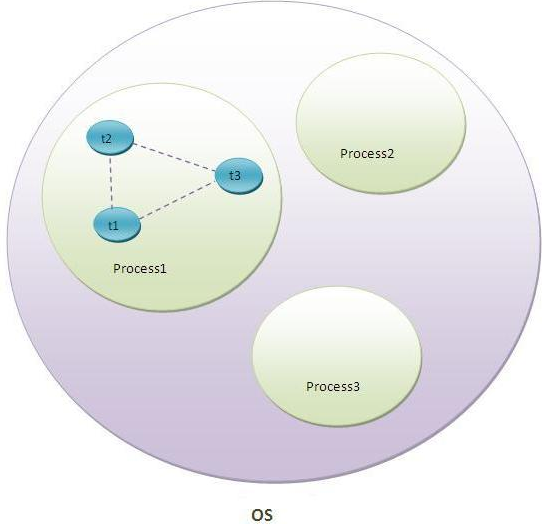
Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.

**2) Thread-based Multitasking (Multithreading)**

Threads share the same address space.

Thread is lightweight.

Cost of communication between the thread is low.

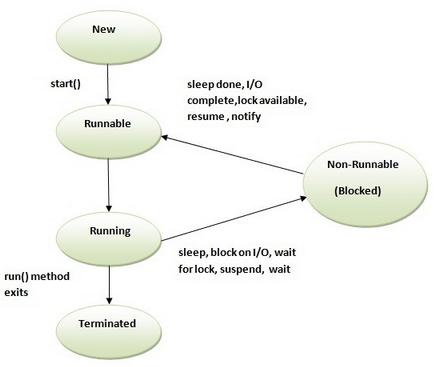
****

**Life cycle of a Thread (Thread States)**

A thread can be in one of the five states. According to Sun microsystem, there is only 4 states in **thread life cycle in java** new, runnable, non-runnable and terminated. There is no running state.

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminated



**1) New**

The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

**2) Runnable**

The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

**3) Running**

The thread is in running state if the thread scheduler has selected it.

**4) Non-Runnable (Blocked)**

This is the state when the thread is still alive, but is currently not eligible to run.

**5) Terminated**

A thread is in terminated or dead state when its run() method exits.

**How to create thread**

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.
3. **Thread class:**

Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface. **Commonly used Constructors of Thread class:**

Thread()

Thread(String name)

Thread(Runnable r)

Thread(Runnable r,String name)

1. **Runnable interface:**

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run(). **public void run():** is used to perform action for a thread.

**1) Java Thread Example by extending Thread class**

class Multi extends Thread{

public void run(){

System.out.println("thread is running...");

}

public static void main(String args[]){

Multi t1=new Multi();

t1.start();

 }

}

Output:thread is running...

**2) Java Thread Example by implementing Runnable interface**

class Multi3 implements Runnable{

public void run(){

System.out.println("thread is running...");

}

public static void main(String args[]){

Multi3 m1=new Multi3();

Thread t1 =new Thread(m1);

t1.start();

 }

}

Output:thread is running...

**Synchronization**

At times when more than one thread try to access a shared resource, we need to ensure that resource will be used by only one thread at a time. The process by which this is achieved is called **synchronization**. The synchronization keyword in java creates a block of code referred to as critical section.

**General Syntax :**

synchronized (object) { //statement to be synchronized }

Every Java object with a critical section of code gets a lock associated with the object. To enter critical section a thread need to obtain the corresponding object's lock.

**Inter-thread communication in Java**

**Inter-thread communication** or **Co-operation** is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of **Object class**:

* wait()
* notify()
* notifyAll()
* **1) wait() method**

Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

* **2) notify() method**

Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. Syntax:

public final void notify()

* **3) notifyAll() method**

Wakes up all threads that are waiting on this object's monitor. Syntax:

public final void notifyAll()

Java provides complete control over multithreaded program. You can develop a multithreaded program which can be suspended, resumed, or stopped completely based on your requirements. There are various static methods which you can use on thread objects to control their behavior. Following table lists down those methods −

**public void suspend()**

This method puts a thread in the suspended state and can be resumed using resume() method.

**public void stop()**

This method stops a thread completely.

**public void resume()**

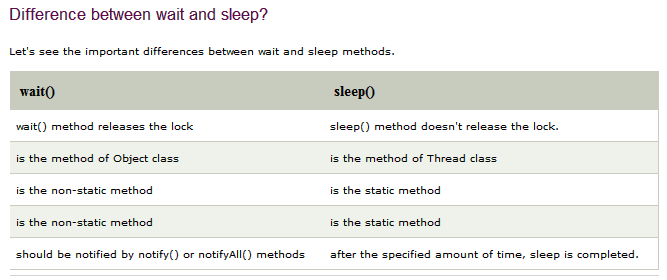
This method resumes a thread, which was suspended using suspend() method.

**public void wait()**

Causes the current thread to wait until another thread invokes the notify().

**public void notify()**

Wakes up a single thread that is waiting on this object's monitor.



**Thread Priorities:-**

1. Every Thread in java has some property. It may be default priority provided be the JVM or customized priority provided by the programmer.

2. The valid range of thread priorities is 1 – 10. Where one is lowest priority and 10 is highest priority.

3. The default priority of main thread is 5. The priority of child thread is inherited from the parent.

4. Thread defines the following constants to represent some standard priorities.

5. Thread Scheduler will use priorities while allocating processor the thread which is having highest priority will get chance first and the thread which is having low priority.

6. If two threads having the same priority then we can’t expect exact execution order it depends upon Thread Scheduler.

7. The thread which is having low priority has to wait until completion of high priority threads.

8. Three constant values for the thread priority.

**a. MIN\_PRIORITY = 1**

**b. NORM\_PRIORITY = 5**

**c. MAX\_PRIORITY = 10**

Thread class defines the following methods to get and set priority of a Thread.

**a. Public final int getPriority()**

**b. Public final void setPriority(int priority)**

Here ‘priority’ indicates a number which is in the allowed range of 1 – 10. Otherwise we will get

Runtime exception saying “IllegalArgumentException.

1. **Exception Handling in Java**

The **exception handling in java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained

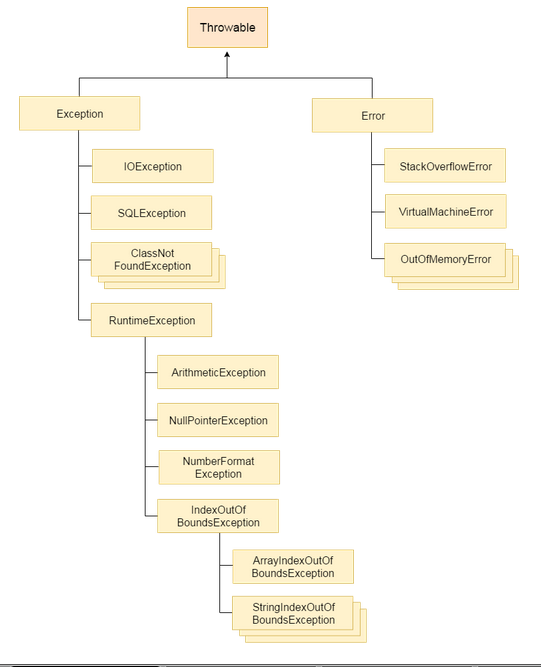
**What is exception**

**Dictionary Meaning:** Exception is an abnormal condition.

In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.  
**What is exception handling**  
Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.

**Advantage of Exception Handling**

The core advantage of exception handling is **to maintain the normal flow of the application**. Exception normally disrupts the normal flow of the application that is why we use exception handling.  
**Hierarchy of Java Exception classes**



**Types of Exception**

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

**Difference between checked and unchecked exceptions**

**1) Checked Exception**

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.  
**2) Unchecked Exception**

The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

**3) Error**

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

**Common scenarios where exceptions may occur**

There are given some scenarios where unchecked exceptions can occur. They are as follows:

**1) Scenario where ArithmeticException occurs**

If we divide any number by zero, there occurs an ArithmeticException.

int a=50/0;//ArithmeticException

**2) Scenario where NullPointerException occurs**

If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

String s=null;

System.out.println(s.length());//NullPointerException

**3) Scenario where NumberFormatException occurs**

The wrong formatting of any value, may occur NumberFormatException. Suppose I have a string variable that have characters, converting this variable into digit will occur NumberFormatException.

String s="abc";

int i=Integer.parseInt(s);//NumberFormatException

**4) Scenario where ArrayIndexOutOfBoundsException occurs**

If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:

int a[]=new int[5];

a[10]=50; //ArrayIndexOutOfBoundsException

**Java Exception Handling Keywords**

There are 5 keywords used in java exception handling.

1. try
2. catch
3. finally
4. throw
5. throws
6. **Java try block**

Java try block is used to enclose the code that might throw an exception. It must be used within the method.

Java try block must be followed by either catch or finally block.

**Syntax of java try-catch**

try{

//code that may throw exception

}catch(Exception\_class\_Name ref){}

**Syntax of try-finally block**

try{

//code that may throw exception

}finally{}

1. **Java catch block**

Java catch block is used to handle the Exception. It must be used after the try block only.

You can use multiple catch block with a single try.

**Problem without exception handling**

Let's try to understand the problem if we don't use try-catch block.

public class Testtrycatch1{

  public static void main(String args[]){

      int data=50/0;//may throw exception

      System.out.println("rest of the code...");

}

}

Output:

Exception in thread main java.lang.ArithmeticException:/ by zero

**Solution by exception handling**

Let's see the solution of above problem by java try-catch block.

public class Testtrycatch2{

  public static void main(String args[]){

   try{

      int data=50/0;

   }catch(ArithmeticException e){System.out.println(e);}

   System.out.println("rest of the code...");

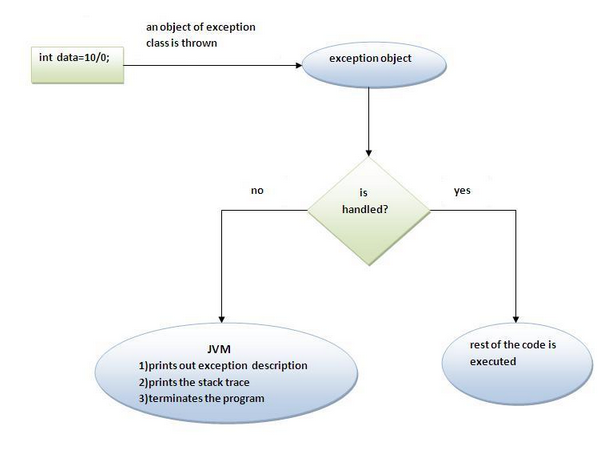
}

}

Output:

Exception in thread main java.lang.ArithmeticException:/ by zero rest of the code...

**Internal working of java try-catch block**



1. **SERVLETS**

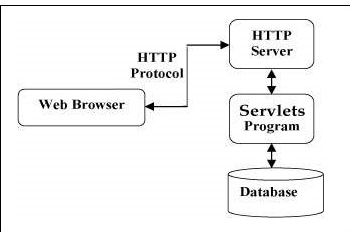
Servlets provide a component-based, platform-independent method for building Webbased applications, without the performance limitations of CGI programs. Servlets have access to the entire family of Java APIs, including the JDBC API to access enterprise databases.

**What are Servlets?**

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI)

**Servlets Architecture**

****

**Servlets perform the following major tasks −**

Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.

Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.

Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.

Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.

Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

**Servlets Packages(API)**

Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.

Servlets can be created using the javax.servlet and javax.servlet.http packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.

**Java Database Connectivity (JDBC):** It is an application program interface (API) specification for database-independent connectivity between the Java programming language and a wide range of databases. It provides methods for querying and updating data in a database.

**Open Database Connectivity (ODBC)**: It is an application programming interface (API) for accessing a database. ODBC enables applications to access data from a variety of database systems regardless of the platform it is on or Database Management Systems (DBMS) it uses.

**Setting up Java Development Kit**

This step involves downloading an implementation of the Java Software Development Kit (SDK) and setting up PATH environment variable appropriately.

You can download SDK from Oracle's Java site − [Java SE Downloads](https://www.oracle.com/technetwork/java/javase/downloads/index.html).

Once you download your Java implementation, follow the given instructions to install and configure the setup. Finally set PATH and JAVA\_HOME environment variables to refer to the directory that contains java and javac, typically java\_install\_dir/bin and java\_install\_dir respectively.

If you are running Windows and installed the SDK in C:\jdk1.8.0\_65, you would put the following line in your C:\autoexec.bat file.

set PATH = C:\jdk1.8.0\_65\bin;%PATH% set JAVA\_HOME = C:\jdk1.8.0\_65

**Setting up Web Server − Tomcat**

A number of Web Servers that support servlets are available in the market. Some web servers are freely downloadable and Tomcat is one of them.

Apache Tomcat is an open source software implementation of the Java Servlet and Java Server Pages technologies and can act as a standalone server for testing servlets and can be integrated with the Apache Web Server. Here are the steps to setup Tomcat on your machine −

**Servlets - Life Cycle**

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

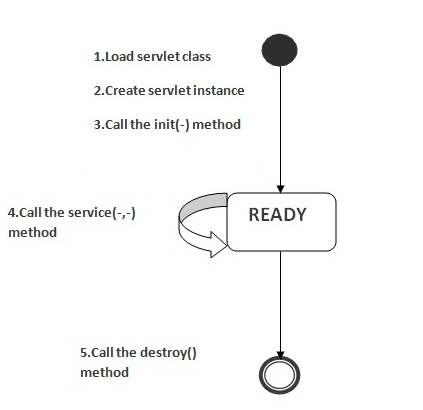
Servlet class is loaded.

Servlet instance is created.

init method is invoked.

service method is invoked.

destroy method is invoked.



**1) Servlet class is loaded**

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

**2) Servlet instance is created**

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

**3) init method is invoked**

The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

public void init(ServletConfig config) throws ServletException

**4) service method is invoked**

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

public void service(ServletRequest request, ServletResponse response)

  throws ServletException, IOException

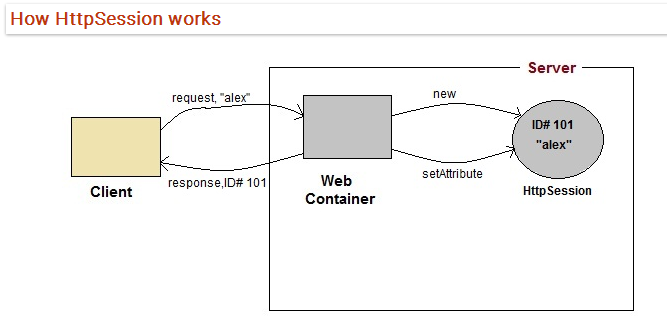
**5) destroy method is invoked**

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:

public void destroy()

**What is HttpSession?**

HttpSession object is used to store entire session with a specific client. We can store, retrieve and remove attribute from HttpSession object. Any servlet can have access to HttpSession object throughout the getSession() method of the HttpServletRequest object.



On client's first request, the Web Container generates a unique session ID and gives it back to the client with response. This is a temporary session created by web container.

The client sends back the session ID with each request. Making it easier for the web container to identify where the request is coming from.

The Web Container uses this ID, finds the matching session with the ID and associates the session with the request.

**COOKIES**

Using Cookies for Session Management

Cookies are small pieces of information that are sent in response from the web server to the client. Cookies are the simplest technique used for storing client state.

