

INTRODUCTION

210301301

CORE JAVA

210301305

PRACTICAL ON CORE JAVA



Unit – 4

Exception Handling in Java

Index

- Exception Handling
 - Introduction
 - Exception Types
 - Try...Catch
 - Throw keyword
 - Throws keyword
 - Finally block
 - Multi-catch
 - User defined exception

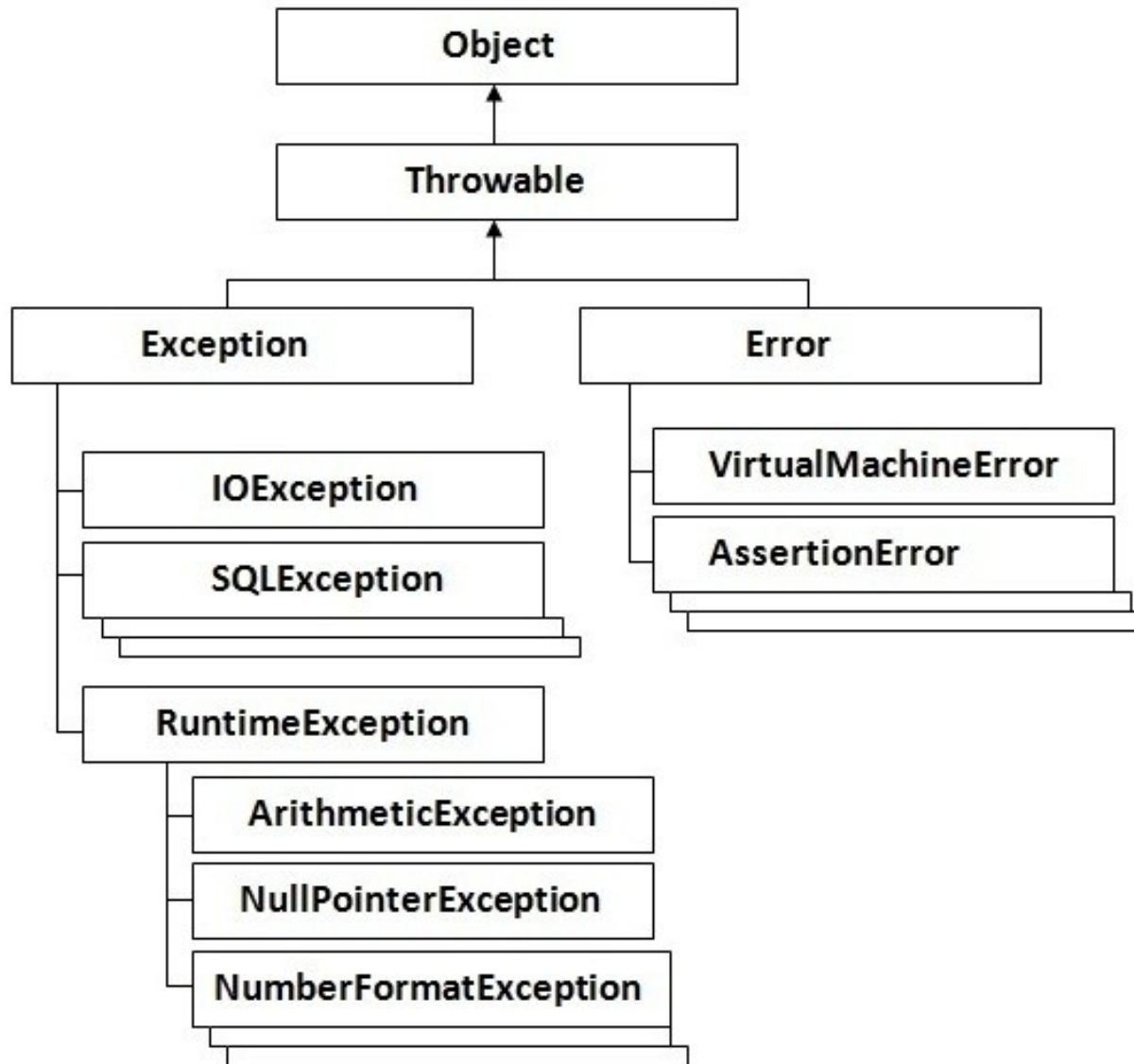
Index

- Multi Threading
 - Introduction
 - Thread Life cycle
 - Java.lang.Thread
 - Main Thread
 - Creation of new Thread class

Exception Handling

- 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.
- **Definition:**
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 ClassNotFoundException, IO, SQL, Remote etc.
- The core advantage of exception handling is to maintain the normal flow of the application.

Exception Handling



Types of Exception

- The sun microsystem says there are three types of exceptions:
 - Checked Exception
 - Unchecked Exception
 - Error

Types of Exception

- **Checked Exception:**

Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler.

e.g. IOException, SQLException etc.

Checked exceptions are checked at compile-time.

- **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.

- **Error**

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

Some scenarios where exceptions may occur

1) Scenario where **ArithmeticException** occurs

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

```
int a=50/0; //ArithmeticException
```

Exc1.java

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
```

Exc2.java

Some scenarios where exceptions may occur

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
```



There are 5 keywords used in java exception handling.

- try
- catch
- finally
- throw
- throws

Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

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{}
```

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.
- At a time only one Exception is occurred and at a time only one catch block is executed.
- All catch blocks must be ordered from most specific to most general i.e. catch for `ArithmeticException` must come before catch for `Exception` .

Exc3.java Exc4.java

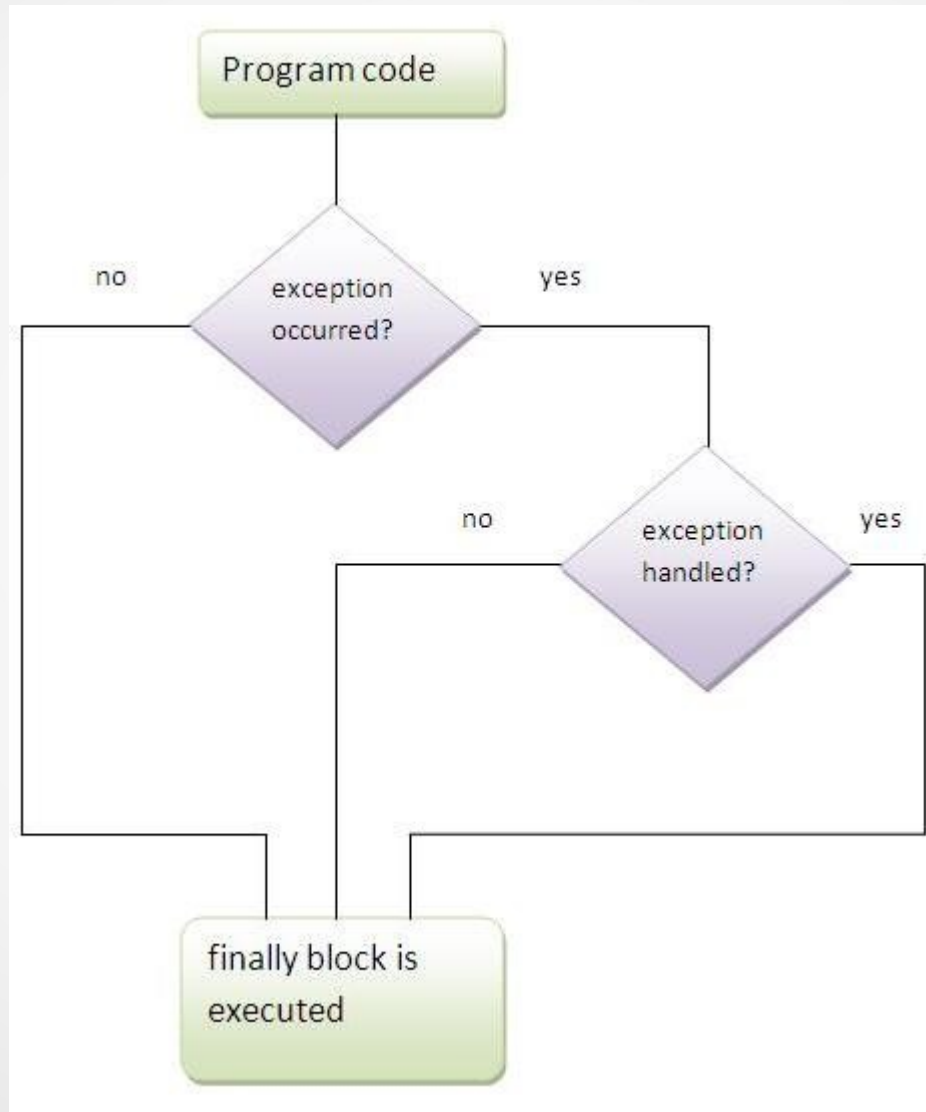
Java finally block

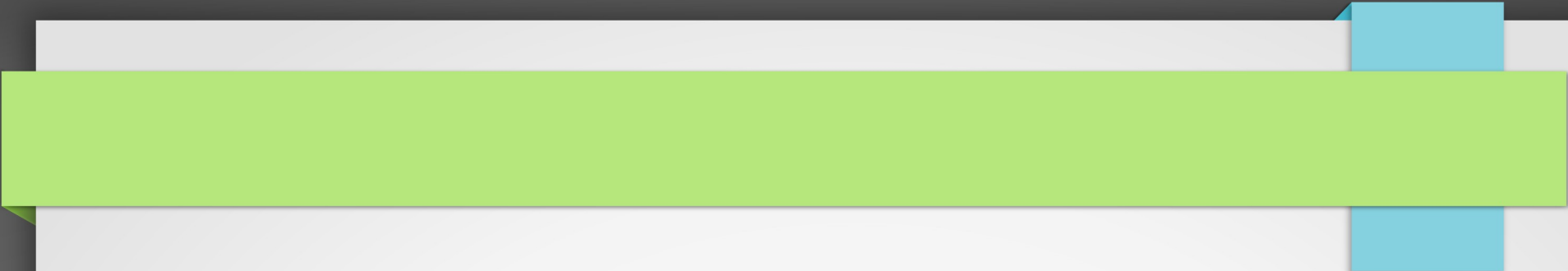
- Java finally block is a block that is used to execute important code such as closing connection, stream etc.
- Java finally block is always executed whether exception is handled or not.
- Java finally block follows try or catch block.
- If you don't handle exception, before terminating the program, JVM executes finally block(if any).
- Finally block in java can be used to put "cleanup" code such as closing a file, closing connection etc.

Exc5.java

Exc6.java

Java finally block



- 
- For each try block there can be zero or more catch blocks, but only one finally block.
 - The finally block will not be executed if program exits. (either by calling `System.exit()` or by causing a fatal error that causes the process to abort).

Java throw keyword

- The Java throw keyword is used to explicitly throw an exception.
- We can throw either checked or unchecked exception in java by throw keyword.
- The throw keyword is mainly used to throw custom exception.
- The syntax of java throw keyword is given below.

throw exception;

Exc7.java

Exc7_1.java

Exc7_2.java

Java throws keyword

- The Java throws keyword is used to declare an exception.
- It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.
- Exception Handling is mainly used to handle the checked exceptions.
- If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.
- Syntax of java throws

```
return_type method_name() throws exception_class_name{
```

```
//method code
```

```
}
```

Exc8.java

Exc8_1.java

UNIT 4

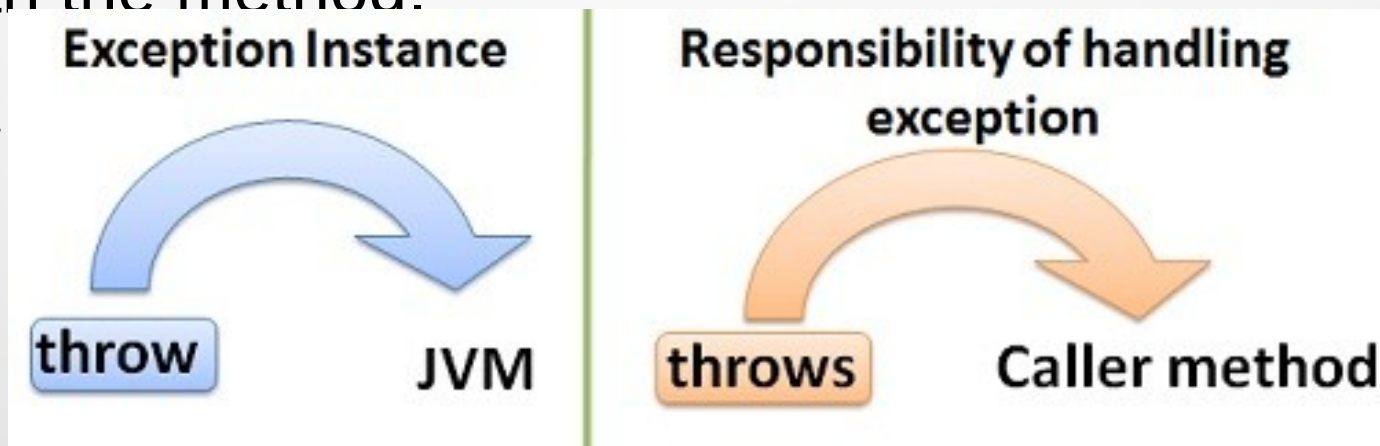
Java throws keyword

- Which exception should be declared
- Ans) checked exception only, because:
 - unchecked Exception: under your control so correct your code.
 - error: beyond your control e.g. you are unable to do anything if there occurs `VirtualMachineError` or `StackOverflowError`.
- **Advantage of Java throws keyword**
 - Now Checked Exception can be propagated (forwarded in call stack).
 - It provides information to the caller of the method about the exception.

- If you are calling a method that declares an exception, you must either caught or declare the exception.
- There are two cases:
 - Case1: You caught the exception i.e. handle the exception using try/catch.
 - Case2: You declare the exception i.e. specifying throws with the method.

Exc9_1.java

Exc10.java



Difference between throw and throws

throw	throws
1. Java throw keyword is used to explicitly throw an exception	1. Java throws keyword is used to declare an exception.
2. <pre>void m(){ throw new ArithmeticException("sorry"); }</pre>	2. <pre>void m()throws ArithmeticException{ //method code }</pre>
3. Checked exception cannot be propagated using throw only.	3. Checked exception can be propagated with throws.
4. Throw is followed by an instance.	4. Throw is followed by a class.
5. Throw is used within the method.	5. Throws is used with the method signature.
6. You cannot throw multiple exceptions.	6. You can declare multiple exceptions e.g. <pre>public void method()throws IOException,SQLException.</pre>

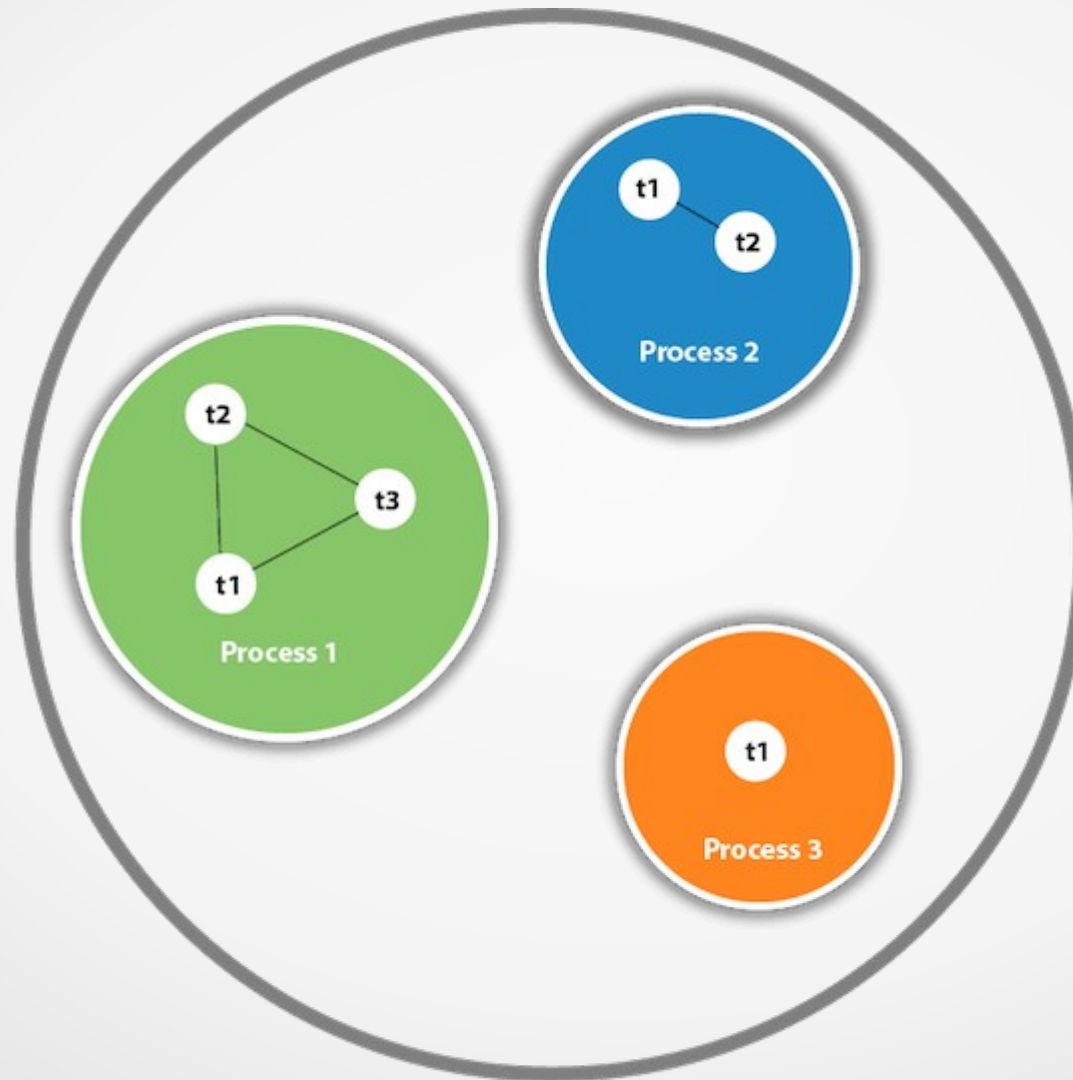
Java Custom Exception

- If you are creating your own Exception that is known as custom exception or user-defined exception.
- Java custom exceptions are used to customize the exception according to user need.
- By the help of custom exception, you can have your own exception and message.

Multithreading in Java

- Multithreading in java is a **process of executing multiple threads simultaneously.**
- **Thread is basically a lightweight sub-process, a smallest unit of processing.**
- **Multiprocessing and multithreading, both are used to achieve multitasking.**
- 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.

Multithreading in Java



Advantages of Java Multithreading

- It **doesn't block the user** because threads are independent and you can perform multiple operations at same time.
- You **can perform many operations together so it saves time.**
- **Threads are independent** so it doesn't affect other threads if exception occur in a single thread.

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)**

Multitasking

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.

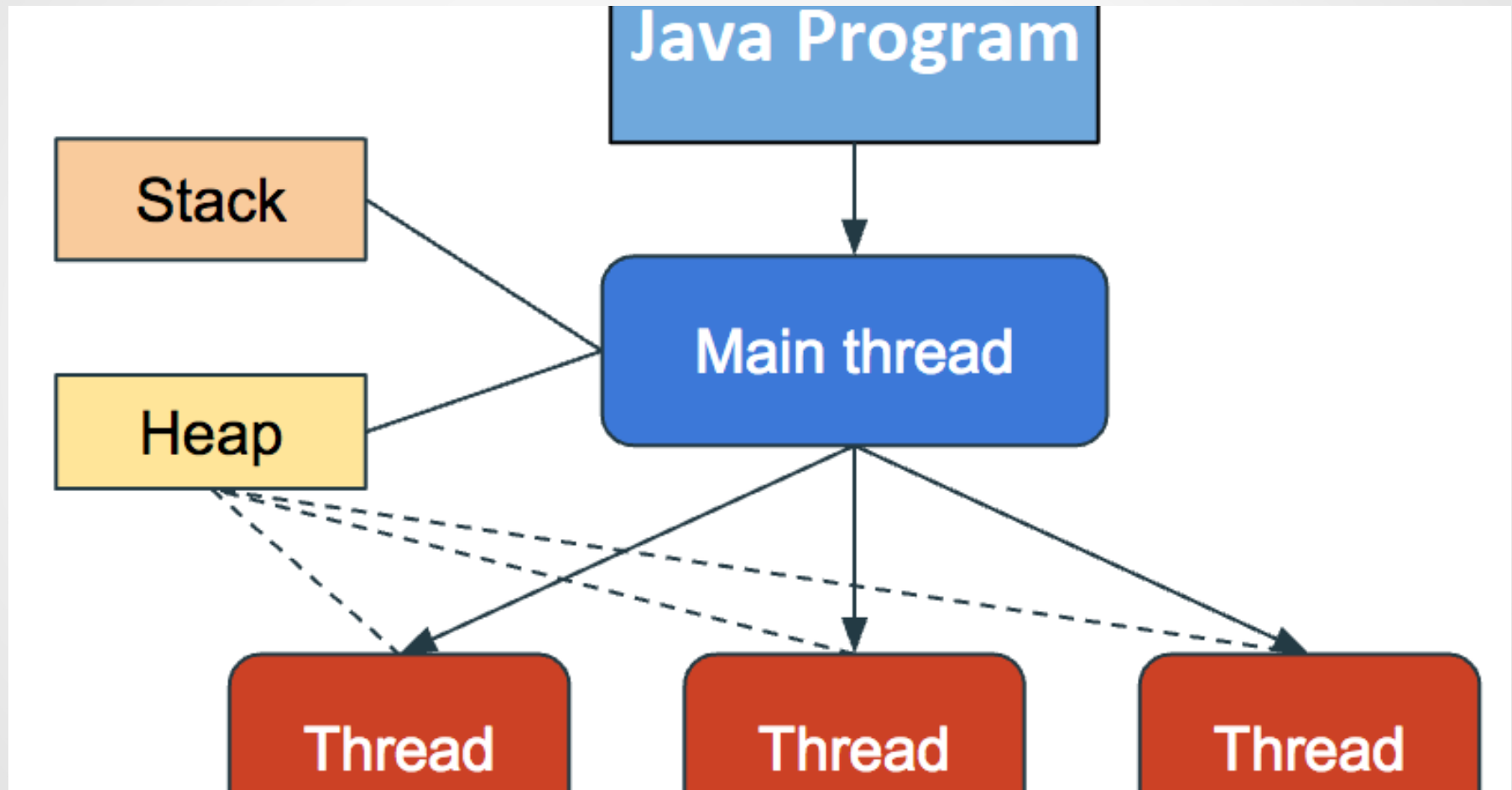
2) **Thread-based Multitasking (Multithreading)**

- Threads share the same address space.
- Thread is lightweight.
- Cost of communication between the thread is low.
- At least one process is required for each thread.

What is Thread in java?

- **A thread is a lightweight sub process, a 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 shares a common memory area.
- At a time one thread is executed only.

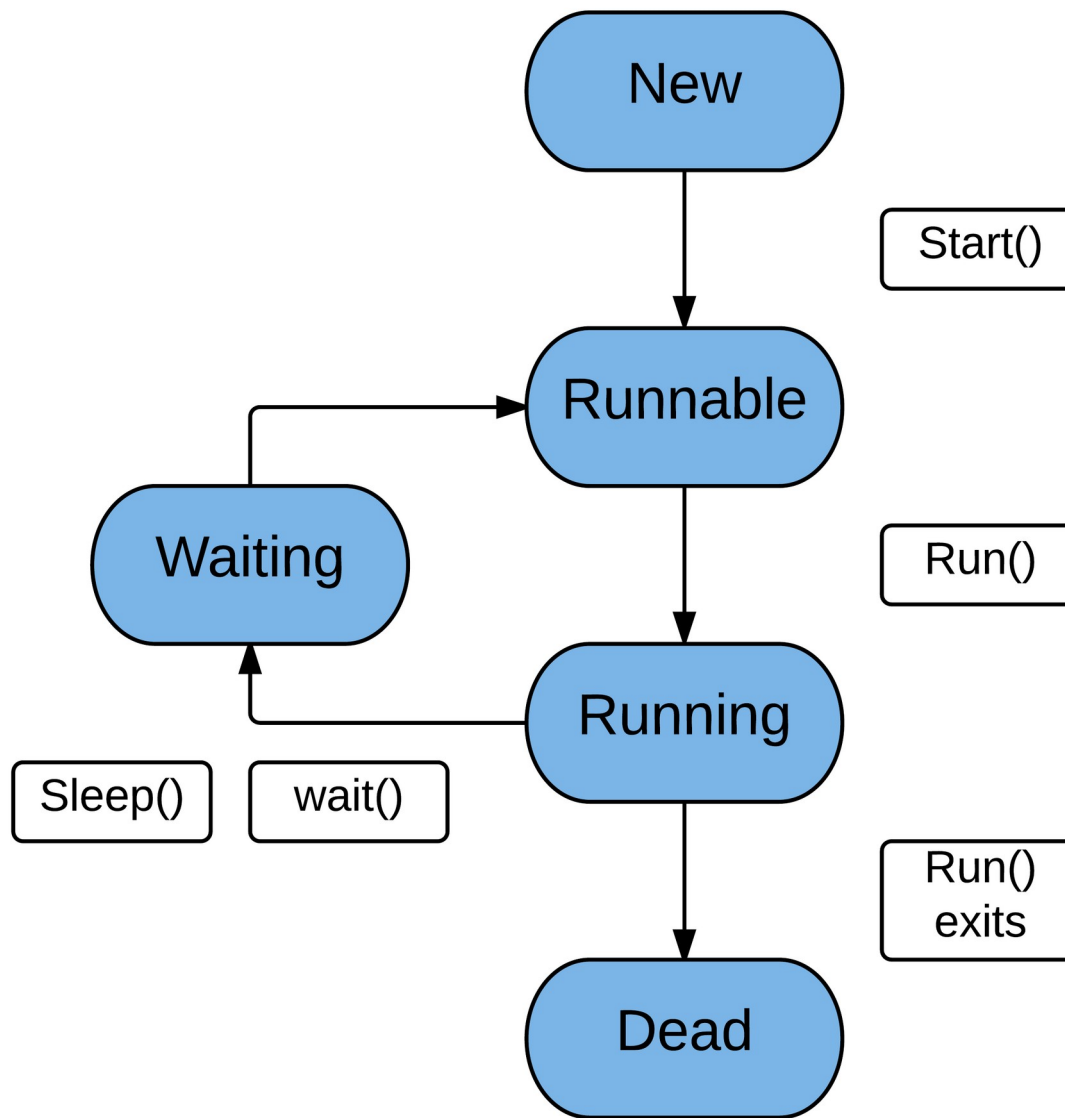
What is Thread in java



Life cycle of a Thread (Thread States)

- A thread can be in one of the five states.
- The life cycle of the thread in java is controlled by JVM.
- The java thread states are as follows:
 - New
 - Runnable
 - Running
 - Non-Runnable (Blocked)
 - Terminated

Life cycle



Life cycle of a Thread

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)

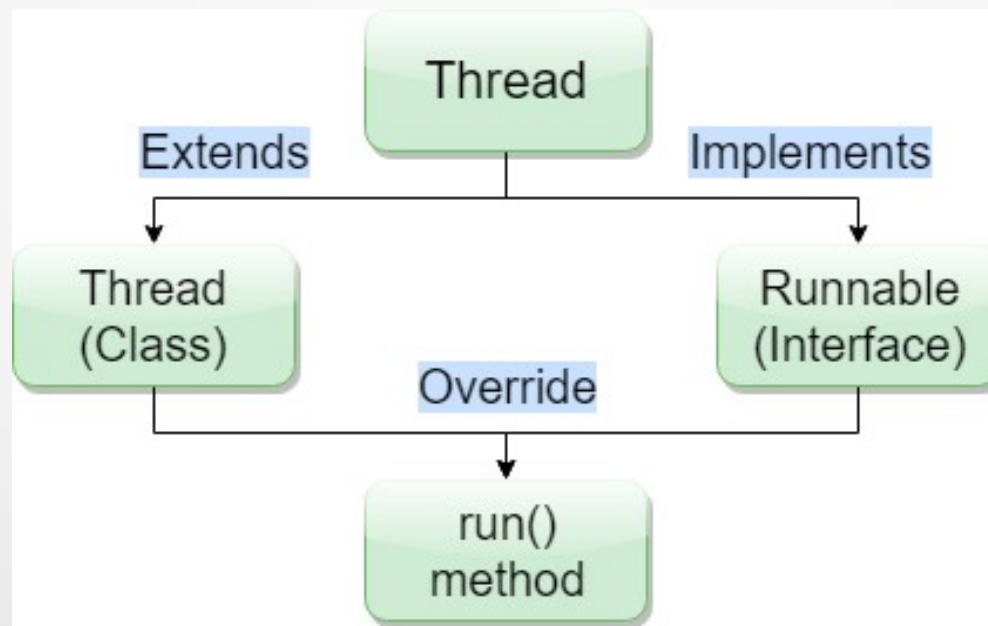
- This is the state when the thread is still alive, but is currently not eligible to run.

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:
- **By extending Thread class**
- **By implementing Runnable interface.**



How to create thread

- **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)

Thread1.java

Commonly used methods of Thread class:

- **public void run()**

is used to perform action for a thread (entry point for a thread).

- **public void start()**

starts the execution of the thread. JVM calls the run() method on the thread.

Thread6.java

Thread3.java

- **public void sleep(long milliseconds)**

Thread5.java

Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.

- **public void join(long milliseconds)**

Thread1_1.java

waits for a thread to die for the specified milliseconds.

Commonly used methods of Thread class:

- **public int getPriority()**
returns the priority of the thread.
- **public int setPriority(int priority)**
changes the priority of the thread.
- **public String getName()**
returns the name of the thread.
- **public void setName(String name)**
changes the name of the thread.
- **public Thread currentThread()**
returns the reference of currently executing thread.
- **public int getId()**
returns the id of the thread.

Thread1_2.java

Thread1_3.java

Thread1_4.java

Commonly used methods of Thread class:

- **public Thread.State getState()**

returns the state of the thread.

Thread1_5.java

- **public boolean isAlive()**

tests if the thread is alive.

Thread1_6.java

- **public void yield()**

causes the currently executing thread object to temporarily pause and allow other threads to execute.

Thread1_7.java

- **public void suspend()**

is used to suspend the thread.

Thread1_8.java

- **public void resume()**

is used to resume the suspended thread.

- **public void stop()**

is used to stop the thread.

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.
- The most common use case of the Runnable interface is when we want only to override the run method.
- When a thread is started by the object of any class which is implementing Runnable, then it invokes the run method in the separately executing thread.

Runnable interface:

Steps to create a new Thread using Runnable :

1. Create a Runnable implementer and implement run() method.
2. Instantiate Thread class and pass the implementer to the Thread, Thread has a constructor which accepts Runnable instance.
3. Invoke start() of Thread instance, start internally calls run() of the implementer. Invoking start(), creates a new Thread which executes the code written in run().

Calling run() directly doesn't create and start a new Thread, it will run in the same thread. To start a new line of execution, call start() on the thread.

Differences between Thread class and Runnable interface

- The significant differences between extending Thread class and implementing Runnable interface:
- When we extend Thread class, we can't extend any other class even we require and When we implement Runnable, we can save a space for our class to extend any other class in future or now.
- When we extend Thread class, each of our thread creates unique object and associate with it. When we implements Runnable, it shares the same object to multiple threads.

Priority of a Thread (Thread Priority)

- Each thread have a priority.
- Priorities are represented by a number between 1 and 10.
- In most cases, thread scheduler schedules the threads according to their priority.
- But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.
- 3 constants defined in Thread class:
- `public static int MIN_PRIORITY`
- `public static int NORM_PRIORITY`
- `public static int MAX_PRIORITY`
- Default priority of a thread is 5 (NORM_PRIORITY).
- The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.