GLS UNIVERSITY

SEM – III

0301301 - CORE JAVA

Dr. Disha Shah
Prof. Vidhi Thakkar

Index

Exception Handling

- Introduction
- ExceptionTypes
- Handling Techniques
  - try..catch
  - throw keyword
  - throws keyword
  - finally keyword
  - Multi-catch
  - User-defined exception

Multi-threading

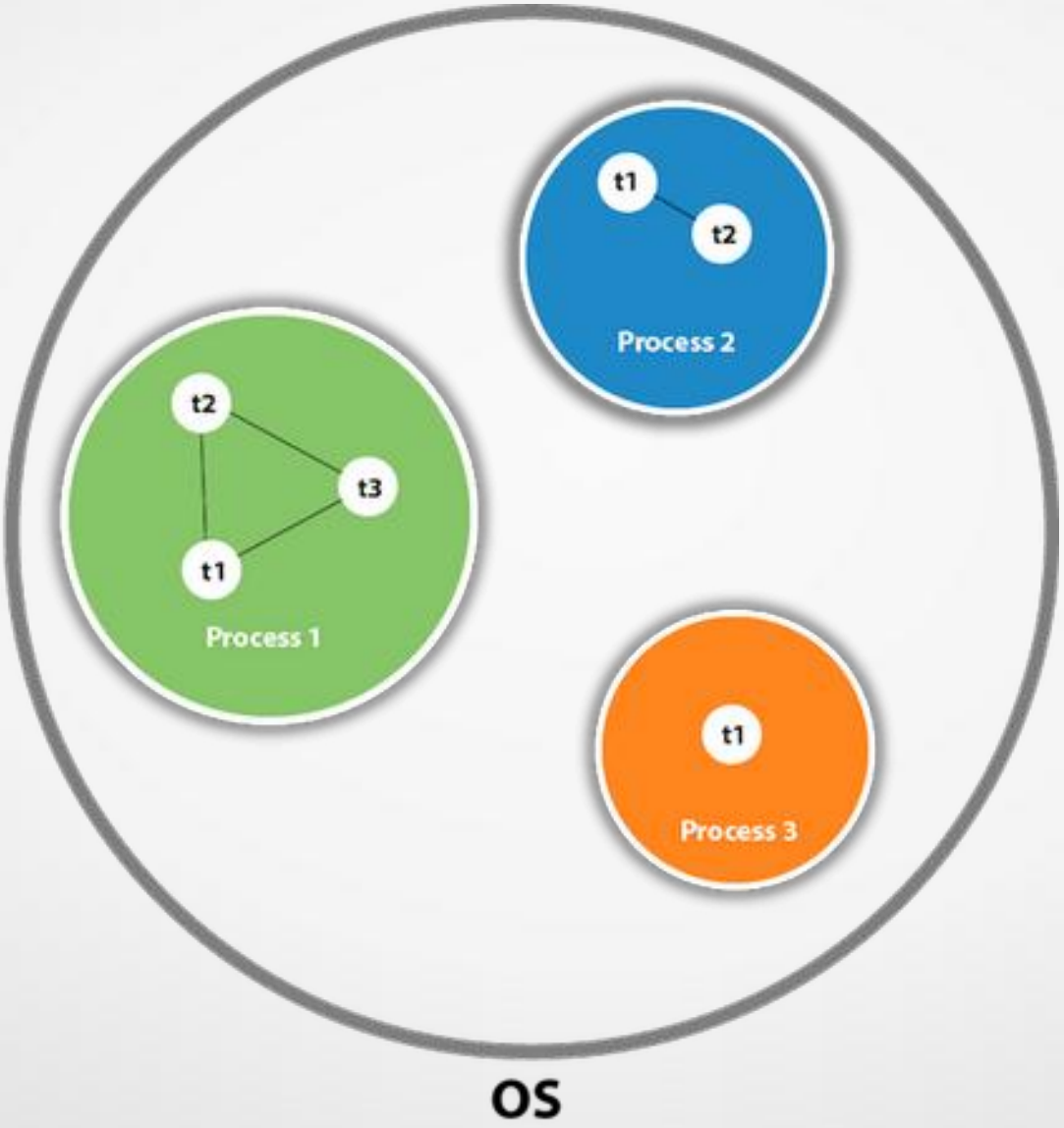
- Introduction
- Thread Life cycle
- java.lang.Thread
- Main thread
- Creation of New Thread
  - By inheriting Thread class
  - By implementing Runnable interface
  - Thread Priorities

Unit – 4
Multithreading in Java

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.

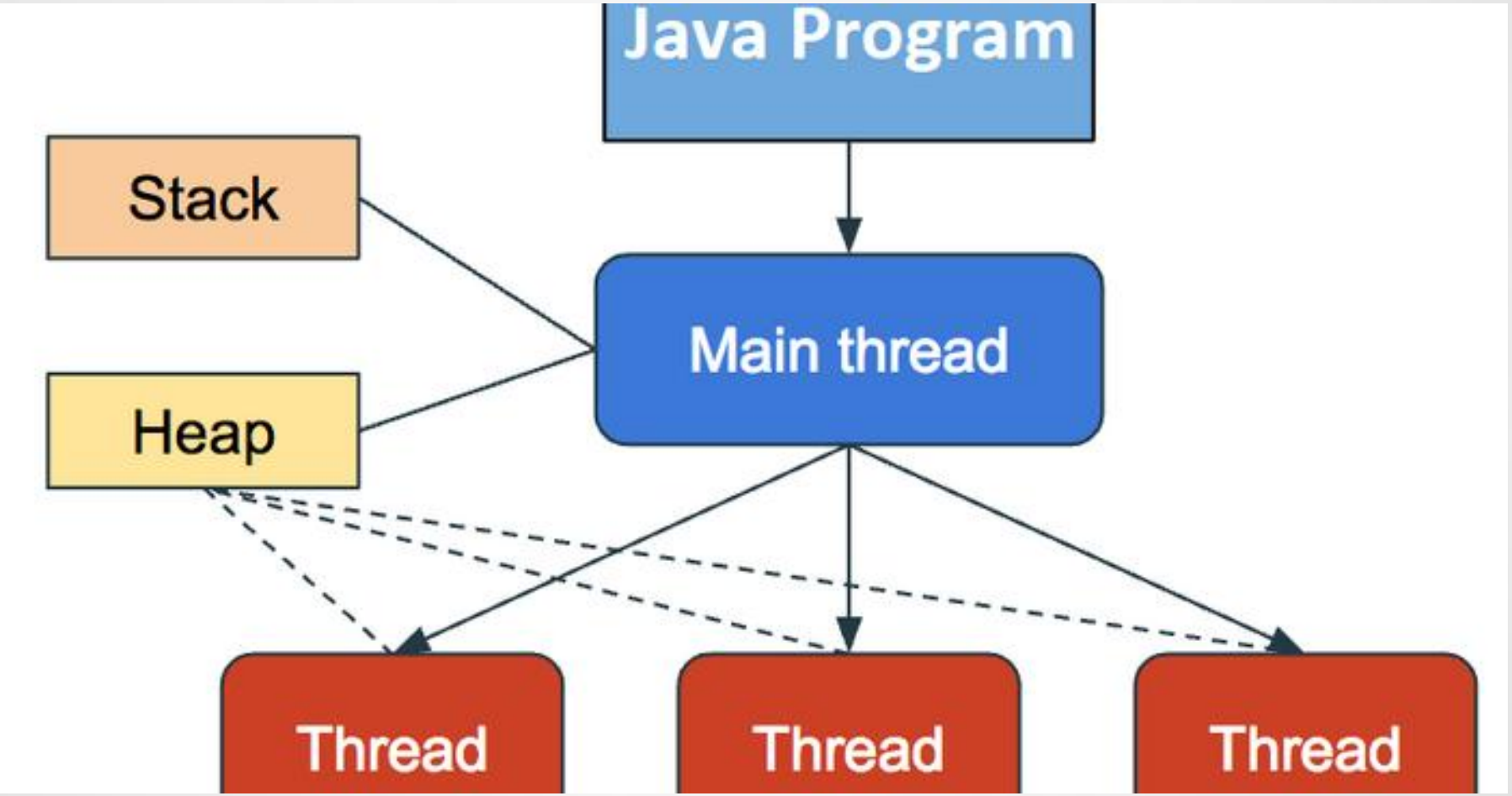
2) Thread-based Multitasking (Multithreading)

- 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.

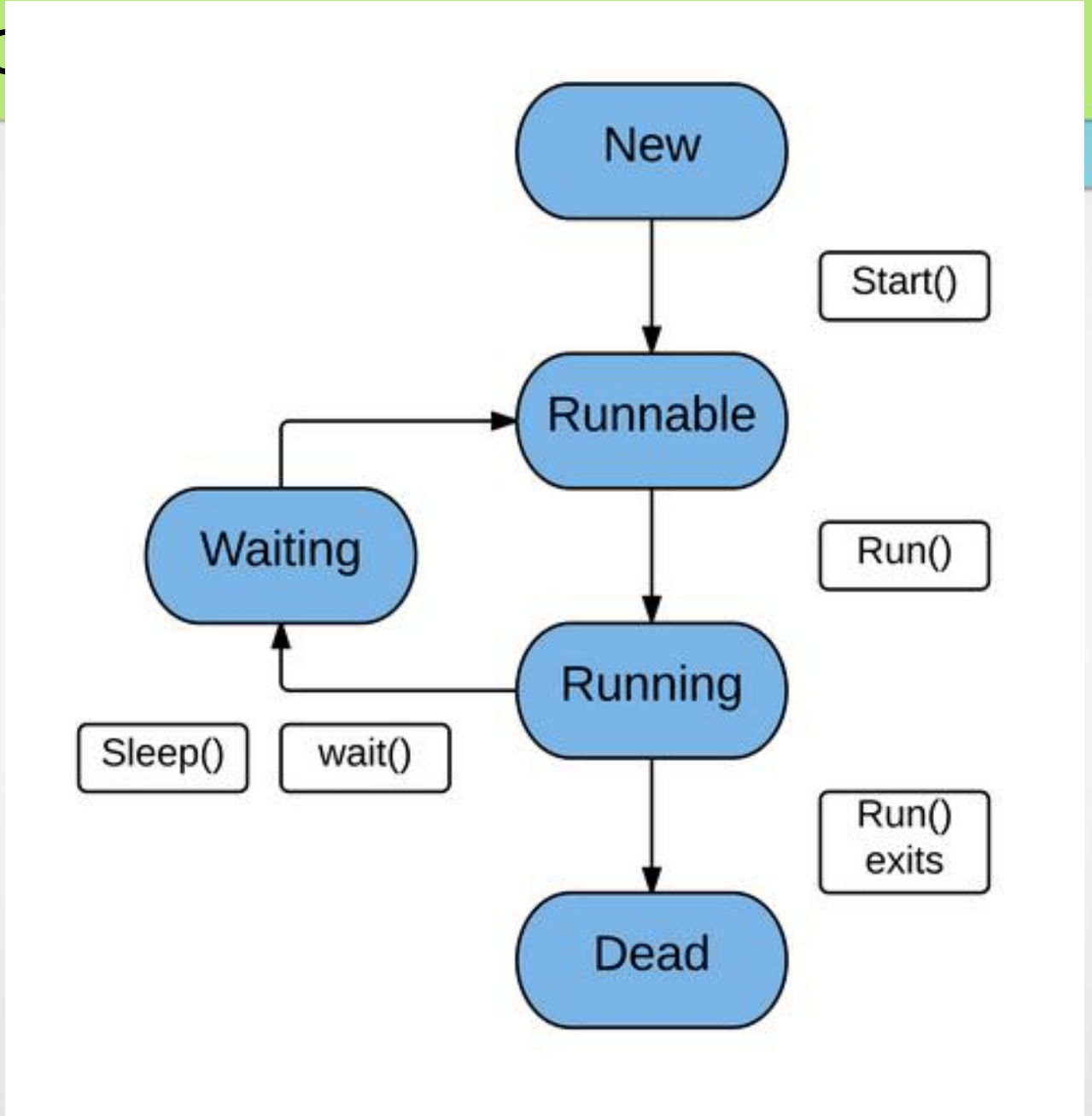
What is Thread in java



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 cyc


New
Start()
Runnable
Waiting
Run()
Running
Sleep()
wait()
Run()
exits
Dead

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)

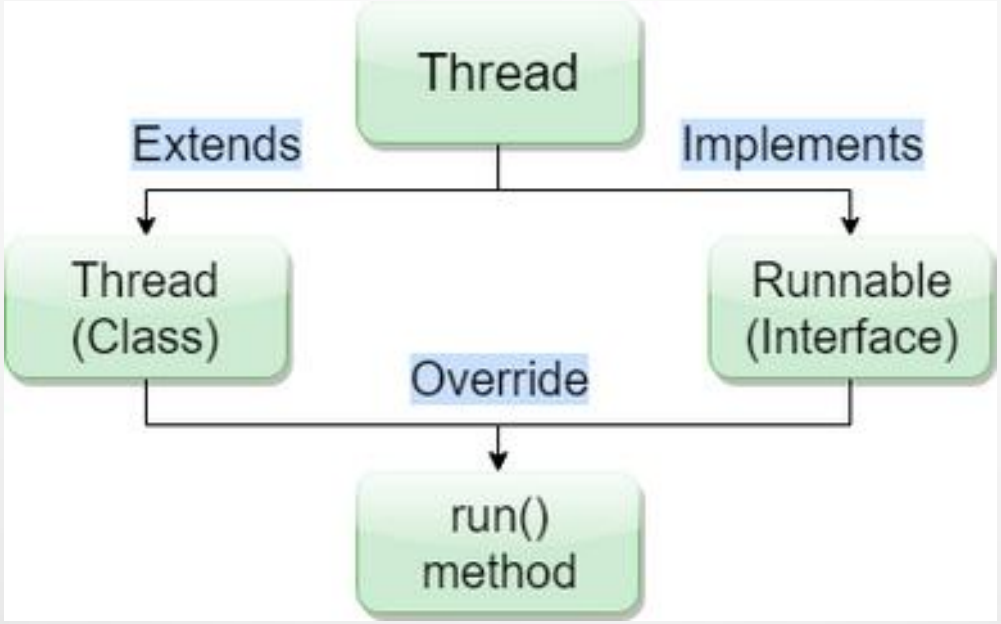
- This is the state when the thread is still alive, but is currently not eligible to run.

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)

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.

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

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

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

  waits for a thread to die for the specified miliseconds.

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.

Commonly used methods of Thread class:

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

  returns the state of the thread.
- **public boolean isAlive()**

  tests if the thread is alive.
- **public void yield()**

  causes the currently executing thread object to temporarily pause and allow other threads to execute.
- **public void suspend()**

  is used to suspend the thread.
- **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 schedular 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 defiend 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.