Java Concurrency in Practice

Chapter-1: Introduction

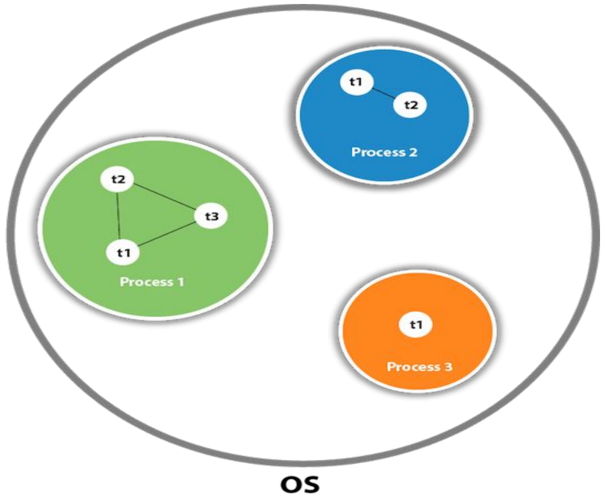
Upcode Software Engineer Team

-2023-

CONTENT

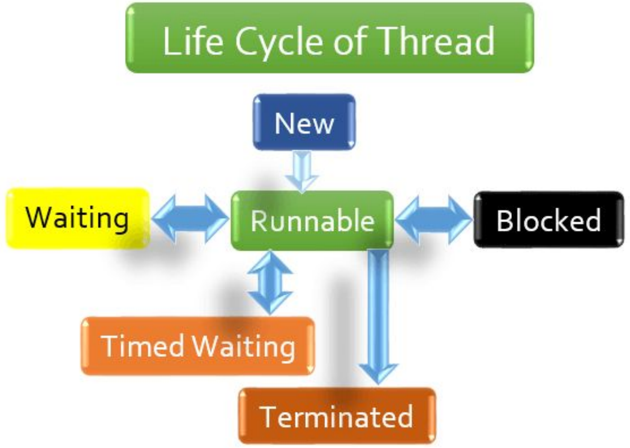
What is Thread? (1/n)

- A thread is a sequence of instructions that can be executed independently by a CPU core.



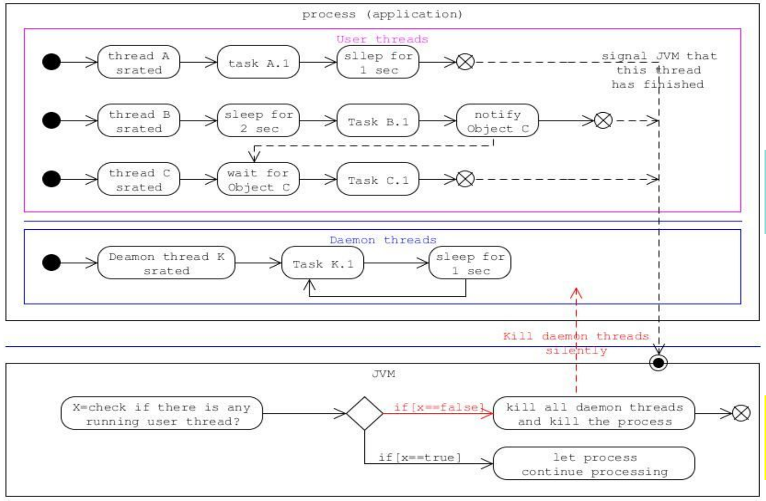
What is Thread? (2/n)

- Threads with higher priority are executed in preference compared to threads with a **lower priority**
- The Java VM continues to execute threads until either of the following occurs thread
  - The exit method of class **Runtime has been called**
  - All user threads have died



Thread Types(1/n) - Daemon and User threads

- The Thread consists of 2 types: **Daemon thread in Java is a low-priority thread** that performs background operations
- **User Threads in Java is a high priority thread that the JVM** waits till the execution is finished.

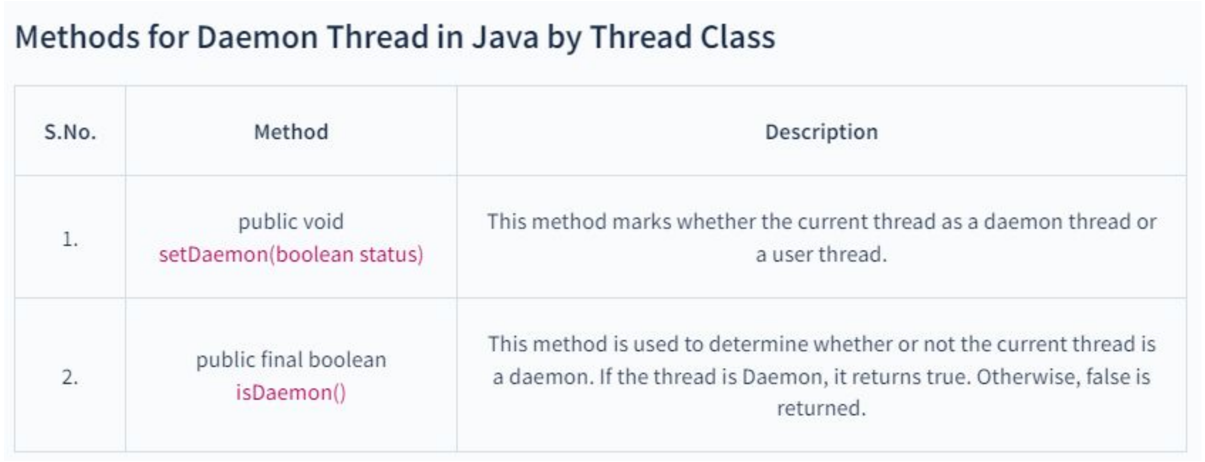


User threads

Daemon threads

Thread Types(2/n)

- Daemon thread in Java is a low-priority thread that performs background operations such as ***garbage collection, finalizer, Action Listeners, Signal dispatches, etc***.
- **Daemon thread in Java is also a service provider thread that helps the user thread.** Its life is at the mercy of user threads; when **all user threads expire, JVM immediately terminates this thread**.

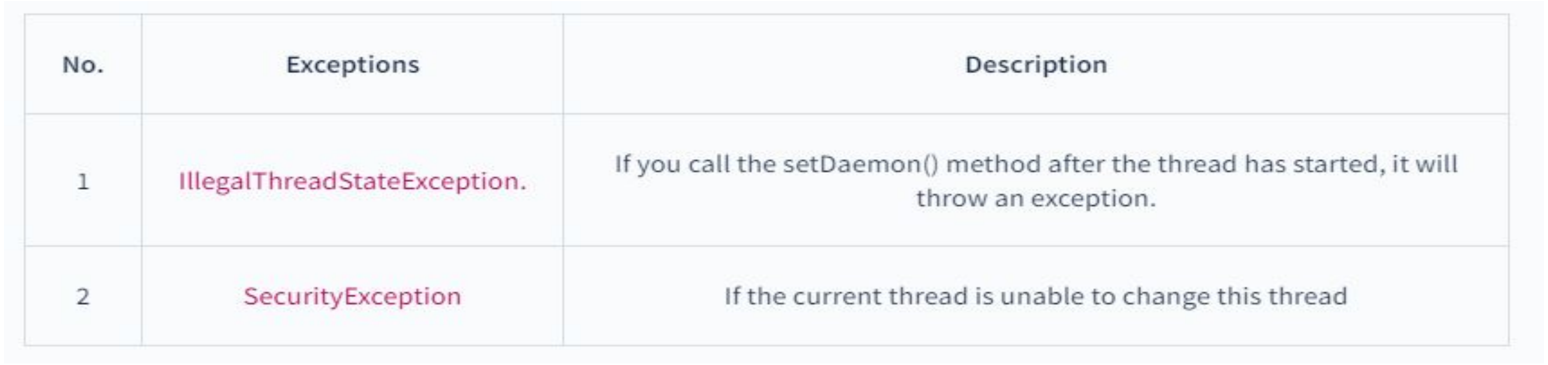


Methods for Daemon Thread in Java by Thread Class

S.No.	Method	Description
1.	public void setDaemon(boolean status)	This method marks whether the current thread as a daemon thread or a user thread.
2.	public final boolean isDaemon()	This method is used to determine whether or not the current thread is a daemon. If the thread is Daemon, it returns true. Otherwise, false is returned.

Thread Types(3/n)-Exceptions in a Daemon

- Daemon thread Exception in Java

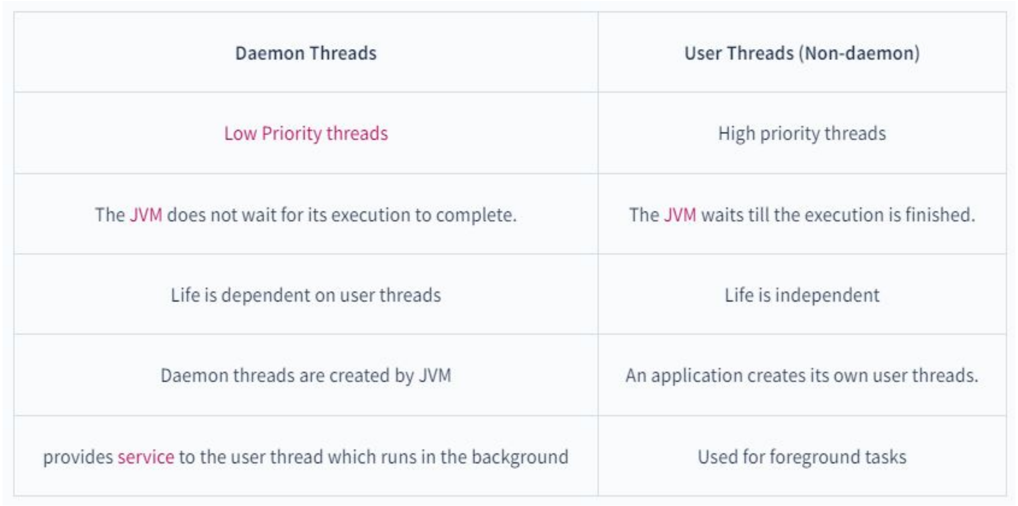


No.	Exceptions	Description
1	IllegalThreadStateException.	If you call the setDaemon() method after the thread has started, it will throw an exception.
2	SecurityException	If the current thread is unable to change this thread

Exception in thread "main" java.lang.IllegalThreadStateException at
java.base/java.lang.Thread.setDaemon(Thread.java:1406)at
DemoDaemonThread.main(DemoDaemonThread.java:18)

Thread Types(4/n)

- With the aid of the table below, learn more about the distinctions between Daemon and User threads
- Every user defined thread is created as non-daemon thread by default, because main thread is a non-daemon thread.



Daemon Threads	User Threads (Non-daemon)
Low Priority threads	High priority threads
The JVM does not wait for its execution to complete.	The JVM waits till the execution is finished.
Life is dependent on user threads	Life is independent
Daemon threads are created by JVM	An application creates its own user threads.
provides service to the user thread which runs in the background	Used for foreground tasks

Why to study Threads ? (1/n)

- **Multitasking:**
  - Threads allow Java programs to perform multiple tasks simultaneously. For example, in a graphical user interface (GUI) application, one thread can handle user input, while another thread can perform background tasks like fetching data from a server. This multitasking capability leads to a more responsive and efficient user experience.

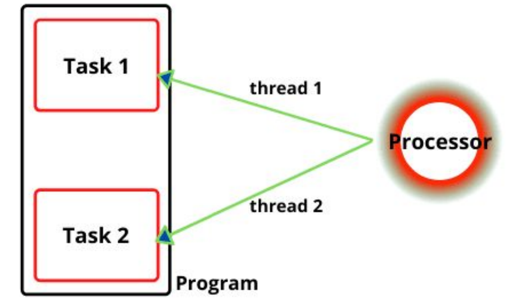
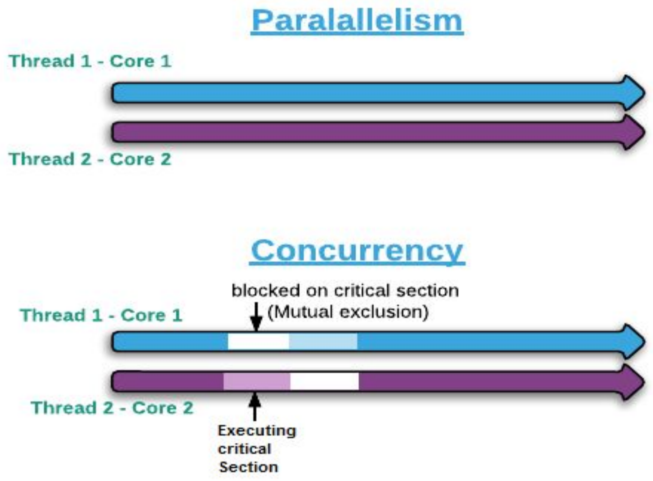


Fig: Thread-based Multitasking

Why to study Threads ? (2/n)

- **Concurrency:**
    - Java threads enable concurrent execution of code. Concurrency is essential for optimizing system resources and improving the overall performance of applications. By dividing tasks into smaller threads, programs can make use of multicore processors effectively, thus achieving parallelism.



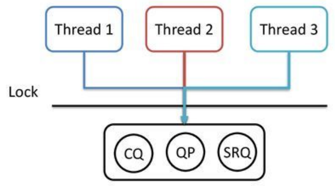
Why to study Threads ? (3/n)

- **Responsiveness:**
    - Threads are used in Java to keep the user interface responsive. Long-running tasks, such as network operations or database queries, can be executed in a separate thread, ensuring that the main user interface thread remains responsive to user interactions.

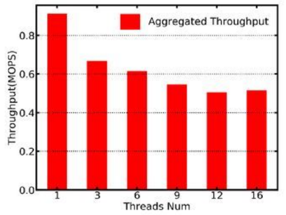
Why to study Threads ? (4/n)

- **Resource Sharing:**
    - Threads can share resources within the same process, allowing data to be accessed and manipulated by multiple threads. Proper synchronization techniques are essential to ensure data consistency and avoid conflicts when multiple threads access shared resources concurrently.

Resource Sharing Problem



(1) Synchronous Resource Sharing Model



(2) Throughput on the Shared QP with Varied number of Threads

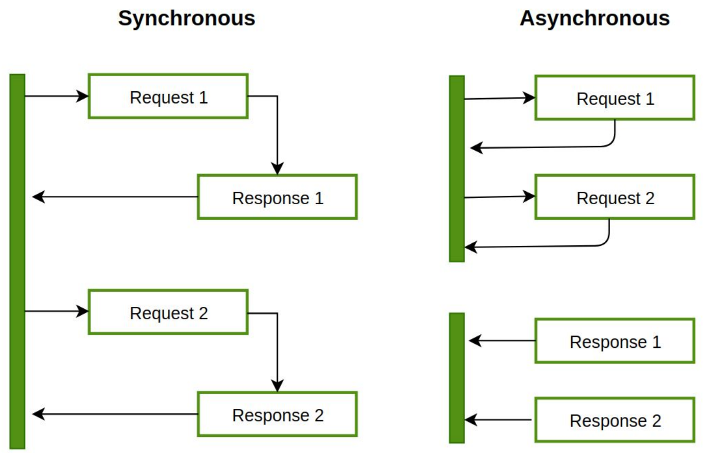
- Lock mechanism must be applied in synchronous sharing.
- The SRQs provided by verbs are application isolated and not effectively shared.

- Serious performance loss when applying lock mechanism.
    - 34% performance loss for 6 threads
    - 44% performance loss for 9 threads.

Resources should be effectively shared among applications

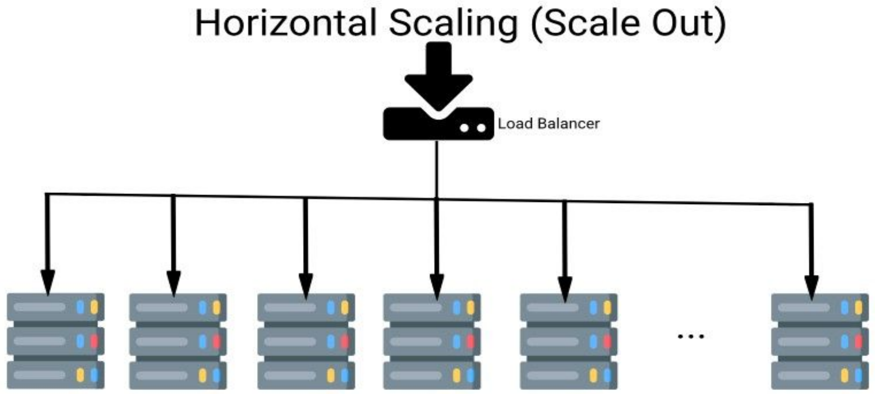
Why to study Threads ? (5/n)

- **Asynchronous Programming:**
  - Threads enable asynchronous programming, where tasks can be executed independently of the main program flow. Asynchronous programming is essential for handling I/O operations efficiently, such as reading from files or making network requests, without blocking the entire program.



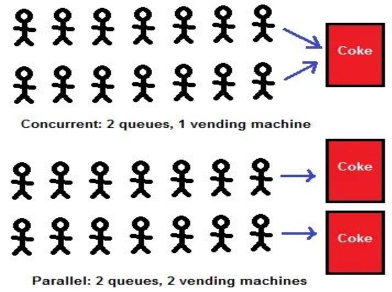
Why to study Threads ? (6/n)

- **Scalability**
  - Multi-threading can enhance the scalability of applications.
  - By efficiently utilizing available system resources, threaded programs can handle a large number of concurrent connections or requests, making them suitable for building scalable server-side applications.

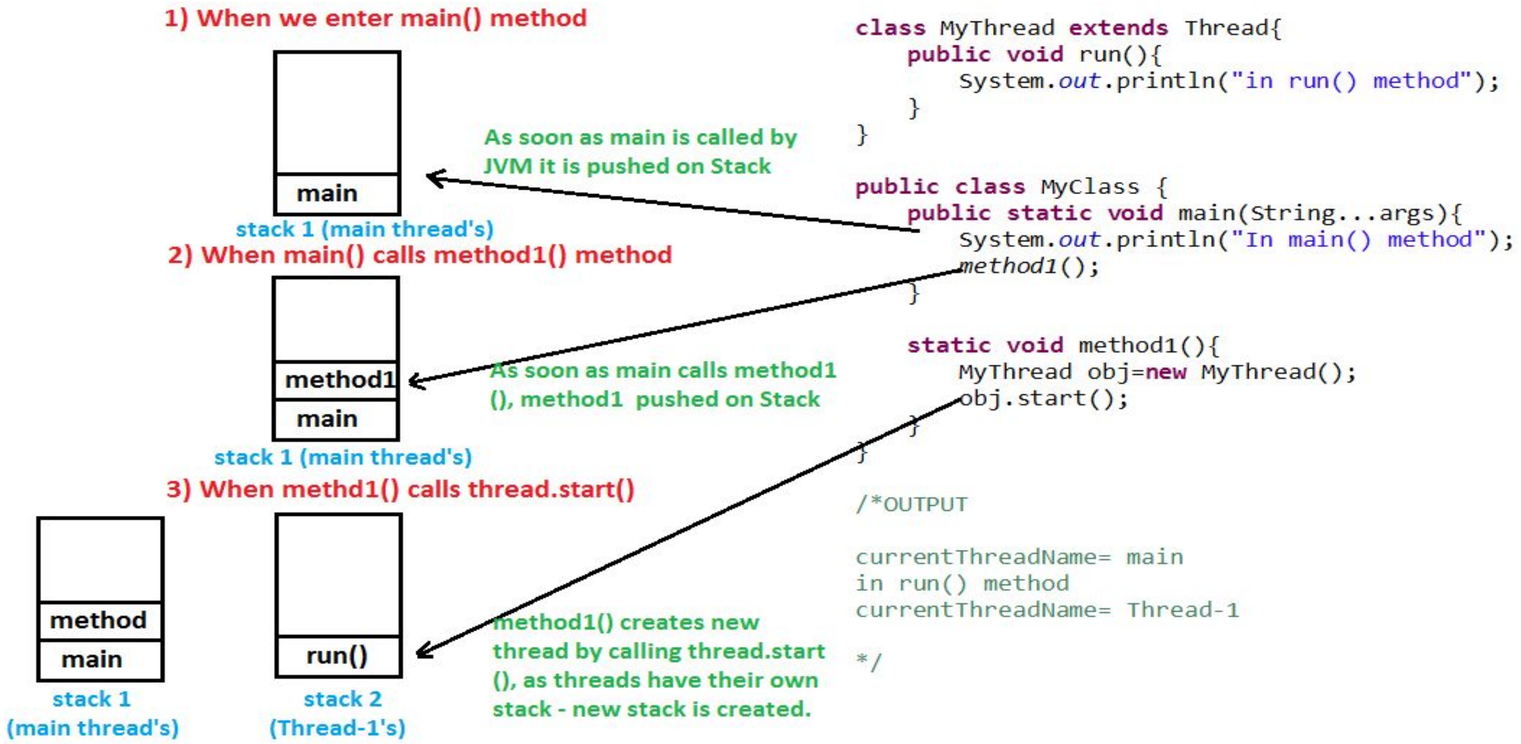


Why to study Threads ? (7/n)

- **Real-time Applications:**
  - In certain applications, especially those requiring real-time processing such as online gaming or financial trading systems, threads are essential to ensure timely response and processing of events.
- **Learning Parallelism:**
  - Understanding threads is a fundamental step towards understanding parallelism, which is becoming increasingly important in the era of multi-core processors. Parallel programming is a key concept for improving the performance of computationally intensive tasks.


Coke
Concurrent: 2 queues, 1 vending machine
Coke
Coke
Parallel: 2 queues, 2 vending machines

Stack and Threads. (1/n)



1) When we enter main() method

As soon as main is called by JVM it is pushed on Stack

2) When main() calls method1() method

As soon as main calls method1 (), method1 pushed on Stack

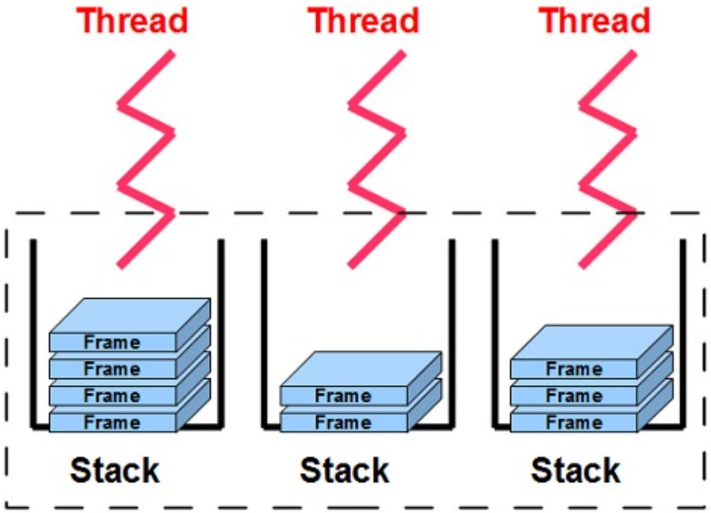
3) When methd1() calls thread.start()

method1() creates new thread by calling thread.start (), as threads have their own stack - new stack is created.

```
class MyThread extends Thread{
    public void run(){
        System.out.println("in run() method");
    }
}

public class MyClass {
    public static void main(String...args){
        System.out.println("In main() method");
        method1();
    }

    static void method1(){
        MyThread obj=new MyThread();
        obj.start();
    }
}

/*OUTPUT

currentThreadName= main
in run() method
currentThreadName= Thread-1

*/
```

Stack and Threads. (2/n)

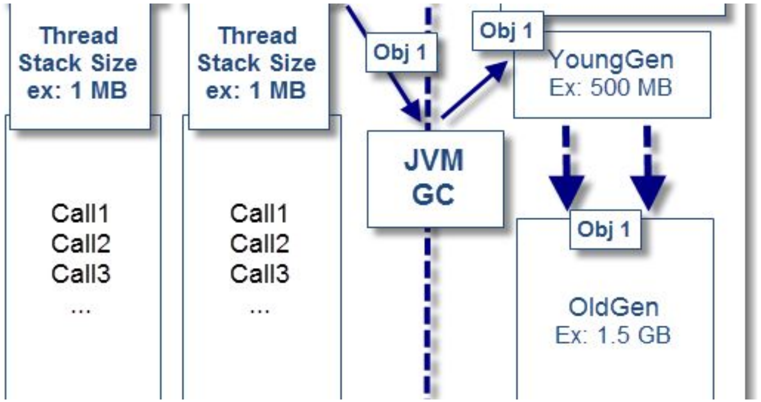
- Each thread in a program has its own stack.
- When a new thread is created, the operating system allocates a new stack for that thread. This stack is used to store information about the thread's function calls, local variables, and other related data.
- When the thread calls a function, the function's local variables and other information are stored on its stack. This allows each thread to have its own isolated space for function calls and local data, ensuring that one thread's function calls and local variables do not interfere with another thread's.



Heap and Threads. (1/n)

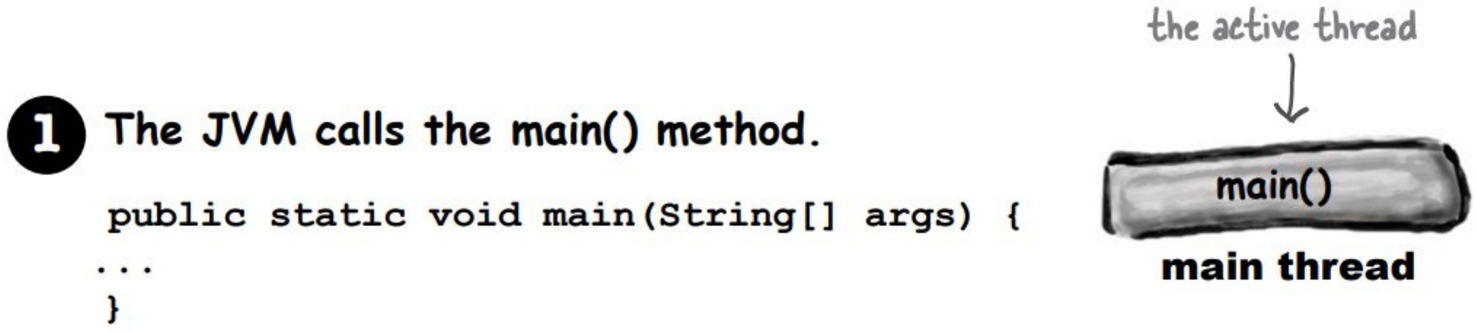
- The heap is a region of a computer's memory used for dynamic memory allocation. It's a common area where objects, data structures, and other variables are allocated memory during a program's runtime.
- Multiple threads within the same program share the same heap. This shared memory space allows threads to access shared data and resources. However, it also requires careful synchronization mechanisms to prevent conflicts when multiple threads try to read or modify shared data simultaneously.
- While threads have their own stacks, they share the same heap.

This means that objects and data allocated on the heap can be accessed and modified by multiple threads.



What does it mean to have more than one call stack?

- With more than one call stack, you get the appearance of having multiple things happen at the same time.
- In reality, only a true multiprocessor system can actually do more than one thing at a time, but with Java threads, it can appear that you're doing several things simultaneously.
- In other words, execution can move back and forth between stacks so rapidly that you feel as though all stacks are executing at the same time.



1 The JVM calls the main() method.

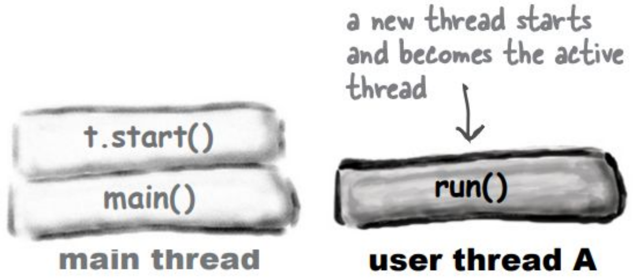
```
public static void main(String[] args) {
...
}
```

What does it mean to have more than one call stack?

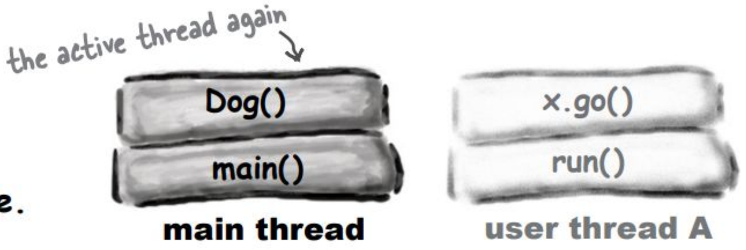
2 main() starts a new thread. The main thread is temporarily frozen while the new thread starts running.

```
Runnable r = new MyThreadJob();
Thread t = new Thread(r);
t.start();
Dog d = new Dog();
```

you'll learn what this means in just a moment...



3 The JVM switches between the new thread (user thread A) and the original main thread, until both threads complete.

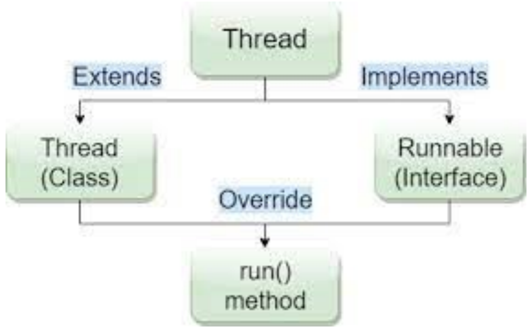


How to launch a new thread ? (1/n)

- **Make a Runnable object (the thread's job).**

  Runnable threadJob = new MyRunnable();

- Runnable is an interface you'll learn about on the next page. You'll write a class that implements the Runnable interface, and that class is where you'll define the work that a thread will perform. In other words, the method that will be run from the thread's new call stack.

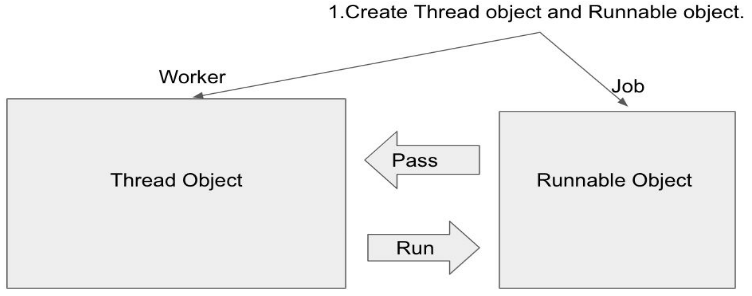


How to launch a new thread ? (2/n)

- **Make a Thread object (the worker) and give it a Runnable (the job).**

  Thread myThread = new Thread(threadJob);

- Pass the new Runnable object to the Thread constructor. This tells the new Thread object which method to put on the bottom of the new stack—the Runnable's run() method.



2. Pass the Runnable Object to the Thread Object.

3. Call the start method of the Thread Object to execute the code defined in the run method in the Runnable Object.

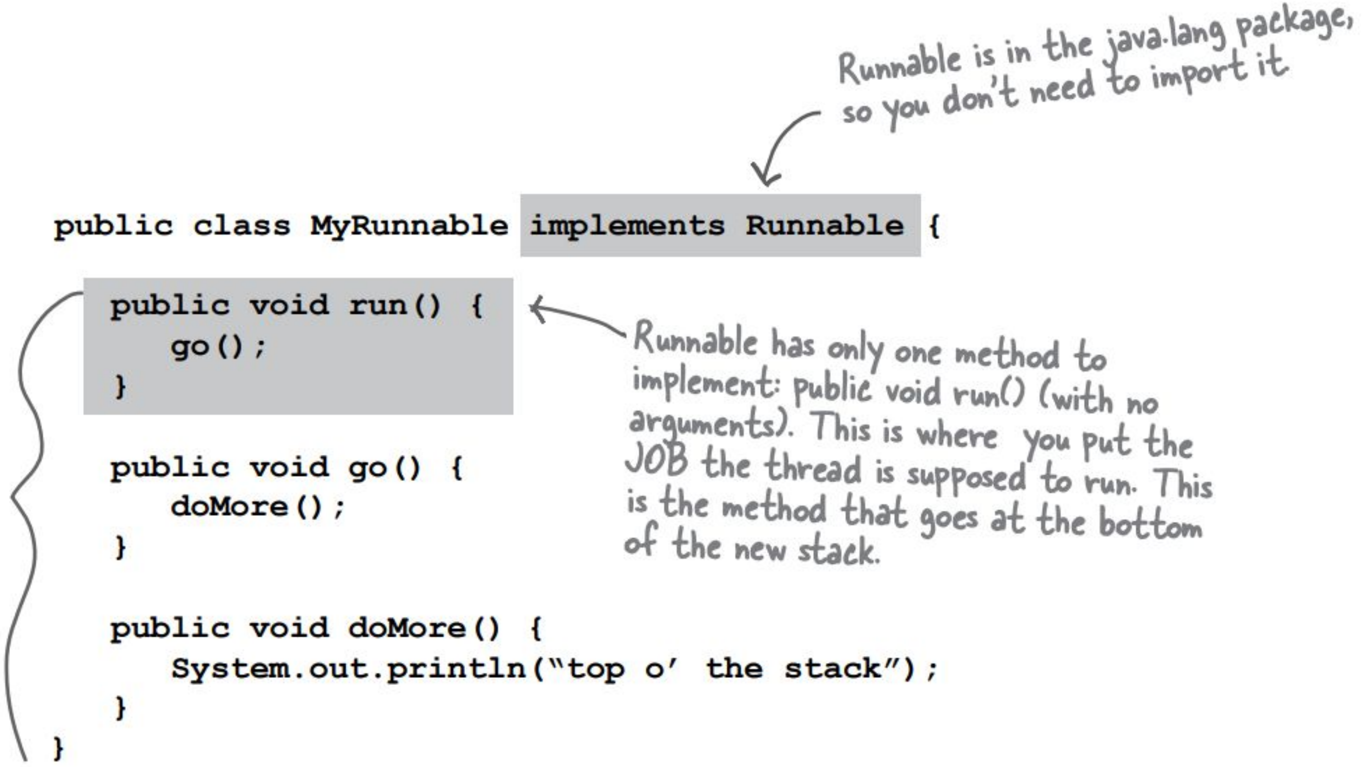
How to launch a new thread ? (3/n)

- **Start the Thread.**

  myThread.start();

- Nothing happens until you call the Thread’s start() method. That’s when you go from having just a Thread instance to having a new thread of execution. When the new thread starts up, it takes the Runnable object’s run() method and puts it on the bottom of the new thread’s stack.

How to launch a new thread ? (4/n)



Runnable is in the java.lang package, so you don't need to import it

```
public class MyRunnable implements Runnable {

    public void run() {
        go();
    }

    public void go() {
        doMore();
    }

    public void doMore() {
        System.out.println("top o' the stack");
    }
}
```

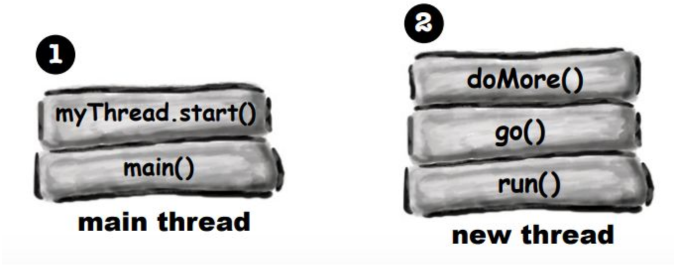
Runnable has only one method to implement: public void run() (with no arguments). This is where you put the JOB the thread is supposed to run. This is the method that goes at the bottom of the new stack.

How to launch a new thread ? (5/n)

```
class ThreadTester {

   public static void main (String[] args) {

      Runnable threadJob = new MyRunnable();
      Thread myThread = new Thread(threadJob);

  1   myThread .start();

      System.out.println("back in main");
   }
}
```

Pass the new Runnable instance into the new Thread constructor. This tells the thread what method to put on the bottom of the new stack. In other words, the first method that the new thread will run.

You won't get a new thread of execution until you call start() on the Thread instance. A thread is not really a thread until you start it. Before that, it's just a Thread instance, like any other object, but it won't have any real 'threadness'.



Summary

- In summary, threads are a fundamental part of modern computing, enabling multitasking, improved performance, and efficient resource sharing within a process. Proper thread management and synchronization are crucial for creating robust, responsive, and efficient applications.

Reference

1.Thread Type: Daemon Thread.

2. Java thread methods .

3. Head First (book).

4. Multithreading Tutorial.

Thank you!

Presented by

Sherjonov Jahongir

(jakhongirsherjonov@gmail.com)