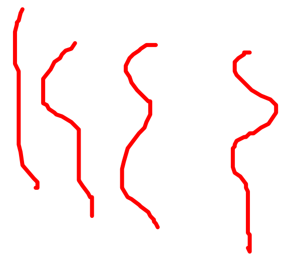
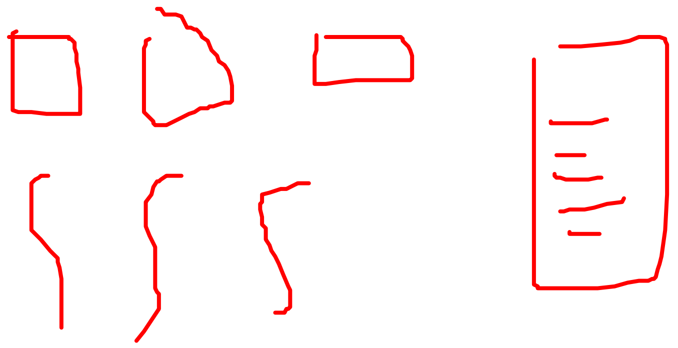
Java Multithreading

Java Multithreading

- **Multithreading in Java** is a process of executing multiple threads simultaneously.
- A thread is a lightweight sub-process, the smallest unit of processing.
- Multiprocessing and multithreading, both are used to achieve multitasking.
- However, we use multithreading than multiprocessing because threads use a shared 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.
- Different tasks within program done simultaneously
- Individual programs appear to do multiple tasks at a time.
- Each sub task is called thread.



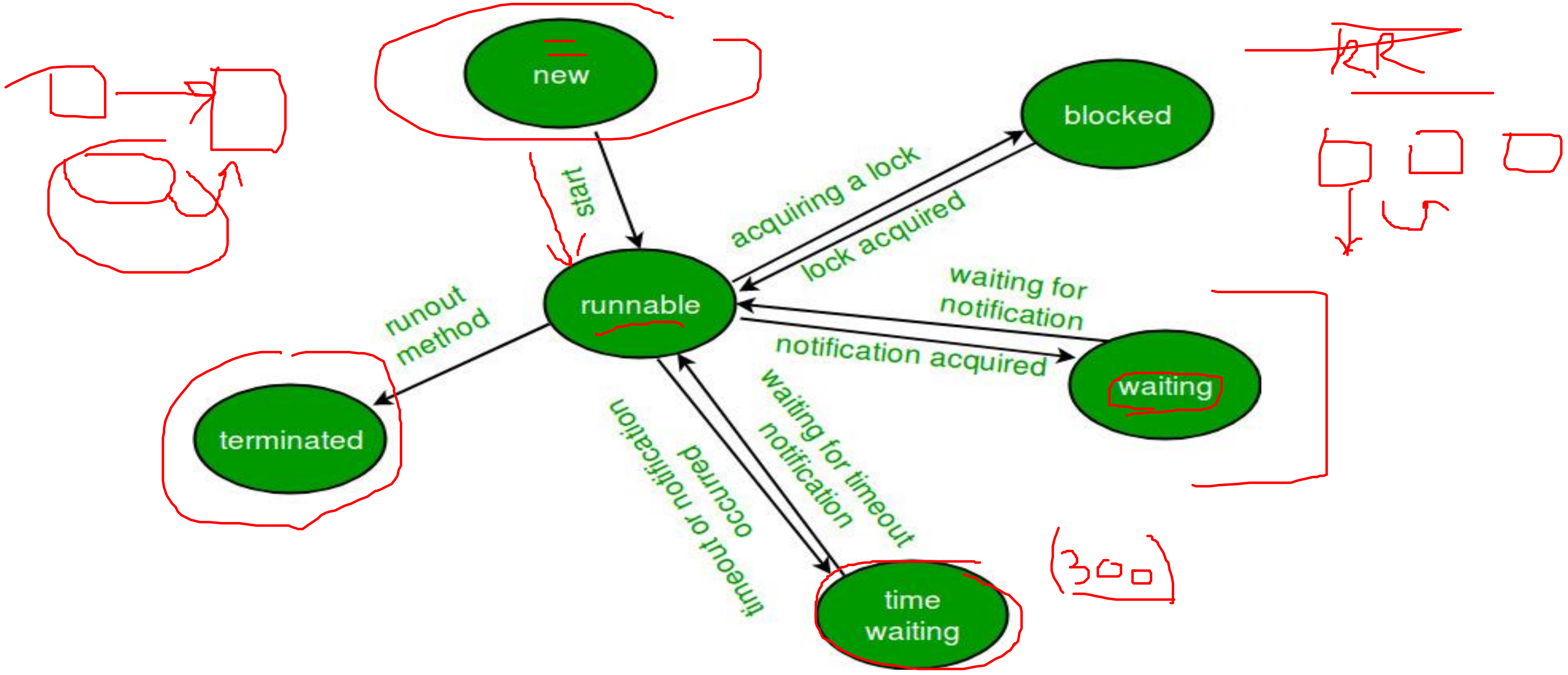


Java Multithreading

- **Life Cycle Of Thread**
- A thread in Java at any point of time exists in any one of the following states. A thread lies only in one of the shown states at any instant:

1. New
2. Runnable
3. Blocked
4. Waiting
5. Timed Waiting
6. Terminated

Java Multithreading



Java Multithreading

1. **New Thread:**

- When a new thread is created, it is in the new state.
- The thread has not yet started to run when thread is in this state.
- When a thread lies in the new state, it’s code is yet to be run and hasn’t started to execute.

Java Multithreading

2. **Runnable State:**

- A thread that is ready to run is moved to runnable state.
- In this state, a thread might actually be running or it might be ready run at any instant of time.
- It is the responsibility of the thread scheduler to give the thread, time to run.
- A multi-threaded program allocates a fixed amount of time to each individual thread.
- Each and every thread runs for a short while and then pauses and relinquishes the CPU to another thread, so that other threads can get a chance to run.
- When this happens, all such threads that are ready to run, waiting for the CPU and the currently running thread lies in runnable state.

Java Multithreading

3. Blocked/Waiting state:

- When a thread is temporarily inactive, then it's in one of the following states:

  1. Blocked and 2. Waiting

- For example, when a thread is waiting for I/O to complete, it lies in the blocked state. It's the responsibility of the thread scheduler to reactivate and schedule a blocked/waiting thread. A thread in this state cannot continue its execution any further until it is moved to runnable state. Any thread in these states does not consume any CPU cycle.
- A thread is in the blocked state when it tries to access a protected section of code that is currently locked by some other thread. When the protected section is unlocked, the schedule picks one of the thread which is blocked for that section and moves it to the runnable state.
- Whereas, a thread is in the waiting state when it waits for another thread on a condition. When this condition is fulfilled, the scheduler is notified and the waiting thread is moved to runnable state.
- If a currently running thread is moved to blocked/waiting state, another thread in the runnable state is scheduled by the thread scheduler to run. It is the responsibility of thread scheduler to determine which thread to run.

Java Multithreading

4. **Timed Waiting:**

- A thread lies in timed waiting state when it calls a method with a time out parameter.
- A thread lies in this state until the timeout is completed or until a notification is received.
- For example, when a thread calls sleep or a conditional wait, it is moved to a timed waiting state.

5. **Terminated State:**

- A thread terminates because of either of the following reasons:
  1. Because it exists normally. This happens when the code of thread has entirely executed by the program.
  2. Because there occurred some unusual erroneous event, like segmentation fault or an unhandled exception.
- A thread that lies in a terminated state does no longer consumes any cycles of CPU.

Java Multithreading

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

- 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.

For Example : Multi.java

Java Multithreading

Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r, String name)
- 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.
- Multi3.java

Java Multithreading

Thread Scheduler in Java

- Thread scheduler in java is the part of the JVM that decides which thread should run.
- There is no guarantee that which runnable thread will be chosen to run by the thread scheduler.
- Only one thread at a time can run in a single process.
- The thread scheduler mainly uses pre-emptive or time slicing scheduling to schedule the threads.

Java Multithreading

Sleep method in java

- The sleep() method of Thread class is used to sleep a thread for the specified amount of time.
- The Thread class provides below method for sleeping a thread:

1. public static void sleep(long miliseconds)throws InterruptedException

- For example: TestSleepMethod1.java

Java Multithreading

Naming Thread

- The Thread class provides methods to change and get the name of a thread.
- By default, each thread has a name i.e. thread-0, thread-1 and so on.
- By we can change the name of the thread by using setName() method.
- The syntax of setName() and getName() methods are given below:

1. public String getName(): is used to return the name of a thread.
2. public void setName(String name): is used to change the name of a thread.

TestMultiNaming1.java

Java Multithreading

Current Thread

The currentThread() method returns a reference of currently executing thread.

TestMultiNaming2 .java

Java Multithreading

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 (known as pre-emptive scheduling).
- But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.
- 3 constants defined in Thread class:

1. public static int MIN_PRIORITY
2. public static int NORM_PRIORITY
3. 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.
- [TestMultiPriority1.java]()