

Multi-tasking :

Executing several tasks simultaneously is called multi-tasking

There are 2 types of multi-tasking

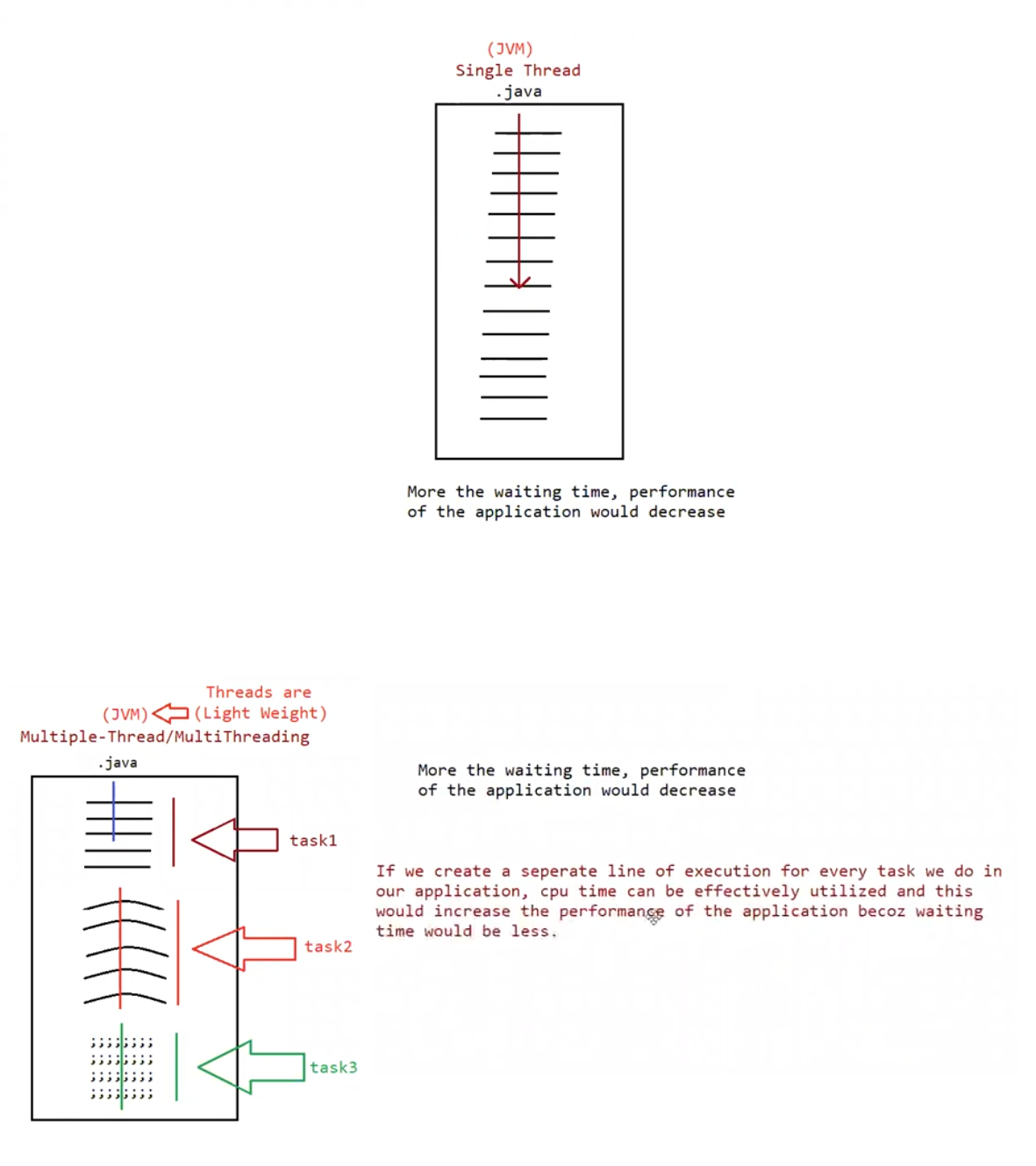
1. Process based multi-tasking
2. Thread based multi-tasking

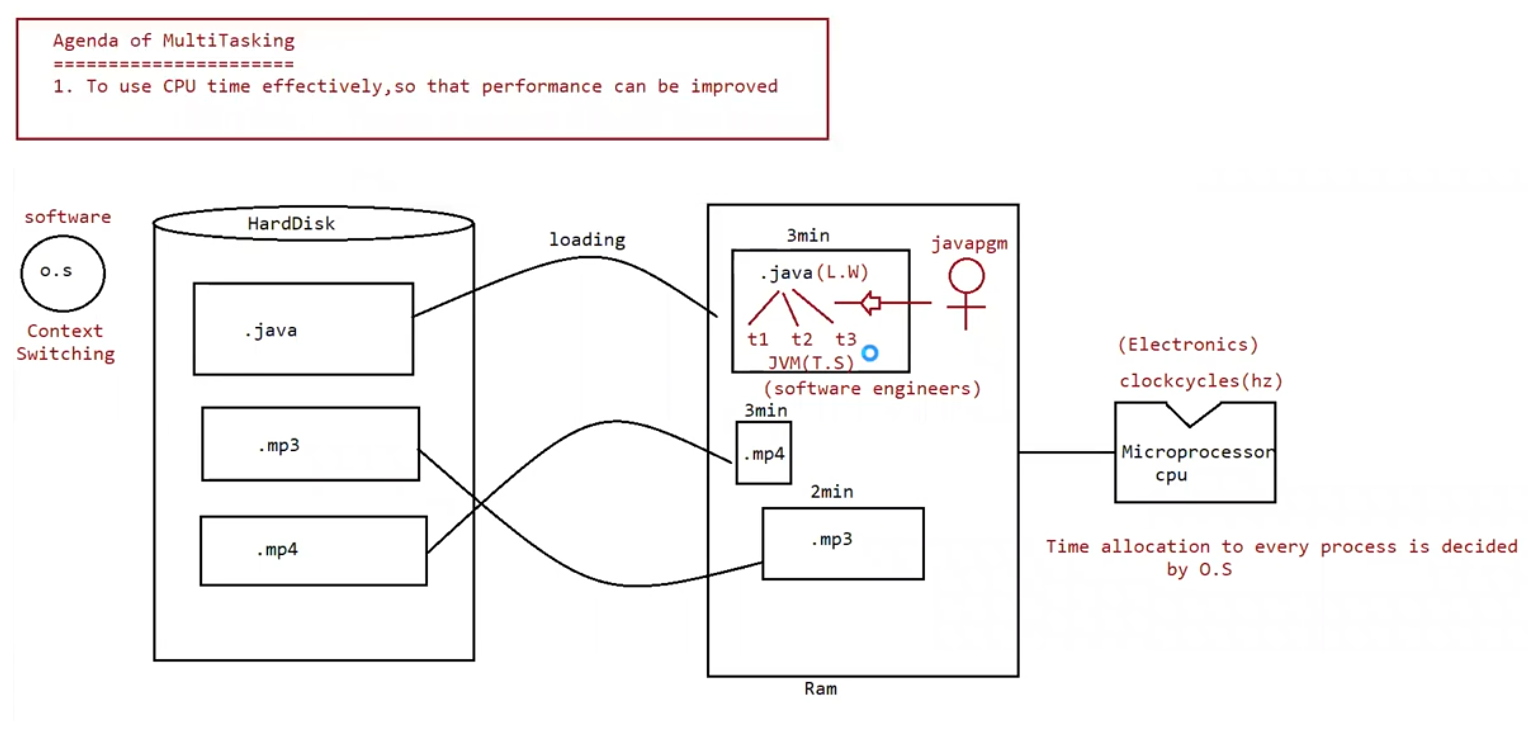
Process based multi-tasking :

Executing several tasks simultaneously where each task is separate independent process such type of multi-tasking is called process based multi-tasking.

Eg: coding a java program , listening songs , downloading some files etc

Process based multi-tasking is best suited at os-level.





Thread based multi-tasking:

Executing several tasks simultaneously where each task is a separate independent part of the program, is called thread based multi-tasking.

Each independent part is called thread.

1. This type of multi-tasking is best suited at programmatic -level

The main advantage of multi-tasking is to reduce the response time ( faster access ) of a system and to improve the performance

1. The main important application areas of multi-threading are

To implement multi media graphics

To develop web application servers

To develop videogames

1. Java provides inbuilt support to work with through API called Thread , Runnable , Threadlocal etc……
2. To work with multi threading , java developers will code only 10% remaining 90% java API will take care.

What is thread ?

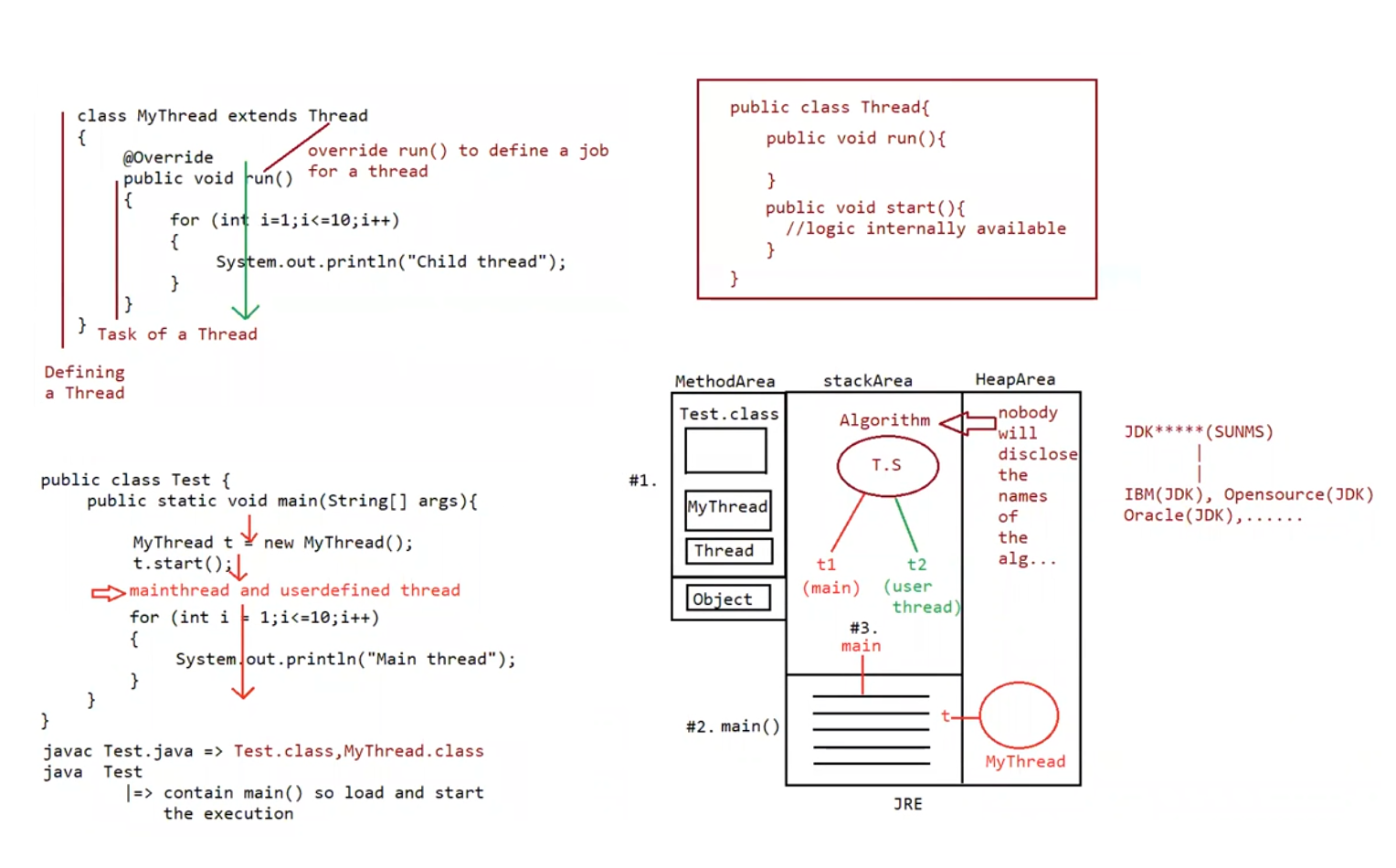
Separate flow/ line of execution is called a thread

If there is only one flow then it is called “Single thread” programming

For every thread there would be a separate job

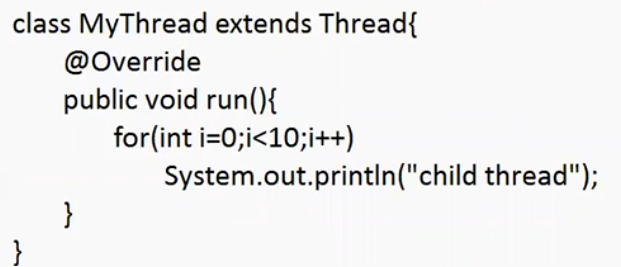
In java we can define a thread in 2 ways

1. Implementing interface
2. Extending thread class



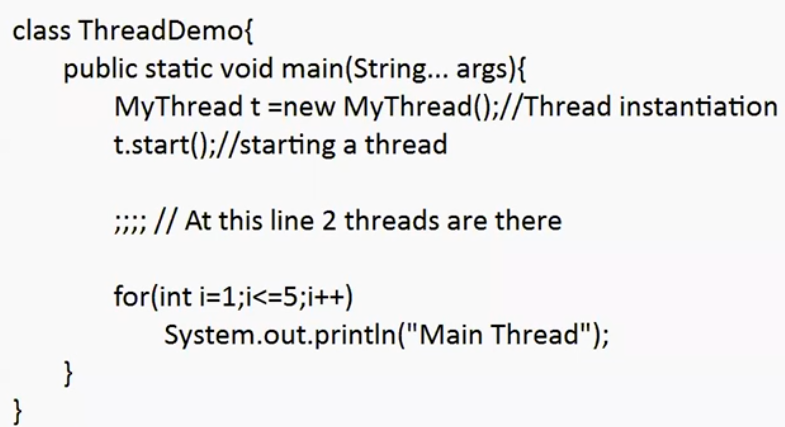
1. By extending a thread class

We can create a thread class by extending the thread class



Defining a thread ( writing a class and extending the thread )

Job a thread ( code written inside run() )



Behind the scenes :

1. Main thread is created automatically by jvm.
2. Main thread creates the child thread starts the child thread

Thread-scheduler:

If multiple threads are waiting to execute, then which thread to execute first is decided by the thread – scheduler which is a part of jvm.

Incase of multi-threading we can’t predict the exact output only possible output we can expect.

Since jobs of threads are important , we are not interested in order of execution it shold just execute such that performance should be improved.

Eg: Thread\_Eg1

Case 2:

Difference between t.start() and t.run()

If we call t.start() a separate thread will be created which is responsible to execute run() method

If we call t.run() no separate thread will be created rather the method will be called just like normal method by main thread.

If we replace t.start() with t.run() then the output of the program will be

Eg: Thread\_With\_Run\_Method

// go through the output

Case 3:

Importance of thread class start() method :

For every thread required mandatory activates like registering the thread with thread scheduler will be taken care by thread class start() method and programmer is just responsible for just doing the job of thread inside run() method

start() acts like an assistance to programmer

public void start() {

1. Register the thread with thread scheduler
2. All other mandatory low level activities
3. Invoke (or) call run() method

}

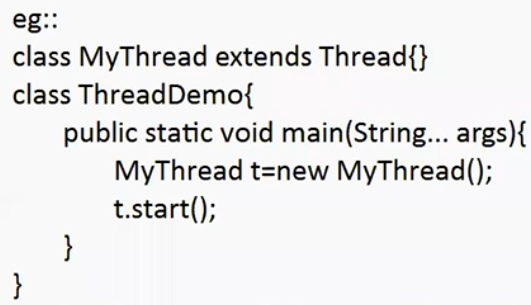
We can conclude that without executing the thread class start() method there is no chance of starting a new thread.

Due to this start() is considered as the heart of the multi-threading.

Case 4:

If we are not overriding run() method

If we are not overriding run() method then thread class run() will be executed which has empty implementation and hence we wont get any output



It is highly recommended to override run() method .

Eg: Thread\_Not\_Overriding\_Run\_Method

// go through the output