

Multi-tasking:

Executing several tasks simultaneously is called multi-tasking

There are 2 types of multi-tasking

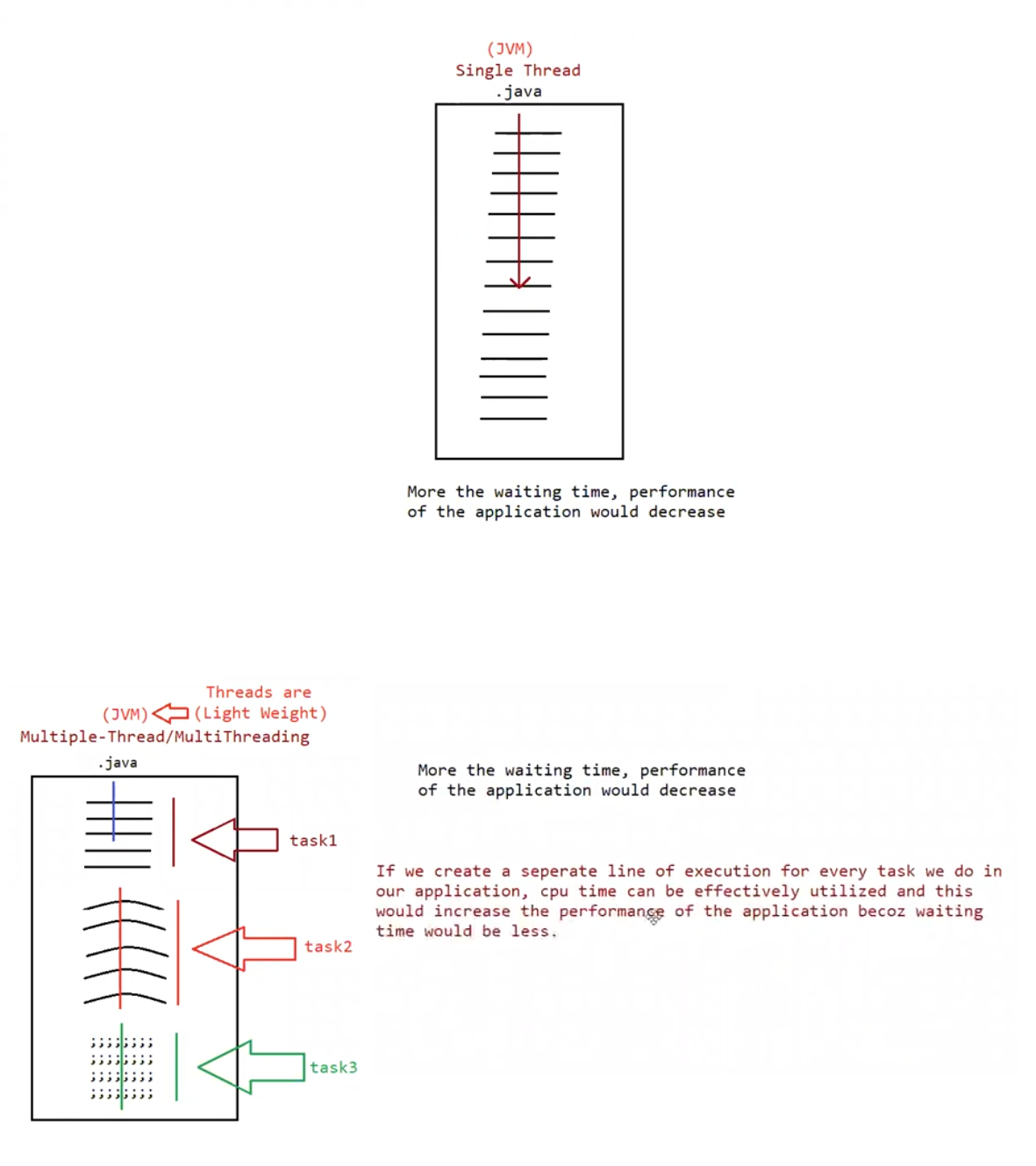
1. Process based multi-tasking
2. Thread based multi-tasking

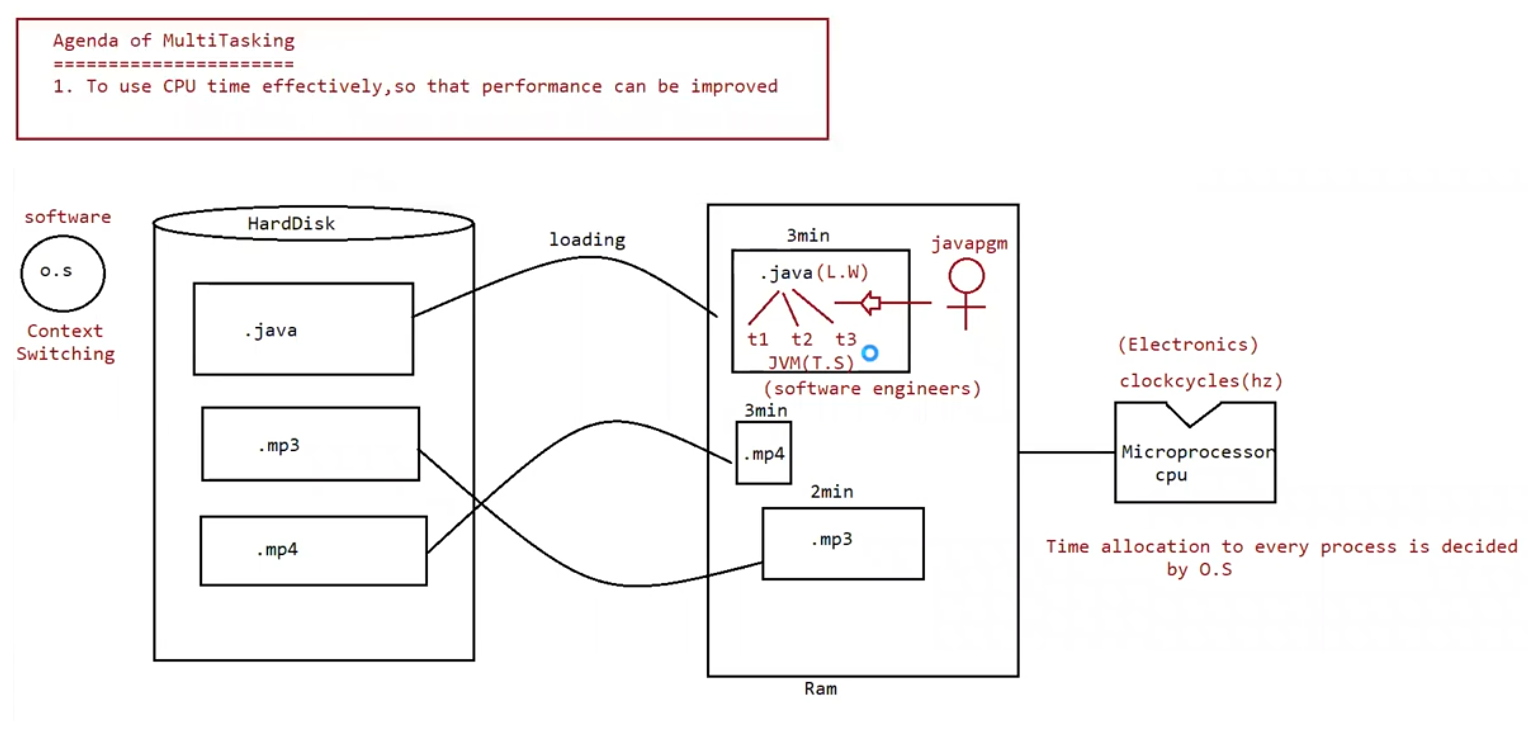
Process based multi-tasking:

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?

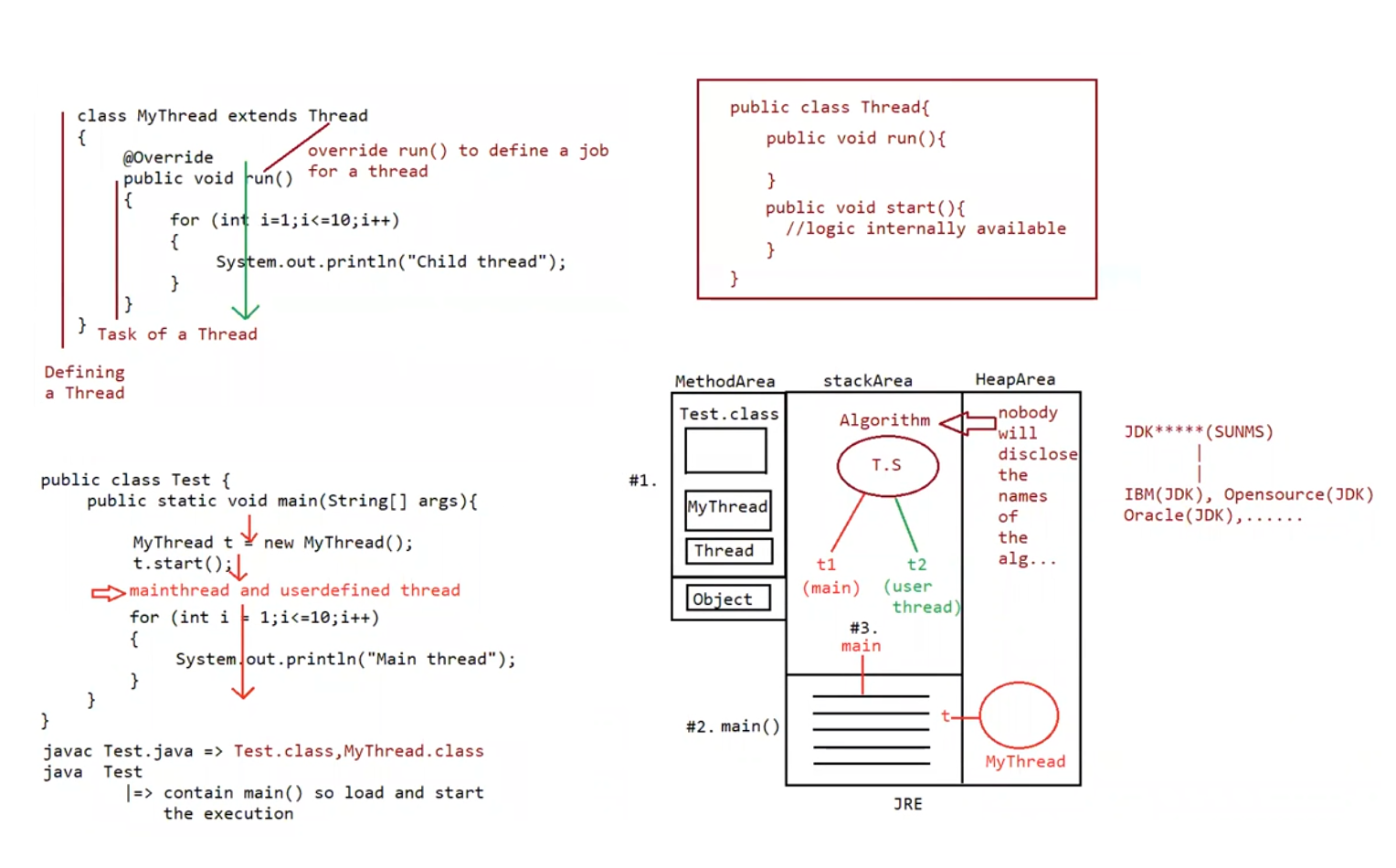
Separate flow/ line of execution is called a thread

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

For every thread there would be a separate job

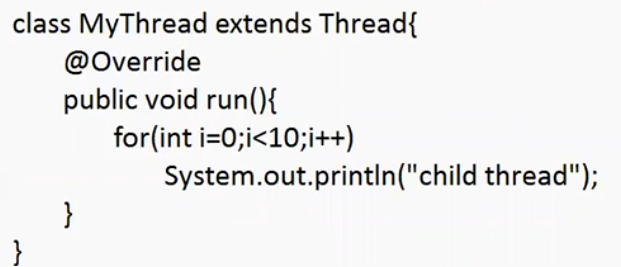
In java we can define a thread in 2 ways

1. Implementing interface
2. Extending thread class



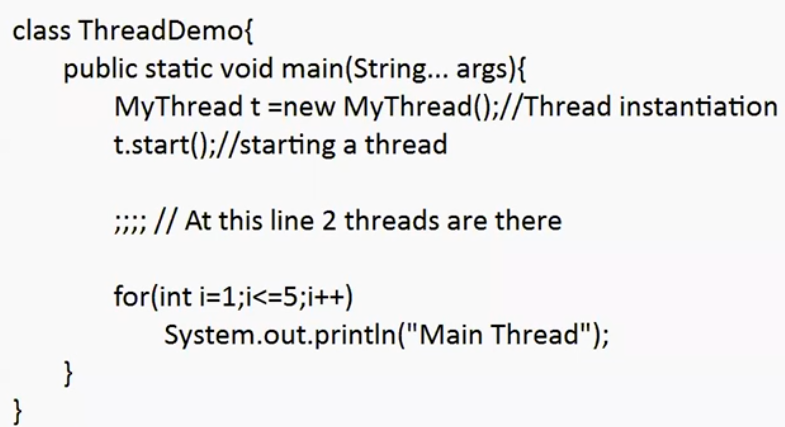
1. By extending a thread class

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 and 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 should just execute such that performance should be improved.

Eg: MyThread\_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

}

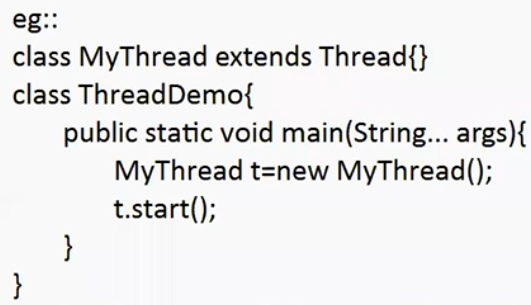
We can conclude that without executing the thread class start() method there is no chance of starting a new thread.

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 won’t get any output



It is highly recommended to override run() method .

Eg: Thread\_Not\_Overriding\_Run\_Method

// go through the output

Case 5:

We can overload run() method, but thread class start method will always call run() with zero argument.

If we overload run with arguments, then we need to explicitly call the argument based run() method and it will be executed just like normal method.

Eg: Thread\_Run\_method\_With\_Argument

Case 6:

If we override start() then our start() method will be executed just like a normal method, but no new thread will be created and no new thread will be started .

Eg: Thread\_Start\_Method\_Overriding

It is never recommended to override start() method.

Case 7:

Eg: User\_Defined\_Start\_Method\_Making\_Call\_To\_Super\_Start\_Method

Here in user defined start() method a call is made to super.start() . so the parent thread class

start() method is called and start() method of thread class will do it tasks like creating a thread ,registering with thread scheduler and other required tasks. And code in userdefined run() method is executed by the new thread.

Case 8 : life cycle of a thread

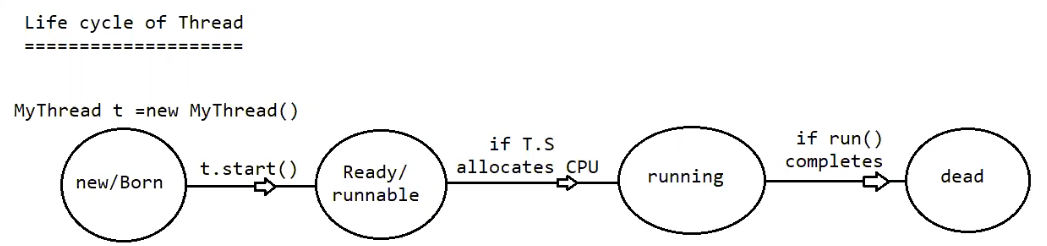
Mythread t = new Mythread(); // thread is in born state

t.start() // thread is in ready / runnable state

if thread scheduler allocates cpu time then we can say thread entered into running state

if run() is completed by thread then we can say thread entered into dead state.

* Once we created a thread object then the thread is said to be in new state (or) born state
* Once we call start() method then the thread will be entered into ready (o) runnable state
* If the thread scheduler allocates the cpu then the thread will be entering into running state
* Once run() method completes then the thread will enter into dead state



Case 9:

After starting the thread, we are not supposed to start the same thread again, if we try to start again it will lead to “IllegalThreadStateException”

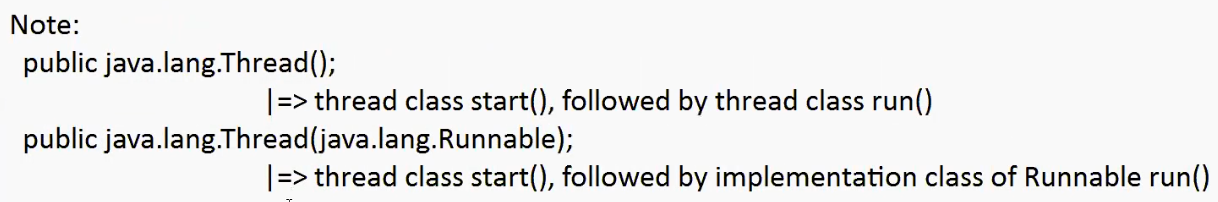
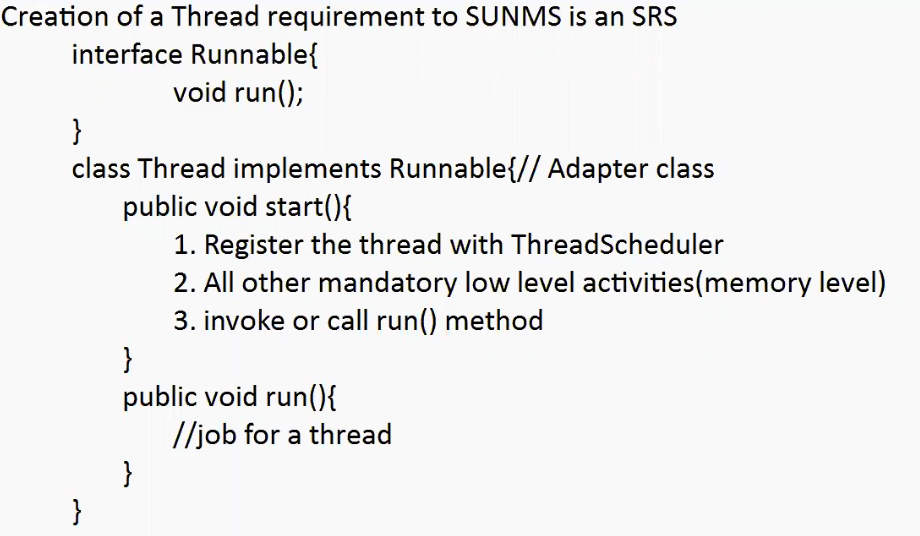
Mythread t = new Mythread();

t.start(); // thread is in ready state

….

T.start(); // IllegalThreadStateException

Creation of a thread using runnable interface:



Eg: Parameterized\_Thread\_Class\_Constructor

Case Study:

MyRunnnable r = MyRunnable();

Thread t1 = new Thread();

Thread t2 = new Thread(r);

Case1:

t1.start() A new thread will be created which is responsible for executing thread class run() method.

Case2:

t2.start() A new thread will be created which is responsible for executing Mythread class run() method.

// go through the above program

Case3:

t1.run()

no new thread class will be created, but Thread class run() method will be executed just like a normal method call.

Case4:

t2.run()

no new thread class will be created, but Mythread class run() method will be executed just like a normal method call.

Case5:

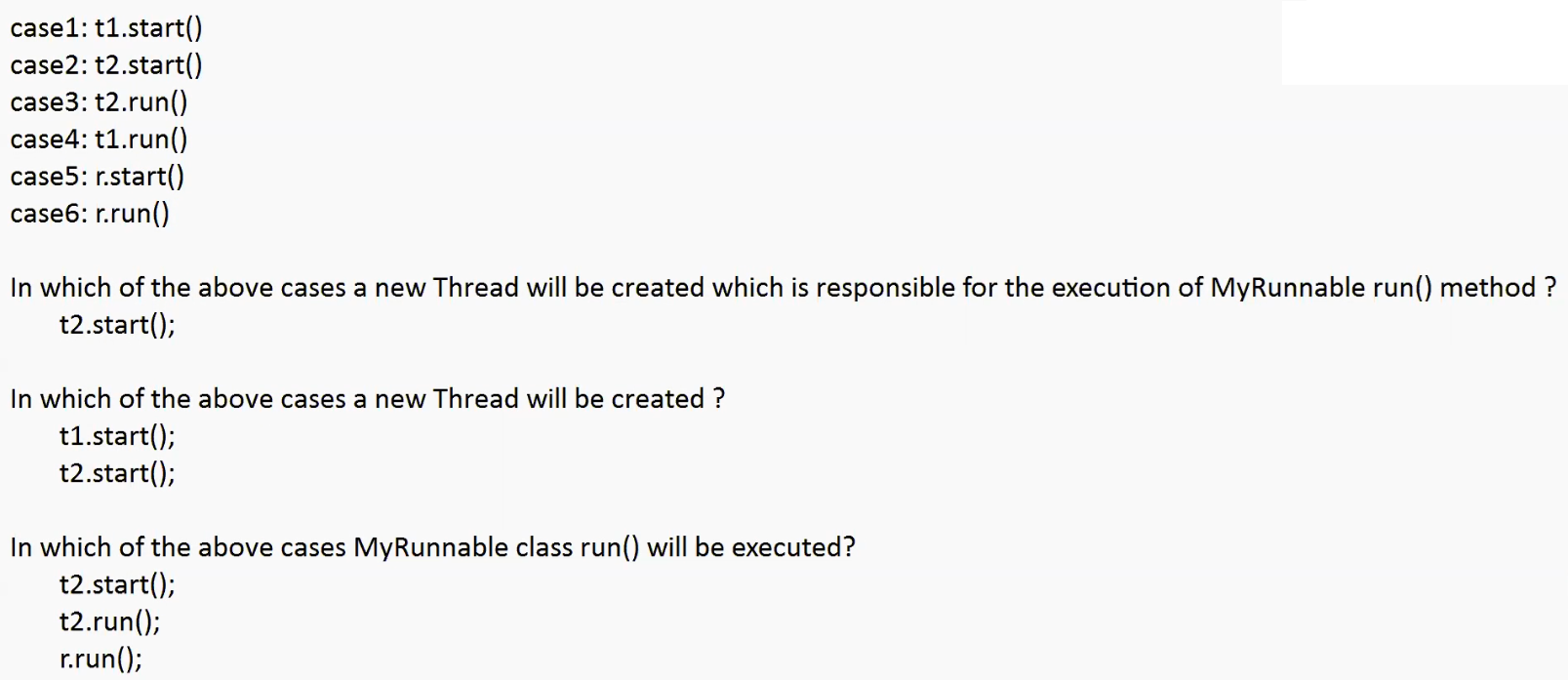
r.start()

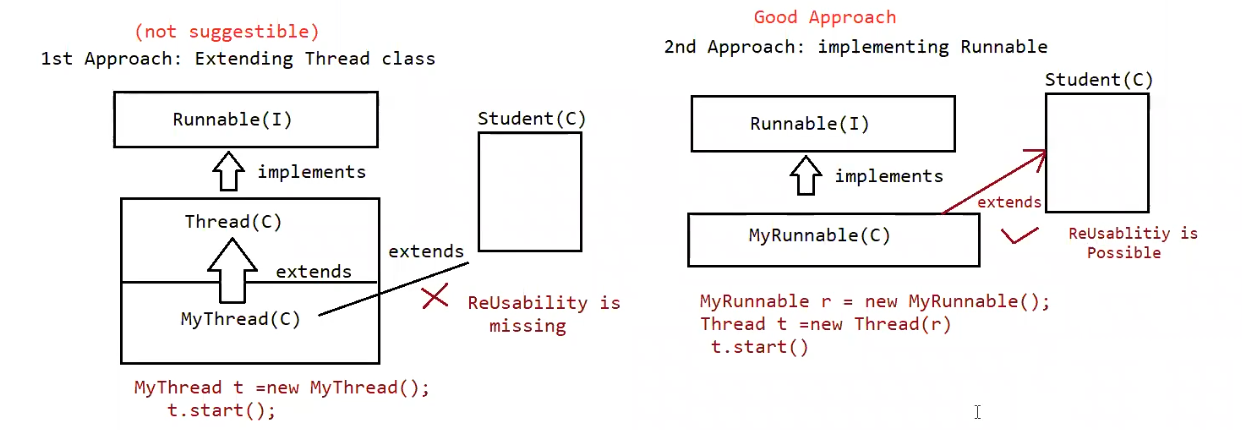
It results in compile time error

Case6:

r.run()

no new thread class will be created, but Mythread class run() method will be executed just like a normal method call.





Different approach for creating a thread

1. Extending the Thread class
2. Implementing Runnable interface

Which approach is best approach

1. implements Runnable interface is recommended because our class can extend other class through which inheritance benefit can brought into our class.

Internally performance and memory level is also good when we work with interface

1. if we work with extends feature then we will miss out inheritance feature benefit because already our class has inherited the feature from “Thread class”, so normally we don’t prefer extends approach rather implements approach is used in real time for working with “Multi-Threading”.

