IMPORTANT LINK: <http://tutorials.jenkov.com/java-concurrency/volatile.html>

MULTITHREADING IN JAVA

* You can overload run method in user defined class as run() and run(int) or run(String), etc. But when you call start() on thread instance it will call run() and not any other overloaded versions of run(). If you want, you can call them explicitly but then they will not be running in a different thread and will not have their own call stacks like a thread should have. Instead they will be invoked on the same/current call stack of the code you made the call (to overloaded version of run()) from, just like any other normal method.
* If you call run() directly on a Thread instance or a Runnable instance, the run() method will be called on the current call stack and no new thread of execution (and its call stack) will be created. To create a new thread of execution (and its call stack), you should always call start().
* start() cannot be called on Runnable instance because it is a method defined in Thread.
* Thread implements Runnable.
* We start a Thread and not a Runnable i.e. we can call start() only on a Thread instance and not on a Runnable instance.
* start() can be called on a Thread instance only once. If we try to call it again (whether the run() method has been called or not or run() has completed or not does not matter) we will get error as IllegalThreadStateException which is a RuntimeException. The compiler will not show this error (no red cross ‘x’ in eclipse).
* 5 states of a thread’s life cycle are – new, running, runnable, waiting/blocked/sleeping and dead/terminated.

Snippet 1:

**public** **class** ThreadDemo **extends** Thread{

**public** **void** run() {

System.***out***.println("Running!!!");

}

**public** **static** **void** main(String[] args) {

Thread t = **new** Thread();

System.***out***.println("Begin!");

t.start();

}

}

Output:

Begin!

Observation:

Here we are extending Thread class (which is ok) and when creating a thread we are creating an instance of Thread and not of user defined class ThreadDemo. As a result when this thread t calls start() it will call run() in Thread and not ThreadDemo.

If we pass ThreadDemo object to Thread’s constructor then we are explicitly providing a target to the thread t to call the run() method on. Output for below changed to above code is-

Thread t = **new** Thread(new ThreadDemo());

Output:

Begin!

Running!!!

Snippet 2:

**public** **class** ThreadDemo **implements** Runnable {

**public** **void** run() {

System.***out***.println("Running!!!");

System.***out***.println(**this**.getName());

}

**public** **static** **void** main(String[] args) {

Thread t = **new** Thread();

System.***out***.println("Begin!");

t.start();

}

}

Output:

Compiler error as: “The method getName() is undefined for the type ThreadDemo”.

Observation:

this refers to an instance of ThreadDemo which is not a Thread type and so it does not have getName() method which is defined in Thread. However, if you change the code such that ThreadDemo now extends Thread and you call getName() on ThreadDemo instance it will correctly call the method as ThreadDemo ‘IS-A’ Thread and will have the method.

**public** **class** ThreadDemo **extends** Thread {

**public** **void** run() {

System.***out***.println("Running!!!");

System.***out***.println(**this**.getName());

}

**public** **static** **void** main(String[] args) {

Thread t = **new** Thread(**new** ThreadDemo());

System.***out***.println("Begin!");

t.start();

}

}

Output:

Begin!

Running!!!

Thread-0