Core Java

Java Threads – Part 1

LEVEL – PRACTITIONER

About the Author

Created By:	Renjith(t-renjith)/ Shanmu (105110)
Credential Information:	Trainer / Sr Architect
Version and Date:	1.0, January 23rd 2012

Cognizant Certified Official Curriculum

Icons Used

Questions

Tools

Hands on Exercise

Coding Standards

Test Your Understanding

Case Study

Demonstration

Best Practices & Industry Standards

Workshop

Objectives

After completing this chapter you will be able to understand:

- What is Multithreading and it's advantages?
- What is Thread life cycle?
- What is meant by thread priority?
- What are the ways of implementing threads?
- How to use Thread class to create threads?

Process Vs Thread

Before we learn about Threads lets first understand the difference between a thread and a process.

Process are executables which run in separate memory space.

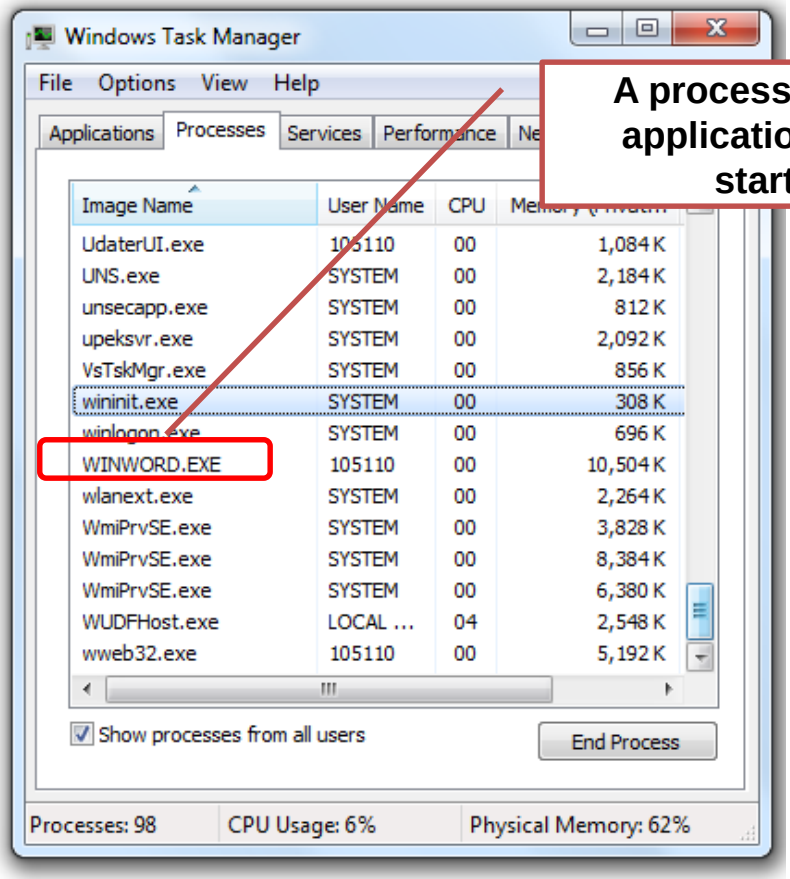
Threads are small process which run in shared memory space within a process.

So in a nutshell Process is a container for Threads, all thread runs inside the process.

Still confused lets look at a example to understand it better.

Process Example

Lets consider Microsoft word application to understand it better, what happens when you start an word application.



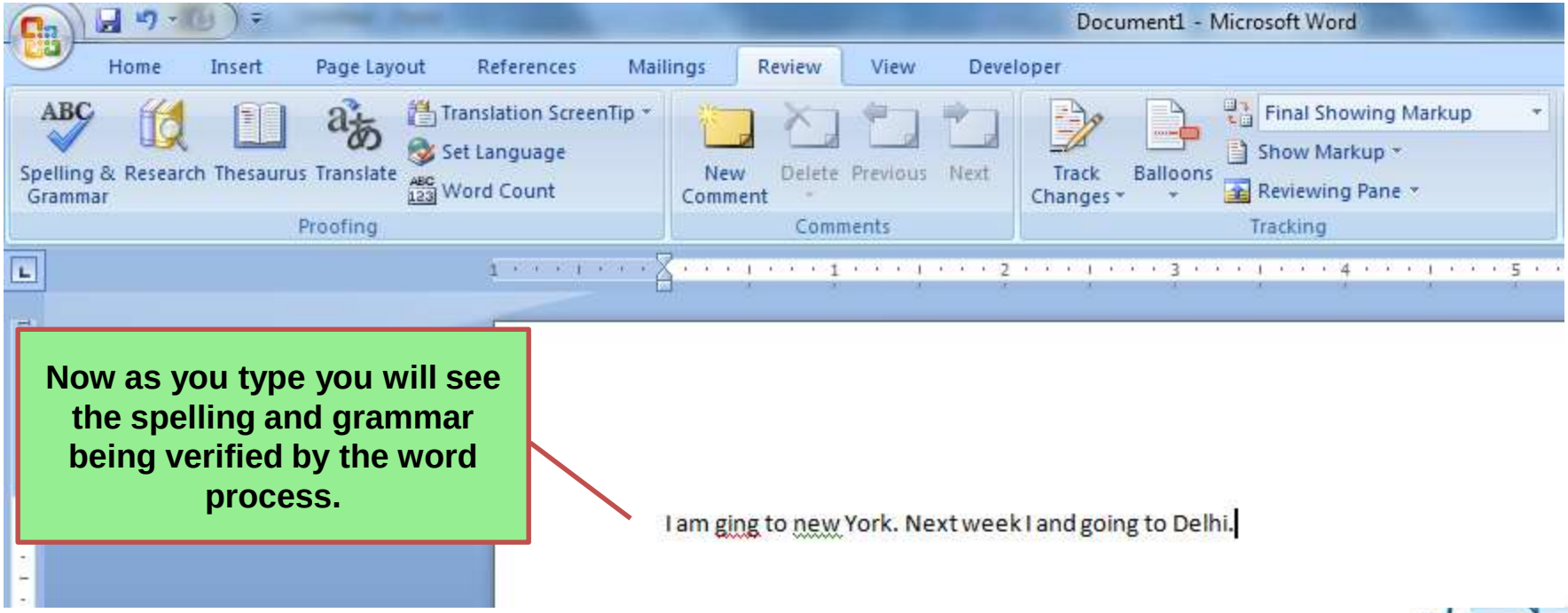
A process for word application will be started.

Now lets see what happens when the user starts using word application.

Thread Example

The spell check has been implemented as a ***thread*** within the worde.exe process which runs continuously and verifies what you type.

Word.exe is the **process** and **spell check** is a **thread** running inside the process.



A Problem statement

Tim was developing an application where he has a requirement where user can register his profile in the application , assume registration has three steps

- ***Validate user details*** – Takes 3 seconds for execution for each user.
- ***Validate user Citizenship -*** Takes 4 seconds for execution for each user.

Now customer wants to complete the registration process is less than 5 seconds.

Guess how Tim would have achieved it?

He used Multi Threading.

Lets see what it is and how it can be implemented in this session.

What is Multitasking and Multithreading ?

What is Multitasking?

- Refers to a computer's ability to perform multiple jobs concurrently
- More than one program are running concurrently,

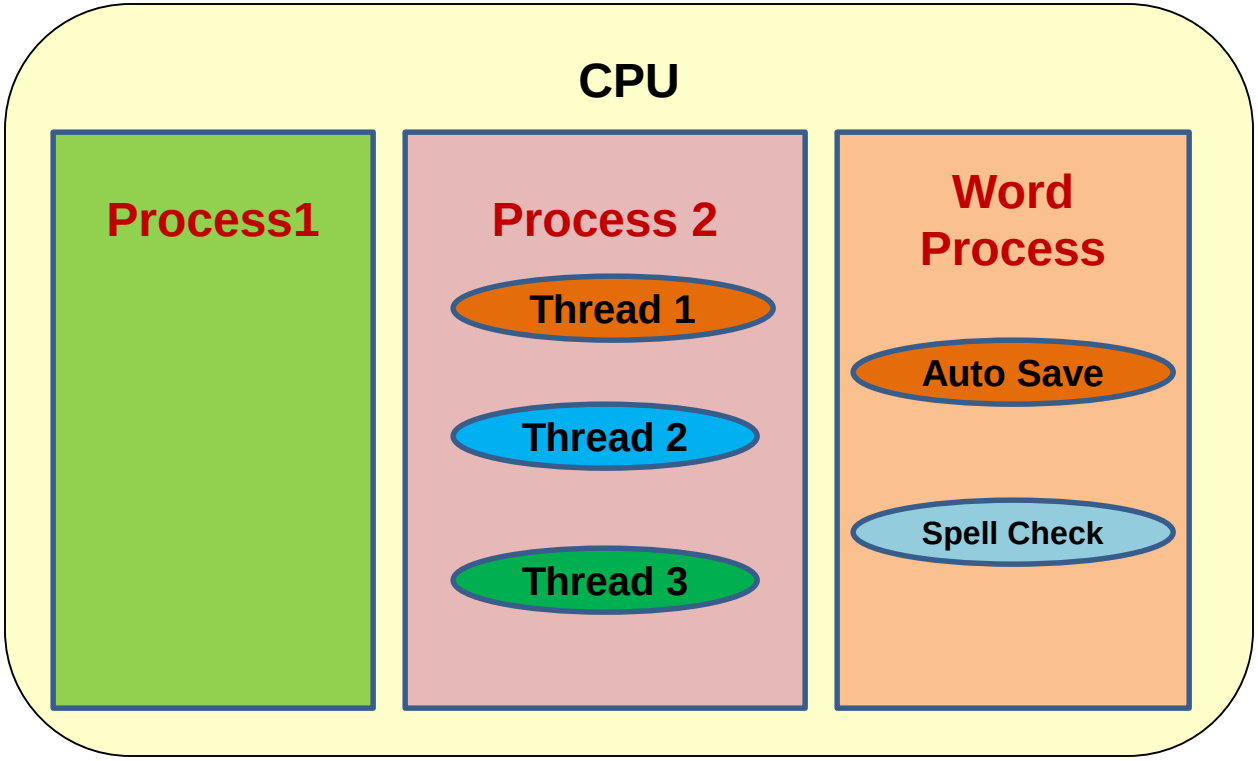
Example: In windows you can run Word, power point , media player at the same time. Yu will working on word and in parallel media player might play some music.

What is Multithreading?

- A thread is a single sequence of execution within a program/process.
- This refers to multiple threads of control within a single program.
- Each program can run multiple threads of control within it.

Example: Microsoft word process having multiple threads like spell check, auto save etc.

Multitasking & Multithreading Illustration



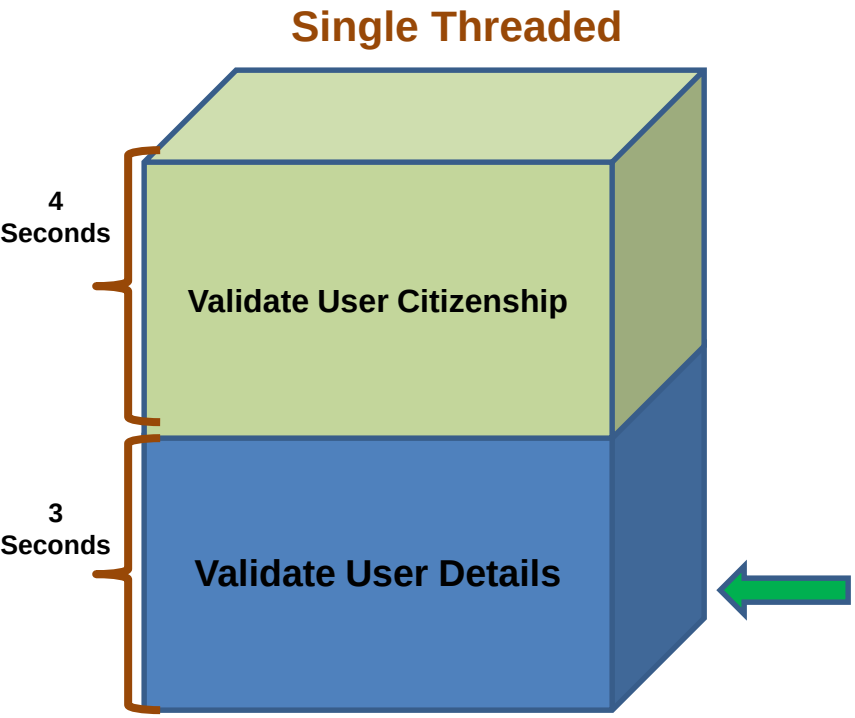
The CPU is running three processes. Process 2 in turn has three threads running inside it.

Benefits of Multithreading

- To reduce response (execution) time of a process.
- Support Parallel Operation of Functions.
- Increase System Efficiency.
- Requires less overheads compared to Multitasking.

How Tim solved the problem?

Tim implemented two threads for processing the **Validate user details** and **Validate user Citizenship.**



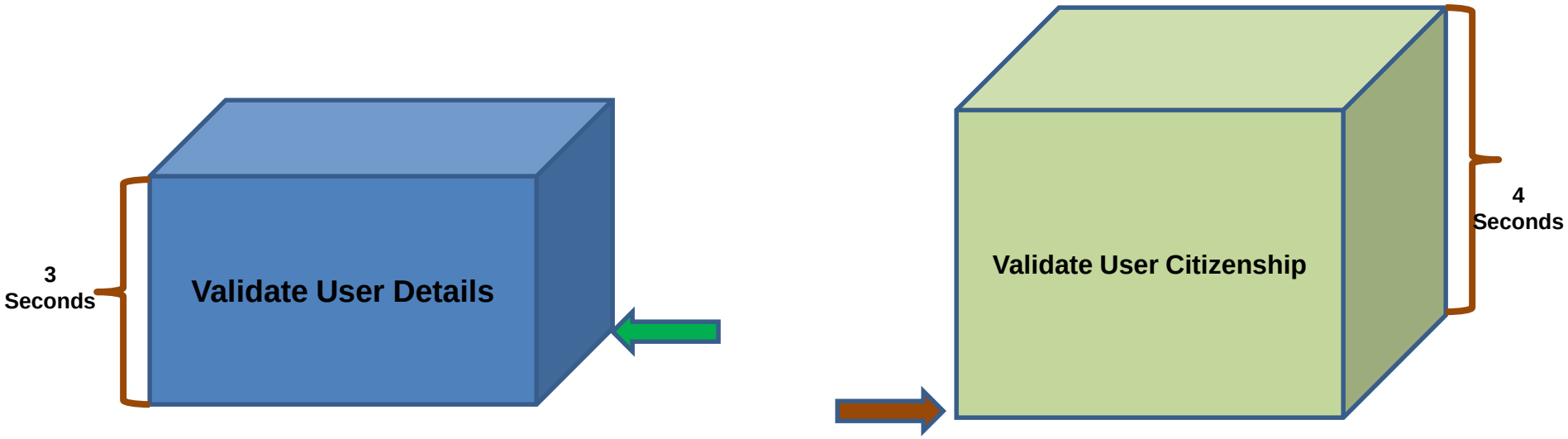
The thread takes 7 seconds for completing the registration process.

Let look how Tim implemented it.

How Tim solved the problem?

Tim implemented two threads one thread for each method **Validate user details** and **Validate user Citizenship.**

Multi Threaded



The process will be completed in 4 seconds. Since both the threads works in parallel.

What is an Application Thread?

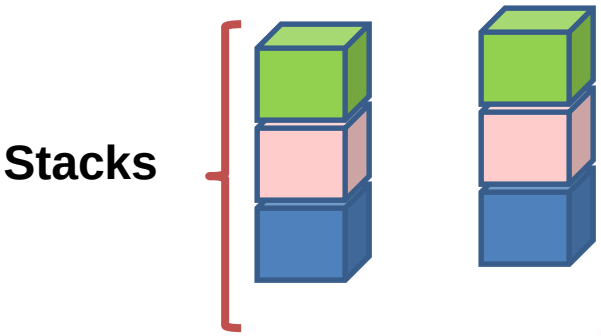
When we execute an application,

1. The JVM creates a thread (**T1**) object which invokes the **main()** method and starts the application.
2. The thread executes the statements of the program one by one (or) starts other threads.
3. After executing all the statements, the method returns and the thread dies.

The thread **T1** which is responsible for starting the application by invoking the main method is called "***Application Thread***".

How Multiple threads run in an application?

- Each thread has its private run-time stack for storing variables and data.
- If two threads execute the same method, each will have its own copy of the stack to store the method local variables.
- The objects instance variables are (Class variables) shared across all the threads, they are not ***thread safe.***



How Multiple threads run in an application?

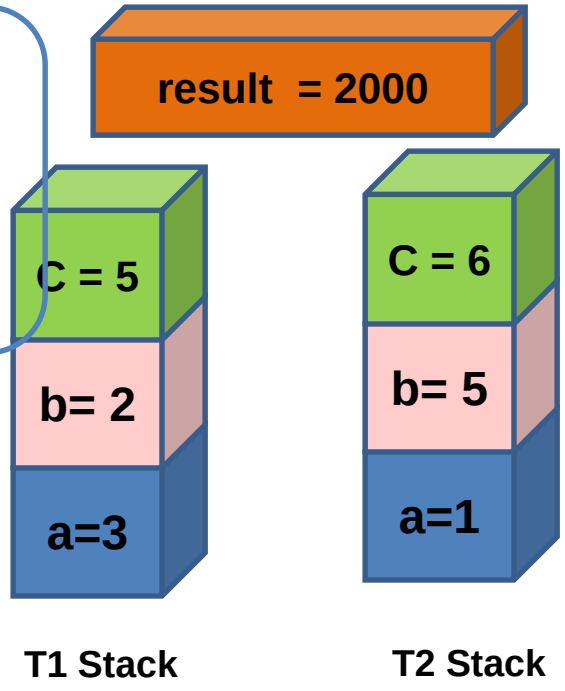
```
public class Calculator{

    int result =100;
    public void add(int a, int b)
    {
        int c = a+b;

    }

}
```

The result instance variable will be shared by both the threads. **NOTE:** For thread T2 the result will be displayed as 2000.



Assume two threads are executing the above methods.
Thread T1 : a=3 & b=2
Thread T2 : a=1 & b=5

Assume thread T1 sets the class variable result to 2000.

Each thread will have its own stack to store the method local variables.

Ways of Implementing Threads

Method 1 : Extend ***Thread*** Class

Method 2 : Implement ***Runnable*** Interface

In this session we will learn about Method 1.

Thread Class Methods

Method	Description
void run()	• The thread logic should be implemented in this method. • This method should be overridden in all the Thread class.
void start()	Creates a new thread and invokes run method of the thread.
getName()	Returns the thread's name.
int getPriority()	Returns the thread's priority
boolean isAlive()	Tests if the thread is alive.

Thread Class Methods

Method	Description
boolean isDaemon()	Tests if the thread is a daemon thread.
setName(String name)	Sets a name for the thread to be equal to the argument name.
setPriority (int newPriority)	Changes the priority of the thread
static void sleep(long millis)	Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
void yield()	Causes the currently executing thread object to temporarily pause and allow other threads of same priority to execute

Method 1 : Steps to develop thread using Thread

Step 1 : Develop a Thread class by extending the ***java.lang.Thread*** class

```
class ThreadDemo extends Thread
```

Step 2 : Override the ***run()*** function for making the code run as a separate thread

```
class ThreadDemo extends Thread {
  void run() {
        // do Something
        }
}
```

Step 3 : Create an entry point for the Thread (main method) which can start the thread.

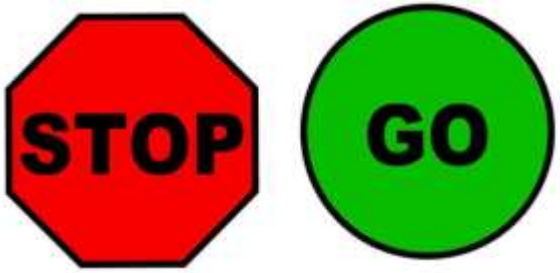
Method 1 : Steps to develop thread using Thread

Step 4 : Create an instance of the Thread class.

Step 5 : Invoke the ***start ()*** method on the ***Thread*** object which in turn invokes the ***run()*** method in the Thread class.

```
class ThreadProgram {
        public static void main(String args[])
            {
              ThreadDemo thread=new ThreadDemo();
              thread.start() ;
            }
      }
```

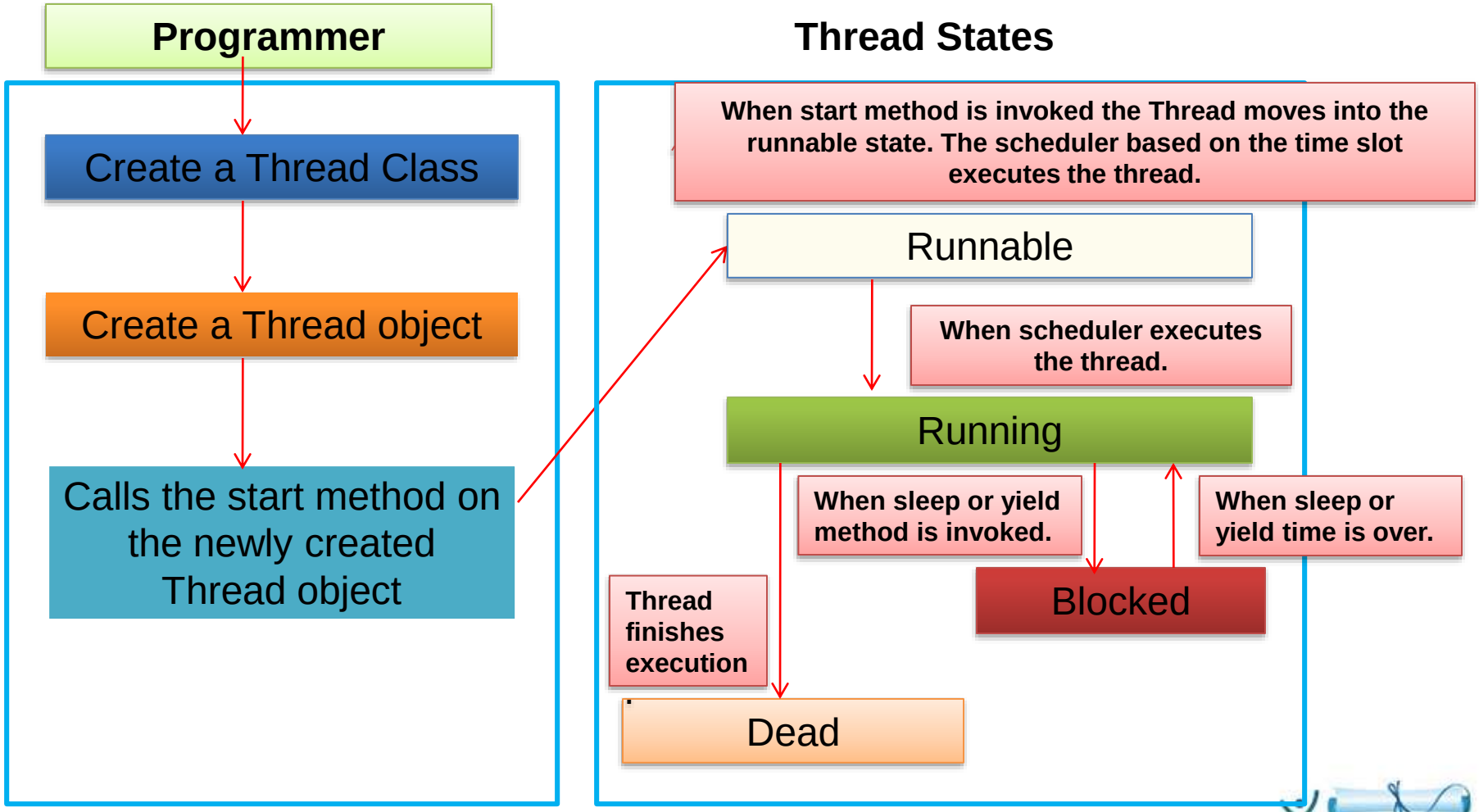
Time To Reflect



Associates to quickly summarize the following before proceeding the session

- What is the difference between Thread and process?
- What is the difference between multitasking and multiprocessing
- What are the two methods of implementing multithreading in java?
- What is the method used to set the priority of a thread?
- How to start a new Thread ?

Thread Life Cycle



Scheduling Threads

- Execution of multiple threads on a single CPU, in some order, is called ***scheduling***.
- The Java runtime supports a very simple, deterministic scheduling preemptive ***priority scheduling*** algorithm.
- This algorithm schedules threads based on their ***priority.***
- At any given time, when multiple threads are ready to be executed, the runtime system chooses the **runnable** thread with the highest priority for execution.
- Low priority thread gets a chance only when threads with high priority move to a non runnable state.

Scheduling Threads(Cont)

- If two threads of the same priority are waiting for the CPU, the scheduler chooses one of them to run in a round-robin fashion. The chosen thread will run until one of the following conditions is true:
- A higher priority thread becomes "Runnable"
- It yields, or its run() method exits

Then the second thread is given a chance to run, and so on, until the interpreter exits.

- The Java runtime system's thread scheduling algorithm is also ***preemptive***.
- If at any time a thread with a higher priority than all other "***Runnable***" threads becomes "***Runnable***", the runtime system chooses the new higher priority thread for execution. The new higher priority thread is said to ***preempt*** the other threads.

Thread Priority

- ***Priorities*** determine, which thread receives CPU control and gets to be executed first.
- When a thread is created, it inherits the priority of the thread that created it.
- The priority values range from 1 to 10, in increasing priority.
- The priority can be adjusted subsequently using the ***setPriority()*** method.
- The priority of a thread may be obtained using ***getPriority().***
- Priority constants are defined in ***Thread*** Class.

- MIN_PRIORITY=1
- MAX_PRIORITY=10
- NORM_PRIORITY=5

- The **main** thread is created with priority NORM_PRIORITY

Important Note

As per the specification, at any given time, the highest priority thread is always running.

However, this is not guaranteed.

This depends on the Thread scheduler as it can run a lower priority thread to avoid starvation without getting a chance to execute.

So don't rely on thread priority for algorithm correctness.

What are Daemon Threads?

- ***Daemon threads*** are “background” threads, that provide services to other threads.

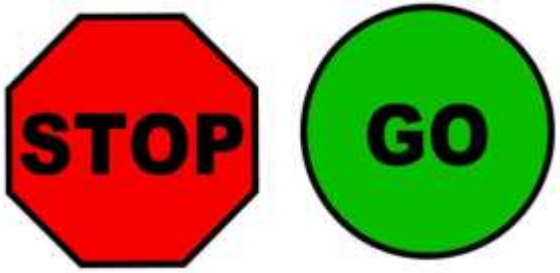
  Example: The garbage collection thread.

- The Java VM **cannot exit** if non-Daemon threads are executing.
- The Java VM **can exit** Daemon threads are executing.
- Daemon threads die when the Java VM exits.

Thread Group

- A ***thread group*** represents a set of threads.
- Thread groups are typically used if a collection of similar threads needs to be collectively managed. These threads will be added to a thread group.
- A thread group can also include other thread groups.
- The thread groups form a tree in which every thread group except the initial thread group has a parent.

Time To Reflect



Associates to quickly summarize the following before ending the session

- What is the thread state when a new thread is created?
- What is the state of the thread when a sleep or yield method is invoked?.
- Can a Thread in the dead state move back to the runnable state?
- A thread with priority 6 creates a new Thread **T1**. What will be the priority of thread **T1**?

Lend a Hand – How to develop a thread?

In this demo we will learn how to

1. Create a Thread by extending the Thread class
2. Override the run method
3. Create a thread object and start the thread using the start method
4. How the following methods operates

a. getName()

b. setName()

c. setPriority()

d. getPriority()

e. sleep()

Components to be developed,

1. **ThreadEX** – The Thread class should loop and print values 0…4.
2. **ThreadExMain** – The main class to execute the Threads

Lend a Hand Solution - How to develop a Thread?

```
public class ThreadEx extends Thread {
    int i = 0;

    public ThreadEx(String name) {
        this.setName(name);

    }

    public void run() {
        for (i = 0; i < 5; i++) {
            System.out.println("Printing from  " + Thread.currentThread().getName()
                    + "  the value of i  :: " + i);
            try {
                Thread.sleep(300);
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

Class **ThreadEx** extends Thread class and becomes a Thread class

Sets the name of the Thread using **setName** method()

Overrides the **run()** method

Prints the name of the current Thread using the **getName()** method

Makes the Thread **sleep** for 300 milliseconds. Invoking the sleep may cause an **InterruptedException** to be thrown which should be handlers.

Lend a Hand Solution– How to develop a Thread?

```
public class ThreadExMain {
    public static void main(String args[]) throws InterruptedException {
        Thread t = Thread.currentThread();
        System.out.println("The Current Thread is " + t.getName());
        System.out.println("The Current Thread Priority is " + t.getPriority());
        t.setName("myThread");
        t.setPriority(6);
        System.out.println("The Current Thread is " + t.getName());
        System.out.println("The Current Thread Priority is " + t.getPriority());
        ThreadEx ex1 = new ThreadEx("first");
        ThreadEx ex2 = new ThreadEx("second");
        System.out.println("The   Thread Priority of ex1 is "
                + ex1.getPriority());
        System.out.println("The   Thread Priority of ex2 is "
                + ex2.getPriority());
        ex1.start();
        ex2.start();
        for (int i = 0; i < 5; i++) {
            System.out.println("Printing from :  "
                    + Thread.currentThread().getName() + " :: " + i);
            Thread.sleep(100);
        }
    }
}
```

Gets the priority and name of the main thread

Sets the priority and name of the main thread

Creates two new Thread objects with name "**first** "and "**second**"

Starts the Thread by invoking the **start ()** which will invoke the **run()** method of the Thread class

Lend a Hand Output - How to develop a Thread?

- Execute the main class – ***ThreadExMain***
- The output will be something as shown below – Output may vary for each execution since Thread execution is based on the underlying operating system.

```
The Current Thread is main
The Current Thread Priority is 5
The Current Thread is myThread
The Current Thread Priority is 6
The   Thread Priority of ex1 is 6
The   Thread Priority of ex2 is 6
Printing from :  myThread :: 0
Printing from  first  the value of i  :: 0
Printing from  second  the value of i  :: 0
Printing from :  myThread :: 1
Printing from :  myThread :: 2
Printing from  first  the value of i  :: 1
Printing from  second  the value of i  :: 1
Printing from :  myThread :: 3
Printing from :  myThread :: 4
Printing from  second  the value of i  :: 2
Printing from  first  the value of i  :: 2
Printing from  second  the value of i  :: 3
Printing from  first  the value of i  :: 3
Printing from  second  the value of i  :: 4
Printing from  first  the value of i  :: 4
```

Different output displayed during different run

```
The Current Thread is main
The Current Thread Priority is 5
The Current Thread is myThread
The Current Thread Priority is 6
The   Thread Priority of ex1 is 6
The   Thread Priority of ex2 is 6
Printing from :  myThread :: 0
Printing from  first  the value of i  :: 0
Printing from  second  the value of i  :: 0
Printing from :  myThread :: 1
Printing from :  myThread :: 2
Printing from  second  the value of i  :: 1
Printing from  first  the value of i  :: 1
Printing from :  myThread :: 3
Printing from :  myThread :: 4
Printing from  second  the value of i  :: 2
Printing from  first  the value of i  :: 2
Printing from  first  the value of i  :: 3
Printing from  second  the value of i  :: 3
Printing from  first  the value of i  :: 4
Printing from  second  the value of i  :: 4
```

Core Java

You have successfully completed – Java Threads Part - 1