Lecture 01: Concurrency (Part 1)

(22 slides)

Why should you study this chapter?

- This chapter will help you developing a program in which some tasks executing concurrently.
- Nowadays, in one program, some tasks execute concurrently. You usually saw them in a web page including texts, sounds, images, games, … changing concurrently.
- Nowadays, operating systems (OS) support many programs running concurrently.
- Open the Task Manager of Windows OS (Ctrl Alt Delete) to see how many programs are running in your computer.

Review

Concepts introduced in the course OOP using Java:

- Object = properties + methods
- Benefits of OOP Implementation: encapsulation, inheritance, polymorphism
- Encapsulation is implemented by modifiers
- Inheritance: Subclass inherits declaration of base class (Java is a single inheritance OOP language, the class Object is the ultimate Java class
- Polymorphism in Java is implemented by overloading and overriding methods

Review

- Class: A template (blueprint) for a group of similar instances
- Interface: the core of some classes
- Abstract class is a class containing abstract methods (some methods are prototyped only)
- Anonymous class: A concrete class developed from an abstract class or an interface but it is not named:
- Nested class: a class which is declared in the declaration of the enclosing class
- Enum type: Declaration of some constants

Review

- **Class Declaration:**

```
[public] class ClassName [extends BaseClass] [implements Interface] {
    [modifiers] DataType field1 [ = initialValue];
    [modifier]  ClassName (params) {  // constructor
    <code>
    }
   [modifier]  methodName(params) {  // method
    <code>
    }
}
```

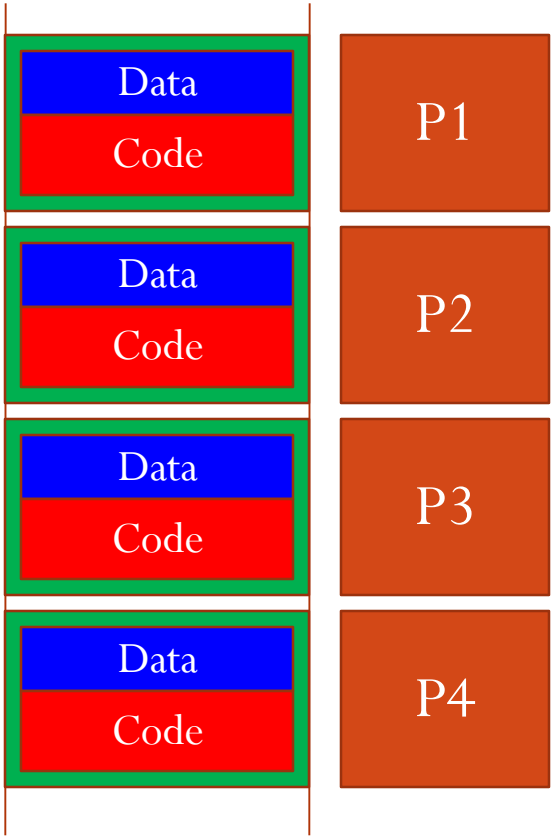
- **Creating objects:** ClassName obj = **new** ClassName(params)
- **Accessing an object**: **obj.field** or **obj.methodeName(args)**

Objectives

- Multi-Processing System
- Threads and Multi-Threading Program
- Thread Fundamentals in Java

1- Processes and Multi-Processing System

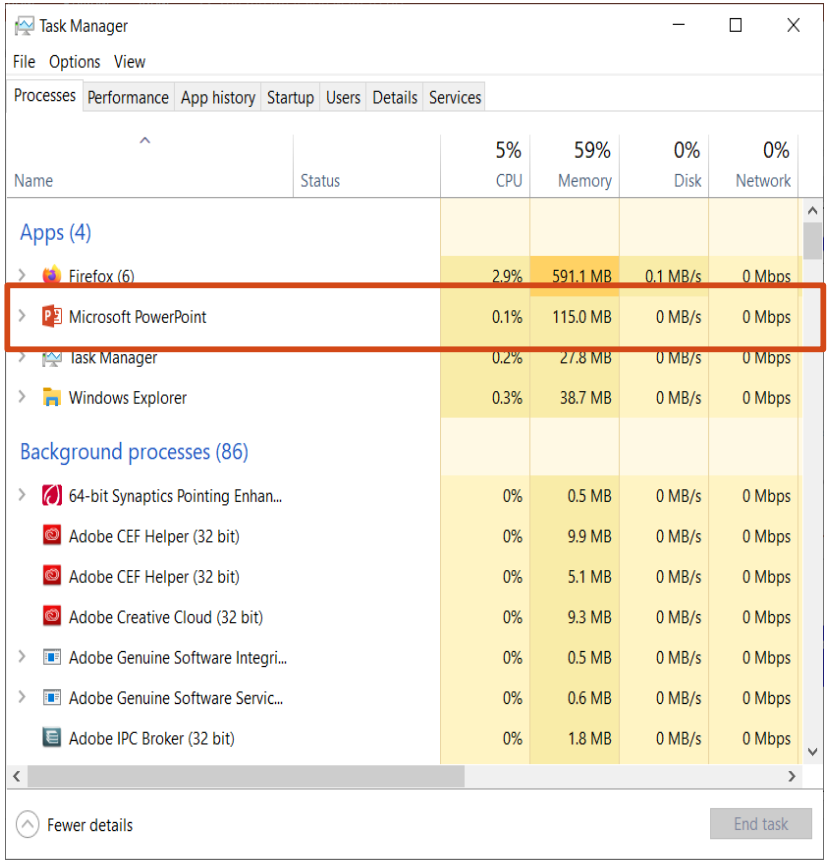
- Program: An executable file (data + code) stored in secondary memory(disk).
- Process: A program in running. It's data and code are loaded to RAM in a contiguous memory block.
- Multi-Processing System: An operating system allows some processes running concurrently

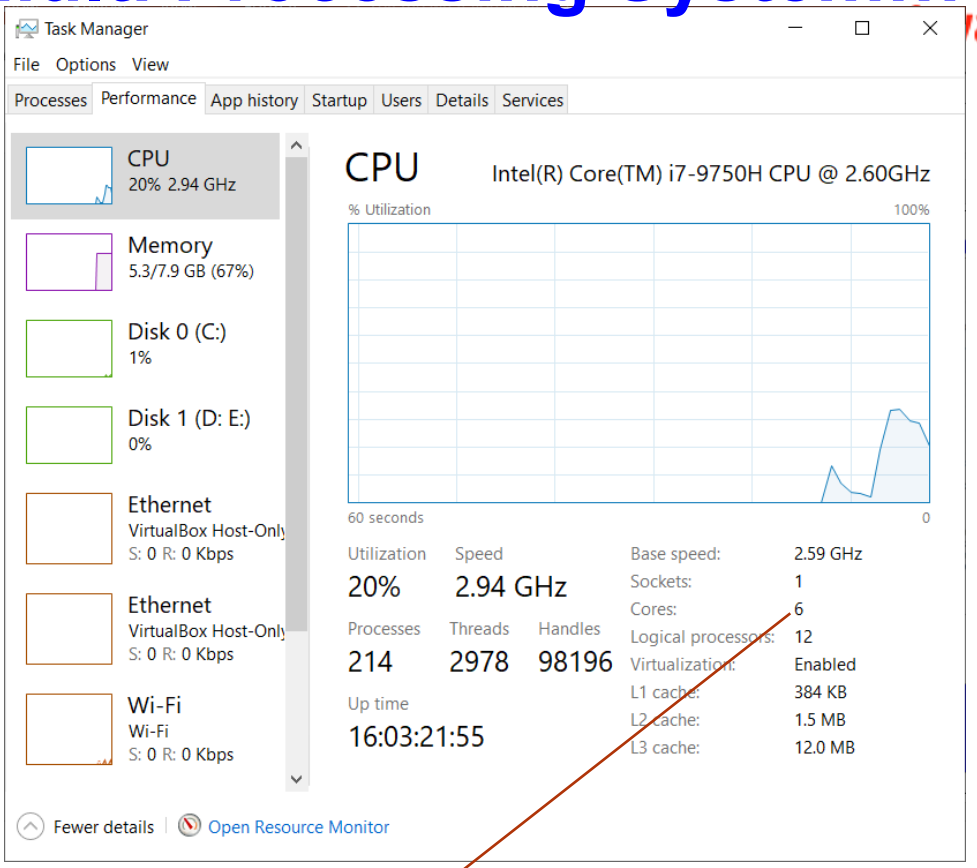


Processes and Multi Processing System…

- A process has a self-contained execution environment. A process generally has a complete, private set of basic run-time resources; in particular, each process has its own memory space.
- Multi Processing/ Multi Tasking System: Almost of operating systems allows many processes executing concurrently.
- Open the ***Task Manager*** of Windows OS (Ctrl+Alt+Del) to see current processes in your computer.
  - Applications tag contains processes which you start them.
  - Processes tag contains processes which automatically run immediately after the startup of OS.

Processes and Multi Processing System...



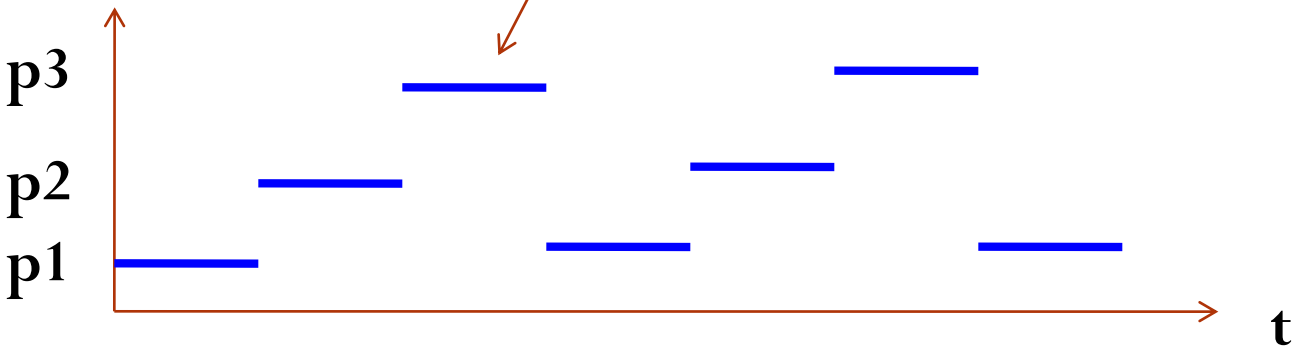


CPU 6 cores → 1 core runs 36 (214/6) processes

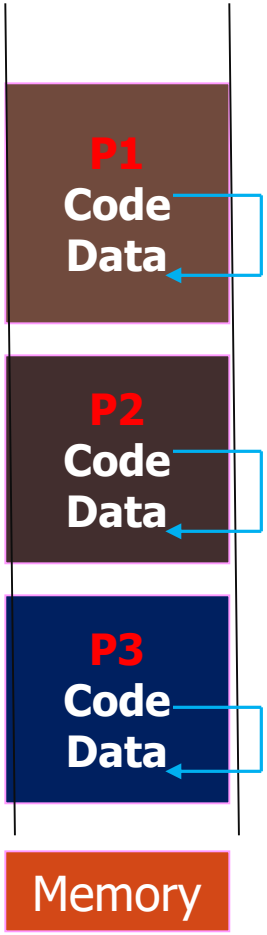
→ Pseudo -parallelism

Processes and Multi Processing System…

- How OS manages processes based on one CPU?
  - Time-sharing mechanism: OS allows each process running in a time slot (time slice, quantum, about 50 to 70 ms). When this duration expires, another process will be chosen to run → The scheduler (a component of OS, trình lập lịch) will choose the current process.

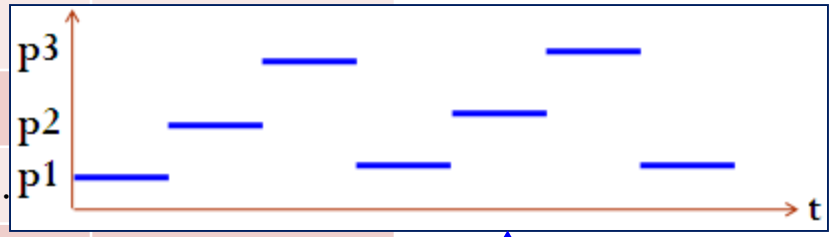


Processes and Multi Processing System...
How can OS manage concurrent processes?



Process Table maintains information of processes.

App	Code Addr	Duration (mili sec)	CPU	…
P1	30320	50	1	
P2	20154	60	2	
P3	10166	70	1	
…	…	…	…	
…	…	…	…	



Time-slicing mechanism. Each process is allocated resources (CPU, …) for executing in a time-slot (such as 50 milliseconds). When the time duration expires, this process will pause to yield resources to the next process which will be chosen by the scheduler of OS.

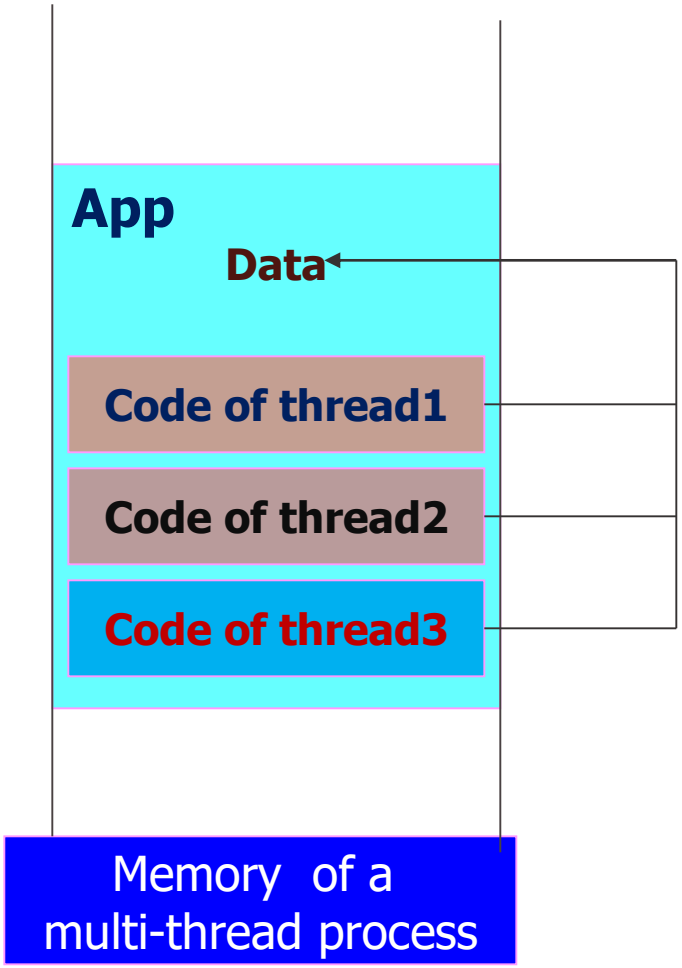
2- Threads and Multi-Threading

- Thread, is sometimes called *lightweight processes, is a running code unit.*
- Threads exist within a process — **every process has at least one thread (main thread)**. Threads share the process's resources, including memory and open files. This makes for efficient, but potentially problematic, communication.
- Both processes and threads are provided an execution environment, but creating a new thread requires fewer resources than creating a new process.
- Multithreaded execution is an essential feature of the Java platform. Threads in a program are managed by the JVM. Scheduler in JVM will choose a current thread.

Threads and Multi-Threading …
How can JVM manage threads

- Thread is a smallest unit code in an application that performs a special job.

➔ A program can have several threads they can be executed concurrently.



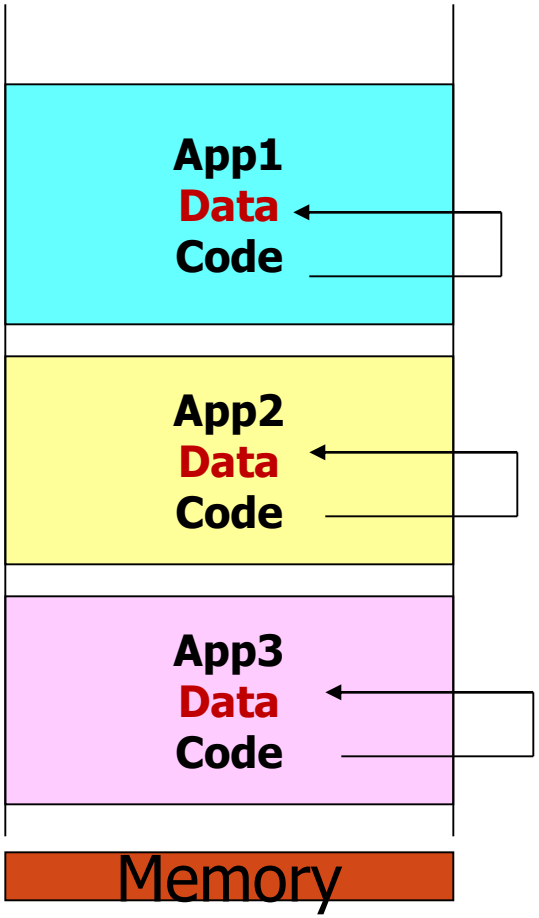
Thread Table maintains information of threads

Thread	Code Addr	Duration (mili sec)	CPU	State
Thread 1	10320	15	1	ready
Thread 2	40154	17	2	ready
Thread 3	80166	22	1	sleep
…	…	…	…	….

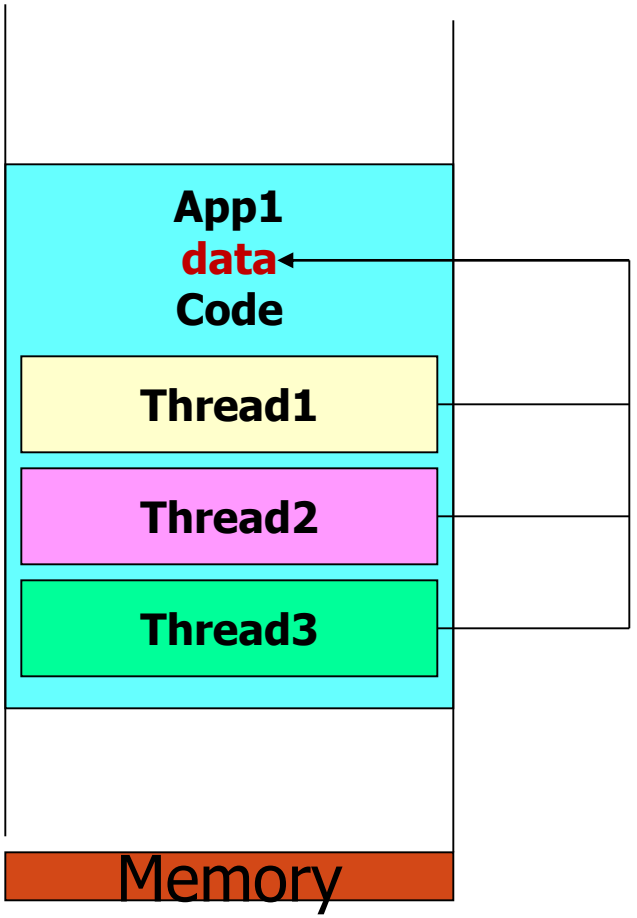
Time-slicing mechanism is used to schedule thread executions also.

Threads and Multi-Threading …

Processes VS Threads



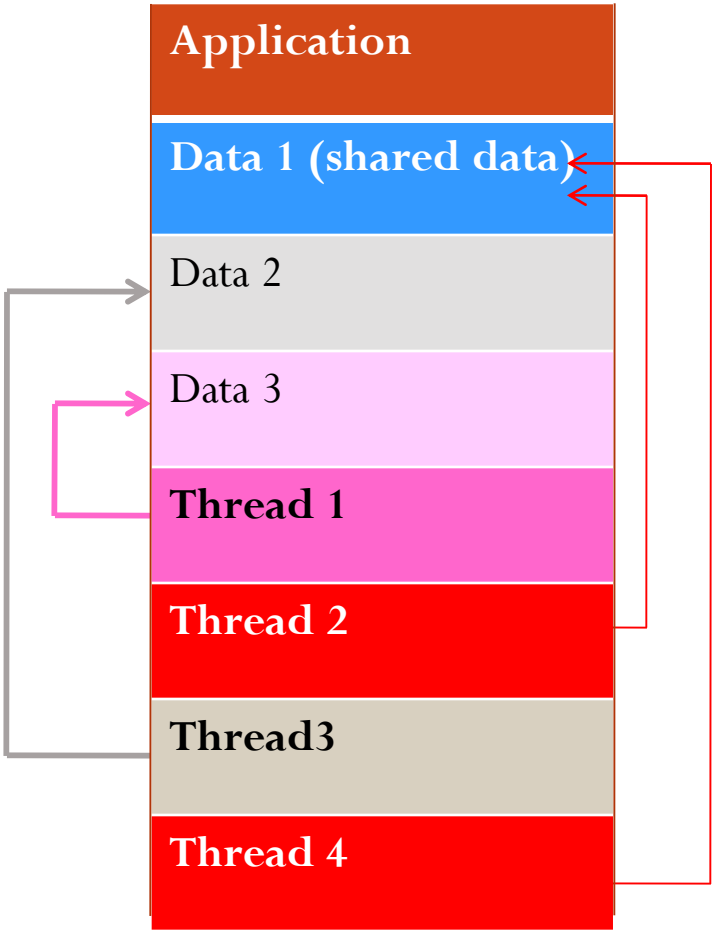
Protection mechanism in OS does not allow this process accessing addresses in others



Threads in a process can access common data of this process

Threads and Multi-Threading …

Race Conditions



Suppose that Thread2 and Thread4 concurrently execute.

Time	Thread 2	Thread4
1	Data1=10	…
2	….	…
3	….	Data1= 100
4	….	…
5	…	…
6	Y= 2*Data1;	…
7		…

Y=?

A race occurs

Need Synchronization

3- Thread Fundamentals in Java

- **Threading supports in Java:**
  - **The java.lang.Thread** class
  - **The java.lang.Object** class
  - **The Java language and JVM (Java Virtual Machine)**
- **How to create a thread in Java?**
  - (1) Create a subclass of the **java.lang.Thread** class and override the method **run()**
  - (2) Create a class implementing the **Runnable interface** and override the **run()** method.

Create a subclass of the Thread class

Thread1.java * | Thread1Using.java

```
1  /* Thread1.java */
2    public class Thread1 extends Thread {
3
4        public Thread1() {
5            super ();
6        }
7        public void run() {
8            System.out.println("I'm a child thread");
9        }
10   }
```

Thread1.java * | Thread1Using.java

```
1  public class Thread1Using {
2      public static void main(String args[])
3      {   Thread1 t= new Thread1();
4          System.out.println("I'm the main thread.");
5          t.start();
6          System.out.println("Hello");
7      }
8  }
```

Thread table

Thread	Code Addr	State	…
main	8000	run	…
t	10320	run	

Every process has at least one thread (main thread).

Output - ThreadDemo (run-single)

```
compile:
run-single:
I'm the main thread.
Hello
I'm a child thread
```

?

The start() method is implemented in the Thread class. This method calls the method run().

This process has 2 threads running concurrently. At a time, order of their execution is decided by the scheduler.

Class implements the Runnable interface

RunnableThread.java * | RunnableClassUsing.java

```
1  public class RunnableThread implements Runnable {
2
3      public RunnableThread() {
4      }
5      public void run() {
6          System.out.println("I'm a child thread");
7      }
8  }
```

RunnableThread.java * | RunnableClassUsing.java *

```
1  public class RunnableClassUsing {
2
3      public static void main(String args[]) {
4          RunnableThread obj= new RunnableThread();
5          Thread t= new Thread (obj);
6          t.start();
7          System.out.println("I'm the main thread.");
8          System.out.println("Hello");
9      }
10 }
```

Thread	Code Addr	State	…
main	8000	run	…
t	10320	run	

Output - ThreadDemo (run-single)

```
compile:
run-single:
I'm the main thread.
Hello
I'm a child thread
```

The java.lang.Thread class

Declaration
public class **Thread** extends **Object** implements **Runnable**

set/get-is

Properties
id
name
state
threadGroup
daemon ['di:mən]
priority

Constructor
Thread()
Thread(Runnable target)
Thread(Runnable target, String name)
Thread(String name)
Thread(ThreadGroup group, Runnable target)
Thread(ThreadGroup group, Runnable target, String name)
Thread(ThreadGroup group, Runnable target, String name, long stackSize)
Thread(ThreadGroup group, String name)

Common Methods
start()
join ()
sleep (milisec)
yield()
notify()
notifyAll()
wait()

static int	MAX_PRIORITY	10
static int	MIN_PRIORITY	1
static int	NORM_PRIORITY	5

Using some methods of the Thread class

ThreadProperties.java

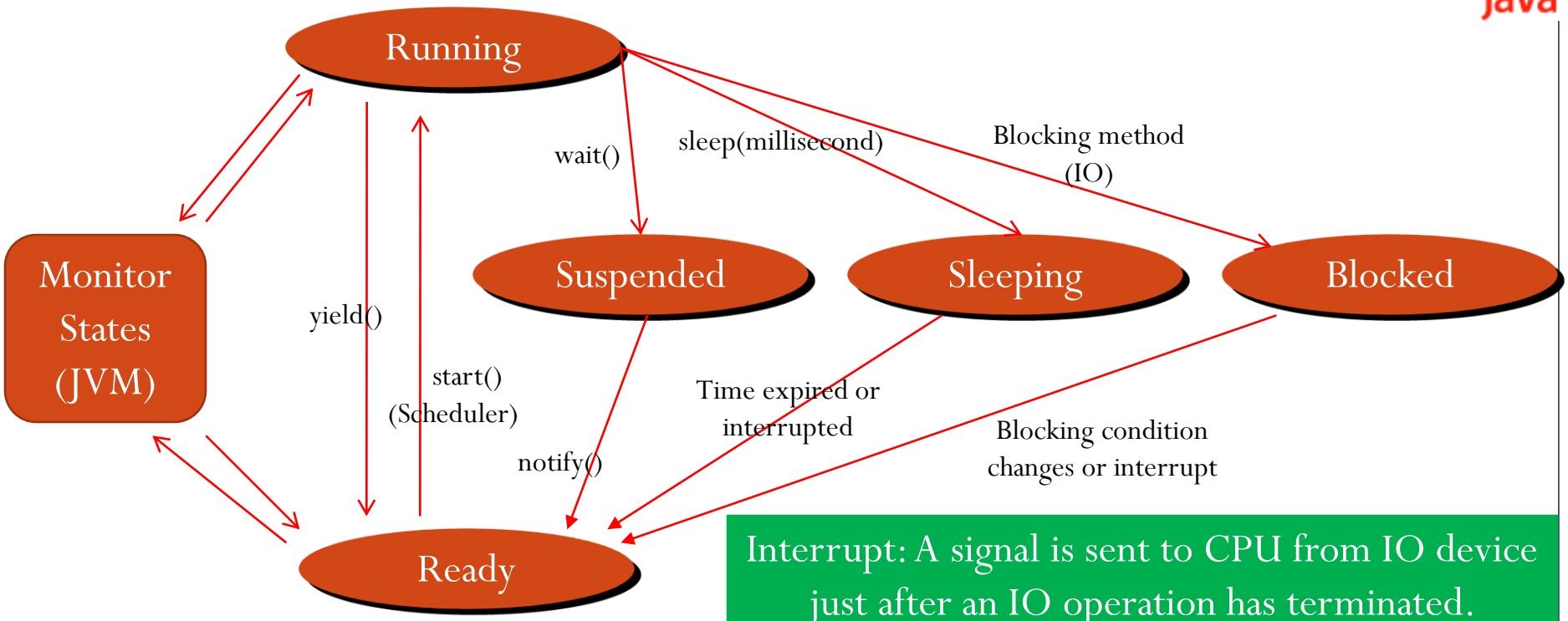
```
1  public class ThreadProperties extends Thread{
2      public ThreadProperties(String threadName) {
3        super(threadName);
4        this.start();
5      }
6      public static void showProperties(Thread t){
7        System.out.println("I'm the " + t.getName() + " thread");
8        System.out.println("--My ID:" + t.getId());
9        System.out.println("--My name:" + t.getName());
10       System.out.println("--My priority:" + t.getPriority());
11       System.out.println("--My state:" + t.getState());
12       System.out.println("--I'm a deamon:" + t.isDaemon());
13       System.out.println("--I'm alive:" + t.isAlive());
14     }
15     public void run(){
16       showProperties(this);
17     }
18     public static void main (String args[]){
19       System.out.println("Thread count:" + Thread.activeCount());
20       Thread t= Thread.currentThread();
21       showProperties(t);
22       ThreadProperties t1= new ThreadProperties("Son1");
23       ThreadProperties t2= new ThreadProperties("Son2");
24       System.out.println("Thread count:" + Thread.activeCount());
25   }
26 }
```

Output

Debugger Console | ThreadD

```
run-single:
Thread count:1
I'm the main thread
--My ID:1
--My name:main
--My priority:5
--My state:RUNNABLE
--I'm a deamon:false
--I'm alive:true
Thread count:3
I'm the Son2 thread
--My ID:9
--My name:Son2
--My priority:5
--My state:RUNNABLE
--I'm a deamon:false
--I'm alive:true
I'm the Son1 thread
--My ID:8
--My name:Son1
--My priority:5
--My state:RUNNABLE
--I'm a deamon:false
--I'm alive:true
```

Thread States



Interrupt: A signal is sent to CPU from IO device just after an IO operation has terminated.

Only one of ready threads will be chosen by the JVM scheduler at a time.

Ready: As soon as it is created , it can enter the **running** state when JVM's processor is assigned to it.

Running: It get full attention of JVM's processor which executes the thread's **run()** method

Dead: When the **run()** method terminates.

Summary

Concepts were introduced:

- Definitions: Program, Process, Thread
- Multi-processing system
- Multi-threading programming in Java
- Thread Fundamentals in Java
- Thread states

Thank You