Brighter Blue

Threads and Garbage Collection

Core Java: Day 3

IBM

Agenda

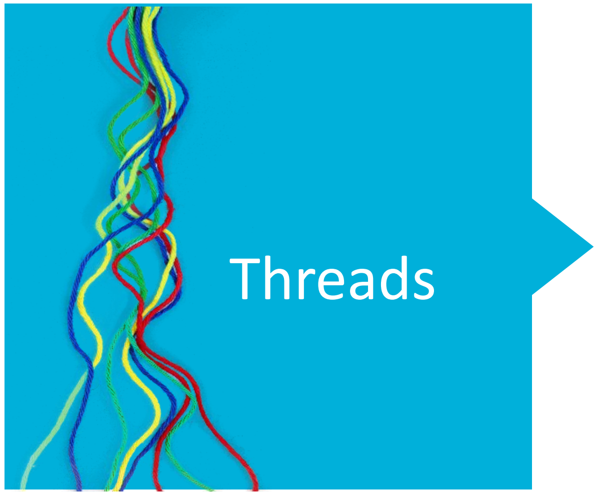
Day 03

01 Threads

02 Garbage Collection

01 Threads

What is Java Thread?



Thread is a piece of code that runs in concurrent with other Threads.

A Thread is defined as the path of execution of a program. It is a sequence of instructions that is executed to define a unique flow of control.

Benefits of Threads versus Processes

Threads have some advantages of (multi) processes, They take less time to:

- Create a new Thread rather than a process
- Terminate a Thread rather than a process
- Switch between two Threads within the same process
- Communication overheads

Why Threads?

The reasons for using Threads are that they help in:

1. Improved performance
2. Minimized system resource usage
3. Simultaneous access to multiple applications
4. Program structure simplification
5. Send and receive data on network
6. Read and write files to disk
7. Perform useful computation (editor, browser, and game)

Process versus Thread

Process **Versus** **Thread**

Definition

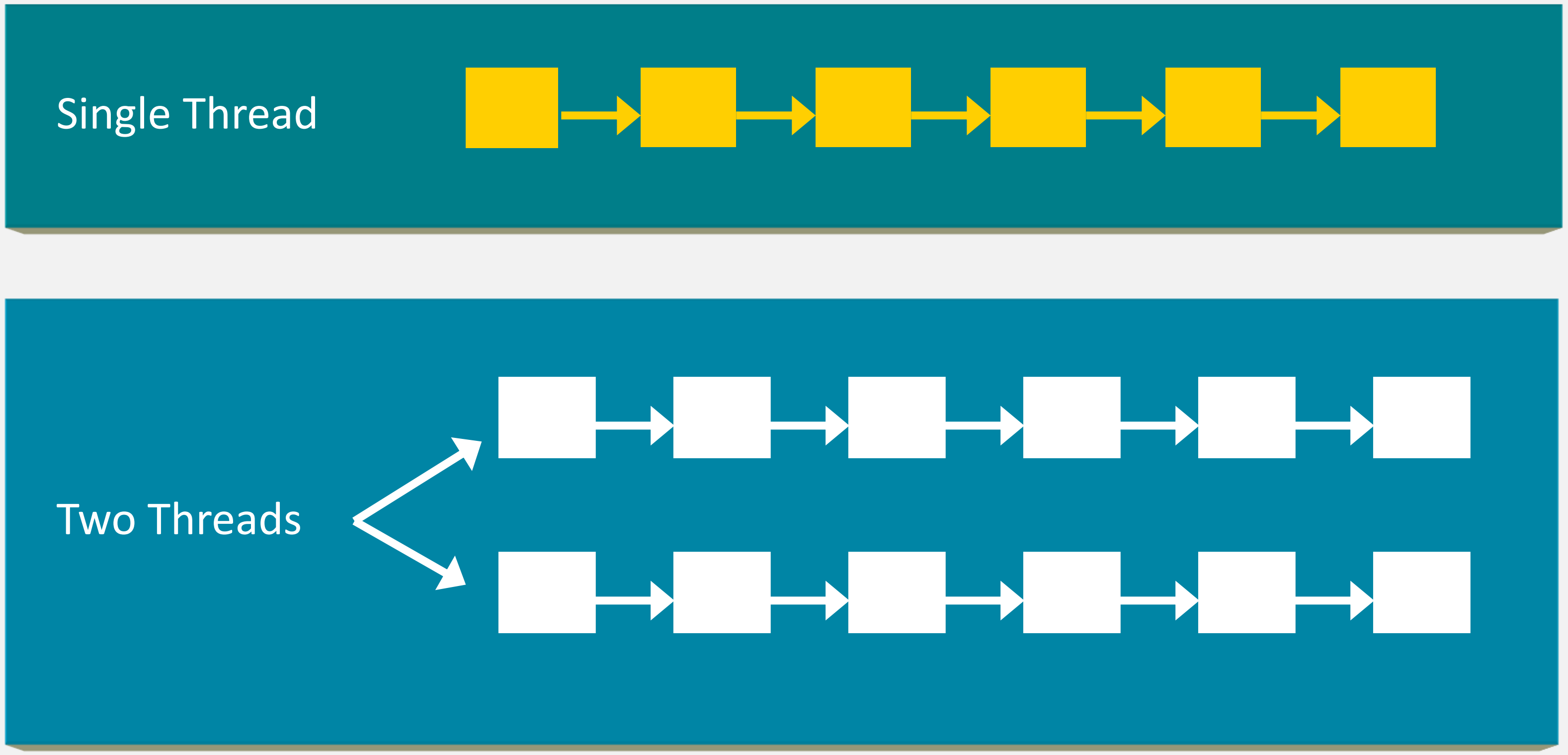
Process	Thread
1. Executable program loaded in memory	1. Sequentially executed stream of instructions

Address Space

Process	Thread
2. Has own address space–variables and data structures (in memory)	2. Shares address space with other Threads
3. Communicate through operating system, files, and network	3. Has own execution context
4. May contain multiple Threads	4. Multiple Thread in process execute same program

Types of Thread

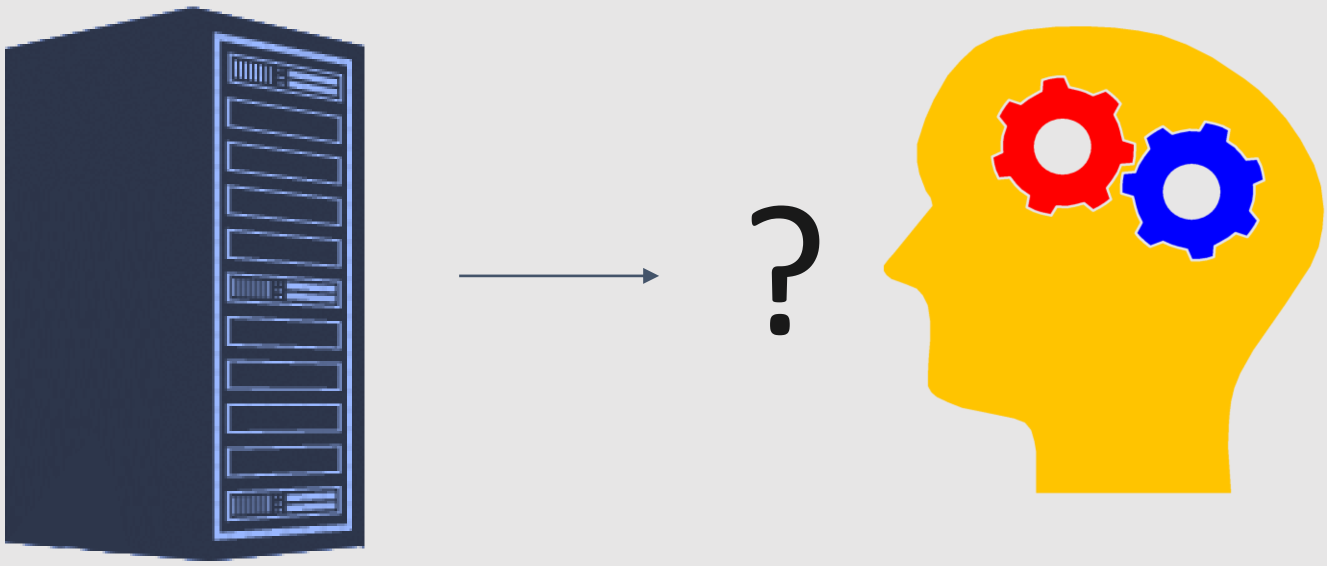
- Single Threaded process
- Multi Threaded process



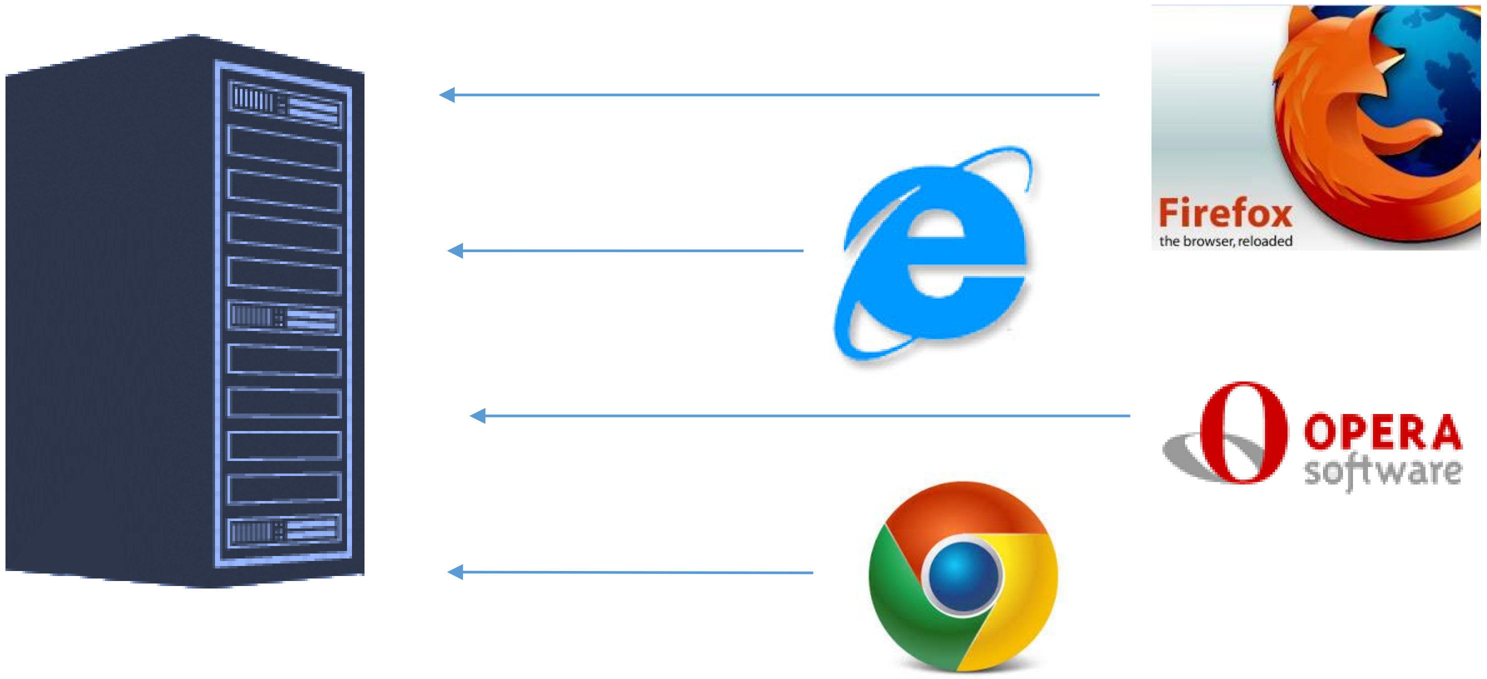
A c t i v i t y

Write / create simple steps to show the Threading performance of a CPU that performs various tasks simultaneously.

Hint: All these processes are handled by separate Threads.



Example: Thread



Web Server uses Threads to handle

Multiple simultaneous web browser requests

Basic concepts of **multitasking**

Multitasking

Multitasking is the ability to execute more than one task at the same time on the same processor.

Multitasking can be divided into two categories:

- Process-based multitasking
- Thread-based multitasking

Process-based multitasking (Multiprocessing):

Refers to working with two programs concurrently.

Thread-based multitasking (Multithreading):

Refers to working with parts of one program.

Multiprocessing vs Multithreading

While multiprocessing and multitasking are both ways to achieve multitasking, each has it’s own characteristics.

Multiprocessing	Multithreading
Each process have its own address in memory i.e. each process allocates separate memory area.	Threads share the same address space..
Cost of communication between the process is high. Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.	Thread is lightweight. Cost of communication between the thread is low.

We use multithreading rather than multiprocessing because threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Disadvantage of multitasking

These are the disadvantages of multitasking:

1 Race condition

2 Deadlock condition

3 Lock starvation

1 Two or more Threads access to same object

2 Two or more Threads are blocked forever, waiting for each other

3 A Thread is unable to gain regular access to shared resources and is unable to make progress

Spot quiz

Brighter Blue

01

What are the types of multitasking?

A	Process based
B	Thread based
C	Single Thread based
D	Multi Thread based

Using Threads in Java

Brighter Blue

Thread Class declaration

```
public class Thread extends Object
        implements Runnable {
    public Thread();
    public Thread(String name); // Thread name
    public Thread(Runnable r); // Thread ⇒ r.run()
    public Thread(Runnable r, String name);
    public Thread(ThreadGroup g,Runnable r);
    public Thread(ThreadGroup g,Runnable r, String name);
    public Thread(ThreadGroup g, String name);

    ...
}
```

Thread class:

- Available in java.lang package.
- Uses to implement multiThreading in java

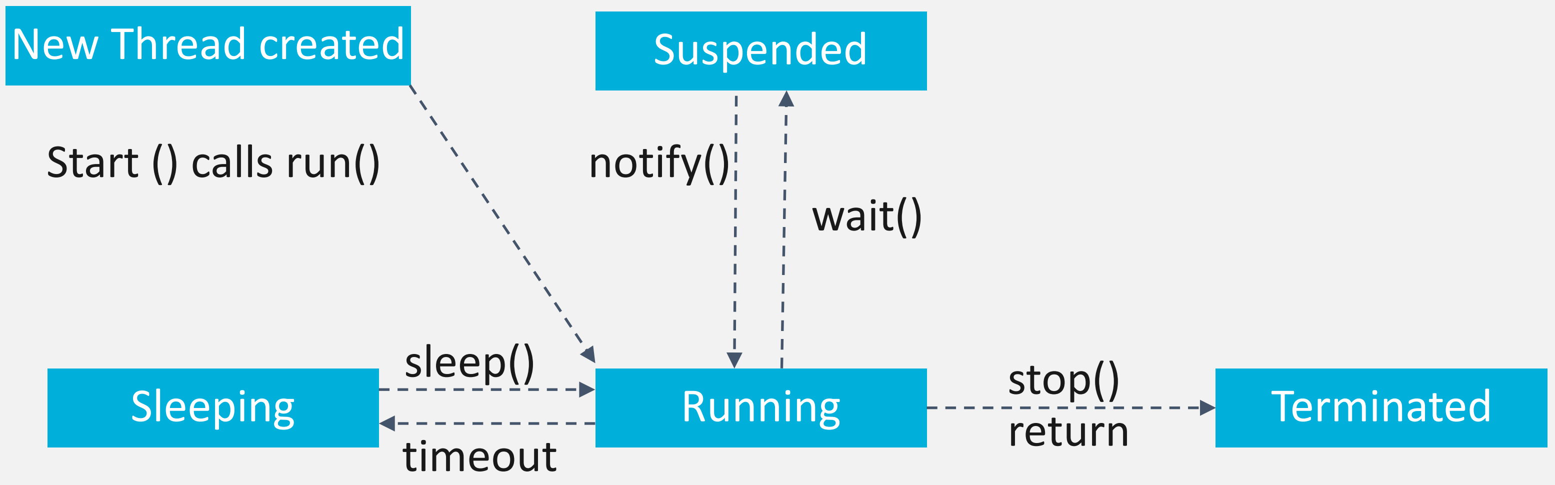
Methods of Thread class

Thread Class Methods

public void start() 1	public final boolean isAlive() 5
public final String getName() 2	public static void sleep(long millis) 6
public static Thread currentThread() 3	public final void join() 7
public void run() 4	public static void yield() 8

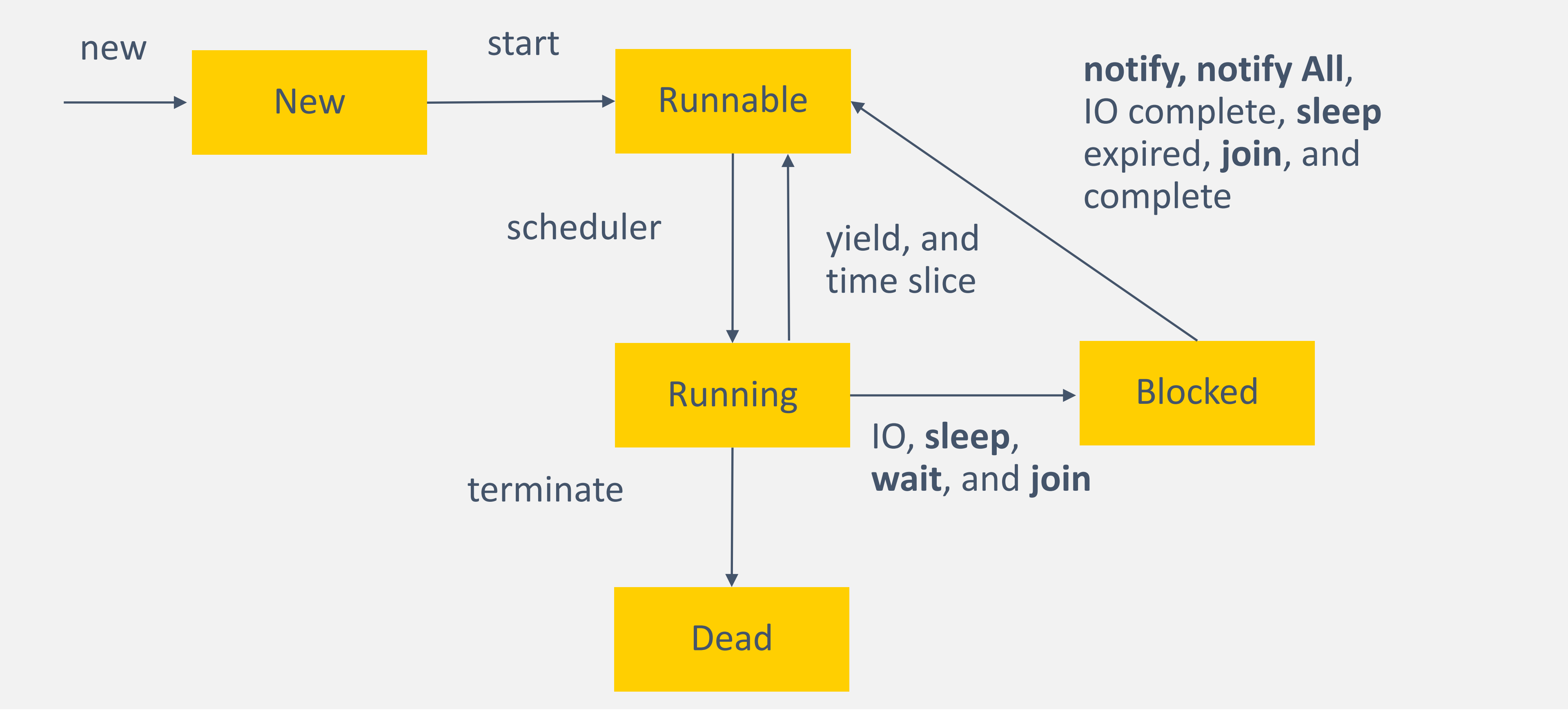
Lifecycle of Thread

Life cycle of Thread



Java Thread **state**

A Thread can be in only one of the states (shown below) at a given point in time.


new
New
start
Runnable
notify, notify All, IO complete, sleep expired, join, and complete
scheduler
yield, and time slice
Running
Blocked
IO, sleep, wait, and join
terminate
Dead

Spot quiz

Brighter Blue

02

In which states Threads can be explained

A	New, Start, Running, Dead, and Terminated
B	Start, Run, Wait, and Sleep
C	New, Runnable, Running, Dead, and Blocked
D	None of these

Creating Threads (1 of 9)

How to create Threads

Two ways of creating Thread in Java:

- Extending from java.lang.Thread class
- Implementing the Runnable Interface

Creating Threads (2 of 9)

Extending from java.lang.Thread class

The class extending the Thread class calls start() method to begin executing child Thread.

Sample code

```
public class MyT extends Thread{
   public void run(){
      …
   }
}
MyT T = new MyT();//create thread
T.start();                        // begin running thread
…
```

Creating Threads (3 of 9)

Extending from java.lang.Thread class

```
class MyThread extends Thread{
  private String name, msg;
    public MyThread(String name, String msg) {
        this.name = name;
        this.msg = msg;
    }
  public void run() {
   System.out.println(name+"starts its execution");
   for (int i = 0; i < =5; i++) {
    System.out.println(name+"says:"+msg+”for“+i+”time/s”);
    try{
      Thread.sleep(2000);
      }
   catch (InterruptedException ie){}

    }// End of For Loop
   System.out.println(name+"finished execution");
    }
}
```

Extends Thread

Overridden Method

It will throw an InterruptedException

Caught by

Creating Threads (4 of 9)

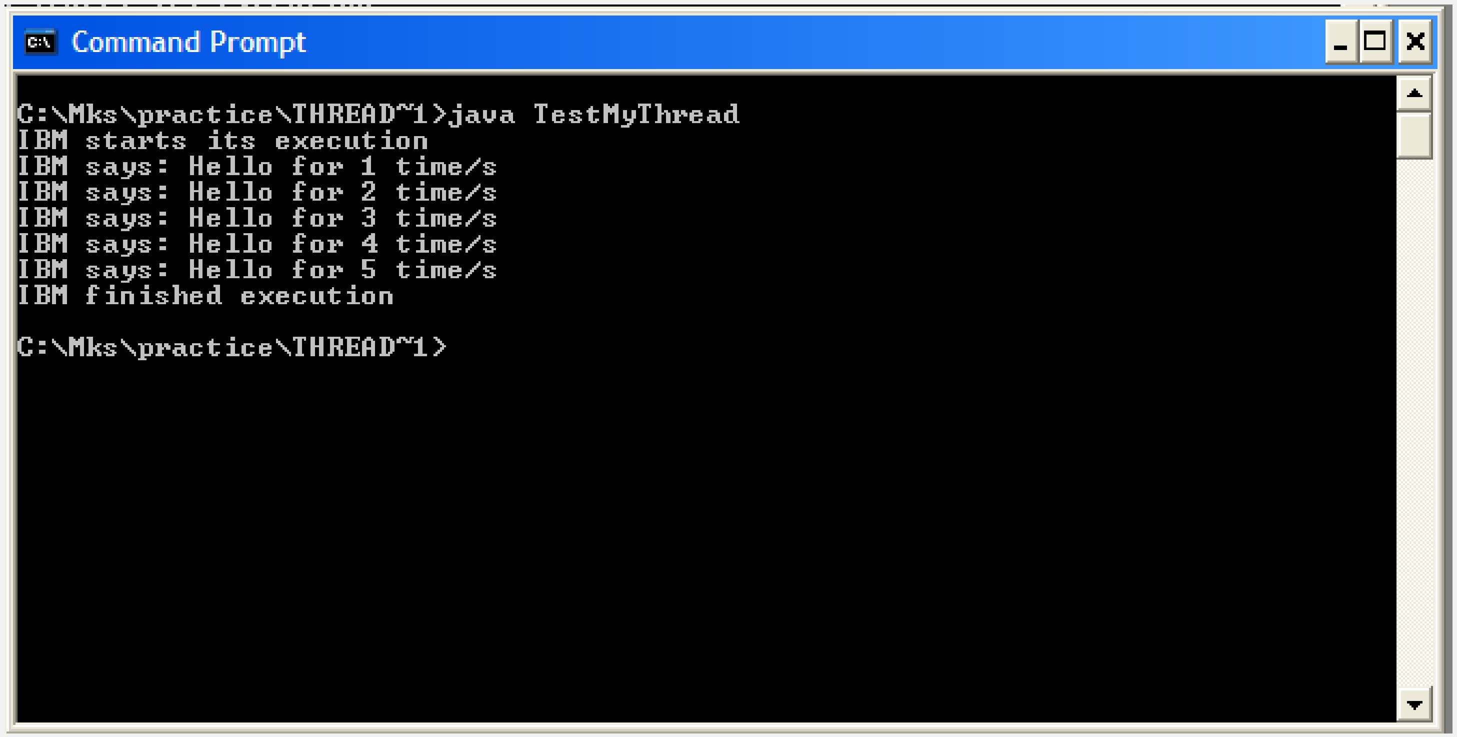
Extending from java.lang.Thread class

This class uses the Thread

```
public class TestMyThread
{
   public static void main(String[] args)
     {
      MyThread mth1 = new MyThread(“IBM", “Hello"); 1
      mth1.start(); 2
     }
}
```

Creating Threads (5 of 9)

Extending from java.lang.Thread class



```
C:\Mks\practice\THREAD~1>java TestMyThread
IBM starts its execution
IBM says: Hello for 1 time/s
IBM says: Hello for 2 time/s
IBM says: Hello for 3 time/s
IBM says: Hello for 4 time/s
IBM says: Hello for 5 time/s
IBM finished execution

C:\Mks\practice\THREAD~1>
```

Spot quiz

Brighter Blue

03

Which of the two methods are used class Thread?

A	wait()
B	start()
C	run()
D	terminate()

Creating Threads (6 of 9)

Implementing the Runnable Interface

- The Runnable Interface only consists of run() method.
- When a program needs to be inherited from a class other than Thread class; you need to implement the Runnable Interface.

```
public class MyT implements Runnable{
    public void run(){
       … //work for thread
    }
}
Thread T = new Thread(new MyT());//create thread
T.start();                       // begin running thread
…
```

Creating Threads (7 of 9)

Brighter Blue

Implementing the Runnable Interface

Code to create Thread using Runnable Interface

```
class MyRunnable implements Runnable{ 1
  private String name, msg;
  public MyRunnable(String name, String msg){
        this.name = name;
        this.msg = msg;
   }
  public void run() { 2
  System.out.println(name+"starts its execution");
for (int i = 1; i < =5; i++) {
  System.out.println(name+"says:"+msg+”for“+i+”time/s”);
  try {
       Thread.sleep(2000);
       }
catch (InterruptedException ie){}
   }// End of For Loop
 System.out.println(name+"finished execution");
```

Creating Threads (8 of 9)

Implementing the Runnable Interface

This class uses the Thread

```
public class TestMyRunnable
 {
  public static void main(String[] args)
         {
        1 MyRunnable myrun = new MyRunnable(“IBM", “Hello");
            Thread mythread = new Thread(myrun); 2
          mythread.start(); 3
         }
}
```

Creating Threads (9 of 9)

Implementing the Runnable Interface

What is the Predicted Output?

Same output as in the case of TestMyThread class

Spot quiz

Brighter Blue

04

Which method is used to start a Thread execution?

A	init()
B	run()
C	start()
D	resume()

Java multi Threading **example** (1 of 4)

Consider a class **DisplayMessage** which implements **Runnable**:

```
public class DisplayMessage implements Runnable
{
    private String message;
    public DisplayMessage(String message)
    {
        this.message = message;
    }
    public void run()
    {
        while(true)
        {
            System.out.println(message);
        }
    }
}
```

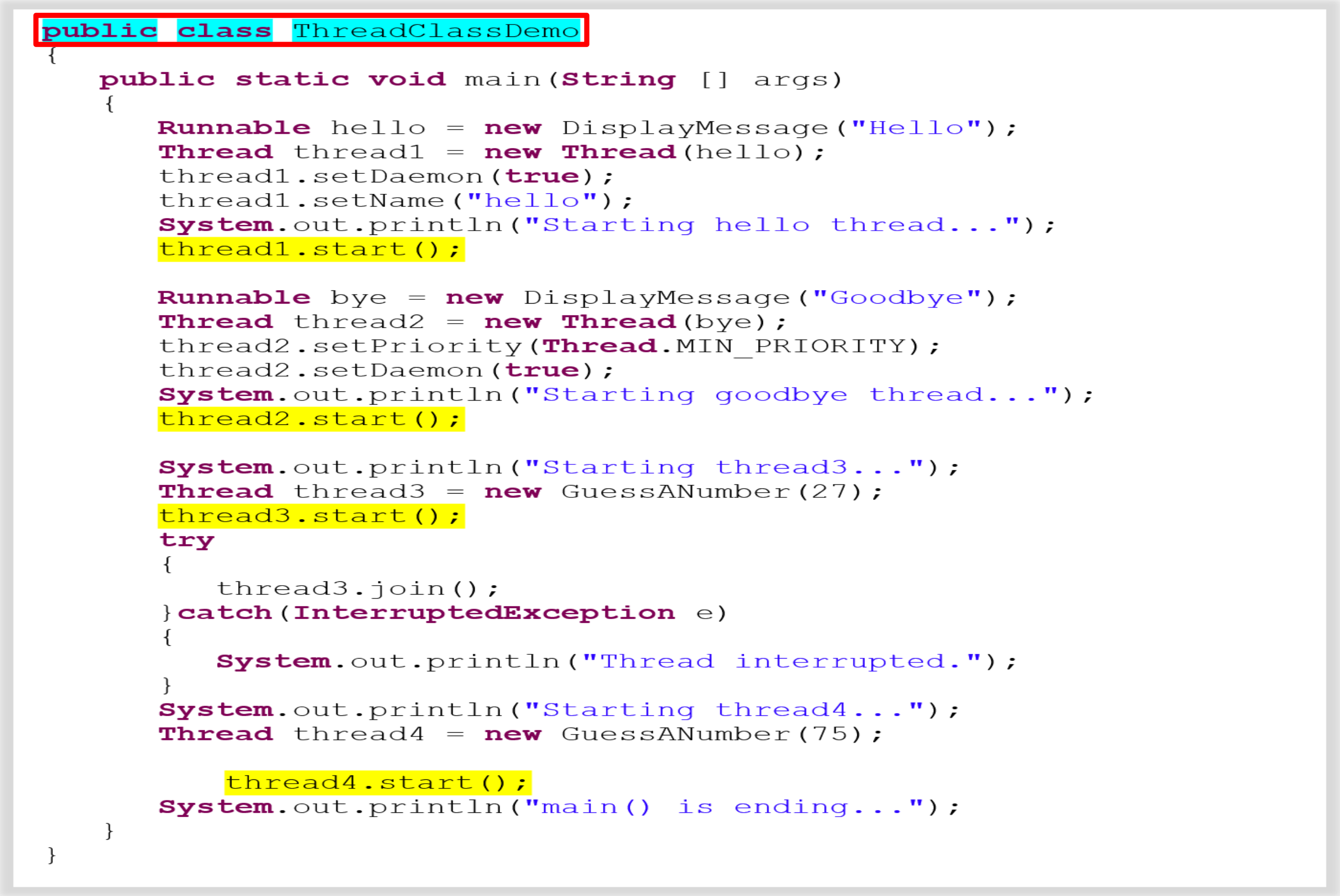
Java multi Threading **example** (2 of 4)

Following is another class which extends Thread class:

```
public class GuessANumber extends Thread
{
    private int number;
    public GuessANumber(int number)
    {
        this.number = number;
    }
    public void run()
    {
        int counter = 0;
        int guess = 0;
        do
        {
            guess = (int) (Math.random() * 100 + 1);
            System.out.println(this.getName()
                             + " guesses " + guess);
            counter++;
        }while(guess != number);
        System.out.println("** Correct! " + this.getName()
                             + " in " + counter + " guesses.**");
    }
}
```

Java multi Threading **example** (3 of 4)

Following is the main program which makes use of above defined classes:



```
public class ThreadClassDemo
{
   public static void main(String [] args)
   {
      Runnable hello = new DisplayMessage("Hello");
      Thread thread1 = new Thread(hello);
      thread1.setDaemon(true);
      thread1.setName("hello");
      System.out.println("Starting hello thread...");
      thread1.start();

      Runnable bye = new DisplayMessage("Goodbye");
      Thread thread2 = new Thread(bye);
      thread2.setPriority(Thread.MIN_PRIORITY);
      thread2.setDaemon(true);
      System.out.println("Starting goodbye thread...");
      thread2.start();

      System.out.println("Starting thread3...");
      Thread thread3 = new GuessANumber(27);
      thread3.start();
      try
      {
         thread3.join();
      }catch(InterruptedException e)
      {
         System.out.println("Thread interrupted.");
      }
      System.out.println("Starting thread4...");
      Thread thread4 = new GuessANumber(75);

         thread4.start();
      System.out.println("main() is ending...");
   }
}
```

Java multi Threading **example** (4 of 4)

What is the possible outputs?

It would produce the following result. You can try this example again and again and you would get different result every time.

```
Starting hello thread...
Starting goodbye thread...
Hello
Hello
Hello
Hello
Hello
Hello
Goodbye
Goodbye
Goodbye
Goodbye
Goodbye
```

Spot quiz

05

Extending Thread class and implementing Runnable interface are the two ways to:

A	Name a Thread
B	Join a Thread
C	Create a Thread
D	Terminate a Thread

Using Threads in Java

Points to be noted

The first Thread to be executed in a multithreaded process is called the main Thread

The main Thread is created automatically on the startup of java program execution

What is Threads **scheduling**?

Thread scheduling

The execution of multiple Threads on a single CPU is called scheduling.

Thread scheduling:

- Determines which runnable Threads to run
- Can be based on Thread priority
- Is a part of OS or Java Virtual Machine (JVM)

Types of scheduling:

- Preemptive
- Non-Preemptive

The Java runtime supports a very simple, deterministic scheduling algorithm known as fixed priority scheduling.

Thread policy

What is Thread policy?

- Thread priorities are the integers in the range of 1 to 10 that specify the priority of one Thread with respect to the priority of another Thread.
- Each Java Thread is given a numeric priority between MIN_PRIORITY and MAX_PRIORITY.
- The Java Run-time system selects the runnable Thread with the highest priority of execution when a number of Threads get ready to execute.
- The Java Run-time Environment executes Threads based on their priority.
- A Thread with higher priority runs before Threads with low priority.
- A given Thread may, at any time, give up its right to execute by calling the yield method. Threads can only yield the CPU to other Threads of the same priority– attempts to yield to a lower priority Thread are ignored.

What is Thread **priority**?

Thread priority

Thread priorities are the integers in the range of 1 to 10 that specify the priority of one Thread with respect to the priority of another Thread.

You can set the Thread priority after it is created using the setPriority() method declared in the Thread class.

What is **Thread synchronization**?

Thread Synchronization

Synchronization of Threads ensures that if two or more Threads need to access a shared resource then that resource is used by only one Thread at a time.

Synchronization is based on the concept of **monitor**.

What is a monitor?

- A monitor is an object that can **block Threads** and **notify** them **when** it is **available**.
- The monitor **controls** the way in which synchronized methods access an object or class.
- To enter an object's monitor, you need to call a synchronized method.

Why use Thread synchronization?

Why use synchronization?

PROBLEM

RACE CONDITION!!

Two or more Threads access to the same object and each call a method that modifies the state of the object.

RESULT

❌ CORRUPTED OBJECT!!!

How to **achieve** synchronization?

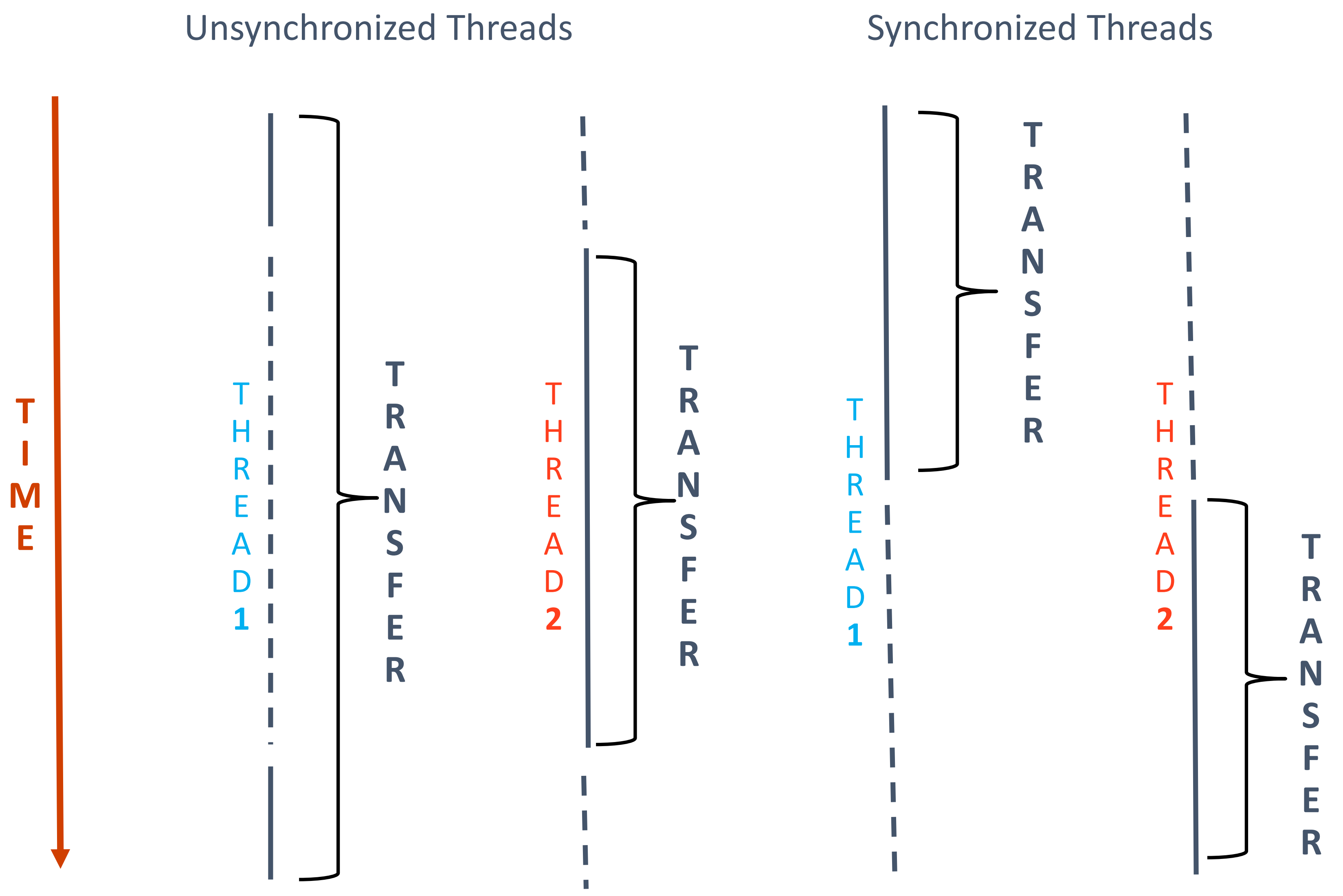
Brighter Blue

To achieve synchronization → Use → **`Synchronized` keyword**

To lock the object: Use the **`synchronized`** keyword.

To unlock the object: Use the **`wait()`** method. This is called inside the **`synchronized()`** method and the current Thread gets blocked and is put into the **wait list**.

Unsynchronized and synchronized Threads



Levels of synchronization

Synchronization can be achieved at the **method** level and the **object** level.

Synchronizing a method:

```
public synchronized void updateRecord()
{
        // critical code goes here …
}
```

Only one Thread will be inside the body of the updateRecord() method. The second call will be blocked until the first call returns or wait() is called inside the synchronized method.

Synchronizing an object:

```
public void updateRecord()
{
    synchronized (this)
     {
        // critical code goes here …
    }
}
```

If you do not need to protect an entire method, you can synchronize on an object. Declaring a method as synchronized is equivalent to synchronizing on *this* for all the method block.

Understanding the problem without synchronization

```
Class Table{
void printTable(int n)  {  //method not
synchronized
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception
e){System.out.println(e);}
   }
 }
}
class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
    this.t=t;
}
public void run(){
     t.printTable(5);
}
}
```

```
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}
class TestSynchronization1{
  public static void main(String
args[]){
  Table obj = new Table();//only one
object
   MyThread1 t1=new MyThread1(obj);
   MyThread2 t2=new MyThread2(obj);
    t1.start();
    t2.start();
}
}
```

Using the Java synchronized method

```
//example of java synchronized method
    class Table{
     synchronized void printTable(int
n){//synchronized method
       for(int i=1;i<=5;i++){
         System.out.println(n*i);
         try{
          Thread.sleep(400);
         }catch(Exception
e){System.out.println(e);}
       }   }
    }
    class MyThread1 extends Thread{
    Table t;
    MyThread1(Table t){
         this.t=t;
    }    public void run(){
         t.printTable(5);
    }
    }
```

```
class MyThread2 extends Thread{
    Table t;
    MyThread2(Table t){
         this.t=t;
    }
    public void run(){
         t.printTable(100);
    }      }
    public class TestSynchronization2{
         public static void main(String
args[]){
                 Table obj = new
Table();//only one object
                 MyThread1 t1=new
MyThread1(obj);
                 MyThread2 t2=new
MyThread2(obj);
                 t1.start();
                 t2.start();
          }
    }
```

Activity

Write the code for the following output using 2 Threads and Synchronized method

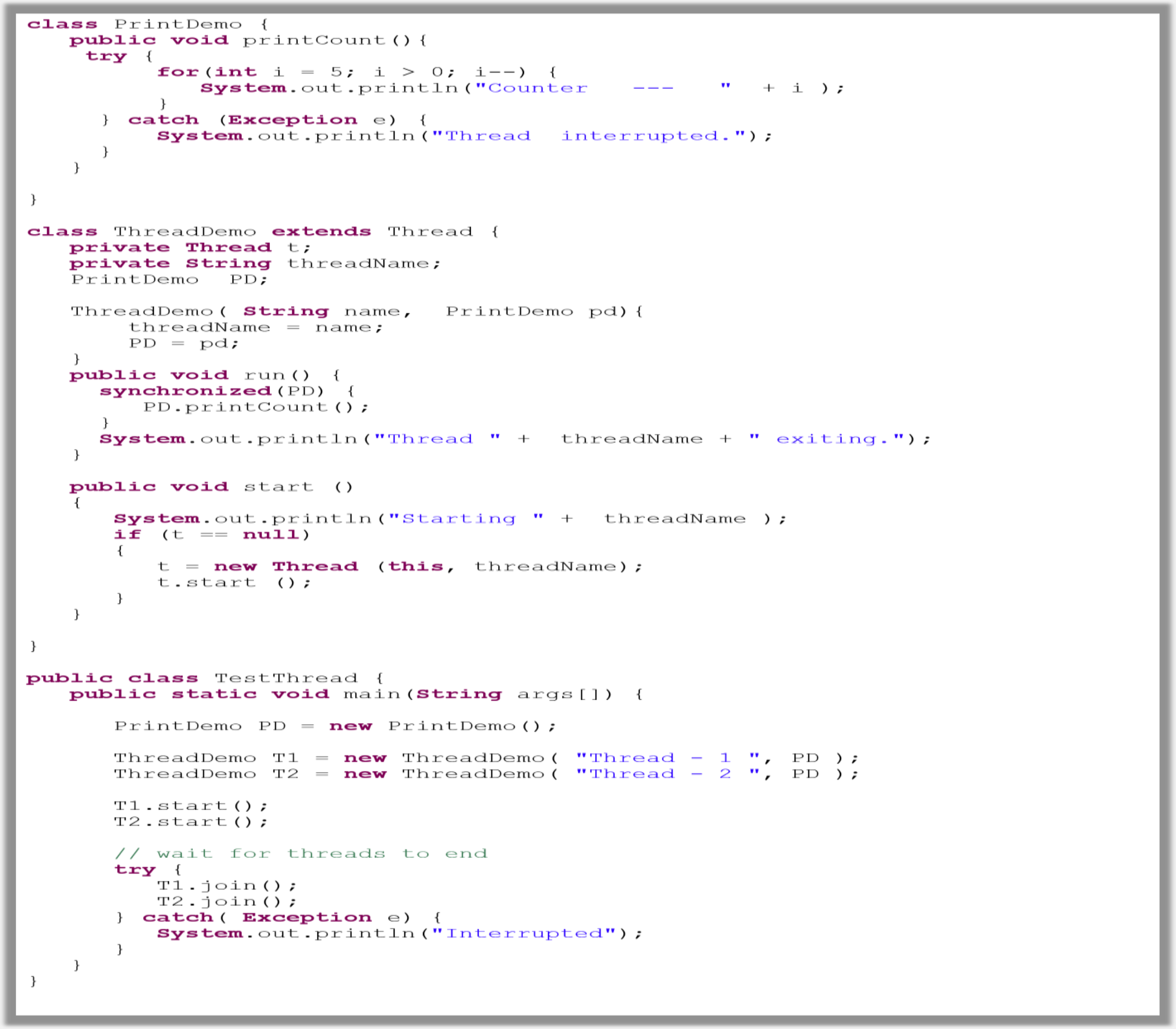
Expected output:

```
5
4
3
2
1
Thread exiting.
```

```
5
4
3
2
1
Thread exiting.
```

A c t i v i t y

What are the output for the given code example using two Threads and Synchronized method



```
class PrintDemo {
   public void printCount(){
    try {
         for(int i = 5; i > 0; i--) {
            System.out.println("Counter   ---   "  + i );
         }
     } catch (Exception e) {
         System.out.println("Thread  interrupted.");
     }
   }

}

class ThreadDemo extends Thread {
   private Thread t;
   private String threadName;
   PrintDemo  PD;

   ThreadDemo( String name,  PrintDemo pd){
       threadName = name;
       PD = pd;
   }
   public void run() {
     synchronized(PD) {
        PD.printCount();
     }
     System.out.println("Thread " +  threadName + " exiting.");
   }

   public void start ()
   {
      System.out.println("Starting " +  threadName );
      if (t == null)
      {
         t = new Thread (this, threadName);
         t.start ();
      }
   }

}

public class TestThread {
   public static void main(String args[]) {

      PrintDemo PD = new PrintDemo();

      ThreadDemo T1 = new ThreadDemo( "Thread - 1 ", PD );
      ThreadDemo T2 = new ThreadDemo( "Thread - 2 ", PD );

      T1.start();
      T2.start();

      // wait for threads to end
      try {
         T1.join();
         T2.join();
      } catch( Exception e) {
         System.out.println("Interrupted");
      }
   }
}
```

What is inter-Thread communication?

Inter-Thread Communication

A Thread may notify another Thread that the task has been completed. This communication between Threads is known as inter-Thread communication.

The various methods() used in inter-Thread communication are:

- **`wait()`:** It is a part of java.lang.Object class that tells a Thread to relinquish a monitor and go into suspension.
- **`notify()`:** It is a part of the java.lang.Object class that tells a Thread that is suspended by wait() to wake up again and regain control of the monitor.
- **`notifyAll()`:** It belongs to java.lang.Object class that wakes up all the Threads that are waiting for control of the monitor.

Spot quiz

Brighter Blue

06

What is synchronization in reference to a Thread?

A It is a process of handling situations when two or more Threads need access to a shared resource.

B It is a process by which many Threads are able to access same shared resource simultaneously.

C It is a process by which a method is able to access many different Threads simultaneously.

D It is a method that allows to many Threads to access any information that they require.

Questions?

02 Garbage Collection

What is **garbage collection**?

It is the feature of Java that helps automatically **to destroy** the **objects created** and **release** their **memory** for **future reallocation**.

Java uses a Thread to collect garbage in the background.

The various activities involved in garbage collection are:

- **Monitoring** the objects used by a program and **determining** when they are not in use.
- **Destroying** objects that are no more in use and **reclaiming** their resources, such as memory space.

Note:
The Java Virtual machine (JVM) acts as the garbage collector that keeps a track of the memory allocated to various objects and the objects being referenced.

Advantages of garbage collection

Garbage Collection

- It makes Java **memory efficient** because garbage collector **removes** the **unreferenced objects** from heap memory.
- It is **automatically done** by the garbage collector so we **do not need** to make **extra efforts**.

How can an object be unreferenced?

- By nulling the reference:
```
Employee e=new Employee();
e=null;
```
- By assigning one reference to another:
```
Employee e1=new Employee();
Employee e2=new Employee();
e1=e2;//
```
- By anonymous object and so on:
```
newEmployee();
```

The **finalize** and **garbage collector** methods

`finalize()` method:
The finalize() method is invoked each time before the object is garbage collected.
This method can be used to perform cleanup processing. This method is defined in object class as:

```
protected void finalize(){}
```

`gc() method:`
The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in system and runtime classes.
Garbage collection is performed by a daemon Thread called garbage collector(gc). This Thread calls the finalize() method before object is garbage collected.

```
public static void gc(){}
```

Note: Even if you call gc () method, **there's no guarantee** that it will be called immediately. It is up to the compiler.

Garbage collection: example

```
public class TestGarbage1{

public void finalize(){System.out.println("object is garbage
collected");}

public static void main(String args[]){
       TestGarbage1 s1=new TestGarbage1();
       TestGarbage1 s2=new TestGarbage1();
       s1=null;
       s2=null;
       System.gc();
  }
}


Output: object is garbage collected
```

Spot quiz

07

Which function is used to perform some action when the object is to be destroyed?

A	`finalize()`	B	`gc()`
C	`delete()`	D	`main()`

Spot quiz

08

Which of the following statements are false?

A The gc() method is used to invoke the garbage collector to perform cleanup processing.

B Garbage collection is performed by a daemon Thread called garbage collector(gc).

C finalize() method is called when an object goes out of scope and is no longer needed.

Questions?

A c t i v i t y

Brighter Blue

Day 03 Practice Exercises:

1. Create and run 5 threads that randomizes a number from 1 to 10 and adds them.