Task01:

**Process:** Any operations being executed on applications can be named as process.

Ex.: Opening an application, closing an application and so on…

Task02:

**Thread:** A thread is the smallest unit with in a process. A process may be made up of multiple threads.

Each thread can be managed independently by the scheduler.

Task03:

class **RunnableDemo** implements **Runnable** {  
 private **Thread** t;  
 private **String** threadName;  
 RunnableDemo( **String** name){  
 threadName = name;  
 **System**.*out*.println("Creating " + threadName );  
 }  
 public void run() {  
 **System**.*out*.println("Running " + threadName );  
 try {  
 for(int i = 4; i > 0; i--) {  
 **System**.*out*.println("Thread: " + threadName + ", " + i);  
// Let the thread sleep for a while.  
 **Thread**.*sleep*(6000);  
 }  
 } catch (**InterruptedException** e) {  
 **System**.*out*.println("Thread " + threadName + " interrupted.");  
 }  
 **System**.*out*.println("Thread " + threadName + " exiting.");  
  
 }  
 public void ThStart()  
 {  
 **System**.*out*.println("Starting " + threadName );  
 if (t == null)  
 {  
 t = new Thread (this, threadName);  
 t.start ();  
 }  
 }  
}  
public class **Task03** {  
 public static void main(**String** args[]) {  
 **RunnableDemo** R1 = new RunnableDemo( "Thread-1");  
 R1.ThStart();  
 **RunnableDemo** R2 = new RunnableDemo( "Thread-2");  
 R2.ThStart();  
 }  
}



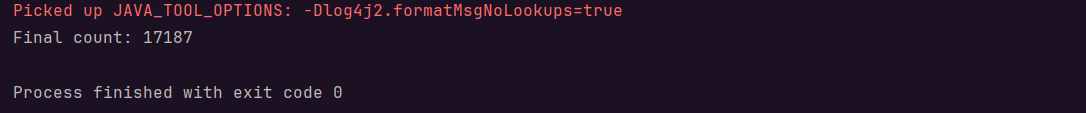
Task04:

class **ThreadDemo** extends **Thread** {  
 private **Thread** t;  
 private **String** threadName;  
 ThreadDemo( **String** name){  
 threadName = name;  
 **System**.*out*.println("Creating " + threadName );  
 }  
 public void run() {  
 **System**.*out*.println("Running " + threadName );  
 try {  
 for(int i = 4; i > 0; i--) {  
 **System**.*out*.println("Thread: " + threadName + ", " + i);  
 // Let the thread sleep for a while.  
 **Thread**.*sleep*(50);  
 }  
 } catch (**InterruptedException** e) {  
 **System**.*out*.println("Thread " + threadName + " interrupted.");  
 }  
 **System**.*out*.println("Thread " + threadName + " exiting.");  
 }  
 public void Thstart () {  
 **System**.*out*.println("Starting " + threadName );  
 if (t == null) {  
 t = new Thread (this, threadName);  
 t.start ();  
 }  
 }  
}  
public class **Task04** {  
 public static void main(**String** args[]) {  
 **ThreadDemo** T1 = new ThreadDemo( "Thread-1");  
 T1.Thstart();  
 **ThreadDemo** T2 = new ThreadDemo( "Thread-2");  
 T2.Thstart();  
 }  
}



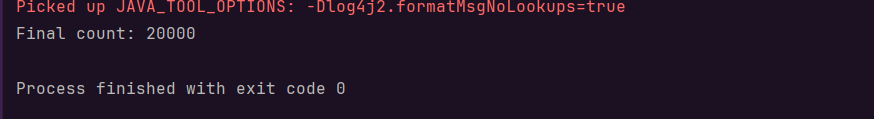
Task05:

class **Counter** {  
 private int count = 0;  
  
 public void increment() {  
 count++;  
 }  
  
 public int getCount() {  
 return count;  
 }  
}  
  
class **ThreadDemo** extends **Thread** {  
 **Counter** counter;  
  
 ThreadDemo(**Counter** counter) {  
 this.counter = counter;  
 }  
  
 public void run() {  
 for (int i = 0; i < 10000; i++) {  
 counter.increment();  
  
 }  
 }  
}  
  
public class **Task05** {  
 public static void main(**String**[] args) {  
 **Counter** counter = new Counter();  
 **ThreadDemo** t1 = new ThreadDemo(counter);  
 **ThreadDemo** t2 = new ThreadDemo(counter);  
  
 t1.start();  
 t2.start();  
  
 try {  
 t1.join();  
 t2.join();  
 } catch (**InterruptedException** e) {  
 e.printStackTrace();  
 }  
  
 **System**.*out*.println("Final count: " + counter.getCount());  
 }  
}



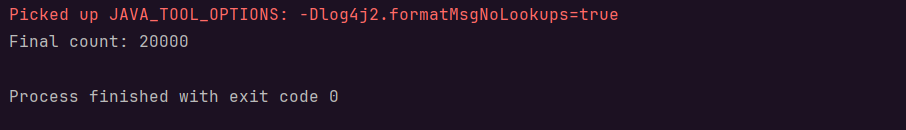
Task06:

class **Counter** {  
 private int count = 0;  
  
 public synchronized void increment() {  
 count++;  
 }  
  
 public int getCount() {  
 return count;  
 }  
}  
  
class **ThreadDemo** extends **Thread** {  
 **Counter** counter;  
  
 ThreadDemo(**Counter** counter) {  
 this.counter = counter;  
 }  
  
 public void run() {  
 for (int i = 0; i < 10000; i++) {  
 counter.increment();  
  
 }  
 }  
}  
  
public class **Task06** {  
 public static void main(**String**[] args) {  
 **Counter** counter = new Counter();  
 **ThreadDemo** t1 = new ThreadDemo(counter);  
 **ThreadDemo** t2 = new ThreadDemo(counter);  
  
 t1.start();  
 t2.start();  
  
 try {  
 t1.join();  
 t2.join();  
 } catch (**InterruptedException** e) {  
 e.printStackTrace();  
 }  
  
 **System**.*out*.println("Final count: " + counter.getCount());  
 }  
}



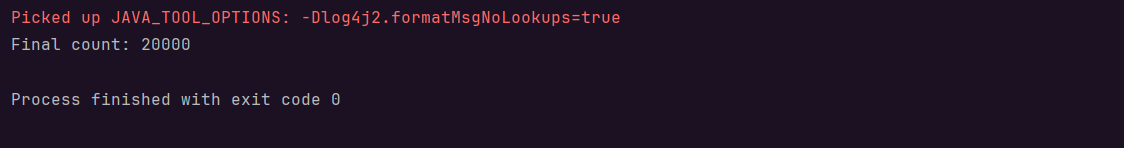
Task07:

class **Counter** {  
 private int count = 0;  
  
 public synchronized void increment() {  
 synchronized (this){  
 count++;  
 }}  
  
 public int getCount() {  
 return count;  
 }  
}  
  
class **ThreadDemo** extends **Thread** {  
 **Counter** counter;  
  
 ThreadDemo(**Counter** counter) {  
 this.counter = counter;  
 }  
  
 public void run() {  
 for (int i = 0; i < 10000; i++) {  
 counter.increment();  
  
 }  
 }  
}  
  
public class **Task06** {  
 public static void main(**String**[] args) {  
 **Counter** counter = new Counter();  
 **ThreadDemo** t1 = new ThreadDemo(counter);  
 **ThreadDemo** t2 = new ThreadDemo(counter);  
  
 t1.start();  
 t2.start();  
  
 try {  
 t1.join();  
 t2.join();  
 } catch (**InterruptedException** e) {  
 e.printStackTrace();  
 }  
  
 **System**.*out*.println("Final count: " + counter.getCount());  
 }  
}



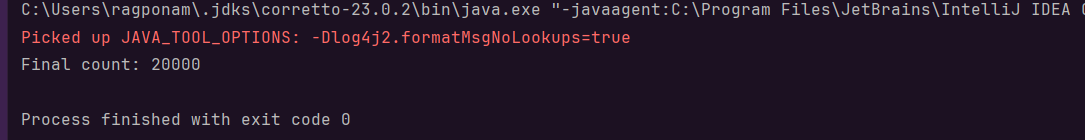
Task08:

class **Counter** {  
 private static int *count* = 0;  
  
 public static synchronized void increment() {  
  
 *count*++;  
 }  
  
 public int getCount() {  
 return *count*;  
 }  
}  
  
class **ThreadDemo** extends **Thread** {  
 **Counter** counter;  
  
 ThreadDemo(**Counter** counter) {  
 this.counter = counter;  
 }  
  
 public void run() {  
 for (int i = 0; i < 10000; i++) {  
 counter.*increment*();  
  
 }  
 }  
}  
  
public class **Task08** {  
 public static void main(**String**[] args) {  
 **Counter** counter = new Counter();  
 **ThreadDemo** t1 = new ThreadDemo(counter);  
 **ThreadDemo** t2 = new ThreadDemo(counter);  
  
 t1.start();  
 t2.start();  
  
 try {  
 t1.join();  
 t2.join();  
 } catch (**InterruptedException** e) {  
 e.printStackTrace();  
 }  
  
 **System**.*out*.println("Final count: " + counter.getCount());  
 }  
}



Task09:

import **java.util.concurrent.locks.Lock**;  
import **java.util.concurrent.locks.ReentrantLock**;  
  
class **Counter** {  
 private int count = 0;  
 private final **Lock** lock = new ReentrantLock();  
  
 public void increment() {  
 lock.lock();  
 try {  
 count++;  
 } finally {  
 lock.unlock();  
 }  
 }  
   
 public int getCount() {  
 return count;  
 }  
}  
  
class **ThreadDemo** extends **Thread** {  
 **Counter** counter;  
  
 ThreadDemo(**Counter** counter) {  
 this.counter = counter;  
 }  
  
 public void run() {  
 for (int i = 0; i < 10000; i++) {  
 counter.increment();  
  
 }  
 }  
}  
  
public class **Task09** {  
 public static void main(**String**[] args) {  
 **Counter** counter = new Counter();  
 **ThreadDemo** t1 = new ThreadDemo(counter);  
 **ThreadDemo** t2 = new ThreadDemo(counter);  
  
 t1.start();  
 t2.start();  
  
 try {  
 t1.join();  
 t2.join();  
 } catch (**InterruptedException** e) {  
 e.printStackTrace();  
 }  
  
 **System**.*out*.println("Final count: " + counter.getCount());  
 }  
}



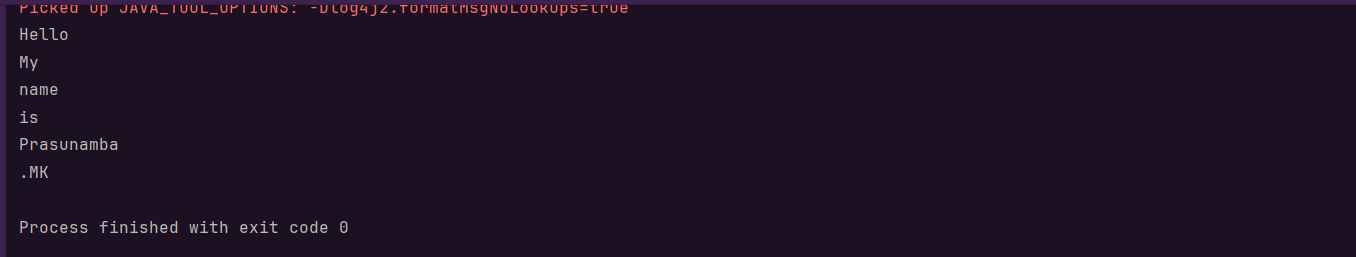
Task011:

class **SharedResource** {  
 private boolean ready = false;  
  
 synchronized void produce() {  
 try {  
 while (ready) {  
 wait();  
 }  
 **System**.*out*.println("Producing...");  
 ready = true;  
 notify();  
 } catch (**InterruptedException** e) {  
 e.printStackTrace();  
 }  
 }  
  
 synchronized void consume() {  
 try {  
 while (!ready) {  
 wait();  
 }  
 **System**.*out*.println("Consuming...");  
 ready = false;  
 notify();  
 } catch (**InterruptedException** e) {  
 e.printStackTrace();  
 }  
 }  
}  
  
public class **Task011** {  
 public static void main(**String**[] args) {  
 **SharedResource** resource = new SharedResource();  
  
 **Thread** producer = new Thread(resource::produce);  
 **Thread** consumer = new Thread(resource::consume);  
  
 producer.start();  
 consumer.start();  
 }  
}



Task012:

import **java.util.stream.**\*;  
class **DoubleColonOp** {  
 public static void main(**String**[] args) {  
 **Stream**<**String**> stream  
 = **Stream**.*of*("Hello", "My",  
 "name", "is",  
 "Prasunamba",  
 ".MK");  
  
 stream.forEach(**System**.*out*::println);  
 }  
}



Task013:

class **InterruptibleThread** extends **Thread** {  
 public void run() {  
 try {  
 while (!**Thread**.*currentThread*().isInterrupted()) {  
 **System**.*out*.println("Thread is running");  
 **Thread**.*sleep*(100);  
 }  
 } catch (**InterruptedException** e) {  
 **System**.*out*.println("Thread was interrupted");  
 }  
 }  
}  
  
public class **Task013** {  
 public static void main(**String**[] args) {  
 **InterruptibleThread** thread = new InterruptibleThread();  
 thread.start();  
  
 try {  
 **Thread**.*sleep*(500);  
 thread.interrupt();  
 } catch (**InterruptedException** e) {  
 e.printStackTrace();  
 }  
 }  
}



Task014:

Daemon thread is a thread that runs in the background assisting the main threads. Daemon thread does not run any functionalities, while the main thread does.

Ex: Garbage collector in JVM, Background monitoring applications, Background logging, etc.

Task015:

Debugging tools are essential software applications or features that help developers find, diagnose, and fix errors (bugs) in their code. They provide a systematic way to troubleshoot problems by allowing developers to observe and control the execution of their programs.

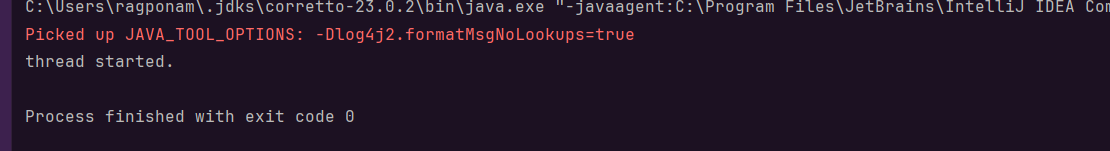
Ex: Debuggers in IDEs, GNU debugger, Wireshark, Raygun, Postman.

Task017:

A **stack trace** is a report of the active **stack frames** at a certain point in time during the execution of a computer program. It's like a snapshot of all the functions (or methods, in object-oriented programming) that were active at the moment an event occurred, typically an error or exception.

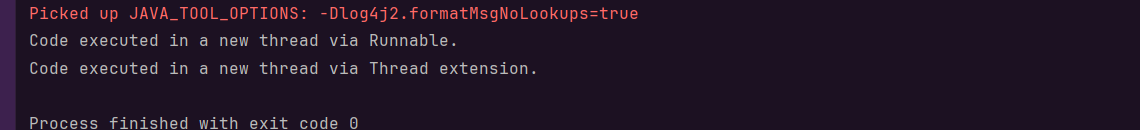
Task018:

class **Test** extends **Thread**{  
 public void run(){  
 **System**.*out*.println("thread started.");  
 }  
}  
  
public class **Task018** {  
 public static void main(**String** args[]){  
  
 **Test** t1 = new Test();  
 t1.run();  
 }  
}



Task019:

class **MyRunnable** implements **Runnable** {  
 @Override  
 public void run() {  
 **System**.*out*.println("Code executed in a new thread via Runnable.");  
 }  
}  
  
  
class **MyThread** extends **Thread** {  
 @Override  
 public void run() {  
 **System**.*out*.println("Code executed in a new thread via Thread extension.");  
 }  
}  
class **Task019** {  
 public static void main(**String**[] args) {  
  
 **MyRunnable** runnableInstance = new MyRunnable();  
 **MyThread** threadInstance = new MyThread();  
  
 **Thread** t1 = new Thread(runnableInstance);  
  
 t1.start(); // For MyRunnable  
 threadInstance.start(); // For MyThread  
  
 }  
}



Task020:

public class **Task020** {  
  
 public static void main(**String**[] args) {  
 *method1*();  
 }  
  
 public static void method1() {  
 *method2*();  
 }  
  
 public static void method2() {  
 *method3*();  
 }  
  
 public static void method3() {  
 **StackTraceElement**[] stackTrace = **Thread**.*currentThread*().getStackTrace();  
  
 **System**.*out*.println("Thread Stack Trace:");  
 // Iterate through the StackTraceElement array and print details  
 for (**StackTraceElement** element : stackTrace) {  
 **System**.*out*.println(" Class: " + element.getClassName() +  
 ", Method: " + element.getMethodName() +  
 ", Line: " + element.getLineNumber());  
 }  
 }  
}

