**ASSIGNMENT – 9**

Dt. 03/11/2022

1. Inherit a class from Thread and override the run( ) method. Inside run( ), print name of thread , and then call sleep( ). Repeat this three times, then return from run( ). Put a start-up message in the constructor. Make your thread object and main thread run to see what happens.

Code:

public class Question1 extends Thread { Question1(String name) {

super(name); System.out.println("Started"); }

public void run() {

for (int i = 0; i < 3; i++) { System.out.println(this.getName()); try {

this.sleep(1000);

} catch (Exception e) { System.out.println(e);

} } }

public static void main(String[] args) { Question1 q1 = new Question1("Mitra");

System.out.println(Thread.currentThread().getName()); System.out.println("Debayudh : " + q1.getName()); q1.start();

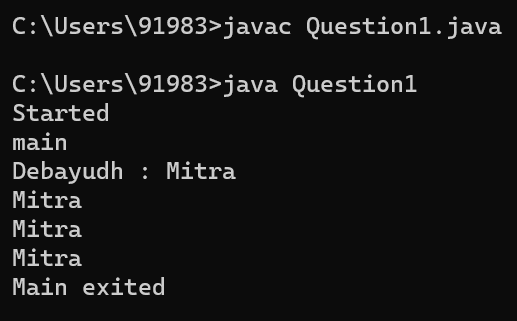
try {

q1.join();

} catch (Exception e) { System.out.println(e); }

System.out.println("Main exited");

} }

Output:

1. Implement a class from Runnable and override the run( ) method. Inside run( ), print full qualified name of thread, and then call sleep( ). Repeat this three times, then return from run( ). Put a start- up message in the constructor. Make your thread object and main thread run to see what happens.

Code:

public class Question2 implements Runnable { Question2(String name) {

System.out.println("Started"); } public void run() {

for (int i = 0; i < 3; i++) { System.out.println(Thread.currentThread().getName()); try {

Thread.sleep(1000);

} catch (Exception e) { System.out.println(e);

}}}

public static void main(String[] args) {

Question2 q2 = new Question2("New Thread"); Thread t = new Thread(q2, "Debayudh");

System.out.println(Thread.currentThread().getName()); System.out.println("Original name: " + t.getName()); t.start();

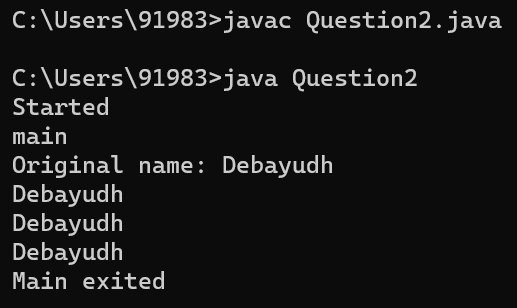
try {

t.join();

} catch (Exception e) { System.out.println(e); }

System.out.println("Main exited");

}}

Output:

1. Make several threads (say 5) with names (by extending thread), set their priority and run them to see what happens.

Code:

public class Question3 extends Thread { Question3(String naam) {

super(naam); } public void run() {

for (int i = 0; i < 3; i++) {

System.out.println(this + " [Timed: " + (i + 1) + "]");

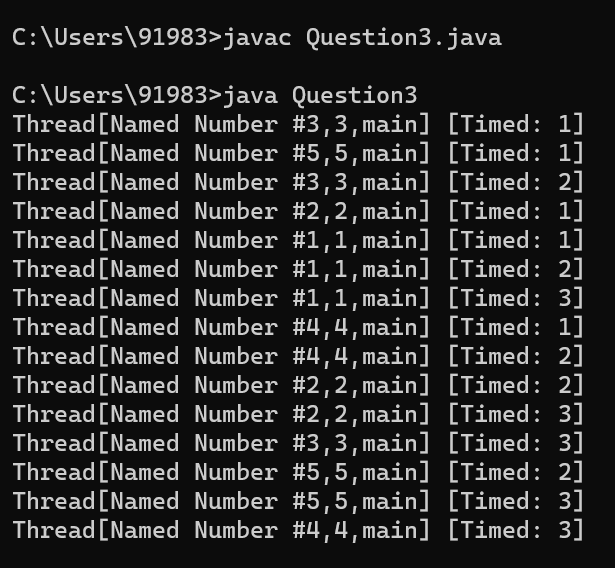
}}

public static void main(String[] args) { Question3 qs[] = new Question3[5]; for (int i = 1; i < 6; i++) {

qs[i - 1] = new Question3("Named Number #" + i); qs[i - 1].setPriority(i);

qs[i - 1].start();

}}}

Output:

1. Make several threads (say 5) with their names (implementing Runnable) set their priority and run them to see what happens.

Code:

public class Question4 implements Runnable { public void run() {

for (int i = 0; i < 3; i++) {

System.out.println(Thread.currentThread().getName() + " [Timed: " + (i + 1) + "]");

}}

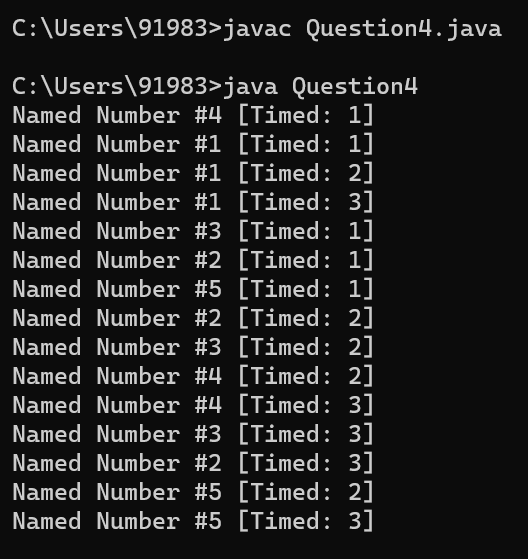
public static void main(String[] args) { Thread qs[] = new Thread[5];

for (int i = 1; i < 6; i++) {

qs[i - 1] = new Thread(new Question4(), "Named Number #" + i); qs[i - 1].setPriority(i);

qs[i - 1].start();

}}}

Output:

1. Write a program to use join() and isAlive() in Multi-Threading. Code:

class MyRunnableClass implements Runnable { public void run() {

for (int i = 0; i < 5; i++) { System.out.println(Thread.currentThread().getName() + " i - " + i); try {

Thread.sleep(100);

} catch (InterruptedException e) { e.printStackTrace();

}}}}

public class Question5 {

public static void main(String[] args) {

Thread t1 = new Thread(new MyRunnableClass(), "t1"); Thread t2 = new Thread(new MyRunnableClass(), "t2"); Thread t3 = new Thread(new MyRunnableClass(), "t3"); t1.start();

t2.start();

t3.start();

System.out.println("t1 Alive - " + t1.isAlive()); System.out.println("t2 Alive - " + t2.isAlive()); System.out.println("t3 Alive - " + t3.isAlive()); try {

t1.join();

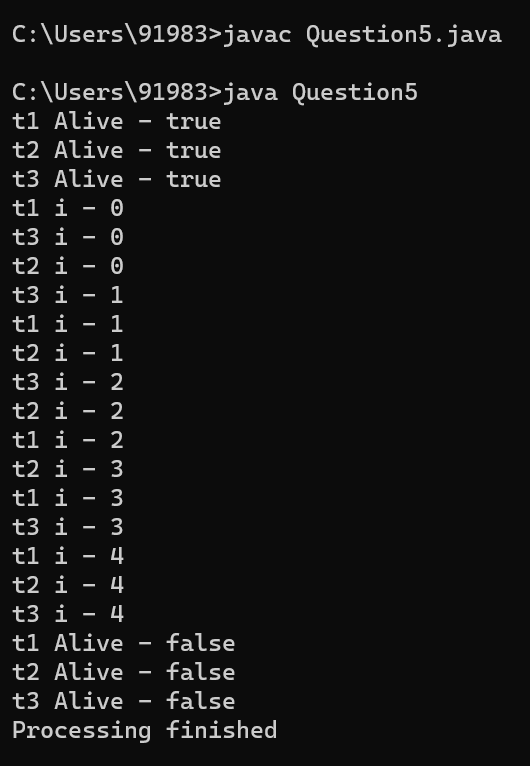
t2.join();

t3.join();

} catch (InterruptedException e) { e.printStackTrace(); }

System.out.println("t1 Alive - " + t1.isAlive()); System.out.println("t2 Alive - " + t2.isAlive()); System.out.println("t3 Alive - " + t3.isAlive()); System.out.println("Processing finished");

}}

Output:

1. Implement a program of locking of a common method by several threads. (Using the keyword ‘synchronized’).

Code:

public class Question6 implements Runnable { public void run() {

for (int i = 0; i < 3; i++) { showMe();

try {

Thread.sleep(500);

} catch (Exception e) { System.out.println(e);

}}}

synchronized void showMe() { for (int i = 0; i < 3; i++) {

System.out.println(Thread.currentThread().getName() + ": " + i); try {

Thread.sleep(100);

} catch (Exception e) { System.out.println(e);

}}}

public static void main(String[] args) throws InterruptedException { Question6 base = new Question6();

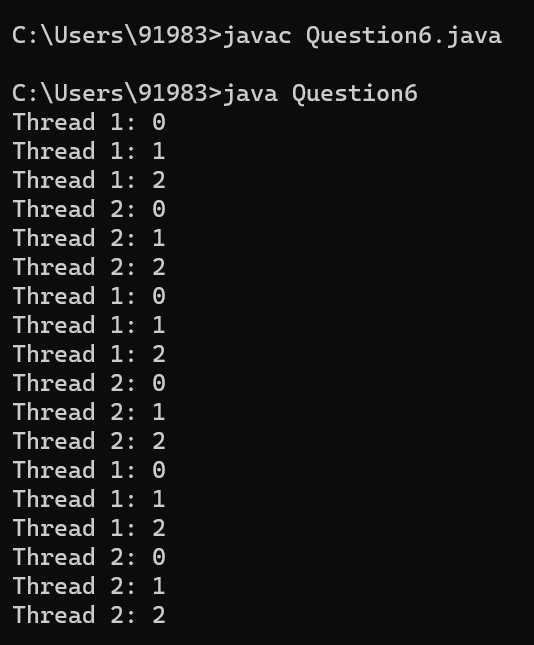
Thread t1 = new Thread(base, "Thread 1"); Thread t2 = new Thread(base, "Thread 2"); t1.start();

t2.start();

t1.join();

t2.join();

}}

Output:

1. Write a program to implement inter-thread communication the consumer consumes items produced by the producer with proper synchronization.

Code:

class ItemQueue { int item;

boolean valueSet = false; synchronized int getItem() {

while (!valueSet) try {

wait();

} catch (InterruptedException e) { System.out.println("InterruptedException caught"); }

System.out.println("Consummed:" + item); valueSet = false;

try {

Thread.sleep(1000);

} catch (InterruptedException e) { System.out.println("InterruptedException caught"); }

notify(); return item; }

synchronized void putItem(int item) { while (valueSet)

try {

wait();

} catch (InterruptedException e) { System.out.println("InterruptedException caught"); }

this.item = item; valueSet = true;

System.out.println("Produced: " + item); try {

Thread.sleep(1000);

} catch (InterruptedException e) { System.out.println("InterruptedException caught");

}

notify(); }}

class Producer implements Runnable { ItemQueue itemQueue;

Producer(ItemQueue itemQueue) { this.itemQueue = itemQueue;

new Thread(this, "Producer").start(); } public void run() {

int i = 0; while (true) {

itemQueue.putItem(i++);

}}}

class Consumer implements Runnable { ItemQueue itemQueue; Consumer(ItemQueue itemQueue) {

this.itemQueue = itemQueue;

new Thread(this, "Consumer").start(); } public void run() {

while (true) { itemQueue.getItem();

}}}

class Question7 {

public static void main(String args[]) { ItemQueue itemQueue = new ItemQueue(); new Producer(itemQueue);

new Consumer(itemQueue);

}}

Output: