**Wrapper Classes**

public class WrapperDemo {

public static void main(String[] args) {

//1. Check if character is a digit

char c = '7';

System.***out***.println(c + (Character.*isDigit*(c) ? " is a digit." : " is NOT a digit."));

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

// 2. Compare two Strings

String s1 = "hello";

String s2 = "world";

if (s1.equals(s2)) {

System.***out***.println("strings are equal.");

} else {

System.***out***.println("strings are NOT equal.");

}

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

// 3. Convert using valueOf method

int num = 100;

Integer intWrapper = Integer.*valueOf*(num);

System.***out***.println("Integer wrapper from int: " + intWrapper);

String str = "123";

Integer intFromString = Integer.*valueOf*(str);

System.***out***.println("Integer wrapper from String: " + intFromString);

double d = 12.34;

Double doubleWrapper = Double.*valueOf*(d);

System.***out***.println("Double wrapper from double: " + doubleWrapper);

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

// 4. Create Boolean Wrapper usage

Boolean bool1 = Boolean.*valueOf*(true);

Boolean bool2 = Boolean.*valueOf*("true");

Boolean bool3 = Boolean.*valueOf*("yes");

System.***out***.println("Boolean from true: " + bool1);

System.***out***.println("Boolean from true: " + bool2);

System.***out***.println("Boolean from yes: " + bool3);

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

// 5. Convert null to wrapper classes

Integer intNull = null;

Boolean boolNull = null;

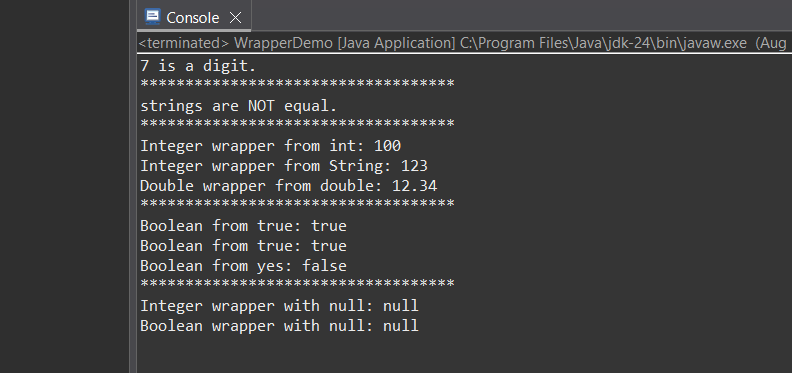
System.***out***.println("Integer wrapper with null: " + intNull);

System.***out***.println("Boolean wrapper with null: " + boolNull);

}

}

**OUTPUT:**

****

**Pass by value:**

1. **Answer**

public class PassByValueDemo {

public static void main(String[] args) {

int num = 10;

System.out.println("before method call, num = " + num);

changeValue(num);

System.out.println("after method call, num = " + num);

}

public static void changeValue(int n) {

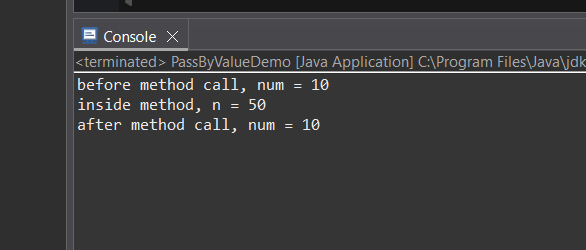
n = 50;

System.out.println("inside method, n = " + n);

}

}

**OUTPUT:**



**2.Answer**

public class SwapDemo {

public static void swap(int a, int b) {

int temp = a;

a = b;

b = temp;

System.*out*.println("inside swap method: a = " + a + ", b = " + b);

}

public static void main(String[] args) {

int x = 10;

int y = 20;

System.*out*.println("before swap method call: x = " + x + ", y = " + y);

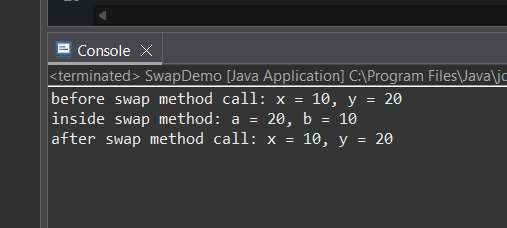
*swap*(x, y);

System.*out*.println("after swap method call: x = " + x + ", y = " + y);

}

}

**OUTPUT:**

****

**3.Answer**

public class PassByDemo {

public static void changeValue(int number) {

System.***out***.println("inside method, before change: " + number);

number = 100;

System.***out***.println("inside method, after change: " + number);

}

public static void main(String[] args) {

int originalNumber = 50;

System.***out***.println("before method call: " + originalNumber);

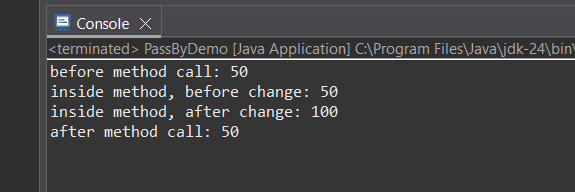
*changeValue*(originalNumber);

System.***out***.println("after method call: " + originalNumber);

}

}

**OUTPUT:**

****

**Call by Reference**

**4.Answer**

class Box {

int length;

Box(int length) {

this.length = length;

}

}

public class CallByRef {

static void modifyBoxLength(Box b) {

b.length = 100;

}

public static void main(String[] args) {

Box myBox = new Box(50);

System.***out***.println("before modify: " + myBox.length);

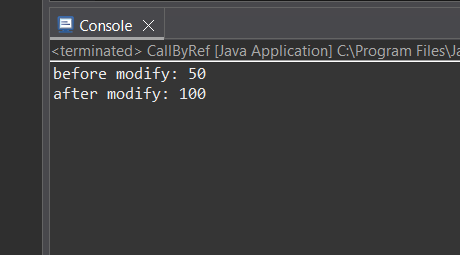
*modifyBoxLength*(myBox);

System.***out***.println("after modify: " + myBox.length);

}

}

**OUTPUT:**

****

**5. Answer**

class Box1 {

int length;

Box1(int length) {

this.length = length;

}

void display() {

System.***out***.println("length: " + length);

}

}

public class CallByRef2 {

static void modifyBox(Box1 b) {

b.length = 100;

System.***out***.println("inside modifyBox method:");

b.display();

}

public static void main(String[] args) {

Box1 box = new Box1(10);

System.***out***.println("before modifyBox call:");

box.display();

*modifyBox*(box);

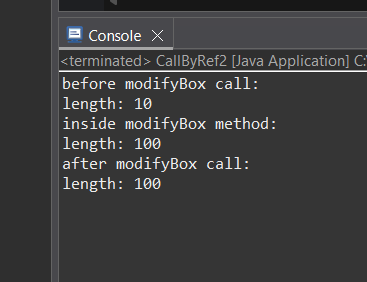
System.***out***.println("after modifyBox call:");

box.display();

}

}

**OUTPUT:**

****

**6.Answer**

class Student {

String name;

int marks;

public Student(String name, int marks) {

this.name = name;

this.marks = marks;

}

public void display() {

System.***out***.println("name: " + name + ", marks: " + marks);

}

}

public class CallByRef3 {

public static void updateMarks(Student s, int newMarks) {

s.marks = newMarks;

}

public static void main(String[] args) {

Student student1 = new Student("Anush", 75);

System.***out***.println("before update:");

student1.display();

*updateMarks*(student1, 90);

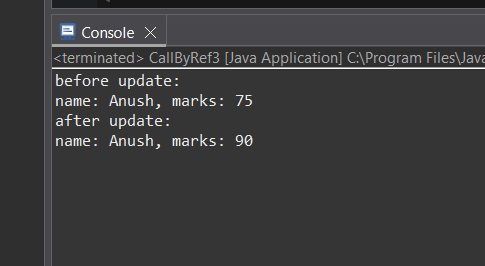
System.***out***.println("after update:");

student1.display();

}

}

**OUTPUT:**

****

**7.Answer**

class Student2 {

String name;

int marks;

Student2(String name, int marks) {

this.name = name;

this.marks = marks;

}

void display() {

System.***out***.println("Name: " + name + ", Marks: " + marks);

}

}

public class CallByValue2 {

static void changeObjectReference(Student2 s) {

s = new Student2("NewStudent", 100);

System.***out***.println("Inside changeObjectReference:");

s.display();

}

static void modifyObjectState(Student2 s) {

s.marks = 999;

System.***out***.println("Inside modifyObjectState:");

s.display();

}

public static void main(String[] args) {

Student2 stu = new Student2("OriginalStudent", 50);

System.***out***.println("Before changeObjectReference:");

stu.display();

*changeObjectReference*(stu);

System.***out***.println("After changeObjectReference:");

stu.display();

*modifyObjectState*(stu);

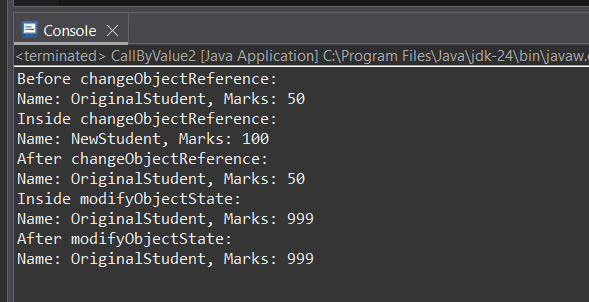
System.***out***.println("After modifyObjectState:");

stu.display();

}

}

**OUTPUT:**

****

**8.Answer**

class Animal {

String name;

Animal(String name) {

this.name = name;

}

void display() {

System.***out***.println("animal name: " + name);

}

}

public class ReferenceT {

static void assignNewAnimal(Animal a) {

a = new Animal("lion");

System.***out***.println("inside method:");

a.display();

}

public static void main(String[] args) {

Animal originalAnimal = new Animal("dog");

System.***out***.println("before method call:");

originalAnimal.display();

*assignNewAnimal*(originalAnimal);

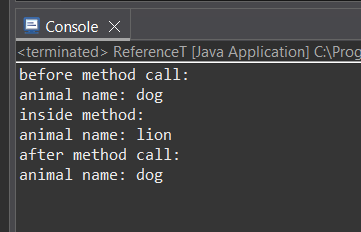
System.***out***.println("after method call:");

originalAnimal.display();

}

}

**OUTPUT:**

****

**9.Answer**

class Person {

String name;

Person(String name) {

this.name = name;

}

}

public class PrimitiveDemo {

public static void main(String[] args) {

// Primitive type

int num = 10;

System.***out***.println("Primitive before: " + num);

*changePrimitive*(num);

System.***out***.println("Primitive after: " + num);

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

// Non-primitive type (modify state)

Person p1 = new Person("Alice");

System.***out***.println("Object before: " + p1.name);

*changeObjectState*(p1);

System.***out***.println("Object after: " + p1.name);

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

// Non-primitive type (assign new object)

Person p2 = new Person("Alice");

System.***out***.println("Reference before: " + p2.name);

*assignNewObject*(p2);

System.***out***.println("Reference after: " + p2.name);

}

static void changePrimitive(int x) {

x = 20;

}

static void changeObjectState(Person person) {

person.name = "Bob";

}

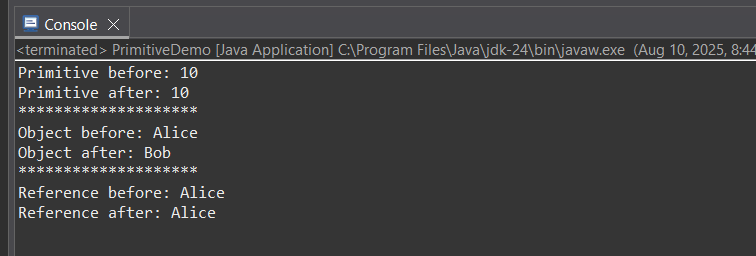
static void assignNewObject(Person person) {

person = new Person("Charlie");

}

}

**OUTPUT:**



**10.Answer**

class IntWrapper {

int value;

IntWrapper(int value) {

this.value = value;

}

}

public class WrapperReference {

static void incrementUsingWrapper(IntWrapper num) {

num.value += 10;

}

static void incrementUsingArray(int[] arr) {

arr[0] += 10;

}

public static void main(String[] args) {

// Using Wrapper Class

IntWrapper number = new IntWrapper(20);

System.***out***.println("Before (Wrapper): " + number.value);

*incrementUsingWrapper*(number);

System.***out***.println("After (Wrapper): " + number.value);

System.***out***.println();

// Using Array

int[] numArray = { 20 };

System.***out***.println("Before (Array) : " + numArray[0]);

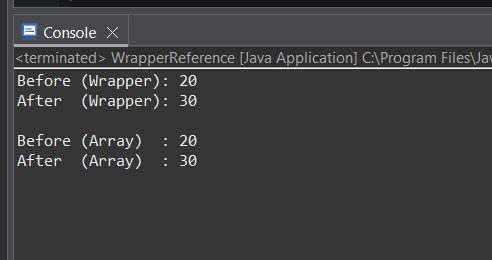
*incrementUsingArray*(numArray);

System.***out***.println("After (Array) : " + numArray[0]);

}

}

**OUTPUT:**



**MultiThreading**

**1.Answer**

class MyThread1 extends Thread {

public void run() {

for (int i = 1; i <= 5; i++) {

System.***out***.println(i);

}

}

}

public class ThreadEg1 {

public static void main(String[] args) {

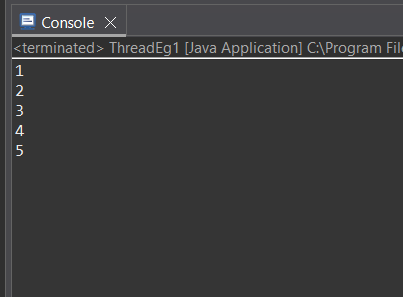
MyThread1 t1 = new MyThread1();

t1.start();

}

}

**OUTPUT:**



**2.Answer**

class MyRunnable implements Runnable {

public void run() {

System.***out***.println("Current Thread: " + Thread.*currentThread*().getName());

}

}

public class ThreadEg2 {

public static void main(String[] args) {

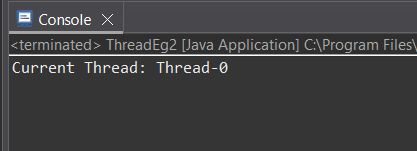
Thread t1 = new Thread(new MyRunnable());

t1.start();

}

}

**OUTPUT:**



**3.Answer**

class MessageThread extends Thread {

String message;

MessageThread(String msg) {

this.message = msg;

}

public void run() {

for (int i = 1; i <= 5; i++) {

System.***out***.println(message);

}

}

}

public class ThreadEg3 {

public static void main(String[] args) {

MessageThread t1 = new MessageThread("Thread 1");

MessageThread t2 = new MessageThread("Thread 2");

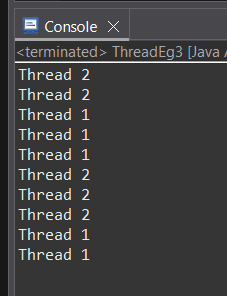
t1.start();

t2.start();

}

}

**OUTPUT:**



**4.Answer**

class SleepThread extends Thread {

public void run() {

try {

for (int i = 1; i <= 3; i++) {

System.***out***.println(i);

Thread.*sleep*(1000);

}

} catch (InterruptedException e) {

System.***out***.println(e);

}

}

}

public class ThreadEg4 {

public static void main(String[] args) {

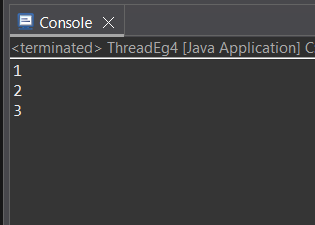
SleepThread t1 = new SleepThread();

t1.start();

}

}

**OUTPUT:**



**5.Answer**

class YieldThread extends Thread {

public void run() {

for (int i = 1; i <= 5; i++) {

System.***out***.println(Thread.*currentThread*().getName() + " - " + i);

Thread.*yield*();

}

}

}

public class ThreadEg5 {

public static void main(String[] args) {

YieldThread t1 = new YieldThread();

YieldThread t2 = new YieldThread();

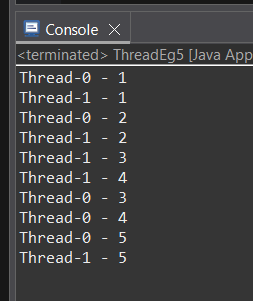
t1.start();

t2.start();

}

}

**OUTPUT:**



**6.Answer**

class EvenPrinter extends Thread {

public void run() {

for (int i = 2; i <= 10; i += 2) {

System.***out***.println("Even: " + i);

}

}

}

class OddPrinter extends Thread {

public void run() {

for (int i = 1; i <= 10; i += 2) {

System.***out***.println("Odd: " + i);

}

}

}

public class EvenOddThreads {

public static void main(String[] args) {

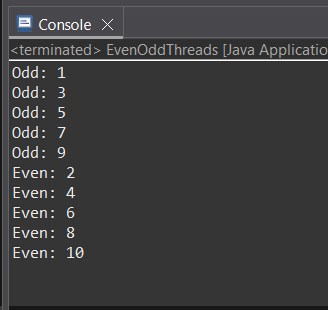
new EvenPrinter().start();

new OddPrinter().start();

}

}

**OUTPUT:**



**7.Answer**

class MyPriorityThread extends Thread {

MyPriorityThread(String name) {

super(name);

}

public void run() {

for (int i = 1; i <= 5; i++) {

System.***out***.println(getName() + " - " + i);

}

}

}

public class ThreadPriorityDemo {

public static void main(String[] args) {

MyPriorityThread t1 = new MyPriorityThread("Low Priority");

MyPriorityThread t2 = new MyPriorityThread("Medium Priority");

MyPriorityThread t3 = new MyPriorityThread("High Priority");

t1.setPriority(Thread.***MIN\_PRIORITY***);

t2.setPriority(Thread.***NORM\_PRIORITY***);

t3.setPriority(Thread.***MAX\_PRIORITY***);

t1.start();

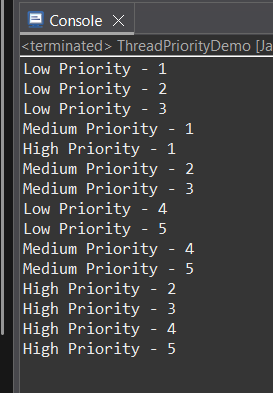
t2.start();

t3.start();

}

}

**OUTPUT:**



**8.Answer**

class WorkerThread extends Thread {

public void run() {

for (int i = 1; i <= 5; i++) {

System.***out***.println("Worker: " + i);

try { Thread.*sleep*(500); } catch (Exception e) {}

}

}

}

public class ThreadJoinDemo {

public static void main(String[] args) {

WorkerThread worker = new WorkerThread();

worker.start();

try {

worker.join();

} catch (Exception e) {

e.printStackTrace();

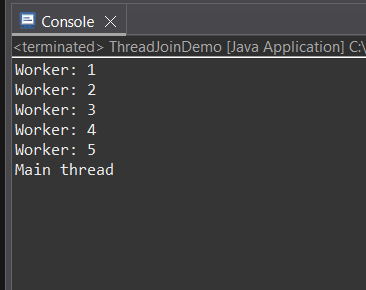
}

System.***out***.println("Main thread");

}

}

**OUTPUT:**



**9.Answer**

class StoppableThread extends Thread {

private volatile boolean running = true;

public void stopThread() {

running = false;

}

public void run() {

while (running) {

System.***out***.println("Thread is running");

try { Thread.*sleep*(500); } catch (InterruptedException e) {}

}

System.***out***.println("Thread stopped");

}

}

public class StopThreadDemo {

public static void main(String[] args) throws InterruptedException {

StoppableThread t = new StoppableThread();

t.start();

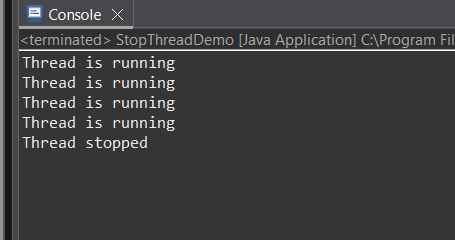
Thread.*sleep*(2000);

t.stopThread();

}

}

**OUTPUT:**

****

**10.Answer**

class Counter {

int count = 0;

}

class CounterThread extends Thread {

Counter counter;

CounterThread(Counter counter) {

this.counter = counter;

}

public void run() {

for (int i = 0; i < 1000; i++) {

counter.count++;

}

}

}

public class RaceConditionDemo {

public static void main(String[] args) throws InterruptedException {

Counter counter = new Counter();

CounterThread t1 = new CounterThread(counter);

CounterThread t2 = new CounterThread(counter);

t1.start();

t2.start();

t1.join();

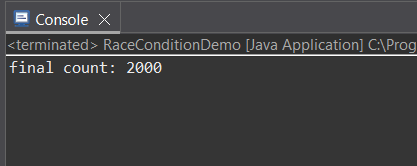
t2.join();

System.***out***.println("final count: " + counter.count);

}

}

**OUTPUT:**

****

**11.Answer**

class Counter1 {

private int count = 0;

public synchronized void increment() {

count++;

}

public int getCount() {

return count;

}

}

class MyThread extends Thread {

Counter1 counter;

MyThread(Counter1 counter) {

this.counter = counter;

}

public void run() {

for (int i = 0; i < 1000; i++) {

counter.increment();

}

}

}

public class SynchronizedExample {

public static void main(String[] args) throws InterruptedException {

Counter1 coun = new Counter1();

MyThread t1 = new MyThread(coun);

MyThread t2 = new MyThread(coun);

t1.start();

t2.start();

t1.join();

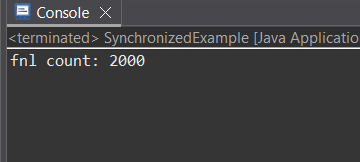
t2.join();

System.*out*.println("fnl count: " + coun.getCount());

}

}

**OUTPUT:**

****

**12.Answer**

class SharedResource {

void printTable(int n) {

synchronized (this) {

for (int i = 1; i <= 5; i++) {

System.***out***.println(Thread.*currentThread*().getName() + " : " + (n \* i));

try {

Thread.*sleep*(500);

} catch (InterruptedException e) {

System.***out***.println(e);

}

}

}

}

}

class MyThread2 extends Thread {

SharedResource resource;

int number;

MyThread2(SharedResource resource, int number, String name) {

super(name);

this.resource = resource;

this.number = number;

}

public void run() {

resource.printTable(number);

}

}

public class SynchronizedBlock2 {

public static void main(String[] args) {

SharedResource obj = new SharedResource();

MyThread2 t1 = new MyThread2(obj, 5, "Thread-1");

MyThread2 t2 = new MyThread2(obj, 10, "Thread-2");

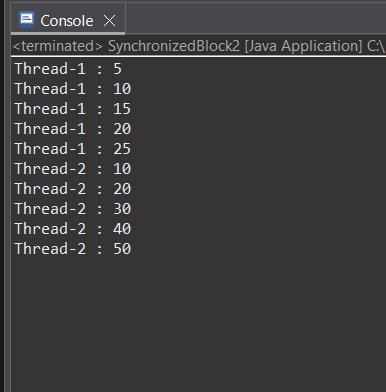
t1.start();

t2.start();

}

}

**OUTPUT:**



**13.Answer**

class BankAccount {

private double balance;

public BankAccount(double initialBalance) {

this.balance = initialBalance;

}

public synchronized void deposit(double amount) {

System.***out***.println(Thread.*currentThread*().getName() + " depositing: " + amount);

balance += amount;

System.***out***.println(Thread.*currentThread*().getName() + " new balance: " + balance);

}

public synchronized void withdraw(double amount) {

System.***out***.println(Thread.*currentThread*().getName() + " withdrawing: " + amount);

if (amount <= balance) {

balance -= amount;

System.***out***.println(Thread.*currentThread*().getName() + " new balance: " + balance);

} else {

System.***out***.println(Thread.*currentThread*().getName() + " insufficient funds. Balance: " + balance);

}

}

}

class BankingTask implements Runnable {

private BankAccount account;

public BankingTask(BankAccount account) {

this.account = account;

}

*@Override*

public void run() {

account.deposit(200);

account.withdraw(150);

account.withdraw(100);

}

}

public class BankAccountDemo {

public static void main(String[] args) {

BankAccount acc = new BankAccount(500);

Thread t1 = new Thread(new BankingTask(acc), "Thread-1");

Thread t2 = new Thread(new BankingTask(acc), "Thread-2");

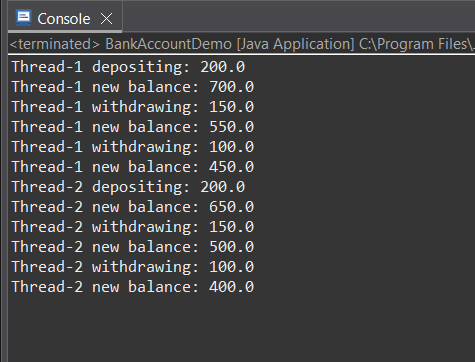
t1.start();

t2.start();

}

}

**OUTPUT:**

****

**14.Answer**

class SharedBuffer {

private int data;

private boolean hasData = false;

public synchronized void produce(int value) {

while (hasData) {

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

data = value;

hasData = true;

System.***out***.println("Produced: " + value);

notify();

}

public synchronized int consume() {

while (!hasData) {

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

hasData = false;

System.***out***.println("Consumed: " + data);

notify();

return data;

}

}

class Producer extends Thread {

private SharedBuffer buffer;

public Producer(SharedBuffer buffer) {

this.buffer = buffer;

}

public void run() {

for (int i = 1; i <= 5; i++) {

buffer.produce(i);

try {

Thread.*sleep*(500);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

class Consumer extends Thread {

private SharedBuffer buffer;

public Consumer(SharedBuffer buffer) {

this.buffer = buffer;

}

public void run() {

for (int i = 1; i <= 5; i++) {

buffer.consume();

try {

Thread.*sleep*(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

public class ProducerConsumer {

public static void main(String[] args) {

SharedBuffer buffer = new SharedBuffer();

Producer producer = new Producer(buffer);

Consumer consumer = new Consumer(buffer);

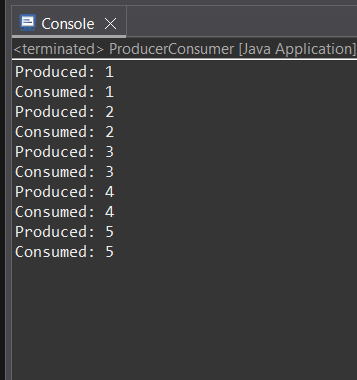
producer.start();

consumer.start();

}

}

**OUTPUT:**

****

**15.Answer**

class Printer {

private boolean letterTurn = true;

public synchronized void printLetter(char letter) {

while (!letterTurn) {

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

System.***out***.print(letter + " ");

letterTurn = false;

notify();

}

public synchronized void printNumber(int number) {

while (letterTurn) {

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

System.***out***.print(number + " ");

letterTurn = true;

notify();

}

}

class LetterThread extends Thread {

private Printer printer;

public LetterThread(Printer printer) {

this.printer = printer;

}

public void run() {

for (char ch = 'A'; ch <= 'Z'; ch++) {

printer.printLetter(ch);

}

}

}

class NumberThread extends Thread {

private Printer printer;

public NumberThread(Printer printer) {

this.printer = printer;

}

public void run() {

for (int i = 1; i <= 26; i++) {

printer.printNumber(i);

}

}

}

public class AlternatePrinting {

public static void main(String[] args) {

Printer printer = new Printer();

Thread t1 = new LetterThread(printer);

Thread t2 = new NumberThread(printer);

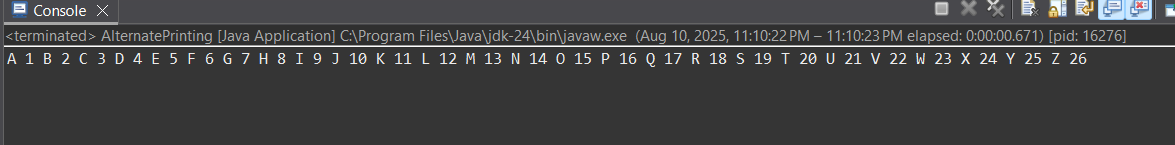
t1.start();

t2.start();

}

}

**OUTPUT:**



**16.Answer**

class SharedResource1 {

private boolean ready = false;

public synchronized void waitUntilReady() {

while (!ready) {

try {

System.***out***.println(Thread.*currentThread*().getName() + " is waiting");

wait();

} catch (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

System.***out***.println(Thread.*currentThread*().getName() + " proceeding");

}

public synchronized void setReady() {

ready = true;

System.***out***.println(Thread.*currentThread*().getName() + " notifying all");

notifyAll();

}

}

public class InterThreadCom {

public static void main(String[] args) {

SharedResource1 resource = new SharedResource1();

for (int i = 1; i <= 3; i++) {

Thread t = new Thread(resource::waitUntilReady, "worker-" + i);

t.start();

}

Thread notifier = new Thread(() -> {

try {

Thread.*sleep*(2000);

} catch (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

resource.setReady();

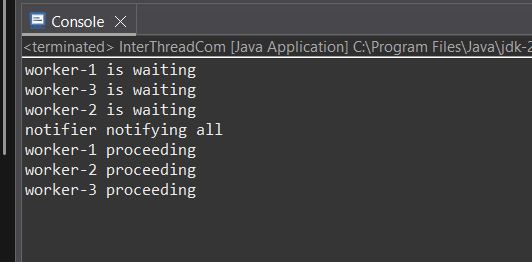
}, "notifier");

notifier.start();

}

}

**OUTPUT:**



**17.Answer**

import java.time.LocalTime;

import java.time.format.DateTimeFormatter;

class TimePrinter extends Thread {

public void run() {

DateTimeFormatter formatter = DateTimeFormatter.*ofPattern*("HH:mm:ss");

while (true) {

System.***out***.println("current Time: " + LocalTime.*now*().format(formatter));

try {

Thread.*sleep*(1000);

} catch (InterruptedException e) {

System.***out***.println("daemon thread");

}

}

}

}

public class DaemonThread {

public static void main(String[] args) {

TimePrinter daemonThread = new TimePrinter();

daemonThread.setDaemon(true);

daemonThread.start();

for (int i = 1; i <= 5; i++) {

System.***out***.println("main thread working" + i);

try {

Thread.*sleep*(2000);

} catch (InterruptedException e) {

e.printStackTrace();

}

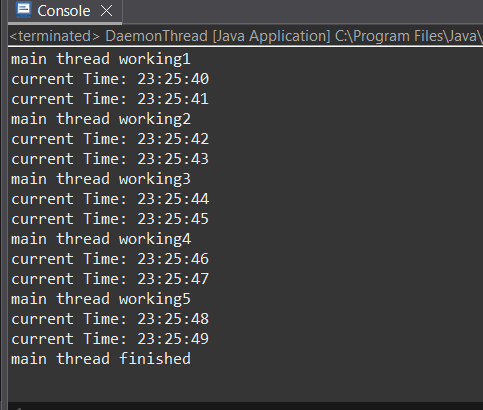
}

System.***out***.println("main thread finished");

}

}

**OUTPUT:**



**18.Answer**

class MyThread3 extends Thread {

public void run() {

System.***out***.println(getName() + " is running");

try {

Thread.*sleep*(2000);

} catch (Exception e) {

e.printStackTrace();

}

System.***out***.println(getName() + " has finished");

}

}

public class ThreadIsAlive {

public static void main(String[] args) {

MyThread3 t1 = new MyThread3();

System.***out***.println("before starting: t1 is alive? " + t1.isAlive());

t1.start();

System.***out***.println("after starting: t1 is alive? " + t1.isAlive());

try {

Thread.*sleep*(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

System.***out***.println("while running: t1 is alive? " + t1.isAlive());

try {

t1.join();

} catch (InterruptedException e) {

e.printStackTrace();

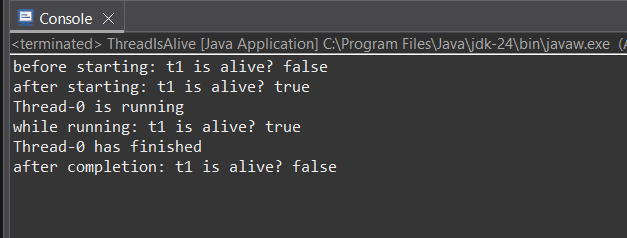
}

System.***out***.println("after completion: t1 is alive? " + t1.isAlive());

}

}

**OUTPUT:**



**19.Answer**

class MyThread4 extends Thread {

MyThread4(ThreadGroup group, String name) {

super(group, name);

}

*@Override*

public void run() {

System.***out***.println(getName() + " started");

try {

Thread.*sleep*(2000);

} catch (Exception e) {

System.***out***.println(getName() + " interrupted");

}

System.***out***.println(getName() + " finished");

}

}

public class ThreadGroupDemo {

public static void main(String[] args) throws InterruptedException {

ThreadGroup group = new ThreadGroup("myThreadGroup");

MyThread4 t1 = new MyThread4(group, "Thread-1");

MyThread4 t2 = new MyThread4(group, "Thread-2");

MyThread4 t3 = new MyThread4(group, "Thread-3");

t1.start();

t2.start();

t3.start();

System.***out***.println("active Thread Count: " + group.activeCount());

System.***out***.println("thread Group name: " + group.getName());

group.list();

t1.join();

t2.join();

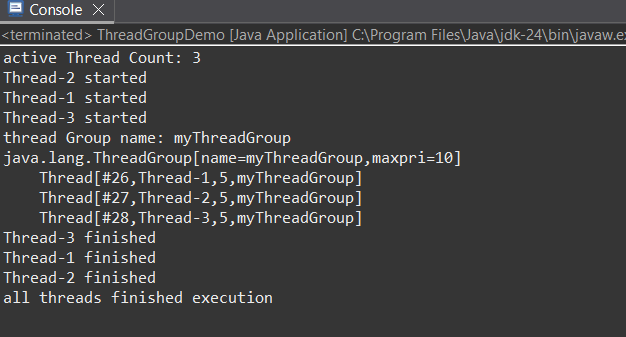
t3.join();

System.***out***.println("all threads finished execution");

}

}

**OUTPUT:**



**20.Answer**

import java.util.concurrent.Callable;

import java.util.concurrent.ExecutionException;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.Future;

public class MultiplicationCallable {

static class MultiplicationTask implements Callable<Integer> {

private int a, b;

public MultiplicationTask(int a, int b) {

this.a = a;

this.b = b;

}

*@Override*

public Integer call() throws Exception {

Thread.*sleep*(1000);

return a \* b;

}

}

public static void main(String[] args) {

ExecutorService executor = Executors.*newSingleThreadExecutor*();

Callable<Integer> task = new MultiplicationTask(5, 7);

Future<Integer> futureResult = executor.submit(task);

try {

Integer result = futureResult.get();

System.***out***.println("multiplication result: " + result);

} catch (InterruptedException | ExecutionException e) {

e.printStackTrace();

} finally {

executor.shutdown();

}

}

}

**OUTPUT:**

