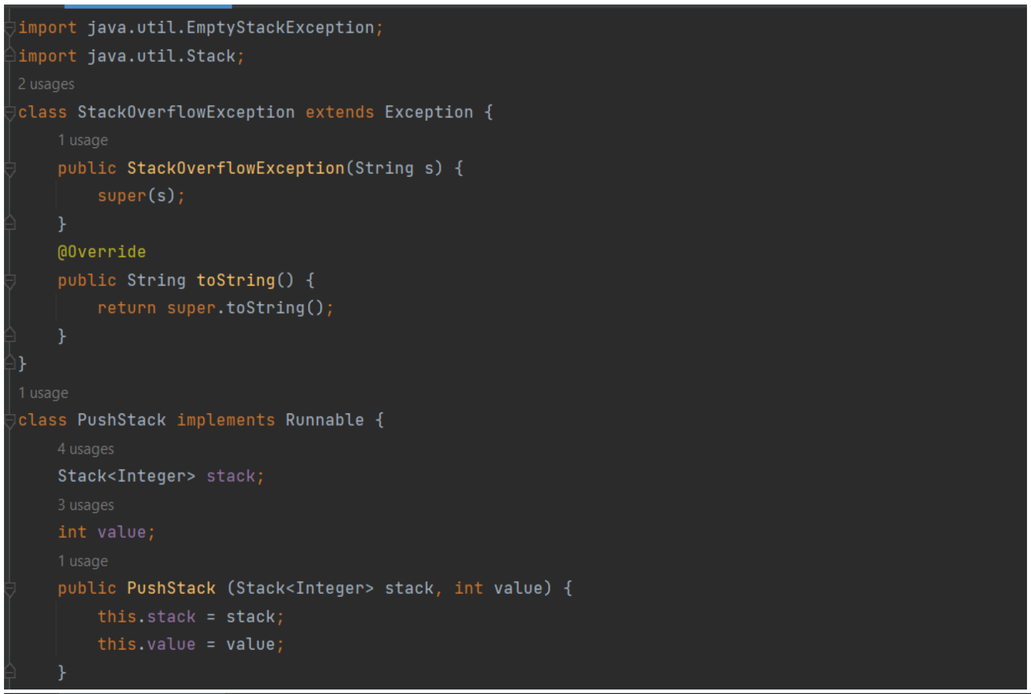
Day 8 – Assignment

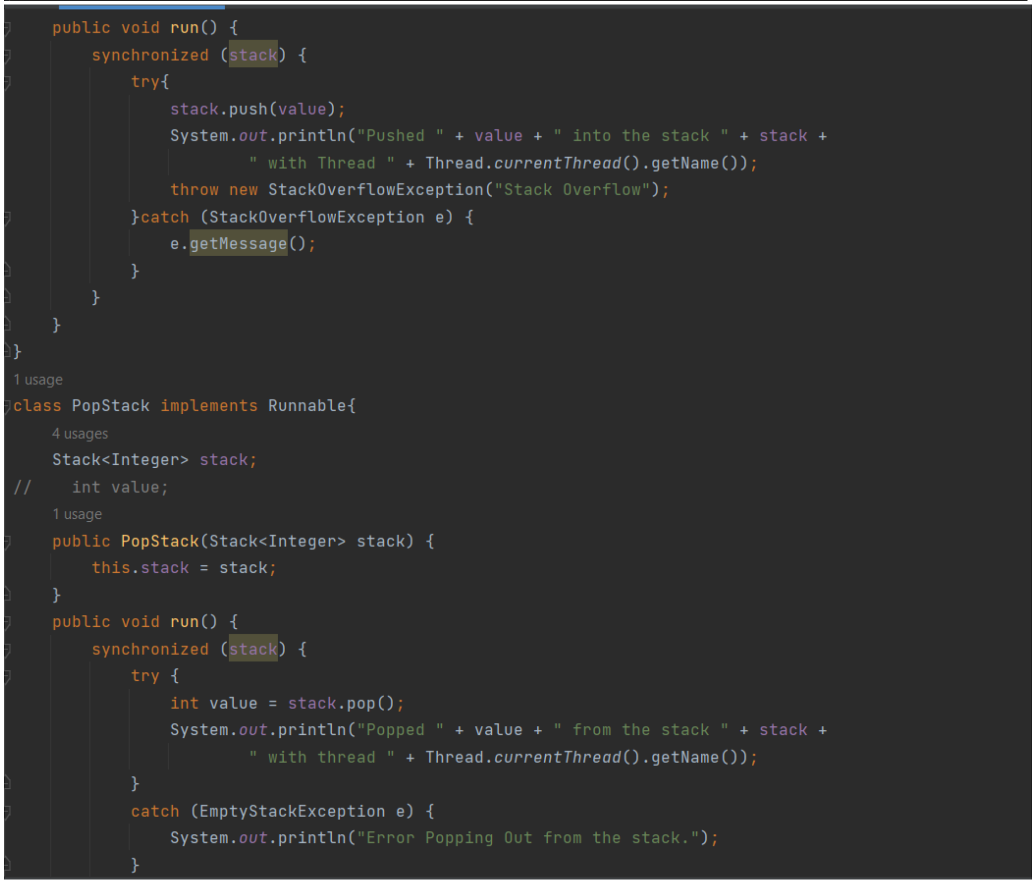
Pratik K Kamble
Employee ID : 46263548
19 – 09 – 2022

Q1: Write a Java program which handles Push operation and Pop operation on stack concurrently.

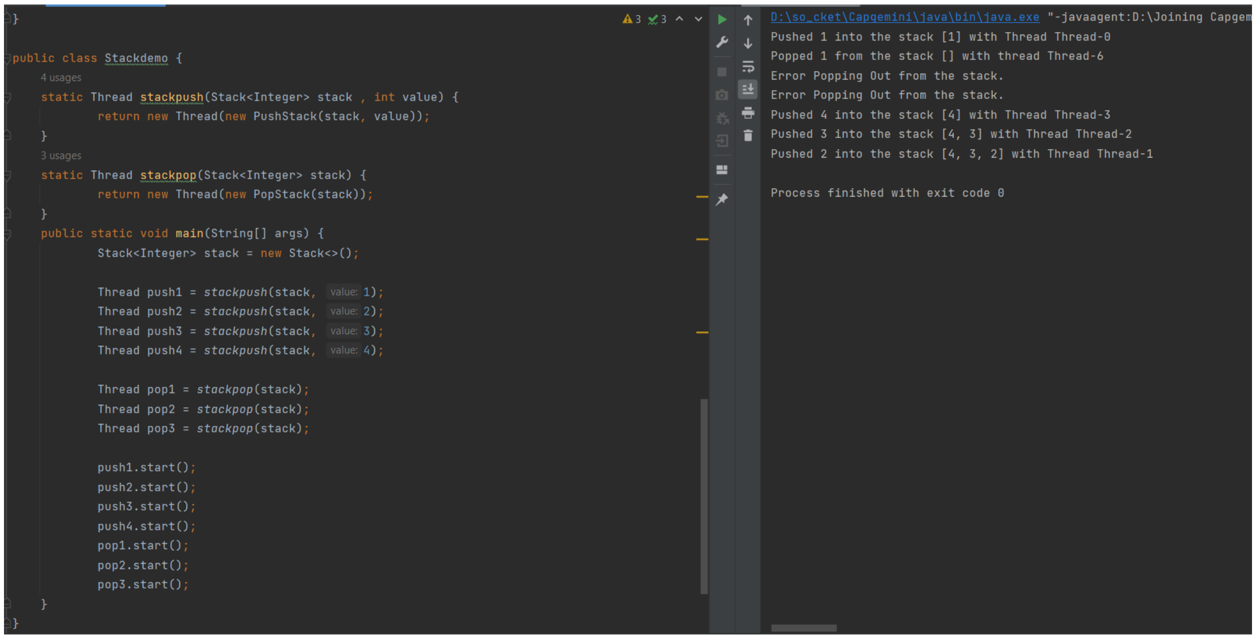
Solution –



```java
import java.util.EmptyStackException;
import java.util.Stack;
2 usages
class StackOverflowException extends Exception {
    1 usage
    public StackOverflowException(String s) {
        super(s);
    }
    @Override
    public String toString() {
        return super.toString();
    }
}
1 usage
class PushStack implements Runnable {
    4 usages
    Stack<Integer> stack;
    3 usages
    int value;
    1 usage
    public PushStack (Stack<Integer> stack, int value) {
        this.stack = stack;
        this.value = value;
    }
```



```java
    public void run() {
        synchronized (stack) {
            try{
                stack.push(value);
                System.out.println("Pushed " + value + " into the stack " + stack +
                        " with Thread " + Thread.currentThread().getName());
                throw new StackOverflowException("Stack Overflow");
            }catch (StackOverflowException e) {
                e.getMessage();
            }
        }
    }
}
1 usage
class PopStack implements Runnable{
    4 usages
    Stack<Integer> stack;
//    int value;
    1 usage
    public PopStack(Stack<Integer> stack) {
        this.stack = stack;
    }
    public void run() {
        synchronized (stack) {
            try {
                int value = stack.pop();
                System.out.println("Popped " + value + " from the stack " + stack +
                        " with thread " + Thread.currentThread().getName());
            }
            catch (EmptyStackException e) {
                System.out.println("Error Popping Out from the stack.");
            }
```

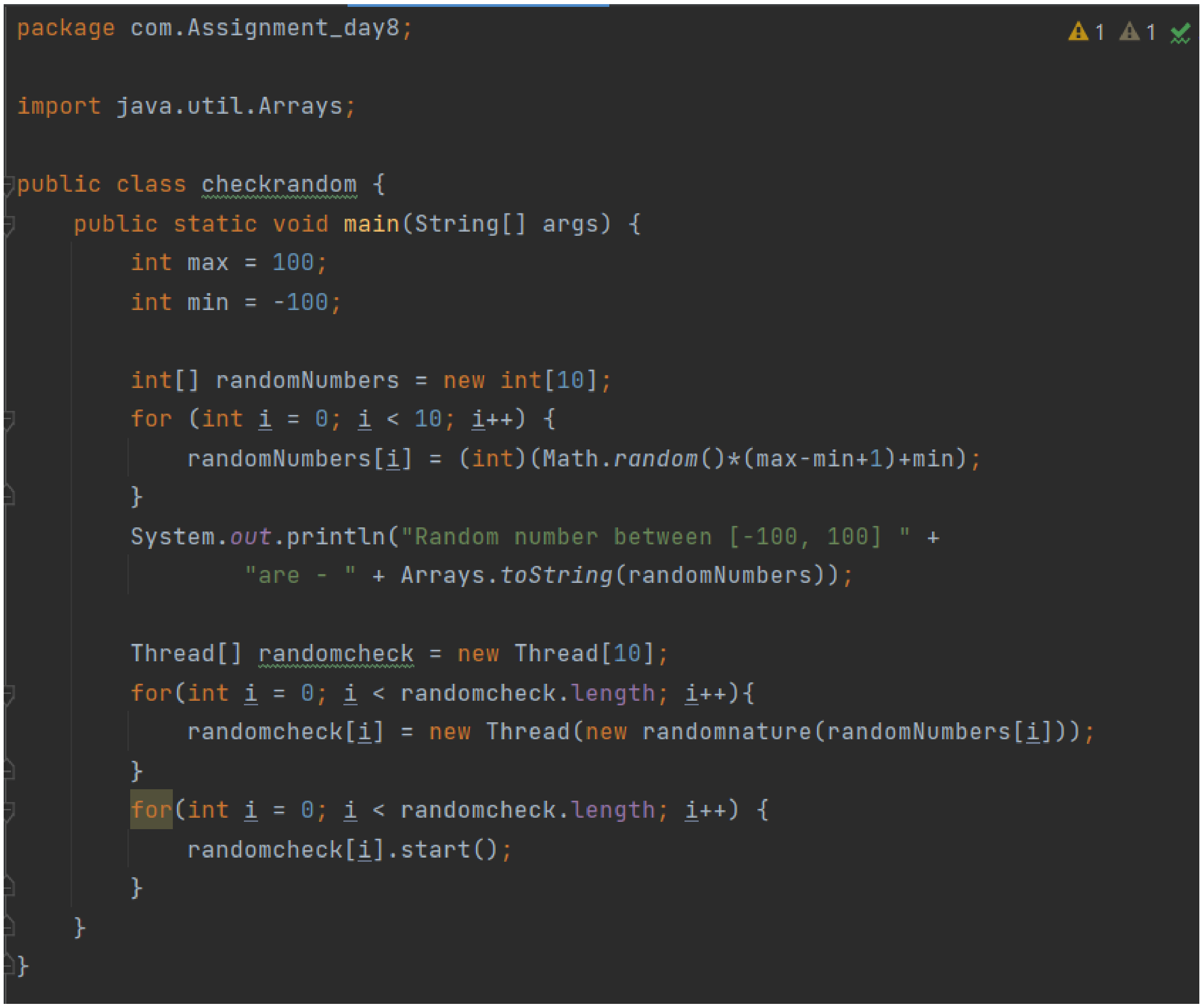


```java
}

public class Stackdemo {
    4 usages
    static Thread stackpush(Stack<Integer> stack , int value) {
            return new Thread(new PushStack(stack, value));
    }
    3 usages
    static Thread stackpop(Stack<Integer> stack) {
            return new Thread(new PopStack(stack));
    }
    public static void main(String[] args) {
            Stack<Integer> stack = new Stack<>();

            Thread push1 = stackpush(stack,  value: 1);
            Thread push2 = stackpush(stack,  value: 2);
            Thread push3 = stackpush(stack,  value: 3);
            Thread push4 = stackpush(stack,  value: 4);

            Thread pop1 = stackpop(stack);
            Thread pop2 = stackpop(stack);
            Thread pop3 = stackpop(stack);

            push1.start();
            push2.start();
            push3.start();
            push4.start();
            pop1.start();
            pop2.start();
            pop3.start();
    }
}
```

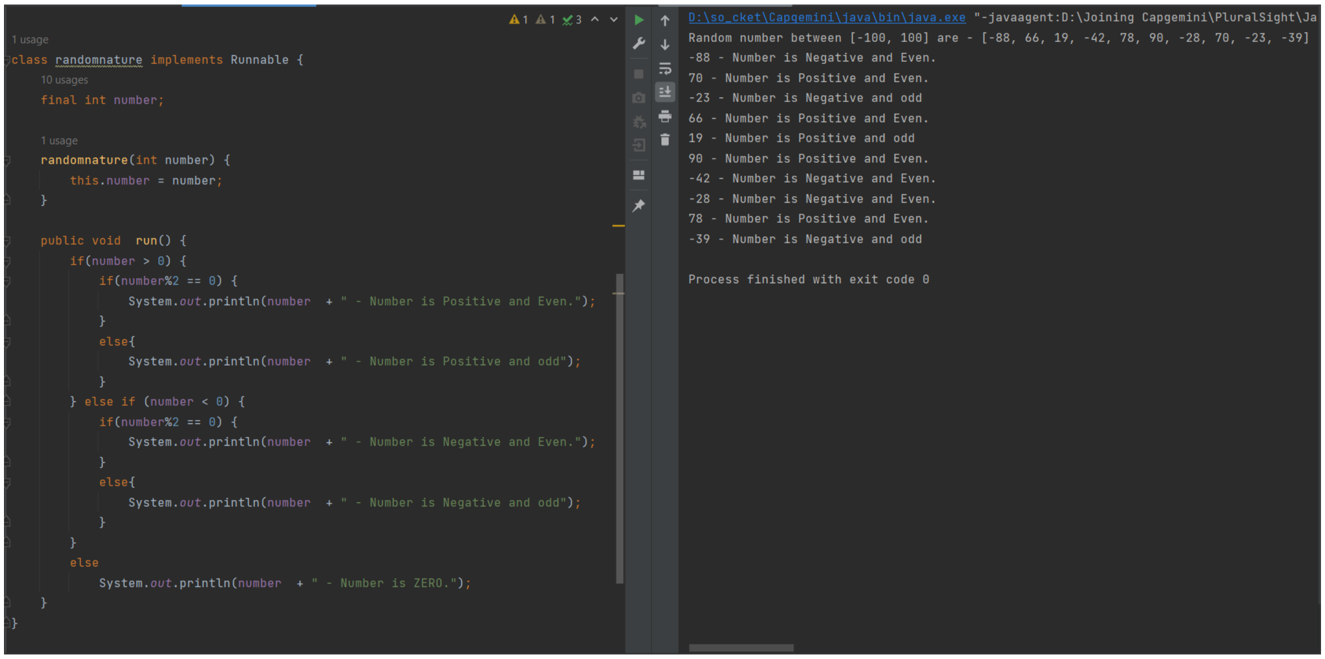
```
D:\so_cket\Capgemini\java\bin\java.exe "-javaagent:D:\Joining Capgem
Pushed 1 into the stack [1] with Thread Thread-0
Popped 1 from the stack [] with thread Thread-6
Error Popping Out from the stack.
Error Popping Out from the stack.
Pushed 4 into the stack [4] with Thread Thread-3
Pushed 3 into the stack [4, 3] with Thread Thread-2
Pushed 2 into the stack [4, 3, 2] with Thread Thread-1

Process finished with exit code 0
```

Q2: Write a Java program which first generates a set of random numbers and then determines negative, positive even, positive odd numbers concurrently.

Solution –



```java
package com.Assignment_day8;

import java.util.Arrays;

public class checkrandom {
    public static void main(String[] args) {
        int max = 100;
        int min = -100;

        int[] randomNumbers = new int[10];
        for (int i = 0; i < 10; i++) {
            randomNumbers[i] = (int)(Math.random()*(max-min+1)+min);
        }
        System.out.println("Random number between [-100, 100] " +
                "are - " + Arrays.toString(randomNumbers));

        Thread[] randomcheck = new Thread[10];
        for(int i = 0; i < randomcheck.length; i++){
            randomcheck[i] = new Thread(new randomnature(randomNumbers[i]));
        }
        for(int i = 0; i < randomcheck.length; i++) {
            randomcheck[i].start();
        }
    }
}
```

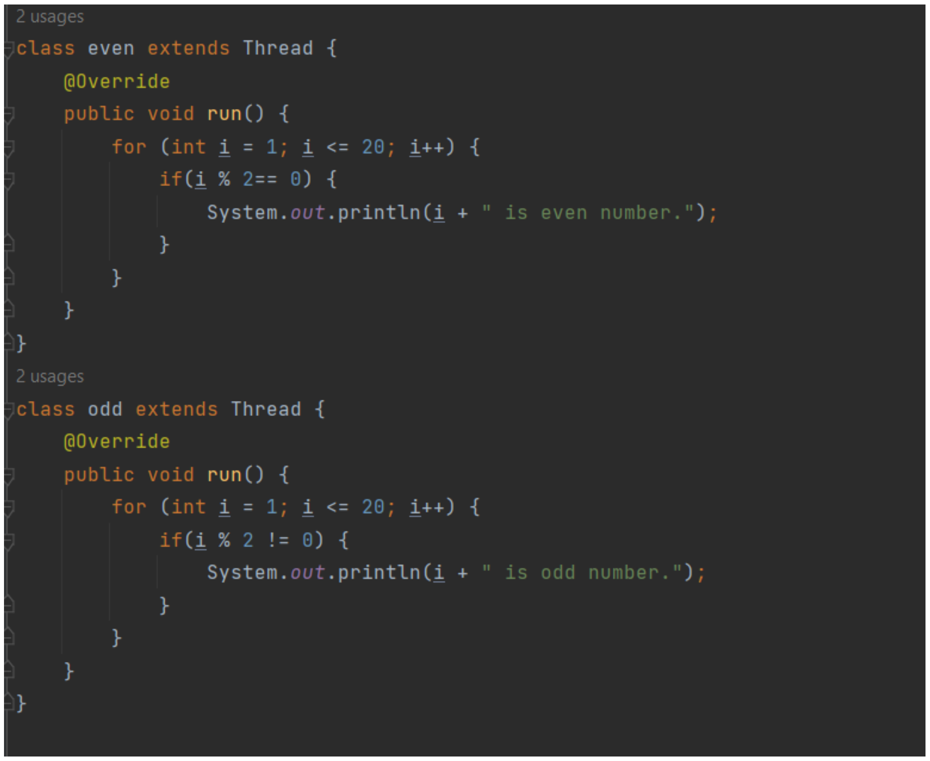


```java
1 usage
class randomnature implements Runnable {
    10 usages
    final int number;

    1 usage
    randomnature(int number) {
        this.number = number;
    }

    public void  run() {
        if(number > 0) {
            if(number%2 == 0) {
                System.out.println(number  + " - Number is Positive and Even.");
            }
            else{
                System.out.println(number  + " - Number is Positive and odd");
            }
        } else if (number < 0) {
            if(number%2 == 0) {
                System.out.println(number  + " - Number is Negative and Even.");
            }
            else{
                System.out.println(number  + " - Number is Negative and odd");
            }
        }
        else
            System.out.println(number  + " - Number is ZERO.");
    }
}
```

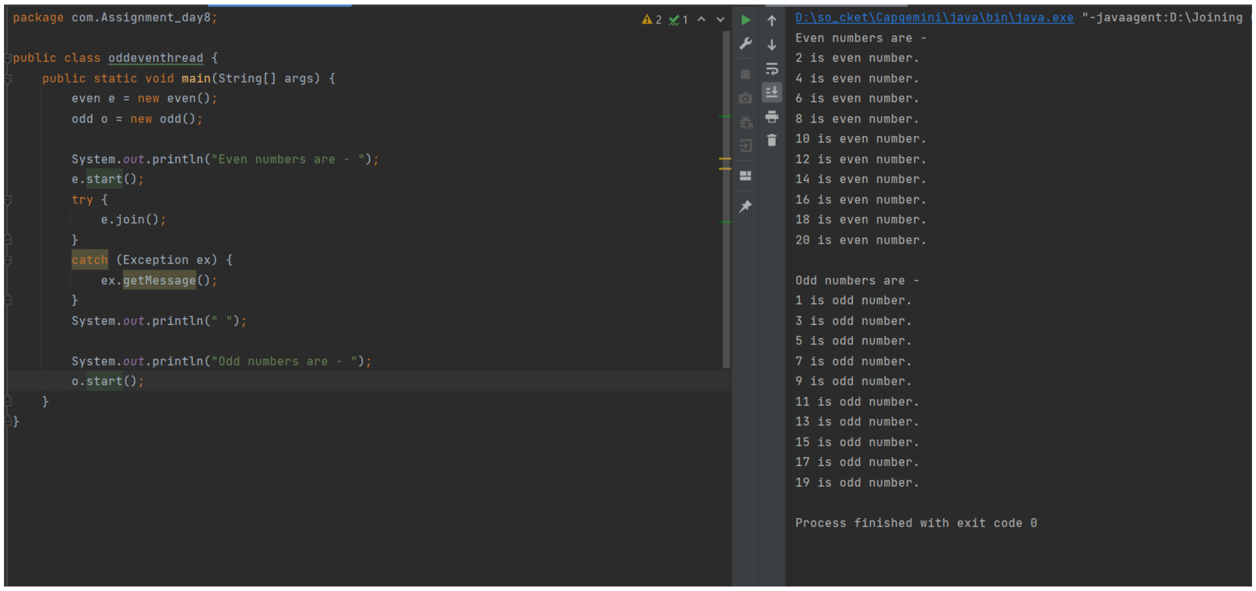
```
D:\so_cket\Capgemini\java\bin\java.exe "-javaagent:D:\Joining Capgemini\PluralSight\Ja
Random number between [-100, 100] are - [-88, 66, 19, -42, 78, 90, -28, 70, -23, -39]
-88 - Number is Negative and Even.
70 - Number is Positive and Even.
-23 - Number is Negative and odd
66 - Number is Positive and Even.
19 - Number is Positive and odd
90 - Number is Positive and Even.
-42 - Number is Negative and Even.
-28 - Number is Negative and Even.
78 - Number is Positive and Even.
-39 - Number is Negative and odd

Process finished with exit code 0
```

Q3: Create two threads, one thread to display all even numbers between 1 & 20, another to display odd numbers between 1 & 20. Note: Display all even numbers followed by odd numbers Hint: use join

Solution –



```java
2 usages
class even extends Thread {
    @Override
    public void run() {
        for (int i = 1; i <= 20; i++) {
            if(i % 2== 0) {
                System.out.println(i + " is even number.");
            }
        }
    }
}
2 usages
class odd extends Thread {
    @Override
    public void run() {
        for (int i = 1; i <= 20; i++) {
            if(i % 2 != 0) {
                System.out.println(i + " is odd number.");
            }
        }
    }
}
```



```java
package com.Assignment_day8;

public class oddeventhread {
    public static void main(String[] args) {
        even e = new even();
        odd o = new odd();

        System.out.println("Even numbers are - ");
        e.start();
        try {
            e.join();
        }
        catch (Exception ex) {
            ex.getMessage();
        }
        System.out.println(" ");

        System.out.println("Odd numbers are - ");
        o.start();
    }
}
```

```
D:\so_cket\Capgemini\java\bin\java.exe "-javaagent:D:\Joining
Even numbers are -
2 is even number.
4 is even number.
6 is even number.
8 is even number.
10 is even number.
12 is even number.
14 is even number.
16 is even number.
18 is even number.
20 is even number.

Odd numbers are -
1 is odd number.
3 is odd number.
5 is odd number.
7 is odd number.
9 is odd number.
11 is odd number.
13 is odd number.
15 is odd number.
17 is odd number.
19 is odd number.

Process finished with exit code 0
```

Q4: I am providing the below code snippet that you need to write in a simple java project. So follow below steps and then attempt to solve the problem statement.

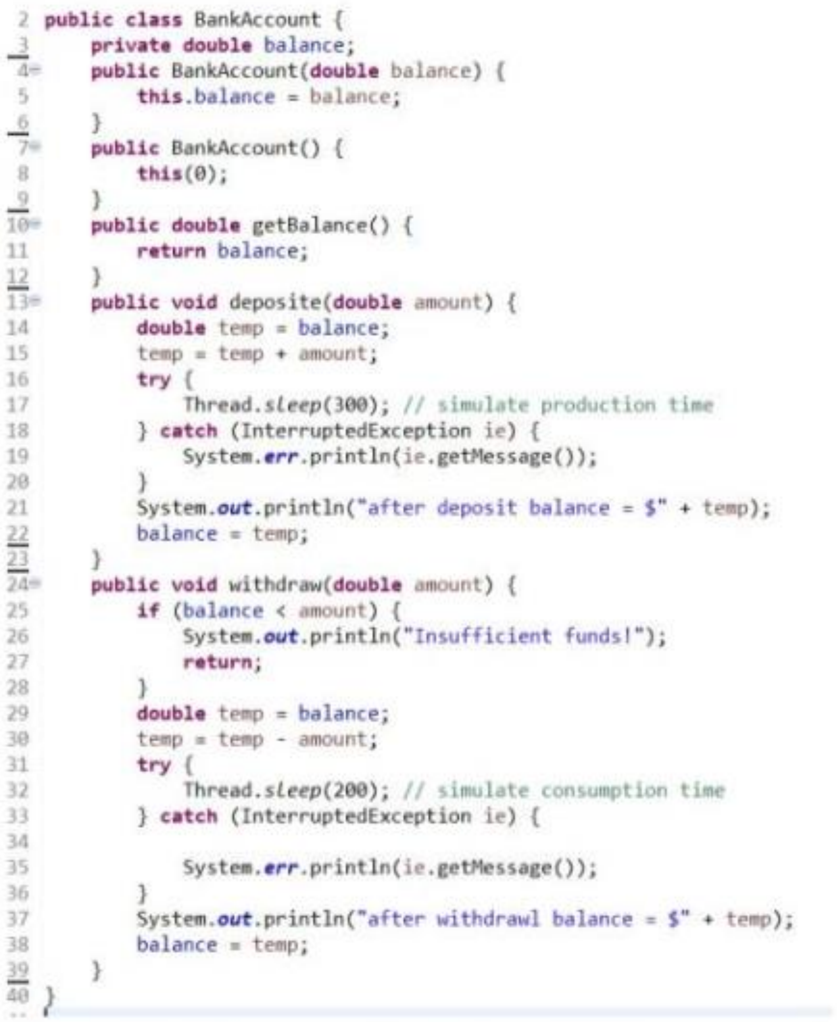
Steps

Step1: Create a core java project with the name as **AdvanceThreadDemo**

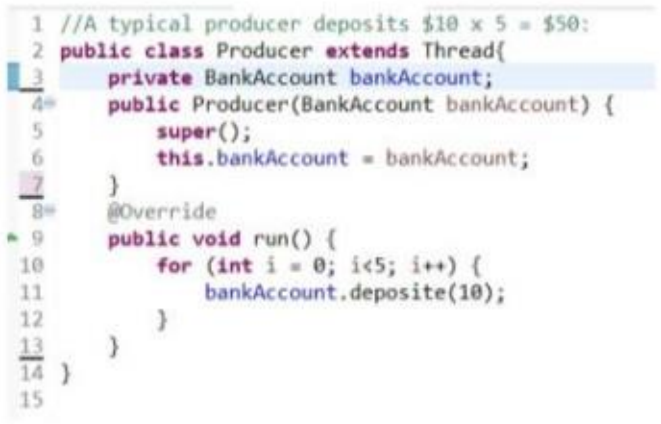
Step2: Create 4 classes separately in src directory. Code is as below.

Classes

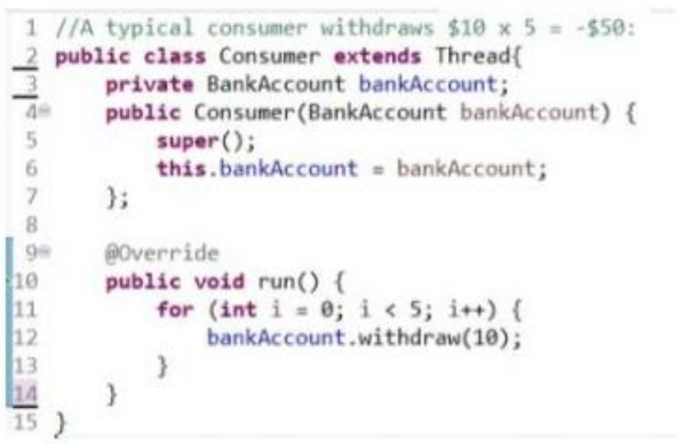
1. BankAccount
2. Producer extends Thread
3. Consumer extends Thread
4. Bank --------- with main method



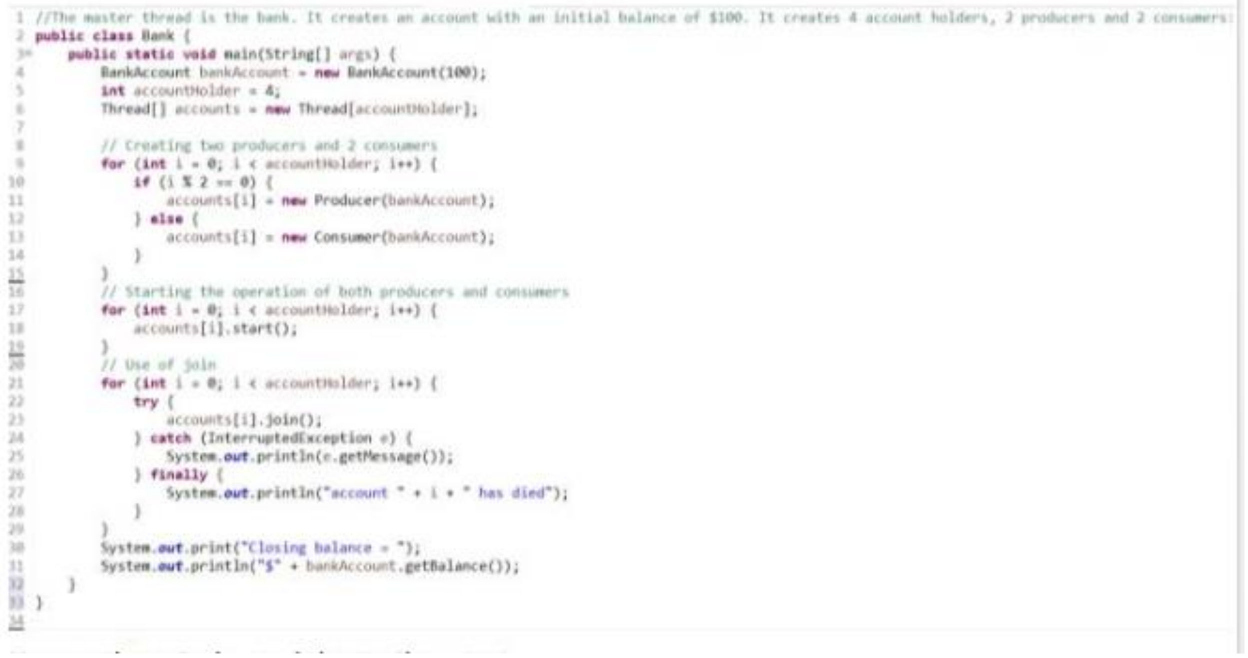
```java
public class BankAccount {
    private double balance;
    public BankAccount(double balance) {
        this.balance = balance;
    }
    public BankAccount() {
        this(0);
    }
    public double getBalance() {
        return balance;
    }
    public void deposite(double amount) {
        double temp = balance;
        temp = temp + amount;
        try {
            Thread.sleep(300); // simulate production time
        } catch (InterruptedException ie) {
            System.err.println(ie.getMessage());
        }
        System.out.println("after deposit balance = $" + temp);
        balance = temp;
    }
    public void withdraw(double amount) {
        if (balance < amount) {
            System.out.println("Insufficient funds!");
            return;
        }
        double temp = balance;
        temp = temp - amount;
        try {
            Thread.sleep(200); // simulate consumption time
        } catch (InterruptedException ie) {

            System.err.println(ie.getMessage());
        }
        System.out.println("after withdrawl balance = $" + temp);
        balance = temp;
    }
}
```



```java
//A typical producer deposits $10 x 5 = $50:
public class Producer extends Thread{
    private BankAccount bankAccount;
    public Producer(BankAccount bankAccount) {
        super();
        this.bankAccount = bankAccount;
    }
    @Override
    public void run() {
        for (int i = 0; i<5; i++) {
            bankAccount.deposite(10);
        }
    }
}
```

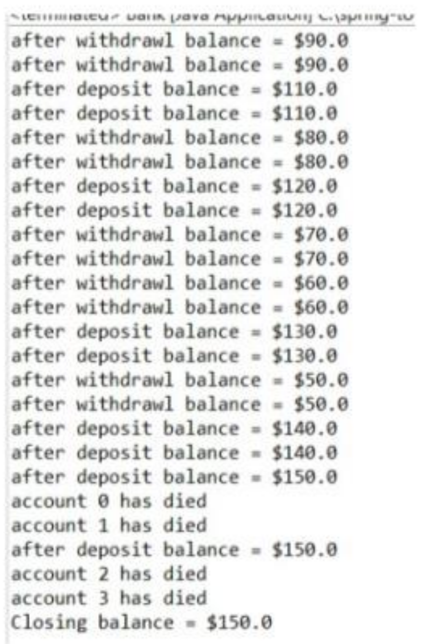


```java
//A typical consumer withdraws $10 x 5 = -$50:
public class Consumer extends Thread{
    private BankAccount bankAccount;
    public Consumer(BankAccount bankAccount) {
        super();
        this.bankAccount = bankAccount;
    };

    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            bankAccount.withdraw(10);
        }
    }
}
```



```java
//The master thread is the bank. It creates an account with an initial balance of $100. It creates 4 account holders, 2 producers and 2 consumers:
public class Bank {
    public static void main(String[] args) {
        BankAccount bankAccount = new BankAccount(100);
        int accountHolder = 4;
        Thread[] accounts = new Thread[accountHolder];

        // Creating two producers and 2 consumers
        for (int i = 0; i < accountHolder; i++) {
            if (i % 2 == 0) {
                accounts[i] = new Producer(bankAccount);
            } else {
                accounts[i] = new Consumer(bankAccount);
            }
        }
        // Starting the operation of both producers and consumers
        for (int i = 0; i < accountHolder; i++) {
            accounts[i].start();
        }
        // Use of join
        for (int i = 0; i < accountHolder; i++) {
            try {
                accounts[i].join();
            } catch (InterruptedException e) {
                System.out.println(e.getMessage());
            } finally {
                System.out.println("account " + i + " has died");
            }
        }
        System.out.print("Closing balance = ");
        System.out.println("$" + bankAccount.getBalance());
    }
}
```

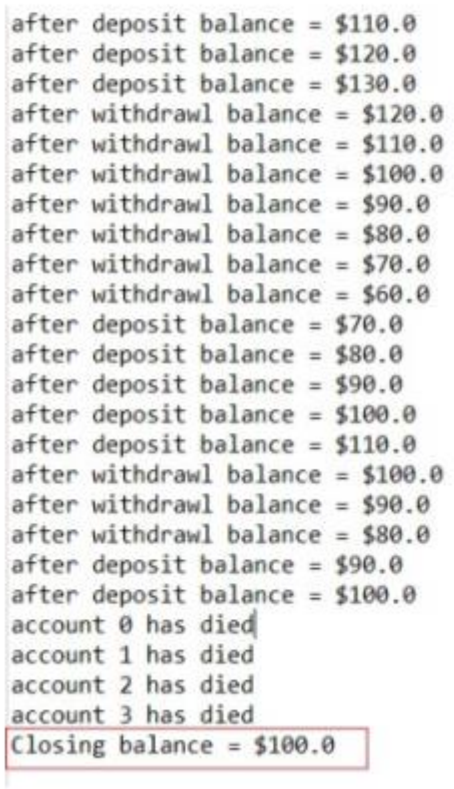
Now run the main class and observe the output.



```
<terminated> Bank [Java Application] C:\spring-to
after withdrawl balance = $90.0
after withdrawl balance = $90.0
after deposit balance = $110.0
after deposit balance = $110.0
after withdrawl balance = $80.0
after withdrawl balance = $80.0
after deposit balance = $120.0
after deposit balance = $120.0
after withdrawl balance = $70.0
after withdrawl balance = $70.0
after withdrawl balance = $60.0
after withdrawl balance = $60.0
after deposit balance = $130.0
after deposit balance = $130.0
after withdrawl balance = $50.0
after withdrawl balance = $50.0
after deposit balance = $140.0
after deposit balance = $140.0
after deposit balance = $150.0
account 0 has died
account 1 has died
after deposit balance = $150.0
account 2 has died
account 3 has died
Closing balance = $150.0
```

Problem Statement:
Notice that the “insufficient funds” message never appeared. That means both consumers successfully withdrew $10 five times each for a total of -$100. Similarly, both producers successfully deposited $10 five times each for a total of $100. The initial balance was $100, so the closing balance should have been:
balance=$100+$100-$100 =
$100 somehow, the bank lost
$50.
It seems pretty clear what went wrong. Look at the first three lines of output:
after withdrawl balance
= $90.0 after withdrawl
balance = $90.0
after deposit balance = $110.0

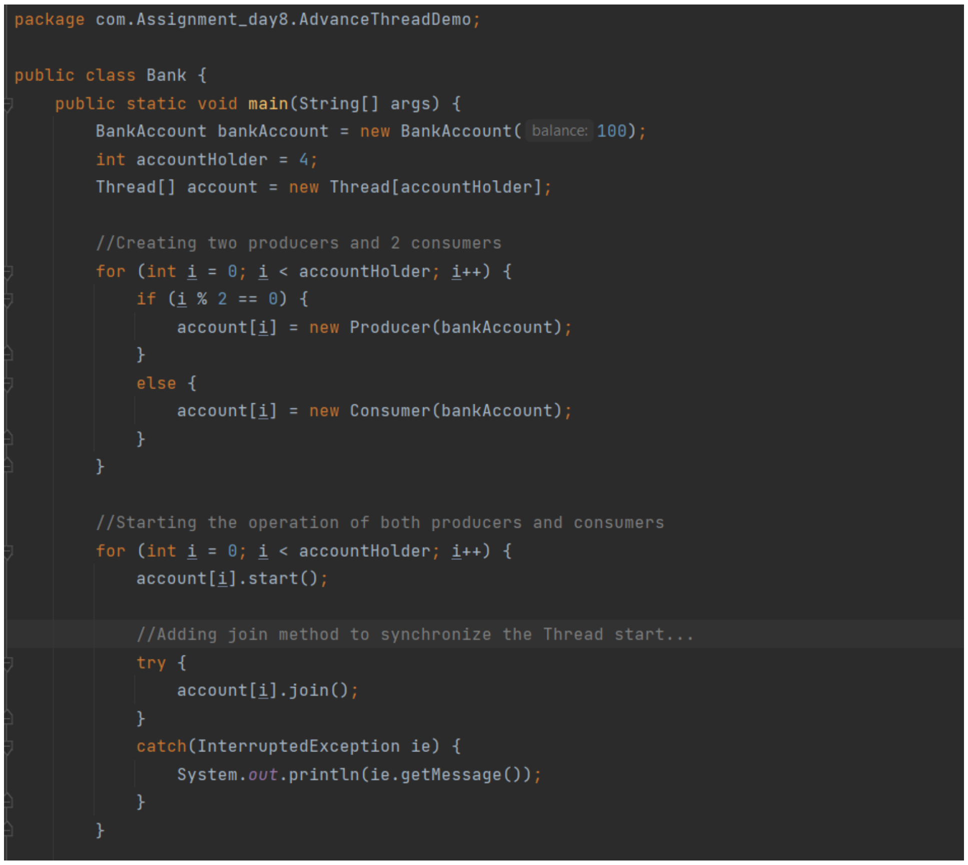
Apparently, when the first producer was in the middle of making a deposit, he was interrupted by a consumer who quicly withdrew $10, leaving the balance at $90. Unfortunately, the last time the producer checked, the balance was $100, so after adding an additional $10, the producer now believes the balance is $110. In other words, the bank lost track of the $10 withdrawn by the consumer.
Make the appropriate changes in your app so that you must get the output as below.



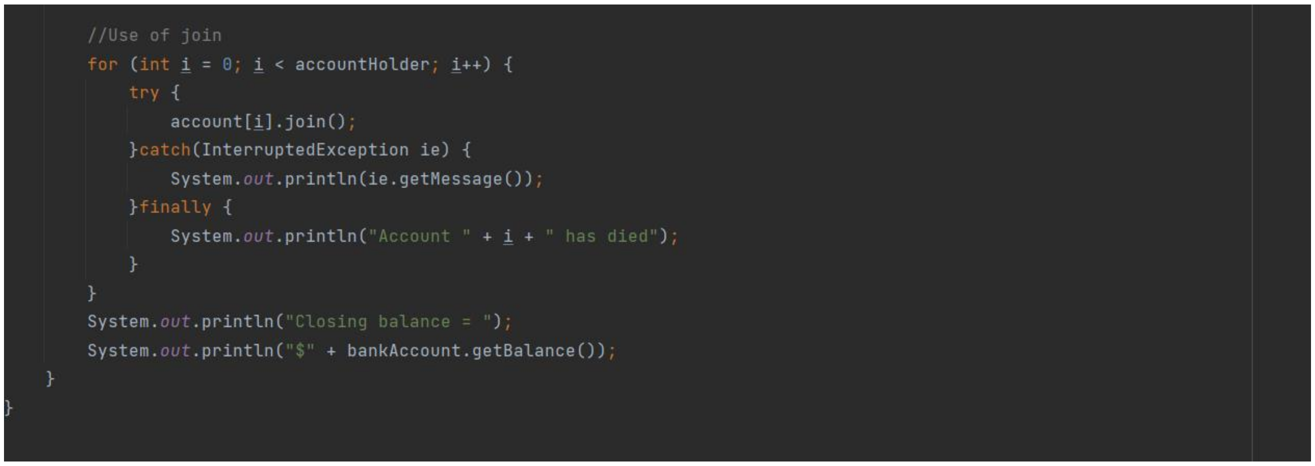
```
after deposit balance = $110.0
after deposit balance = $120.0
after deposit balance = $130.0
after withdrawl balance = $120.0
after withdrawl balance = $110.0
after withdrawl balance = $100.0
after withdrawl balance = $90.0
after withdrawl balance = $80.0
after withdrawl balance = $70.0
after withdrawl balance = $60.0
after deposit balance = $70.0
after deposit balance = $80.0
after deposit balance = $90.0
after deposit balance = $100.0
after deposit balance = $110.0
after withdrawl balance = $100.0
after withdrawl balance = $90.0
after withdrawl balance = $80.0
after deposit balance = $90.0
after deposit balance = $100.0
account 0 has died
account 1 has died
account 2 has died
account 3 has died
Closing balance = $100.0
```

Note : the sequence of deposit or withdraw can be changed. But finally the closing balance must be $100. Because same number of withdraw and deposits are happening with same amount in the system, so ideally closing balance must be initial balance only i.e. $100;

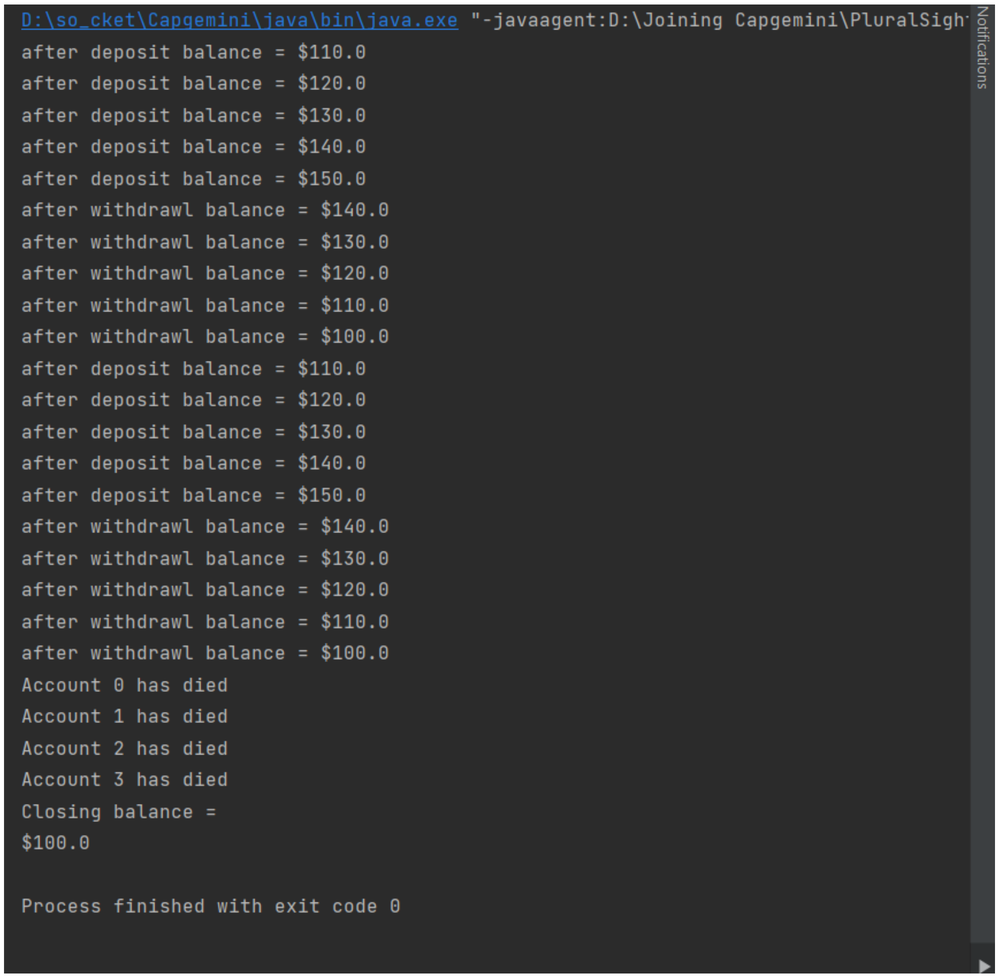
Solution –



```java
package com.Assignment_day8.AdvanceThreadDemo;

public class Bank {
    public static void main(String[] args) {
        BankAccount bankAccount = new BankAccount( balance: 100);
        int accountHolder = 4;
        Thread[] account = new Thread[accountHolder];

        //Creating two producers and 2 consumers
        for (int i = 0; i < accountHolder; i++) {
            if (i % 2 == 0) {
                account[i] = new Producer(bankAccount);
            }
            else {
                account[i] = new Consumer(bankAccount);
            }
        }

        //Starting the operation of both producers and consumers
        for (int i = 0; i < accountHolder; i++) {
            account[i].start();

            //Adding join method to synchronize the Thread start...
            try {
                account[i].join();
            }
            catch(InterruptedException ie) {
                System.out.println(ie.getMessage());
            }
        }
```



```java
        //Use of join
        for (int i = 0; i < accountHolder; i++) {
            try {
                account[i].join();
            }catch(InterruptedException ie) {
                System.out.println(ie.getMessage());
            }finally {
                System.out.println("Account " + i + " has died");
            }
        }
        System.out.println("Closing balance = ");
        System.out.println("$" + bankAccount.getBalance());
    }
}
```



```
D:\so_cket\Capgemini\java\bin\java.exe "-javaagent:D:\Joining Capgemini\PluralSigh
after deposit balance = $110.0
after deposit balance = $120.0
after deposit balance = $130.0
after deposit balance = $140.0
after deposit balance = $150.0
after withdrawl balance = $140.0
after withdrawl balance = $130.0
after withdrawl balance = $120.0
after withdrawl balance = $110.0
after withdrawl balance = $100.0
after deposit balance = $110.0
after deposit balance = $120.0
after deposit balance = $130.0
after deposit balance = $140.0
after deposit balance = $150.0
after withdrawl balance = $140.0
after withdrawl balance = $130.0
after withdrawl balance = $120.0
after withdrawl balance = $110.0
after withdrawl balance = $100.0
Account 0 has died
Account 1 has died
Account 2 has died
Account 3 has died
Closing balance =
$100.0

Process finished with exit code 0
```