BSCCS2005: Graded with Solutions
Week 10

1. Consider the program given below.

```
class Counter implements Runnable{
    boolean stopRequested = false;
    long count = 0;
    public void run() {
        while (!stopRequested) {
            count++;
            if (count==1000000) {
                stopRequested = true;
            }
        }
    }
    public void setStop(boolean stop){
        stopRequested = stop;
    }
    public long getCount(){
        return count;
    }
}
public class ThreadEx {
    public static void main(String[] args) throws InterruptedException {
        Counter ctr = new Counter();
        Thread backgroundThread = new Thread(ctr);
        backgroundThread.start();
        Thread.sleep(1);
        ctr.setStop(true);
        System.out.println(ctr.getCount());
    }
}
```

What will the output be?

- ○ 0
- ○ 1000000
- ✓ Some whole number between 1 and 1000000
- ○ 999999

2. A teacher has to find the maximum marks in a subject for a class of 50 students. The teacher splits the class into two equal groups, and compute the maximum in each group simultaneously, and then find the maximum of the two. Based on this information, consider the Java code given below.

```
import java.util.*;

class SumCompute implements Runnable{
    ArrayList<Integer> half_batch = new ArrayList();
    int max;
    public SumCompute(ArrayList<Integer> hb) {
        half_batch = (ArrayList<Integer>)hb.clone();
        max = -1;
    }
    public void run() {
        max = Collections.max(half_batch);
    }
    public int getMax() {
        return max;
    }
}
public class MaxMarks {
    public static void main(String[] args) {
        int max = -1;

        //Accept the marks into
        //two ArrayList<Integer> objects batch1 and batch2

        SumCompute sc1 = new SumCompute(batch1);
        SumCompute sc2 = new SumCompute(batch2);
        Thread t1 = new Thread(sc1);
        Thread t2 = new Thread(sc2);
        t1.start();
        t2.start();
        //Line 1
        if (sc1.getMax() >= sc2.getMax()) {
            max = sc1.getMax();
        }
        else {
            max = sc2.getMax();
        }
        System.out.println(max);
    }
}
```

What should be added at `Line 1` so that the program will always give the correct result?

- ○ Thread.sleep(1000);
- ✓ `while(sc1.getMax() == -1 || sc2.getMax() == -1) {}`
- ○ `while(sc1.getMax() == -1 && sc2.getMax() == -1) {}`
- ○ No code is required at `Line 1`. The code will always generate the correct output.
- ○ No line of code at `Line 1` can ensure that this code will always generate the correct ouput.

3. Match the following.

a) Test-and-set	1) One thread is locked out permanently
b) Starvation	2) Access control to shared resources using counters
c) Semaphore	3) Check for a value and set it in an atomic step
d) Monitors	4) Attaches synchronization control to data

Choose the option that gives the most appropriate matching.

- ○ a-1, b-2, c-3, d-4
- [✓ a-3, b-1, c-2, d-4]
- ○ a-3, b-2, c-4, d-1
- ○ a-3, b-4, c-2, d-1

Bank account A has ₹1000 as balance, and bank account B has ₹500 as balance. The Java program given below transfers a sum of ₹200 from account A to account B. Based on this code, answer questions 4 and 5.

```java
class Account implements Runnable{
    private int balance;
    final int TRANSFER_AMT = 200;
    public int getBalance() {
        return balance;
    }
    public void setBalance(int amt) {
        balance = amt;
    }
    public int getAmountTransfer() {
        return TRANSFER_AMT;
    }
    public void run() {
    }
}
class SourceAccount extends Account{
    public SourceAccount(int amt) {
        setBalance(amt);
    }
    public void run() {
        setBalance(getBalance()-getAmountTransfer());
    }
}
class TargetAccount extends Account{
    public TargetAccount(int amt) {
        setBalance(amt);
    }
    public void run() {
        setBalance(getBalance()+getAmountTransfer());
    }
}
public class BankTransfer {
    public void transfer(SourceAccount src, TargetAccount tgt) {
        Thread t1 = new Thread(src);
        Thread t2 = new Thread(tgt);
        t1.start();
        t2.start();
    }
    public static void main(String[] args) {
        Account A = new SourceAccount(1000);
41          Account B = new TargetAccount(500);
42          BankTransfer bt = new BankTransfer();
43          bt.transfer((SourceAccount)A, (TargetAccount)B);
44          System.out.println(A.getBalance()+" "+B.getBalance());
45      }
46  }
```

4. Assume that the program terminates normally. Immediately after the execution of which of the following line/s of code, are we guaranteed to see the sum of balances in account A and account B to be equal to ₹1500?

   ○ Line 36

   ○ Line 37

   ○ Line 44

   [✓ None of the above]

5. From among the given options, choose the correct options for the method `run()` of both `SourceAccount` and `TargetAccount` such that at the end of Line 44, we see that the sum of the balances of account A and account B is ₹1500. Assume that P(S) and V(S) are atomic functions that acquire and release, respectively, the control of execution.

[✓]

```
//SourceAccount                         //TargetAccount
public void run() {                     public void run() {
    P(S);                                   P(S);
    setBalance(getBalance()                 setBalance(getBalance()
        -getAmountTransfer());                      +getAmountTransfer());
    V(S);                                   V(S);
}                                       }
```

○

```
//SourceAccount                         //TargetAccount
public void run() {                     public void run() {
    int b = getBalance();                   int b = getBalance();
    int a = getAmountTransfer();            int a = getAmountTransfer();
    P(S);                                   P(S);
    setBalance(b - a);                      setBalance(b  + a);
    V(S);                                   V(S);
}                                       }
```

√
```
//SourceAccount                        //TargetAccount
public void run() {                    public void run() {
    P(S);                                  P(S);
    int b = getBalance();                  int b = getBalance();
    int a = getAmountTransfer();           int a = getAmountTransfer();
    setBalance(b - a);                     setBalance(b  + a);
    V(S);                                  V(S);
}                                      }
```

◯ P(S) and V(S) are insufficient to ensure consistency of data in the given example.

6. Since `Runnable` is a functional interface, the method `run()` can be implemented as a lambda function. Identify the appropriate option(s) to fill in the blank at `LINE 1`, which enable(s) the thread to print the words of the given string `text` on separate lines.

```
import java.util.*;
import java.util.stream.*;
public class FClass{
    public static void main(String[] args){
        var text = "solutions based on test-and-set are low level and
                            prone to programming errors";
        List<String> words = List.of(text.split(" "));
        Thread t = new Thread(__________________);    //LINE 1
        t.start();
    }
}
```

- [✓ `() -> words.stream().forEach(System.out::println)`]
- ◯
```
(words) -> words.stream()
                .forEach(System.out::println)
```
- ◯
```
words -> { for(int i = 0; i < words.length; i++)
                System.out.println(words.get(i));
         }
```
- [✓
```
() -> { for(String s : words)
                System.out.println(s);
         }
```
]

Solution: Since the `run()` method takes no argument as input, option-2 and -3 are wrong.

7. Consider the code given below.

```
class ProdList{
    private String[] items = {"pen", "pencil", "paper"};
    public void show(){
        for(String s : items)
            System.out.print(s + " ");
    }
}
class PrlProdList _____________________________ {    //LINE 1
    public void run(){
        show();
    }
}
public class FClass{
    public static void main(String[] args){
        Thread t1 = new Thread(new PrlProdList());
        Thread t2 = new Thread(new PrlProdList());
        t1.start();
        t2.start();
    }
}
```

Identify the appropriate option to fill in the blank at `LINE 1`, such that the threads `t1` and `t2` execute the `show()` function in parallel.

- ○ `implements Runnable`
- ○ `extends ProdList`
- ○ `extends ProdList, Thread`
- [✓ `extends ProdList implements Runnable`]

Solution: Since, `PrlProdList` uses the `show()` method in `ProdList` without any object reference, it must inherit `ProdList` class. `PrlProdList` class object is passed as argument in `Thread` connstructor, which must be an `Runnable` object. Thus, `PrlProdList` must implements `Runnable`.

8. Consider the code given below.

```
class PrlCls1 extends Thread{
    public void run(){
        for(int i = 1; i <= 10; i++){
            System.out.print(i + " ");
            try{
                Thread.sleep(1000);
            }
            catch(InterruptedException e){}
        }
    }
}
class PrlCls2 extends Thread{
    public void run(){
        for(int i = 11; i <= 20; i++){
            System.out.print(i + " ");
            try{
                Thread.sleep(1000);
            }
            catch(InterruptedException e){}
        }
    }
}
public class Main{
    public static void main(String[] args){
        Thread t1 = new PrlCls1();
        Thread t2 = new PrlCls2();
        t1.run();
        t2.start();
    }
}
```

Choose the correct option regarding the code.

- [✓ It always prints 1 to 10 first, followed by 11 to 20.]
- ◯ It always prints 11 to 20 first, followed by 1 to 10.
- ◯ It may prints 11 to 20 and 1 to 10 in an interleaved manner.
- ◯ It generates a compiler error because the method `run()` cannot be invoked explicitly.

Solution: Since, the `main()` executes the `run()` method first and then starts the thread `t`, it always prints 1 to 10 first, followed by 11 to 20.

9. Suppose there are two threads T1 and T2. T1 will execute a code to deposit ₹100 to an account and thread T2 will execute a code to withdraw ₹200 from the same account. Two semaphores `A` and `B` are used for attaining mutual exclusion in this process. It is given that initially the values of both `A` and `B` are 1 and the initial balance of the account is ₹500.
Consider the following code executed by T1 and T2 threads concurrently.
T1:

```
Line No.   Instructions
1.           P(A)
2.           temp1 = balance
3.           temp1 = temp1 + 100
4.           P(B)
5.           balance = temp1
6.           V(B)
7.           V(A)
```

T2:

```
Line No.   Instructions
1.           P(B)
2.           temp2 = balance
3.           temp2 = temp2 - 200
4.           P(A)
5.           balance = temp2
6.           V(A)
7.           V(B)
```

Choose the correct option regarding the program execution.

- ◯ After execution is finished the balance can be either ₹600 or ₹400.
- ◯ After execution is finished the balance can be either ₹300 or ₹600.
- √ After execution is finished the balance will be ₹400.
- √ The program attains mutual exclusion but may run into a deadlock.

Solution:
In this example there can be three possibilities.

1. T1 starts and completes its execution and then T2 starts
2. T2 starts and completes its execution and then T1 starts

3. The program will run into a deadlock, if T1 executes Line 1 and then T2 executes Line 1.

In the first two cases balance would be Rs 400 after program completes. Therefore the account will never have a balance of Rs 600 or Rs 300 i.e. in other words if deadlock does not occur then it can be said, none of the updates are lost. One should be careful while using semaphores

10. Consider the following code.

```
class ATM implements Runnable{
  private String name;
  static double balance = 100.0;
  public ATM(String name){
    this.name = name;
  }
  public void deposit(double a){
    balance = balance + a;
  }
  public void withdraw(double a){
    balance = balance - a;
  }
  public void displayBalance(){
    System.out.println(balance);
  }
  public void run(){
    if(name.equals("deposit")){
        deposit(100);
        displayBalance();
    }
    if(name.equals("withdraw")){
        withdraw(200);
        displayBalance();
    }
  }
}
public class cardUser {
  public static void main(String[] args) {
    ATM m1 = new ATM("deposit");
    ATM m2 = new ATM("withdraw");
    Thread t1 = new Thread(m1);
    Thread t2 = new Thread(m2);
    t1.start();
    t2.start();

  }
}
```

Select the most appropriate option.

◯ The program may generate the output:
```
-100.0
0.0
```

- ○ The program may generate the output:

  ```
  0.0
  0.0
  ```

- ○ The program may generate the output:

  ```
  200.0
  0.0
  ```

- [✓ All the above]