Chapter 8 – Threads

Question 1:

Which one statement below is true concerning the following code?

```
1. class Hevron extends java.util.Vector implements Runnable
2. {
3.          public void run(int counter)
4.          {
3.          System.out.println("in run() method: " + counter);
5.          }
6. }
7.
8. class HevronTest
9. {
10.          public static void main(String args[])
11.          {
12.                Hevron g = new Hevron();
13.               Thread t = new Thread(g);
14.               t.start();
15.          }
16. }
```

() A) There will be a compiler error because class Hevron doesn't implement the Runnable interface.

() B) There will be a compiler error at line 13 because you must pass two parameters to the constructor of a Thread.

() C) The code will compile correctly but will crash with an exception at line 13.

() D) The code will compile correctly but will crash with an exception at line 14.

() E) The code will execute without throwing any exception.

Question 2:

Regarding the following code:

```
1.  class Jerusalem extends Thread
2.  {
3.      public void run()
4.      {
5.          System.out.print("Hello");
6.          yield();
7.          System.out.println(" Israel");
8.      }
9.      public static void main(String args[])
10.     {
11.         Thread t = new Jerusalem();
12.         t.start();
```

```
13.                 }
14.         }
```

() A) The output will be: Hello Israel

() B) The output will be: Hello

() C) The output will be: Israel

() D) The compilation will fail

() E) The compilation will succeed but on execution nothing will be printed out

Question 3:

Regarding the following code:

```
1.  class Jerusalem extends Thread
2.  {
3.          public void run()
4.          {
5.                  System.out.print("Hello");
6.                  yield();
7.                  resume();
8.                  System.out.println(" Israel");
9.          }
10.         public static void main(String args[])
11.         {
```

```
12.             Thread t = new Jerusalem();
13.             t.start();
14.         }
15. }
```

() A) The output will be: Hello Israel

() B) The output will be Hello

() C) The output will be Israel

() D) The compilation will fail

() E) The compilation will succeed but on execution nothing will be printed out

Question 4:

Regarding the following code:

```
1.  class Jerusalem extends Thread
2.  {
3.          public void run()
4.          {
5.                  System.out.print("Hello");
6.                  suspend();
7.                  System.out.println(" Israel");
8.          }
9.          public static void main(String args[])
```

```
10.         {
11.                 Thread t = new Jerusalem();
12.                 t.start();
13.         }
14.  }
```

() A) The output will be: Hello Israel

() B) The output will be Hello

() C) The output will be Israel

() D) The compilation will fail

() E) The compilation will succeed but on execution nothing will be printed out

Question 5:

Regarding the following code:

```
1.    class Jerusalem extends Thread
2.    {
3.          public void run()
4.          {
5.                System.out.print("Hello");
6.                suspend();
7.                resume();
8.                System.out.println(" Israel");
9.          }
10.         public static void main(String args[])
```

```
11.         {
12.                 Thread t = new Jerusalem();
13.                 t.start();
14.         }
15. }
```

() A) The output will be: Hello Israel

() B) The output will be Hello

() C) The output will be Israel

() D) The compilation will fail

() E) The compilation will succeed but on execution nothing will be printed out

Question 6:

```
1.  class Try55 extends Thread
2.  {
3.                  String str;
4.                  Try55(String str)
5.          {
6.                  this.str = str;
7.          }
8.          public void run()
9.          {
10.                 while (true) {System.out.println(str);
```

```
11.         }
12.         public static void main(String args[])
13.         {
14.                 Thread t1,t2;
15.                 t1 = new Try55(“t1”);
16.                 t2 = new Try55(“t2”);
17.                 t1.setPriority(8);
18.                 t2.setPriority(8);
19.                 t1.start();
20.                 t2.start();
21.         }
22. }
```

[] A) Assuming that the application is running on Windows platform The output will inclue t1 only.

[] B) Assuming that the application is running on Windows platform The output will inclue t2 only.

[] C) Assuming that the application is running on Windows platform The output will inclue both t1 and t2 only.

[] D) Assuming that the application is running on Solaris platform The output will inclue t1 only.

[] E) Assuming that the application is running on Solaris platform The output will inclue t2 only.

[] F) Assuming that the application is running on Solaris platform The output will inclue both t1 and t2 only.

Question 7:

True or False:

Using the notify method it is possible to specify a specific thread to get notified and by that moving it from the waiting pool to the lock (ready) state.

() A) true

() B) false

Question 8:

True or False:

A popular mechanism for handling critical sections is called a monitor. A java class as a whole isn't a monitor in the conventional sense unless all of its nonprivate methods marked as synchronized and all its variables are marked as private.

() A) true

() B) false

Question 9:

True or False:

A thread that was sent to sleep using the sleep method for 100 milliseconds will resume its action exactly after 100 milliseconds.

() A) true

() B) false

Question 10:

True or False:

It is possible to notify a specific thread from a group of waiting threads.

() A) true

() B) false

Question 11

True or False:

The following two lines were taken out from the run method that was defined in a class that extends the Thread class. It can be said that the second line has no effect.

.

.

.

yield();

resume();

.

.

.

() A) true

() B) false

Question 12

True or False:

Calling the suspend() method on an other thread will make it un-runnable for an indefinite period of time.

() A) true

() B) false

Question 13:

True or False:

A thread that was stopped by calling the sleep method for a 2000 milliseconds will resume its running in exactly 2 seconds.

() A) true

() B) false

Question 14:

True or False:

The following code will compile successfully.

```
class Walla extends Yahoo implements Runnable
{
        public void run(int num)
        {
                for(int index=0; index<num; index++)
                {
                        System.out.println("index="+index);
                }
        }

        public static void main(String args[])
        {
                Walla w = new Walla();
                Thread t = new Thread(w);
                t.start();
        }
}
```

() A) true

() B) false

Question 15:

True or False:

The following code will compile successfully.

```
class Walla extends Yahoo implements Runnable
{
        public void run()
        {
                for(int index=0; index<num; index++)
                {
                        System.out.println("index="+index);
                }
        }

        public static void main(String args[])
        {
                Walla w = new Walla();
                Thread t = new Thread(w);
                t.start(1200);
        }
}
```

() A) true

() B) false

Question 16:

Which of the following sentences (one or more) is true ?

[] A) When a thread becomes dead the Thread object continues to exist and it is still possible to access its data.

[] B) It isn't possible to make a dead thread run again.

[] C) It is possible to make a dead thread run again.

[] D) When a thread finishes its run method it is possible to call the start method again.

[] E) The stop() method (declared in the Thread class) is deprecated since jdk1.2 because it can cause data corruption or deadlock if it is used on a thread which is in a critical section of the code.

Question 17:

Which of the following sentences (one or more) is true ?

[] A) If there are more than one thread that is waiting to be chosen by the scheduler and run (by the CPU) there is no guarantee that the thread chosen will be the one that has been waiting the longest.

[] B) Programs that rely on manipulating thread priorities might not run consistently on every platform.

[] C) When using the setPriority method in order to give a priority to a thread it is recommended to use the constants that the Thread class declares: MAX_PRIORITY,

MIN_PRIORITY and NORM_PRIORITY.

[] D) The suspend() and the resume() methods are deprecated because using them might cause a dead lock situation.

[] E) The sleep method and the yield method are both static, and both of them operate on the currently executed thread.

Question 18:

Given the code below,

```
1.  public class SynchronizedThreadExample
2.  {
3.      public static void main(String args[])
4.      {
5.          MyCashier cashier = new MyCashier();
6.          Depositer dep1 = new Depositer(cashier,1);
7.          Depositer dep2 = new Depositer(cashier,2);
8.          Withdrawer with1 = new Withdrawer(cashier,1);
9.          Withdrawer with2 = new Withdrawer(cashier,2);
10.         Thread dep1Th = new Thread(dep1);
11.         Thread dep2Th = new Thread(dep2);
12.         Thread with1Th = new Thread(with1);
13.         Thread with2Th = new Thread(with2);
14.         dep1Th.start();
```

```
15.             dep2Th.start();
16.             with1Th.start();
17.             with2Th.start();
18.         }
19. }
20.
21. class MyCashier
22. {
23.         private int sum;
24.         public int getSum()
25.         {
26.             return sum;
27.         }
28.         public synchronized void withdraw(int sumToWithdraw)
29.         {
30.             System.out.println("BINGO");
31.             while (sum-sumToWithdraw<1000)
32.             {
33.                 try
34.                 {
35.                     wait();
36.                 }
37.                 catch(InterruptedException e) {}
38.             }
39.             sum-=sumToWithdraw;
```

```
40.             System.out.println(sumToWithdraw +
41.                     " was withdrawed. The balance now is " + sum);
42.         }
43.         public synchronized void deposit(int sumToDeposit)
44.         {
45.             System.out.println("TANGA");
46.             sum+=sumToDeposit;
47.             if (sum>=1000)
48.                 notify();
49.             System.out.println(sumToDeposit +
50.                     " was deposit. The balance now is " + sum);
51.         }
52. }
53.
54. class Depositer implements Runnable
55. {
56.         private MyCashier cash;
57.         private int id;
58.         public Depositer (MyCashier cashier, int idNum)
59.         {
60.             cash = cashier;
61.             id = idNum;
62.         }
63.         public void run()
64.         {
```

```
65.             int sumOfMoney;
66.             for (int i=0; i<2; i++)
67.             {
68.                     sumOfMoney = (int)(Math.random()*100);
69.                     cash.deposit(sumOfMoney);
70.                     System.out.println("depositer number " + id +
71.                             " has deposited " + sumOfMoney);
72.                     try
73.                     {
74.                             Thread.sleep((int)(Math.random()*5000));
75.                     }
76.                     catch(Exception e) {}
77.             }
78.             for (int i=0; i<2; i++)
79.             {
80.                     sumOfMoney = (int)(Math.random()*4000);
81.                     cash.deposit(sumOfMoney);
82.                     System.out.println("depositer number " + id +
83.                             "has deposited " + sumOfMoney);
84.                     try
85.                     {
86.                             Thread.sleep((int)(Math.random()*5000));
87.                     }
88.                     catch(Exception e) {}
89.             }
```

```
90.         }
91. }
92.
93. class Withdrawer implements Runnable
94. {
95.         private MyCashier cash;
96.         private int id;
97.         public Withdrawer (MyCashier cashier, int idNum)
98.         {
99.                 cash = cashier;
100.                        id = idNum;
101.                }
102.                public void run()
103.                {
104.                        int sumToWithdraw;
105.                        for (int i=0; i<4; i++)
106.                        {
107.                                sumToWithdraw = (int)(Math.random()*1000);
108.                                cash.withdraw(sumToWithdraw);
109.                                System.out.println("withdrawer number " + id +
110.                                        " has withdrawed " + sumToWithdraw);
111.                               try
112.                                {
113.                                        Thread.sleep((int)(Math.random()*5000));
114.                                }
```

```
115.                                    catch(InterruptedException e) {}
116.                                }
117.                            }
118.                    }
```

the output will include the word TANGA:

() A) x 8

() B) x 7

() C) x 6

() D) x 5

() E) Can’t be predicted for sure. The number of times that the word TANGA will be printed out depends on the platform.

Question 19:

Given the code below,

```
1.  public class SynchronizedThreadExample
2.  {
3.          public static void main(String args[])
4.          {
5.                  MyCashier cashier = new MyCashier();
6.                  Depositer dep1 = new Depositer(cashier,1);
7.                  Depositer dep2 = new Depositer(cashier,2);
8.                  Withdrawer with1 = new Withdrawer(cashier,1);
9.                  Withdrawer with2 = new Withdrawer(cashier,2);
10.                Thread dep1Th = new Thread(dep1);
11.                Thread dep2Th = new Thread(dep2);
12.                Thread with1Th = new Thread(with1);
13.                Thread with2Th = new Thread(with2);
14.                dep1Th.start();
15.                dep2Th.start();
16.                with1Th.start();
17.                with2Th.start();
18.         }
19. }
20.
21. class MyCashier
22. {
23.         private int sum;
24.         public int getSum()
25.         {
26.                 return sum;
27.         }
```

```
28.         public synchronized void withdraw(int sumToWithdraw)
29.         {
30.                 System.out.println("BINGO");
31.                 while (sum-sumToWithdraw<1000)
32.                 {
33.                         try
34.                         {
35.                                 wait();
36.                         }
37.                         catch(InterruptedException e) {}
38.                 }
39.                 sum-=sumToWithdraw;
40.                 System.out.println(sumToWithdraw +
41.                         " was withdrawed. The balance now is " + sum);
42.         }
43.         public synchronized void deposit(int sumToDeposit)
44.         {
45.                 System.out.println("TANGA");
46.                 sum+=sumToDeposit;
47.                 if (sum>=1000)
48.                         notify();
49.                 System.out.println(sumToDeposit +
50.                         " was deposit. The balance now is " + sum);
51.         }
52. }
53.
54. class Depositer implements Runnable
55. {
56.         private MyCashier cash;
57.         private int id;
58.         public Depositer (MyCashier cashier, int idNum)
59.         {
60.                 cash = cashier;
```

```
61.             id = idNum;
62.         }
63.         public void run()
64.         {
65.             int sumOfMoney;
66.             for (int i=0; i<2; i++)
67.             {
68.                 sumOfMoney = (int)(Math.random()*100);
69.                 cash.deposit(sumOfMoney);
70.                 System.out.println("depositer number " + id +
71.                     " has deposited " + sumOfMoney);
72.                 try
73.                 {
74.                     Thread.sleep((int)(Math.random()*5000));
75.                 }
76.                 catch(Exception e) {}
77.             }
78.             for (int i=0; i<2; i++)
79.             {
80.                 sumOfMoney = (int)(Math.random()*4000);
81.                 cash.deposit(sumOfMoney);
82.                 System.out.println("depositer number " + id +
83.                     "has deposited " + sumOfMoney);
84.                 try
85.                 {
86.                     Thread.sleep((int)(Math.random()*5000));
87.                 }
88.                 catch(Exception e) {}
89.             }
90.         }
91. }
92.
93. class Withdrawer implements Runnable
```

```
94. {
95.         private MyCashier cash;
96.         private int id;
97.         public Withdrawer (MyCashier cashier, int idNum)
98.         {
99.                 cash = cashier;
100.                        id = idNum;
101.                }
102.                public void run()
103.                {
104.                        int sumToWithdraw;
105.                        for (int i=0; i<4; i++)
106.                        {
107.                                sumToWithdraw = (int)(Math.random()*1000);
108.                                cash.withdraw(sumToWithdraw);
109.                                System.out.println("withdrawer number " + id +
110.                                        " has withdrawed " + sumToWithdraw);
111.                               try
112.                               {
113.                                        Thread.sleep((int)(Math.random()*5000));
114.                               }
115.                               catch(InterruptedException e) {}
116.                       }
117.               }
118.        }
```

the output will include the word BINGO:

() A) x 8

() B) x 7

() C) x 6

() D) x 5

() E) Can't be predicted for sure. The number of times that the word TANGA will be printed out depends on the platform.

Question 20:

Which of the following sentences (one or more) is true ?

[] A) If a waiting thread, that was moved to a waiting pool as a result of calling the wait() method, receives an interrupt () call it moves immediately to the lock pool. (ready state).

[] B) The thread that calls the wait() method gives up the CPU, gives up the lock, and goes into the monitor's waiting pool.

[] C) As a result of calling the notify() method one thread gets moved out of the monitor's waiting pool into lock pool (ready state) in which it must re-acquire the monitor's lock before it can proceed.

[] D) As a result of calling the notifyAll() method all of the threads are moved out of the monitor's waiting pool into lock pool (ready state) in which they must re-acquire the monitor's lock before they can proceed.

[] E) Using the notify() method it is possible to specify which thread is to be notified.

Question 21:

Given the code below,

```
1.  import java.awt.*;
2.  public class Thready extends Thread
3.  {
4.      private static int numOfThreadys = 1;
5.      private Object ob = new Object();
6.      public static void main(String args[])
7.      {
8.          Object locky = new Object();
9.          Thread [] threads = new Thread[50];
10.         for (int index=0; index<50; index++)
11.         {
12.             threads[index] = new Thready();
13.             threads[index].start();
14.         }
15.     }
16.     public void run()
17.     {
18.         synchronized(ob)
19.         {
20.             System.out.println(numOfThreadys);
21.             numOfThreadys++;
```

```
22.             }
23.         }
24. }
```

() A) The numbers 1, 2, 3 … 50 will be printed in this order.

() B) 50 numbers will be printed out. Each one of them is between 1(included) to 50 (included).

() C) A deadlock situation will happen in this program.

() D) The numbers: 1, 2, 3, 4 … 50 will be printed out, but their order isn't guarantee.

Question 22

Given the code below,

```
25. import java.awt.*;
26. public class Thready extends Thread
27. {
28.         private static int numOfThreadys = 1;
29.         private static Object ob = new Object();
30.         public static void main(String args[])
31.         {
32.             Object locky = new Object();
33.             Thread [] threads = new Thread[50];
34.             for (int index=0; index<50; index++)
```

```
35.             {
36.                     threads[index] = new Thready();
37.                     threads[index].start();
38.             }
39.     }
40.
41.     public void run()
42.     {
43.             synchronized(ob)
44.              {
45.                     System.out.println(numOfThreadys);
46.                     numOfThreadys++;
47.             }
48.     }
49. }
```

() A) The numbers 1, 2, 3 … 50 will be printed in this order.

() B) 50 numbers will be printed out. Each one of them is between 1(included) to 50(included).

() C) A deadlock situation will happen in this program.

() D) The numbers: 1, 2, 3, 4 … 50 will be printed out, but their order isn't guarantee.

Question 23:

True or False:

The given code:

```
public class Tara extends Thread
{
        public static void main(String args[])
        {
                Thread t1,t2;
                t1 = new Tara();
                t2 = new Tara();
                t1.start();
                t2.start();
        }
}
```

doesn't compile successfully.

() A) True

() B) False

Question 24:

Given the code below,

```
public class Tara extends Thread
{
        private int lucky = 1;
        public static void main(String args[])
        {
                Thread t1,t2;
                t1 = new Tara();
                t2 = new Tara();
                t1.start();
                t2.start();
         }

        public void run()
        {
                while(true)
                {
                        synchronized(this)
                        {
                                System.out.println("lucky=" + lucky
                                        + " The current thread is : " + Thread.currentThread().getName());
                                lucky++;
```

```
                }
                if (lucky>100)
                        break;
            }
        }
}
```

() A) This application prints 200 numbers between 1 and 100.

() B) This application prints 100 numbers between 1 and 100.

() C) This application never ends.

() D) This application doesn't compile.

Question 25:

Given the code below:

```
1.  public class Tara extends Thread
2.  {
3.          private static int lucky = 1;
4.          public static void main(String args[])
5.          {
6.                  Thread threads[] = new Thread[10];
7.                  for(int i=0; i<threads.length; i++)
8.                  }
```

```
9.                          threads[i] = new Tara();
10.             {
11.             for(int i=0; i<threads.length; i++)
12.             {
13.                     threads[i].start();
14.             }
15.         }
16.         public void run()
17.         {
18.             while(true)
19.             {
20.                     synchronized(this)
21.                     {
22.                             System.out.println("lucky=" + lucky
23.                             + “The current thread is : " + Thread.currentThread().getName());
24.                             lucky++;
25.                     }
26.                     if (lucky>25)
27.                             break;
28.             }
29.         }
30. }
```

() A) It is possible that more than 25 numbers will be printed out.

() B) 250 numbers will be printed out.

() C) The code doesn’t compile

Question 26:

True or False:

It is possible to activate the start() method that belongs to the Thread class more the once.

() A) True

() B) False

Question 27:

Given the code below:

```
1.  public class Lopez extends Thread implements Runnable
2.  {
3.      static int lupu;

4.      public void run()
5.      {
6.          for(lupu=0; lupu<10; lupu++)
7.          {
8.              System.out.println(lupu);
9.          }
10.     }
```

```
11.     public static void main(String args[])
12.     {
13.         Lopez lopez1 = new Lopez();
14.         lopez1.start();
15.         Lopez lopez2 = new Lopez();
16.         lopez2.start();
17.         for(lupu=0; lupu<10; lupu++)
18.         {
19.             System.out.println("TOPAZ");
20.         }
21.     }
22. }
```

() A) The word "TOPAZ" is printed out exactly 10 times.

() B) The word "TOPAZ" is printed out more than 20 times.

() C) The word "TOPAZ" is printed out exactly 30 times.

() D) None of the above answers is true.

() E) The code doesn't compile

Question 28:

Given the code below:

```
1.  public class Magic extends Thread
2.  {
3.      static int num;

4.      public void run()
5.      {
6.          while(num<100)
7.          {
8.              System.out.println(num);
9.              num++;
10.         }
11.     }

12.     public static void main(String args[])
13.     {
14.         Thread vec[] = new Thread[10];
15.         for(int i=0; i<vec.length; i++)
16.             vec[i] = new Magic();
17.         for(int i=0; i<vec.length; i++)
18.             vec[i].start();
```

```
19.         }
20.  }
```

() A) The output will include exactly 1000 numbers

() B) The output will include more than 1000 numbers

() C) The output will include less than 1000 numbers

() D) The output will include exactly 100 numbers

() E) The output will include more than 100 numbers

() F) The output will include less than 100 numbers

() G) None of the above answers is true

Question 29:

Given the code below:

```
1.    public class Magic2 extends Thread
2.    {
3.          static int num;

4.          public void run(int number)
5.          {
6.                 while(num<100)
7.                 System.out.println(++num);
8.          }
```

```
9.          public static void main(String args[])
10.       {
11.             Thread vec[] = new Thread[10];
12.             for(int i=0; i<vec.length; i++)
13.             vec[i] = new Magic2();
14.             for(int i=0; i<vec.length; i++)
15.                     vec[i].start();
16.             System.out.println(num);
17.       }
18. }
```

The output is:

() A) 99

() B) 100

() C) 101

() D) 98

() E) 0

() F) 102

Question 30:

Given the code below:

```
21. public class Magic extends Thread
22. {
23.         int num;

24.         public void run()
25.         {
26.                 while(num<100)
27.                 {
28.                         System.out.println(num);
29.                         num++;
30.                 }
31.         }
32.         public static void main(String args[])
33.         {
34.                  Thread vec[] = new Thread[10];
35.                  for(int i=0; i<vec.length; i++)
36.                         vec[i] = new Magic();
37.                 for(int i=0; i<vec.length; i++)
38.                         vec[i].start();
39.         }
40. }
```

() A) The output will include exactly 1000 numbers

() B) The output will include more than 1000 numbers

() C) The output will include less than 1000 numbers

() D) The output will include exactly 100 numbers

() E) The output will include more than 100 numbers

() F) The output will include less than 100 numbers

() G) None of the above answers is true

Question 31:

Given the code below:

```
public class ThreadDemo1 extends Thread
{
        String text;
        static int count;

        public ThreadDemo1(String str, int count)
        {
                text = str;
                this.count = count;
        }

        public void run()
```

```
    {
        for(int i=0; i<count; i++)
        {
            System.out.println(text);
        }
    }

    public static void main(String args[])
    {
        ThreadDemo1 td1A = new ThreadDemo1("GEZER", 3);
        ThreadDemo1 td1B = new ThreadDemo1("GEZER", 6);
        td1A.start();
        td1B.start();
    }

}
```

The word “GEZER” will be printed out :

() A) 12 times

() B) 24 times

() C) 9 times

() D) 6 rimes

() E) 3 times

Question 32:

Given the code below:

```
public class ThreadDemo1 extends Thread
{
	String text;
	int count;

	public ThreadDemo1(String str, int count)
	{
		text = str;
		this.count = count;
	}

	public void run()
	{
		for(int i=0; i<count; i++)
		{
			System.out.println(text);
		}
	}

	public static void main(String args[])
	{
```

```
        ThreadDemo1 td1A = new ThreadDemo1("GEZER", 3);
        ThreadDemo1 td1B = new ThreadDemo1("GEZER", 6);
        td1A.start();
        td1B.start();
    }

}
```

The word “GEZER” will be printed out :

() A) 12 times

() B) 24 times

() C) 9 times

() D) 6 rimes

() E) 3 times