Multi-Threading

For more details on SUN Certifications, visit JavaScjpDumps

Q: 01 Click the Exhibit button.
What is the result?

```
 1.  class Computation extends Thread {
 2.
 3.    private int num;
 4.    private boolean isComplete;
 5.    private int result;
 6.
 7.    public Computation(int num) { this.num
= num; }
 8.
 9.    public synchronized void run() {
10.      result = num * 2;
11.      isComplete = true;
12.      notify();
13.    }
14.
15.    public synchronized int getResult() {
16.      while (!isComplete) {
17.        try {
18.          wait();
19.        } catch (InterruptedException e)
{}
20.      }
21.      return result;
22.    }
23.
24.    public static void main(String[] args)
{
25.      Computation[] computations = new
Computation[4];
26.      for (int i = 0; i <
computations.length; i++) {
27.        computations[i] = new
Computation(i);
28.        computations[i].start();
29.      }
30.      for (Computation c : computations)
31.        System.out.print(c.getResult() + "
");
32.    }
33.  }
```

A. The code will deadlock.
B. The code may run with no output.
C. An exception is thrown at runtime.
D. The code may run with output "0 6".

E. The code may run with output "2 0 6 4".
F. The code may run with output "0 2 4 6".
Answer: F

Q: 02 Given:

```
1. public class Threads2 implements Runnable {
2.
3. public void run() {
4. System.out.println("run.");
5. throw new RuntimeException("Problem");
6. }
7. public static void main(String[] args) {
8. Thread t = new Thread(new Threads2());
9. t.start();
10. System.out.println("End of method.");
11. }
12. }
```

Which two can be results? (Choose two.)
A. java.lang.RuntimeException: Problem
B. run.
java.lang.RuntimeException: Problem
C. End of method.
java.lang.RuntimeException: Problem
D. End of method.
run.
java.lang.RuntimeException: Problem
E. run.
java.lang.RuntimeException: Problem
End of method.
Answer: D, E

Q: 03 Given:

```
public class TestSeven extends Thread {
private static int x;
public synchronized void doThings() {
 int current = x;
 current++;
 x = current;
 }
public void run() {
 doThings();
 }
}
```

Which statement is true?

A. Compilation fails.
B. An exception is thrown at runtime.
C. Synchronizing the run() method would make the class thread-safe.
D. The data in variable "x" are protected from concurrent access problems.
E. Declaring the doThings() method as static would make the class thread-safe.
F. Wrapping the statements within doThings() in a synchronized(new Object()) { } block would make the class
thread-safe.
Answer: E

Q: 04 Given:
```
1. public class Threads3 implements Runnable {
2. public void run() {
3. System.out.print("running");
4. }
5. public static void main(String[] args) {
6. Thread t = new Thread(new Threads3());
7. t.run();
8. t.run();
9. t.start();
10. }
11. }
```
What is the result?
A. Compilation fails.
B. An exception is thrown at runtime.
C. The code executes and prints "running".
D. The code executes and prints "runningrunning".
E. The code executes and prints "runningrunningrunning".
Answer: E

Q: 05 Given:
```
public class NamedCounter {
private final String name;
private int count;
public NamedCounter(String name) { this.name = name; }
public String getName() { return name; }
public void increment() { count++; }
public int getCount() { return count; }
public void reset() { count = 0; }
}
```
Which three changes should be made to adapt this class to be used safely by multiple threads? (Choose three.)
A. declare reset() using the synchronized keyword

B. declare getName() using the synchronized keyword
C. declare getCount() using the synchronized keyword
D. declare the constructor using the synchronized keyword
E. declare increment() using the synchronized keyword
Answer: A, C, E

Q: 06 Given:
7. void waitForSignal() {
8. Object obj = new Object();
9. synchronized (Thread.currentThread()) {
10. obj.wait();
11. obj.notify();
12. }
13. }
Which statement is true?
A. This code may throw an InterruptedException.
B. This code may throw an IllegalStateException.
C. This code may throw a TimeoutException after ten minutes.
D. This code will not compile unless "obj.wait()" is replaced with "((Thread) obj).wait()".
E. Reversing the order of obj.wait() and obj.notify() may cause this method to complete normally.
F. A call to notify()or notifyAll() from another thread may cause this method to complete normally.
Answer: B

Q: 07 Which two code fragments will execute the method doStuff() in a separate thread? (Choose two.)
A. new Thread() {
public void run() { doStuff(); }
};
B. new Thread() {
public void start() { doStuff(); }
};
C. new Thread() {
public void start() { doStuff(); }
}.run();
D. new Thread() {
public void run() { doStuff(); }
}.start();
E. new Thread(new Runnable() {
public void run() { doStuff(); }
}).run();
F. new Thread(new Runnable() {
public void run() { doStuff(); }

}).start();
Answer: D, F

Q: 08 Given:
1. public class TestOne implements Runnable {
2. public static void main (String[] args) throws Exception {
3. Thread t = new Thread(new TestOne());
4. t.start();
5. System.out.print("Started");
6. t.join();
7. System.out.print("Complete");
8. }
9. public void run() {
10. for (int i = 0; i < 4; i++) {
11. System.out.print(i);
12. }
13. }
14. }
What can be a result?
A. Compilation fails.
B. An exception is thrown at runtime.
C. The code executes and prints "StartedComplete".
D. The code executes and prints "StartedComplete0123".
E. The code executes and prints "Started0123Complete".
Answer: E

Q: 09Given:
1. public class TestOne {
2. public static void main (String[] args) throws Exception {
3. Thread.sleep(3000);
4. System.out.println("sleep");
5. }
6. }
What is the result?
A. Compilation fails.
B. An exception is thrown at runtime.
C. The code executes normally and prints "sleep".
D. The code executes normally, but nothing is printed.
Answer: C

Q: 10 Given:
11. public class Test {
12. public enum Dogs {collie, harrier, shepherd};
13. public static void main(String [] args) {

14. Dogs myDog = Dogs.shepherd;
15. switch (myDog) {
16. case collie:
17. System.out.print("collie ");
18. case default:
19. System.out.print("retriever ");
20. case harrier:
21. System.out.print("harrier ");
22. }
23. }
24. }
What is the result?
A. harrier
B. shepherd
C. retriever
D. Compilation fails.
E. retriever harrier
F. An exception is thrown at runtime.
Answer: D

Q: 11 Given:
11. Runnable r = new Runnable() {
12. public void run() {
13. System.out.print("Cat");
14. }
15. };
16. Thread t = new Thread(r) {
17. public void run() {
18. System.out.print("Dog");
19. }
20. };
21. t.start();
What is the result?
A. Cat
B. Dog
C. Compilation fails.
D. The code runs with no output.
E. An exception is thrown at runtime.
Answer: B

Q: 12 Given:
1. public class Threads4 {
2. public static void main (String[] args) {
3. new Threads4().go();

```
4. }
5. public void go() {
6. Runnable r = new Runnable() {
7. public void run() {
8. System.out.print("foo");
9. }
10. };
11. Thread t = new Thread(r);
12. t.start();
13. t.start();
14. }
15. }
```

What is the result?

A. Compilation fails.
B. An exception is thrown at runtime.
C. The code executes normally and prints "foo".
D. The code executes normally, but nothing is printed.

Answer: B

Q: 13 Given:

```
1. public class TestFive {
2. private int x;
3. public void foo() {
4. int current = x;
5. x = current + 1;
6. }
7. public void go() {
8. for(int i = 0; i < 5; i++) {
9. new Thread() {
10. public void run() {
11. foo();
12. System.out.print(x + ", ");
13. } }.start();
14. } }
```

Which two changes, taken together, would guarantee the output: 1, 2, 3, 4, 5, ? (Choose two.)

A. move the line 12 print statement into the foo() method
B. change line 7 to public synchronized void go() {
C. change the variable declaration on line 2 to private volatile int x;
D. wrap the code inside the foo() method with a synchronized(this) block
E. wrap the for loop code inside the go() method with a synchronized block synchronized(this) { // for loop code here }

Answer: A, D

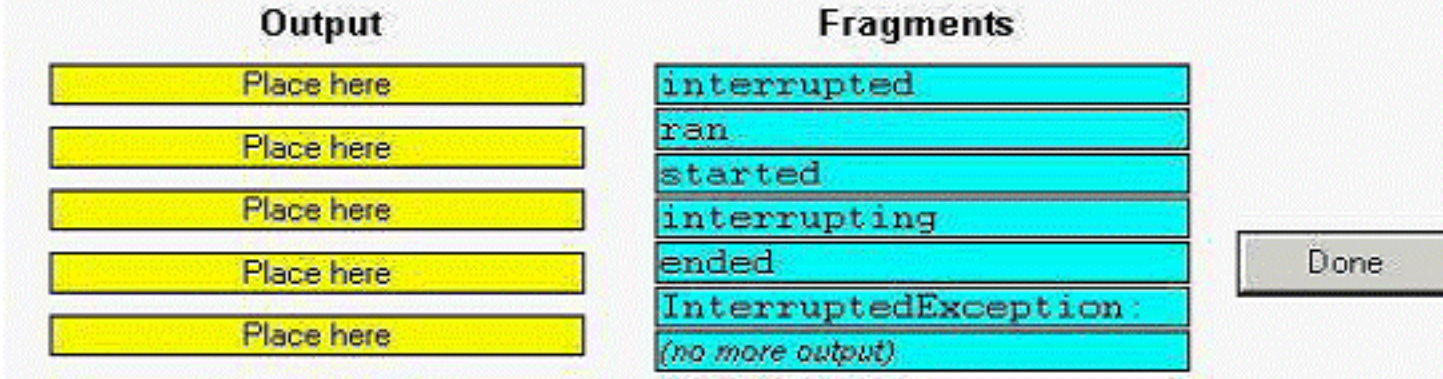
Q: 14 Click the Task button.

Given:

```
10.  Runnable r = new Runnable() {
11.    public void run() {
12.      try {
13.        Thread.sleep(1000);
14.      } catch (InterruptedException e) {
15.        System.out.println("interrupted");
16.      }
17.      System.out.println("ran");
18.    }
19.  };
20.  Thread t = new Thread(r);
21.  t.start();
22.  System.out.println("started");
23.  t.sleep(2000);
24.  System.out.println("interrupting");
25.  t.interrupt();
26.  System.out.println("ended");
```

Assume that sleep(n) executes in exactly n milliseconds, and all other code executes in an insignificant amount of time.

Place the fragments in the output area to show the result of running this code.



Solution:
1.started
2.ran
3.interrupting
4.ended
5.(no more out put)

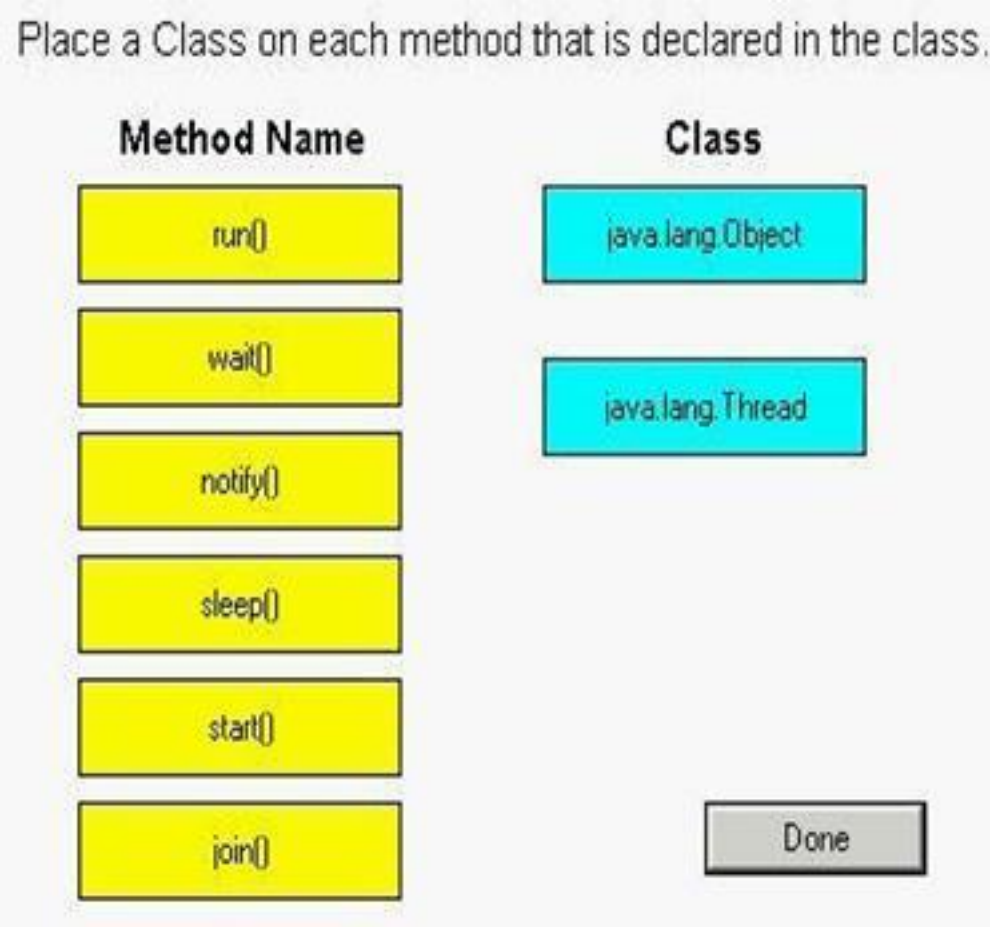
Q:15 Click the Exhibit button.
What is the output if the main() method is run?

Given:

```
10.  public class Starter extends Thread {
11.    private int x = 2;
12.    public static void main(String[] args)
throws Exception {
13.      new Starter().makeItSo();
14.    }
15.    public Starter() {
16.      x = 5;
17.      start();
18.    }
19.    public void makeItSo() throws
Exception {
20.      join();
21.      x = x - 1;
22.      System.out.println(x);
23.    }
24.    public void run() { x *= 2; }
25.  }
```

A. 4 B. 5 C. 8
D. 9 E. Compilation fails.
F. An exception is thrown at runtime.
G. It is impossible to determine for certain.
Answer: D

Q: 16 Click the Task button.



Solution:
java.lang.object ---------- wait( ); notify( );
java.lang.Thread----------run(), sleep( ); start( ); join ( )

Q: 17 Given:
foo and bar are public references available to many other threads. foo refers to a Thread and bar is an
Object. The thread foo is currently executing bar.wait().
From another thread, what provides the most reliable way to ensure that foo will stop executing wait()?

A. foo.notify(); B. bar.notify();
C. foo.notifyAll(); D. Thread.notify();
E. bar.notifyAll(); F. Object.notify();

Answer: E

Q: 18 Given:
1. public class MyLogger {
2. private StringBuilder logger = new StringBuuilder();
3. public void log(String message, String user) {
4. logger.append(message);

5. logger.append(user);
6. }
7. }
The programmer must guarantee that a single MyLogger object works properly for a multi-threaded system.
How must this code be changed to be thread-safe?
A. synchronize the log method
B. replace StringBuilder with StringBuffer
C. replace StringBuilder with just a String object and use the string concatenation (+=) within the log method
D. No change is necessary, the current MyLogger code is already thread-safe.
Answer: A

Q: 19 Given:
1. public class TestSeven extends Thread {
2. private static int x;
3. public synchronized void doThings() {
4. int current = x;
5. current++;
6. x = current;
7. }
8. public void run() {
9. doThings();
10. }
11.}
Which statement is true?
A. Compilation fails.
B. An exception is thrown at runtime.
C. Synchronizing the run() method would make the class thread-safe.
D. The data in variable "x" are protected from concurrent access problems.
E. Declaring the doThings() method as static would make the class thread-safe.
F. Wrapping the statements within doThings() in a synchronized(new Object()) { } block would make the class thread-safe.
Answer: E

Q:20 Click the Exhibit button.
Which two statements are true if this class is compiled and run? (Choose two.)

```
 1.  import java.util.*;
 2.
 3.  public class NameList {
 4.    private List names = new ArrayList();
 5.    public synchronized void add(String
name) { names.add(name); }
 6.    public synchronized void printAll() {
 7.      for (int i = 0; i < names.size();
i++) {
 8.        System.out.print(names.get(i) + "
");
 9.      }
10.    }
11.    public static void main(String[] args)
{
12.      final NameList sl = new NameList();
13.      for (int i = 0; i < 2; i++) {
14.        new Thread() {
15.          public void run() {
16.            sl.add("A");
17.            sl.add("B");
18.            sl.add("C");
19.            sl.printAll();
20.          }
21.        }.start();
22.      }
23.    }
24.  }
```

A. An exception may be thrown at runtime.
B. The code may run with no output, without exiting.
C. The code may run with no output, exiting normally.
D. The code may run with output "A B A B C C ", then exit.
E. The code may run with output "A B C A B C A B C ", then exit.
F. The code may run with output "A A A B C A B C C ", then exit.
G. The code may run with output "A B C A A B C A B C ", then exit.
Answer: E, G

Q: 21 Given:
1. public class Threads5 {
2. public static void main (String[] args) {
3. new Thread(new Runnable() {
4. public void run() {
5. System.out.print("bar");
6. }}).start();

7. }
8. }
What is the result?
A. Compilation fails.
B. An exception is thrown at runtime.
C. The code executes normally and prints "bar".
D. The code executes normally, but nothing prints.
Answer: C

Q: 22 Which three will compile and run without exception? (Choose three.)
A. private synchronized Object o;
B. void go() {
synchronized() { /* code here */ }
C. public synchronized void go() { /* code here */ }
D. private synchronized(this) void go() { /* code here */ }
E. void go() {
synchronized(Object.class) { /* code here */ }
F. void go() {
Object o = new Object();
synchronized(o) { /* code here */ }
Answer: C, E, F

Q: 23 Click the Task button.

Place the code elements in position so that the Flags2 class will compile and make appropriate use of the wait/notify mechanism. Note: You may reuse code elements.

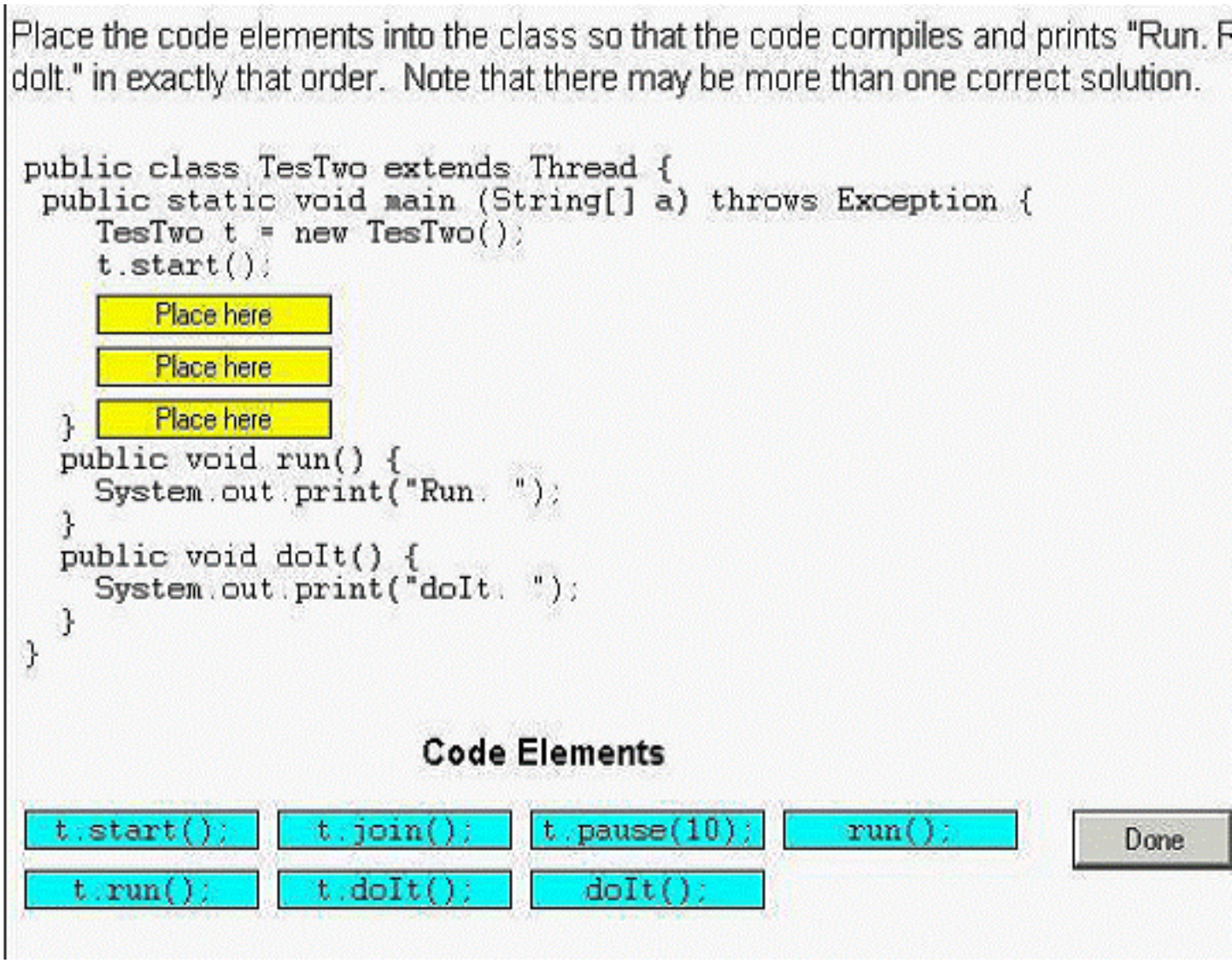
```
public class Flags2 {
   private boolean isReady = false;

   public [Place here] void produce() {
      isReady = true;
      [Place here];
   }
   public [Place here] void consume() {
      while (! isReady) {
        try {
            [Place here];
        } catch (Exception ex) { }
      }
      isReady = [Place here];
    }
}
```

Code Elements

synchronized	true	false	wait()
volatile	synchronized()	notifyAll()	synchronize

Solution:
1.synchronized
2.notifyAll( )
3.synchronized
4.wait( )
5.false

Q: 24 Click the Task button.



Solution:
t.join( );
t.run( );
t.dolt( );

Q: 25 Click the Exhibit button.
Which two are possible results? (Choose two.)

```
 1. public class Threads1 {
 2.  int x = 0;
 3.  public class Runner implements Runnable
{
 4.    public void run() {
 5.      int current = 0;
 6.      for(int i = 0; i < 4; i++) {
 7.        current = x;
 8.        System.out.print(current + ", ");
 9.        x = current + 2;
10.      }
11.    }
12.  }
13.
14.  public static void main(String[] args) {
15.    new Threads1().go();
16.  }
17.
18.  public void go() {
19.    Runnable r1 = new Runner();
20.    new Thread(r1).start();
21.    new Thread(r1).start();
22.  }
23.}
```

A. 0, 2, 4, 4, 6, 8, 10, 6,
B. 0, 2, 4, 6, 8, 10, 2, 4,
C. 0, 2, 4, 6, 8, 10, 12, 14,
D. 0, 0, 2, 2, 4, 4, 6, 6, 8, 8, 10, 10, 12, 12, 14, 14,
E. 0, 2, 4, 6, 8, 10, 12, 14, 0, 2, 4, 6, 8, 10, 12, 14,
Answer: A, C

Question: 26
Click the Exhibit button.
Given:
1**. public class TwoThreads {**
2
3. private static Object resource = new Object();
4.
5. **private static void delay(long n) {**
6. try { Thread.sleep(n); }

```
7. catch (Exception e) { System.out.print(”Error “); }
8. }
9
10. public static void main(String[] args) {
11. System.out.print(”StartMain “);
12. new Thread1().start();
13. delay(1000);
14. Thread t2 = new Thread2();
15. t2.start();
16. delay(1000);
17. t2.interrupt
18. delay(1000);
19. System.out.print(”EndMain “);
20. }
21.
22. static class Thread 1 extends Thread {
23. public void run() {
24. synchronized (resource) {
25. System.out.print(”Startl “);
26. delay(6000);
27. System.out.print(”End1 “);
28. }
29. }
30. }
31.
32. static class Thread2 extends Thread {
33. public void run() {
34. synchronized (resource) {
35. System.out.print(”Start2 “);
36. delay(2000);
37. System.out.print(”End2 “);
38. }
39. }
40. }
41. }
```

Assume that sleep(n) executes in exactly m milliseconds, and all other code executes in an insignificant amount of time. What is the output if the main() method is run?

A. Compilation fails.
B. Deadlock occurs.
C. StartMain Start1 Error EndMain End1
D. StartMain Start1 EndMain End1 Start2 End2
E. StartMain Start1 Error Start2 EndMain End2 End1
F. StartMain Start1 Start2 Error End2 EndMain End1

G. StartMain Start1 EndMain End1 Start2 Error End2
Answer: G

Question: 27
Click the Exhibit button.

```
10. public class Transfers {
11. public static void main(String[] args) throws Exception {
12. Record r1 = new Record();
13. Record r2 = new Record();
14. doTransfer(r1, r2, 5);
15. doTransfer(r2, r1, 2);
16. doTransfer(r1, r2, 1);
17. // print the result
18. System.out.println("rl = " + r1.get() +", r2=" + r2.get());
19. }
20. private static void doTransfer(
21. final Record a, final Record b, final int amount) {
22. Thread t = new Thread() {
23. public void run() {
24. new Clerk().transfer(a, b, amount);
25. }
26. };
27. t.start();
28. }
29. }
30. class Clerk {
31. public synchronized void transfer(Record a, Record b, int amount){
32. synchronized (a) {
33. synchronized (b) {
34. a.add(-amount);
35. b.add(amount);
36. }
37. }
38. }
39. }
40. class Record {
41.int num=10;
42. public int get() { return num; }
43. public void add(int n) { num = num + n; }
44. }
```

If Transfers.main() is run, which three are true? (Choose three.)
A. The output may be "r1 = 6, r2 = 14".
B. The output may be "r1 = *5,* r2 = *15".*
C. The output may be "r1 = 8, r2 = 12".

D. The code may run (and complete) with no output.
E. The code may deadlock (without completing) with no output.
F. M IllegalStateException or InterruptedException may be thrown at runtime.
Answer: ABE

28. Given:

```
public class Messager implements Runnable {
public static void main(String[] args) {
new Thread(new Messager("Wallace")).start();
new Thread(new Messager("Gromit")).start();
}
private String name;
public Messager(String name) { this.name = name; }
public void run() {
message(1); message(2);
}
private synchronized void message(int n) {
System.out.print(name + "-" + n + " ");
}
}
```

Which of the following is a possible result? (Choose all that apply.)
A. Wallace-1 Wallace-2 Gromit-1
B. Wallace-1 Gromit-2 Wallace-2 Gromit-1
C. Wallace-1 Gromit-1 Gromit-2 Wallace-2
D. Gromit-1 Gromit-2
E. Gromit-2 Wallace-1 Gromit-1 Wallace-2
F. The code does not compile.
G. An error occurs at run time.
Answer:
-> **C** is correct. Both threads will print two messages each. Wallace-1 must be before Wallace-2, and Gromit-1 must be before Gromit-2. Other than that, the Wallace and Gromit messages can be intermingled in any order.
-> **A, B, D, E, F,** and **G** are incorrect based on the above.

29. Given:

```
public class Letters extends Thread {
private String name;
public Letters(String name) { this.name = name; }
public void write() {
System.out.print(name);
System.out.print(name);
}
public static void main(String[] args) {
```

new Letters("X").start();
new Letters("Y").start();
} }
We want to guarantee that the output can be either XXYY or YYXX, but never XYXY or any other combination. Which of the following method definitions could be added to the Letters class to make this guarantee? (Choose all that apply.)
A. public void run() { write(); }
B. public synchronized void run() { write(); }
C. public static synchronized void run() { write(); }
D. public void run() { synchronized(this) { write(); } }
E. public void run() { synchronized(Letters.class) { write(); } }
F. public void run() { synchronized(System.out) { write(); } }
G. public void run() { synchronized(System.out.class) { write(); } }
Answer:
-> **E** and **F** are correct. **E** and **F** both cause both threads to lock on the same object, which will prevent the threads from running simultaneously, and guarantee XXYY or YYXX. It's a bit unusual to lock on an object like System.out, but it's perfectly legal, and both threads are locking on the same object.
-> **A** can't guarantee anything since it has no synchronization. **B** and **D** both synchronize on an instance of the Letters class—but since there are two different instances in the main() method, the two threads do not block each other and may run simultaneously, resulting in output like XYXY. **C** won't compile because it tries to override run() with a static method, and also calls a non-static method from a static method. **G** won't compile because System.out.class is nonsense. A class literal must start with a class name. System.out is a field not a class, so System.out.class is not a valid class literal.

30. The following block of code creates a Thread using a Runnable target:
Runnable target = new MyRunnable();
Thread myThread = new Thread(target);
Which of the following classes can be used to create the target, so that the preceding code compiles correctly?
A. public class MyRunnable extends Runnable{public void run(){}}
B. public class MyRunnable extends Object{public void run(){}}
C. public class MyRunnable implements Runnable{public void run(){}}
D. public class MyRunnable implements Runnable{void run(){}}
E. public class MyRunnable implements Runnable{public void start(){}}
Answer:
-> **C** is correct. The class implements the Runnable interface with a legal run() method.
-> **A** is incorrect because interfaces are implemented, not extended. **B** is incorrect because even though the class has a valid public void run() method, it does not implement the Runnable interface. **D** is incorrect because the run() method must be public. **E** is incorrect because the method to implement is run(), not start().

31. Given the following,

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

What is the result of this code?

A. Compilation fails.
B. An exception occurs at runtime.
C. Thread one. Thread two.
D. The output cannot be determined.

Answer:

-> **B** is correct. When the start() method is attempted a second time on a single Thread object, the method will throw an IllegalThreadStateException. (Although this behavior is specified in the API, some JVMs don't consistently throw an exception in this case). Even if the thread has finished running, it is still illegal to call start() again.
-> **A** is incorrect because compilation will succeed. For the most part, the Java compiler only checks for illegal syntax, rather than class-specific logic. **C** and **D** are incorrect because of the logic explained above.

32. Given the following

```
3. class MyThread extends Thread {
4. public static void main(String [] args) {
5. MyThread t = new MyThread();
6. Thread x = new Thread(t);
7. x.start();
8. }
9. public void run() {
10. for(int i=0;i<3;++i) {
11. System.out.print(i + "..");
12. } } }
```

What is the result of this code?

A. Compilation fails. B. 1..2..3.. C. 0..1..2..3..
D. 0..1..2.. E. An exception occurs at runtime.

Answer:

-> **D** is correct. The thread MyThread will start and loop three times (from 0 to 2).
-> **A** is incorrect because the Thread class implements the Runnable interface; therefore, in line 5, Thread can take an object of type Thread as an argument in the constructor (this is NOT recommended). **B** and **C** are incorrect because the variable i in the for loop starts with a value of

0 and ends with a value of 2. **E** is incorrect based on the above.

33. Given the following
3. class Test {
4. public static void main(String [] args) {
5. printAll(args);
6. }
7. public static void printAll(String[] lines) {
8. for(int i=0;i<lines.length;i++){
9. System.out.println(lines[i]);
10. Thread.currentThread().sleep(1000);
11. } } }
The static method Thread.currentThread() returns a reference to the currently executing Thread object. What is the result of this code?
A. Each String in the array lines will print, with exactly a 1-second pause between lines.
B. Each String in the array lines will print, with no pause in between because this method is not executed in a Thread.
C. Each String in the array lines will print, and there is no guarantee there will be a pause because currentThread() may not retrieve this thread.
D. This code will not compile.
E. Each String in the lines array will print, with at least a one-second pause between lines.
Answer:
-> **D** is correct. The sleep() method must be enclosed in a try/catch block, or the method printAll() must declare it throws the InterruptedException.
-> **E** is incorrect, but it would be correct if the InterruptedException was dealt with (**A** is too precise). **B** is incorrect (even if the InterruptedException was dealt with) because all Java code, including the main() method, runs in threads. **C** is incorrect. The sleep() method is static, it always affects the currently executing thread.

34. A ssume you have a class that holds two private variables: a and b. Which of the following pairs can prevent concurrent access problems in that class? (Choose all that apply.)
A. public int read(){return a+b;}
public void set(int a, int b){this.a=a;this.b=b;}
B. public synchronized int read(){return a+b;}
public synchronized void set(int a, int b){this.a=a;this.b=b;}
C. public int read(){synchronized(a){return a+b;}}
public void set(int a, int b){synchronized(a){this.a=a;this.b=b;}}
D. public int read(){synchronized(a){return a+b;}}
public void set(int a, int b){synchronized(b){this.a=a;this.b=b;}}
E. public synchronized(this) int read(){return a+b;}
public synchronized(this) void set(int a, int b){this.a=a;this.b=b;}
F. public int read(){synchronized(this){return a+b;}}
public void set(int a, int b){synchronized(this){this.a=a;this.b=b;}}

Answer:
-> **B** and **F** are correct. By marking the methods as synchronized, the threads will get the lock of the this object before proceeding. Only one thread will be setting or reading at any given moment, thereby assuring that read() always returns the addition of a valid pair.
-> **A** is incorrect because it is not synchronized; therefore, there is no guarantee that the values added by the read() method belong to the same pair. **C** and **D** are incorrect; only objects can be used to synchronize on. **E** fails— it is not possible to select other objects (even this) to synchronize on when declaring a method as synchronized.

35. Which are methods of the Object class? (Choose all that apply.)
A. notify();
B. notifyAll();
C. isInterrupted();
D. synchronized();
E. interrupt();
F. wait(long msecs);
G. sleep(long msecs);
H. yield();
Answer:
-> **A , B,** and **F** are correct. They are all related to the list of threads waiting on the specified object.
-> **C, E, G,** and **H** are incorrect answers. The methods isInterrupted() and interrupt()are instance methods of Thread. The methods sleep() and yield() are static methods of Thread. **D** is incorrect because synchronized is a keyword and the synchronized()construct is part of the Java language.

36. Given the following
```
1. public class WaitTest {
2. public static void main(String [] args) {
3. System.out.print("1 ");
4. synchronized(args){
5. System.out.print("2 ");
6. try {
7. args.wait();
8. }
9. catch(InterruptedException e){}
10. }
11. System.out.print("3 ");
12. } }
```
What is the result of trying to compile and run this program?
A. It fails to compile because the IllegalMonitorStateException of wait() is not dealt with inline 7.
B. 1 2 3
C. 1 3
D. 1 2

E. At runtime, it throws an IllegalMonitorStateException when trying to wait.
F. It will fail to compile because it has to be synchronized on the this object.
Answer:
-> **D** is correct. 1 and 2 will be printed, but there will be no return from the wait call because no other thread will notify the main thread, so 3 will never be printed. It's frozen at line 7.
-> **A** is incorrect; IllegalMonitorStateException is an unchecked exception. **B** and **C** are incorrect; 3 will never be printed, since this program will wait forever. **E** is incorrect because IllegalMonitorStateException will never be thrown because the wait()is done on args within a block of code synchronized on args. **F** is incorrect because any object can be used to synchronize on and this and static don't mix.

37. Assume the following method is properly synchronized and called from a thread A on an object B:
wait(2000);
After calling this method, when will the thread A become a candidate to get another turn at the CPU?
A. After object B is notified, or after two seconds.
B. After the lock on B is released, or after two seconds.
C. Two seconds after object B is notified.
D. Two seconds after lock B is released.
Answer:
-> **A** is correct. Either of the two events will make the thread a candidate for running again.
-> **B** is incorrect because a waiting thread will not return to runnable when the lock is released, unless a notification occurs. **C** is incorrect because the thread will become a candidate immediately after notification. **D** is also incorrect because a thread will not come out of a waiting pool just because a lock has been released.

38. Which are true? (Choose all that apply.)
A. The notifyAll() method must be called from a synchronized context.
B. To call wait(), an object must own the lock on the thread.
C. The notify() method is defined in class java.lang.Thread.
D. When a thread is waiting as a result of wait(), it release its lock.
E. The notify() method causes a thread to immediately release its lock.
F. The difference between notify() and notifyAll() is that notifyAll() notifies all waiting threads, regardless of the object they're waiting on.
Answer:
-> **A** is correct because notifyAll() (and wait() and notify()) must be called from within a synchronized context. **D** is a correct statement.
-> **B** is incorrect because to call wait(), the thread must own the lock on the object that wait() is being invoked on, not the other way around. **C** is wrong because notify() is defined in java.lang.Object. **E** is wrong because notify() will not cause a thread to release its locks. The thread can only release its locks by exiting the synchronized code. **F** is wrong because notifyAll() notifies all the threads waiting on a particular locked object, not all threads waiting on *any* object.

39. Given the scenario: This class is intended to allow users to write a series of messages, so that each message is identified with a timestamp and the name of the thread that wrote the message:

```
public class Logger {
private StringBuilder contents = new StringBuilder();
public void log(String message) {
contents.append(System.currentTimeMillis());
contents.append(": ");
contents.append(Thread.currentThread().getName());
contents.append(message);
contents.append("\n");
}
public String getContents() { return contents.toString(); }
}
```

How can we ensure that instances of this class can be safely used by multiple threads?

A. This class is already thread-safe.
B. Replacing StringBuilder with StringBuffer will make this class thread-safe.
C. Synchronize the log() method only.
D. Synchronize the getContents() method only.
E. Synchronize both log() and getContents().
F. This class cannot be made thread-safe.

Answer:

-> **E** is correct. Synchronizing the public methods is sufficient to make this safe, so **F** is false. This class is not thread-safe unless some sort of synchronization protects the changing data.
-> **B** is not correct because although a StringBuffer is synchonized internally, we call append() multiple times, and nothing would prevent two simultaneous log() calls from mixing up their messages. **C** and **D** are not correct because if one method remains unsynchronized, it can run while the other is executing, which could result in reading the contents while one of the messages is incomplete, or worse. (You don't want to call getString() on the StringBuffer as it's resizing its internal character array.)

40. Given:

```
public static synchronized void main(String[] args) throws
InterruptedException {
Thread t = new Thread();
t.start();
System.out.print("X");
t.wait(10000);
System.out.print("Y");
}
```

What is the result of this code?

A. It prints X and exits.
B. It prints X and never exits.

C. It prints XY and exits almost immeditately.
D. It prints XY with a 10-second delay between X and Y.
E. It prints XY with a 10000-second delay between X and Y.
F. The code does not compile.
G. An exception is thrown at runtime.
Answer:
-> **G** is correct. The code does not acquire a lock on t before calling t.wait(), so it throws an IllegalThreadStateException. The method is synchronized, but it's not synchronized on t so the exception will be thrown. If the wait were placed inside a synchronized(t) block, then the answer would have been **D**.
-> **A, B, C, D, E,** and **F** are incorrect based the logic described above.

41. Given the following:

```
class MyThread extends Thread {
MyThread() {
System.out.print(" MyThread");
}
public void run() { System.out.print(" bar"); }
public void run(String s) { System.out.print(" baz"); }
}
public class TestThreads {
public static void main (String [] args) {
Thread t = new MyThread() {
public void run() { System.out.print(" foo"); }
};
t.start();
} }
```

What is the result?
A. foo
B. MyThread foo
C. MyThread bar
D. foo bar
E. foo bar baz
F. bar foo
G. Compilation fails.
H. An exception is thrown at runtime.
Answer:
-> **B** is correct. The first line of main we're constructing an instance of an anonymous innerclass extending from MyThread. So the MyThread constructor runs and prints MyThread. Next, main() invokes start() on the new thread instance, which causes the overridden run() method (the run() method in the anonymous inner class) to be invoked.
-> **A, C, D, E, F, G** and **H** are incorrect based on the logic described above.

42. Given

```
public class ThreadDemo {
synchronized void a() { actBusy(); }
static synchronized void b() { actBusy(); }
static void actBusy() {
try { Thread.sleep(1000); }
catch (InterruptedException e) {}
}
public static void main(String[] args) {
final ThreadDemo x = new ThreadDemo();
final ThreadDemo y = new ThreadDemo();
Runnable runnable = new Runnable() {
public void run() {
int option = (int) (Math.random() * 4);
switch (option) {
case 0: x.a(); break;
case 1: x.b(); break;
case 2: y.a(); break;
case 3: y.b(); break;
} }
};
Thread thread1 = new Thread(runnable);
Thread thread2 = new Thread(runnable);
thread1.start();
thread2.start();
} }
```

Which of the following pairs of method invocations could NEVER be executing at the same
time? (Choose all that apply.)

A. x.a() in thread1, and x.a() in thread2
B. x.a() in thread1, and x.b() in thread2
C. x.a() in thread1, and y.a() in thread2
D. x.a() in thread1, and y.b() in thread2
E. x.b() in thread1, and x.a() in thread2
F. x.b() in thread1, and x.b() in thread2
G. x.b() in thread1, and y.a() in thread2
H. x.b() in thread1, and y.b() in thread2

Answer:

-> **A , F** and **H**. **A** is incorrect because synchronized instance methods called on the same instance, block each other. **F** and **H** could not happen because synchronized static methods in the same class block each other, regardless of which instance was used to call the methods. (An instance is not required to call static methods; only the class.)

-> **C** could happen because synchronized instance methods called on different instances do not block each other. **B, D, E,** and **G** could all happen because instance methods and static methods lock on different objects, and do not block each other.

43. Given the following,

```
1. public class Test {
2. public static void main (String [] args) {
3. final Foo f = new Foo();
4. Thread t = new Thread(new Runnable() {
5. public void run() {
6. f.doStuff();
7. }
8. });
9. Thread g = new Thread() {
10. public void run() {
11. f.doStuff();
12. }
13. };
14. t.start();
15. g.start();
16. }
17. }
1. class Foo {
2. int x = 5;
3. public void doStuff() {
4. if (x < 10) {
5. // nothing to do
6. try {
7. wait();
8. } catch(InterruptedException ex) { }
9. } else {
10. System.out.println("x is " + x++);
11. if (x >= 10) {
12. notify();
13. }
14. }
15. }
16. }
```

What is the result?

A. The code will not compile because of an error on line 12 of class Foo.
B. The code will not compile because of an error on line 7 of class Foo.
C. The code will not compile because of an error on line 4 of class Test.
D. The code will not compile because of some other error in class Test.
E. An exception occurs at runtime.
F. x is 5
x is 6

Answer:

-> **E** is correct because the thread does not own the lock of the object it invokes wait() on. If the method were synchronized, the code would run without exception.
-> **A, B, C,** and **D** are incorrect because the code compiles without errors. **F** is incorrect because the exception is thrown before there is any output.

44. Given:

```
public class TwoThreads {
static Thread laurel, hardy;
public static void main(String[] args) {
laurel = new Thread() {
public void run() {
System.out.println("A");
try {
hardy.sleep(1000);
} catch (Exception e) {
System.out.println("B");
}
System.out.println("C");
}
};
hardy = new Thread() {
public void run() {
System.out.println("D");
try {
laurel.wait();
} catch (Exception e) {
System.out.println("E");
}
System.out.println("F");
}
};
laurel.start();
hardy.start();
}
}
```

Which letters will eventually appear somewhere in the output? (Choose all that apply.)

A. A B. B
C. C D. D
E. E F. F
G. The answer cannot be reliably determined.
H. The code does not compile.

Answer:

-> **A , C, D, E** and **F** are correct. This may look like laurel and hardy are battling to cause the other to sleep() or wait()—but that's not the case. Since sleep() is a static method, it affects the

current thread, which is laurel (even though the method is invoked using a reference to hardy). That's misleading but perfectly legal, and the Thread laurel is able to sleep with no exception, printing A and C (after a 1-second delay). Meanwhile hardy tries to call laurel.wait()—but hardy has not synchronized on laurel, so calling laurel.wait() immediately causes an IllegalThreadStateException, and so hardy prints D**,** E**,** and F. Although the *order* of the output is somewhat indeterminate (we have no way of knowing whether A is printed before D, for example) it is guaranteed that A**,** C**,** D**,** E**,** and F will all be printed in some order, eventually—so **G** is incorrect.
-> **B, G** and **H** are incorrect based on the above