SELF TEST

The following questions will help you measure your understanding of the material presented in this chapter. If you have a rough time with some of these at first, don't beat yourself up. Some of these questions are long and intricate, expect long and intricate questions on the real exam too!

1. 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(){}}`

2. Given:

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

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

3. Given:

```
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.       }
12.    }
13. }
```

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 output, with a 1-second pause between lines

B. Each String in the array `lines` will output, with no pause in between because this method is not executed in a Thread

C. Each String in the array `lines` will output, 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

4. Assume 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;}}
```

5. Given:

```
 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.     }
13.  }
```

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 in line 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

6. 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

7. 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 releases 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

8. 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

9. 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

10. Given:

```
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

11. 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`

12. 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

13. Given:

```
 3. public class Starter implements Runnable {
 4.   void go(long id) {
 5.     System.out.println(id);
 6.   }
 7.   public static void main(String[] args) {
 8.     System.out.print(Thread.currentThread().getId() + " ");
 9.     // insert code here
10.   }
11.   public void run() { go(Thread.currentThread().getId()); }
12. }
```

And given the following five fragments:

```
I.   new Starter().run();
II.  new Starter().start();
III. new Thread(new Starter());
IV.  new Thread(new Starter()).run();
V.   new Thread(new Starter()).start();
```

When the five fragments are inserted, one at a time at line 9, which are true? (Choose all that apply.)

A. All five will compile
B. Only one might produce the output `4 4`
C. Only one might produce the output `4 2`
D. Exactly two might produce the output `4 4`
E. Exactly two might produce the output `4 2`
F. Exactly three might produce the output `4 4`
G. Exactly three might produce the output `4 2`

14. Given:

```
 3. public class Leader implements Runnable {
 4.   public static void main(String[] args) {
 5.     Thread t = new Thread(new Leader());
 6.     t.start();
 7.     System.out.print("m1 ");
 8.     t.join();
 9.     System.out.print("m2 ");
10.   }
11.   public void run() {
12.     System.out.print("r1 ");
13.     System.out.print("r2 ");
14.   }
15. }
```

Which are true? (Choose all that apply.)

A. Compilation fails

B. The output could be `r1 r2 m1 m2`

C. The output could be `m1 m2 r1 r2`

D. The output could be `m1 r1 r2 m2`

E. The output could be `m1 r1 m2 r2`

F. An exception is thrown at runtime

15. Given:

```
 3. class Dudes {
 4.   static long flag = 0;
 5.   // insert code here
 6.     if(flag == 0) flag = id;
 7.     for(int x = 1; x < 3; x++) {
 8.       if(flag == id) System.out.print("yo ");
 9.       else System.out.print("dude ");
10.     }
11.   }
12. }
13. public class DudesChat implements Runnable {
14.   static Dudes d;
15.   public static void main(String[] args) {
16.     new DudesChat().go();
17.   }
18.   void go() {
19.     d = new Dudes();
```

```
20.     new Thread(new DudesChat()).start();
21.     new Thread(new DudesChat()).start();
22.   }
23.   public void run() {
24.     d.chat(Thread.currentThread().getId());
25.   }
26. }
```

And given these two fragments:

```
I.  synchronized void chat(long id) {
II. void chat(long id) {
```

When fragment I or fragment II is inserted at line 5, which are true? (Choose all that apply.)

A. An exception is thrown at runtime
B. With fragment I, compilation fails
C. With fragment II, compilation fails
D. With fragment I, the output could be `yo dude dude yo`
E. With fragment I, the output could be `dude dude yo yo`
F. With fragment II, the output could be `yo dude dude yo`

16. Given:

```
 3. class Chicks {
 4.   synchronized void yack(long id) {
 5.     for(int x = 1; x < 3; x++) {
 6.       System.out.print(id + " ");
 7.       Thread.yield();
 8.     }
 9.   }
10. }
11. public class ChicksYack implements Runnable {
12.   Chicks c;
13.   public static void main(String[] args) {
14.     new ChicksYack().go();
15.   }
16.   void go() {
17.     c = new Chicks();
18.     new Thread(new ChicksYack()).start();
19.     new Thread(new ChicksYack()).start();
20.   }
21.   public void run() {
```

```
22.      c.yack(Thread.currentThread().getId());
23.    }
24. }
```

Which are true? (Choose all that apply.)

A. Compilation fails
B. The output could be `4 4 2 3`
C. The output could be `4 4 2 2`
D. The output could be `4 4 4 2`
E. The output could be `2 2 4 4`
F. An exception is thrown at runtime

17. Given:

```
 3. public class Chess implements Runnable {
 4.   public void run() {
 5.     move(Thread.currentThread().getId());
 6.   }
 7.   // insert code here
 8.     System.out.print(id + " ");
 9.     System.out.print(id + " ");
10.   }
11.   public static void main(String[] args) {
12.     Chess ch = new Chess();
13.     new Thread(ch).start();
14.     new Thread(new Chess()).start();
15.   }
16. }
```

And given these two fragments:

```
I.  synchronized void move(long id) {
II.  void move(long id) {
```

When either fragment I or fragment II is inserted at line 7, which are true? (Choose all that apply.)

A. Compilation fails
B. With fragment I, an exception is thrown
C. With fragment I, the output could be `4 2 4 2`
D. With fragment I, the output could be `4 4 2 3`
E. With fragment II, the output could be `2 4 2 4`

SELF TEST ANSWERS

1. 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()`. (Objective 4.1)

2. Given:

```
 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. (Objective 4.1)

3. Given:

```
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. (Objective 4.2)

4. Assume 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`. (Objective 4.3)

5. Given:

```
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 in line 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. (Objective 4.4)

6. 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. (Objective 4.4)

7. 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 releases 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. (Objective 4.4)

8. 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.) (Objective 4.3)

9. 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 `IllegalMonitorStateException`. 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. (Objective 4.2)

10. Given:

```
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 inner class 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. (Objective 4.1)

11. 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 a right answer because when `synchronized` instance methods are called on the same *instance*, they block each other. **F** and **H** can't 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. (Objective 4.3)

12. 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 at least 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 `IllegalMonitorStateException`, 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. (Objective 4.4)

13. Given:

```
3. public class Starter implements Runnable {
4.   void go(long id) {
5.     System.out.println(id);
6.   }
7.   public static void main(String[] args) {
8.     System.out.print(Thread.currentThread().getId() + " ");
9.     // insert code here
```

```
10.   }
11.   public void run() { go(Thread.currentThread().getId()); }
12. }
```

And given the following five fragments:

```
I.   new Starter().run();
II.  new Starter().start();
III. new Thread(new Starter());
IV.  new Thread(new Starter()).run();
V.   new Thread(new Starter()).start();
```

When the five fragments are inserted, one at a time at line 9, which are true? (Choose all that apply.)

A. All five will compile
B. Only one might produce the output `4 4`
C. Only one might produce the output `4 2`
D. Exactly two might produce the output `4 4`
E. Exactly two might produce the output `4 2`
F. Exactly three might produce the output `4 4`
G. Exactly three might produce the output `4 2`

Answer:

☑ **C** and **D** are correct. Fragment I doesn't start a new thread. Fragment II doesn't compile. Fragment III creates a new thread but doesn't start it. Fragment IV creates a new thread and invokes `run()` directly, but it doesn't start the new thread. Fragment V creates *and* starts a new thread.

☒ **A, B, E, F,** and **G** are incorrect based on the above. (Objective 4.1)

14. Given:

```
 3. public class Leader implements Runnable {
 4.   public static void main(String[] args) {
 5.     Thread t = new Thread(new Leader());
 6.     t.start();
 7.     System.out.print("m1 ");
 8.     t.join();
 9.     System.out.print("m2 ");
10.   }
```

```
11.   public void run() {
12.     System.out.print("r1 ");
13.     System.out.print("r2 ");
14.   }
15. }
```

Which are true? (Choose all that apply.)

A. Compilation fails

B. The output could be `r1 r2 m1 m2`

C. The output could be `m1 m2 r1 r2`

D. The output could be `m1 r1 r2 m2`

E. The output could be `m1 r1 m2 r2`

F. An exception is thrown at runtime

Answer:

☑ **A** is correct. The `join()` must be placed in a try/catch block. If it were, answers B and D would be correct. The `join()` causes the main thread to pause and join the end of the other thread, meaning `"m2"` must come last.

☒ **B, C, D, E,** and **F** are incorrect based on the above. (Objective 4.2)

15. Given:

```
 3. class Dudes {
 4.   static long flag = 0;
 5.   // insert code here
 6.     if(flag == 0) flag = id;
 7.     for(int x = 1; x < 3; x++) {
 8.       if(flag == id) System.out.print("yo ");
 9.       else System.out.print("dude ");
10.     }
11.   }
12. }
13. public class DudesChat implements Runnable {
14.   static Dudes d;
15.   public static void main(String[] args) {
16.     new DudesChat().go();
17.   }
18.   void go() {
19.     d = new Dudes();
```

```
20.     new Thread(new DudesChat()).start();
21.     new Thread(new DudesChat()).start();
22.   }
23.   public void run() {
24.     d.chat(Thread.currentThread().getId());
25.   }
26. }
```

And given these two fragments:

```
I.  synchronized void chat(long id) {
II. void chat(long id) {
```

When fragment I or fragment II is inserted at line 5, which are true? (Choose all that apply.)

A. An exception is thrown at runtime
B. With fragment I, compilation fails
C. With fragment II, compilation fails
D. With fragment I, the output could be `yo dude dude yo`
E. With fragment I, the output could be `dude dude yo yo`
F. With fragment II, the output could be `yo dude dude yo`

Answer:

☑ **F** is correct. With fragment I, the `chat` method is synchronized, so the two threads can't swap back and forth. With either fragment, the first output must be `yo`.

☒ **A, B, C, D,** and **E** are incorrect based on the above. (Objective 4.3)

16. Given:

```
 3. class Chicks {
 4.   synchronized void yack(long id) {
 5.     for(int x = 1; x < 3; x++) {
 6.       System.out.print(id + " ");
 7.       Thread.yield();
 8.     }
 9.   }
10. }
11. public class ChicksYack implements Runnable {
12.   Chicks c;
13.   public static void main(String[] args) {
14.     new ChicksYack().go();
15.   }
```

```
16.   void go() {
17.     c = new Chicks();
18.     new Thread(new ChicksYack()).start();
19.     new Thread(new ChicksYack()).start();
20.   }
21.   public void run() {
22.     c.yack(Thread.currentThread().getId());
23.   }
24. }
```

Which are true? (Choose all that apply.)

A. Compilation fails

B. The output could be `4 4 2 3`

C. The output could be `4 4 2 2`

D. The output could be `4 4 4 2`

E. The output could be `2 2 4 4`

F. An exception is thrown at runtime

Answer:

☑ **F** is correct. When `run()` is invoked, it is with a new instance of `ChicksYack` and `c` has not been assigned to an object. If `c` were static, then because `yack` is synchronized, answers C and E would have been correct.

☒ **A, B, C, D,** and **E** are incorrect based on the above. (Objective 4.3)

17. Given:

```
3. public class Chess implements Runnable {
4.   public void run() {
5.     move(Thread.currentThread().getId());
6.   }
7.   // insert code here
8.     System.out.print(id + " ");
9.     System.out.print(id + " ");
10.  }
11.  public static void main(String[] args) {
12.    Chess ch = new Chess();
13.    new Thread(ch).start();
14.    new Thread(new Chess()).start();
15.  }
16. }
```

And given these two fragments:

```
I.  synchronized void move(long id) {
II.  void move(long id) {
```

When either fragment I or fragment II is inserted at line 7, which are true? (Choose all that apply.)

A. Compilation fails
B. With fragment I, an exception is thrown
C. With fragment I, the output could be `4 2 4 2`
D. With fragment I, the output could be `4 4 2 3`
E. With fragment II, the output could be `2 4 2 4`

Answer:

☑ **C** and **E** are correct. **E** should be obvious. **C** is correct because even though `move()` is synchronized, it's being invoked on two different objects.

☒ **A, B,** and **D** are incorrect based on the above. (Objective 4.3)