SELF TEST

1. Given two files:

```
1. class One {
2.   public static void main(String[] args) {
3.     int assert = 0;
4.   }
5. }
```

```
1. class Two {
2.   public static void main(String[] args) {
3.     assert(false);
4.   }
5. }
```

And the four command-line invocations:

```
javac -source 1.3 One.java
javac -source 1.4 One.java
javac -source 1.3 Two.java
javac -source 1.4 Two.java
```

What is the result? (Choose all that apply.)

A. Only one compilation will succeed
B. Exactly two compilations will succeed
C. Exactly three compilations will succeed
D. All four compilations will succeed
E. No compiler warnings will be produced
F. At least one compiler warning will be produced

2. Given:

```
class Plane {
  static String s = "-";
  public static void main(String[] args) {
    new Plane().s1();
    System.out.println(s);
  }
  void s1() {
    try { s2(); }
    catch (Exception e) { s += "c"; }
  }
  void s2() throws Exception  {
```

```
    s3();  s += "2";
    s3();  s += "2b";
  }
  void s3() throws Exception {
    throw new Exception();
  }
}
```

What is the result?

A. `-`
B. `-c`
C. `-c2`
D. `-2c`
E. `-c22b`
F. `-2c2b`
G. `-2c2bc`
H. Compilation fails

3. Given:

```
try { int x = Integer.parseInt("two"); }
```

Which could be used to create an appropriate `catch` block? (Choose all that apply.)

A. `ClassCastException`
B. `IllegalStateException`
C. `NumberFormatException`
D. `IllegalArgumentException`
E. `ExceptionInInitializerError`
F. `ArrayIndexOutOfBoundsException`

4. Which are true? (Choose all that apply.)

A. It is appropriate to use assertions to validate arguments to methods marked `public`
B. It is appropriate to catch and handle assertion errors
C. It is NOT appropriate to use assertions to validate command-line arguments
D. It is appropriate to use assertions to generate alerts when you reach code that should not be reachable
E. It is NOT appropriate for assertions to change a program's state

5. Given:

```
1. class Loopy {
2.   public static void main(String[] args) {
3.     int[] x = {7,6,5,4,3,2,1};
4.     // insert code here
5.       System.out.print(y + " ");
6.     }
7.   }
8. }
```

Which, inserted independently at line 4, compiles? (Choose all that apply.)

A. `for(int y : x) {`

B. `for(x : int y) {`

C. `int y = 0; for(y : x) {`

D. `for(int y=0, z=0; z<x.length; z++) { y = x[z];`

E. `for(int y=0, int z=0; z<x.length; z++) { y = x[z];`

F. `int y = 0; for(int z=0; z<x.length; z++) { y = x[z];`

6. Given:

```
class Emu {
  static String s = "-";
  public static void main(String[] args) {
    try {
      throw new Exception();
    } catch (Exception e) {
        try {
          try { throw new Exception();
          } catch (Exception ex) { s += "ic "; }
          throw new Exception(); }
        catch (Exception x) { s += "mc "; }
        finally { s += "mf "; }
    } finally { s += "of "; }
    System.out.println(s);
} }
```

What is the result?

A. `-ic of`

B. `-mf of`

C. `-mc mf`

D. `-ic mf of`

E. `-ic mc mf of`

F. `-ic mc of mf`

G. Compilation fails

7. Given:

```
 3. class SubException extends Exception { }
 4. class SubSubException extends SubException { }
 5.
 6. public class CC { void doStuff() throws SubException { } }
 7.
 8. class CC2 extends CC { void doStuff() throws SubSubException { } }
 9.
10. class CC3 extends CC { void doStuff() throws Exception { } }
11.
12. class CC4 extends CC { void doStuff(int x) throws Exception { } }
13.
14. class CC5 extends CC {  void doStuff()  { } }
```

What is the result? (Choose all that apply.)

A. Compilation succeeds

B. Compilation fails due to an error on line 8

C. Compilation fails due to an error on line 10

D. Compilation fails due to an error on line 12

E. Compilation fails due to an error on line 14

8. Given:

```
 3. public class Ebb {
 4.   static int x = 7;
 5.   public static void main(String[] args) {
 6.     String s = "";
 7.     for(int y = 0; y < 3; y++) {
 8.       x++;
 9.       switch(x) {
10.         case 8: s += "8 ";
11.         case 9: s += "9 ";
12.         case 10: { s+= "10 "; break; }
13.         default: s += "d ";
14.         case 13: s+= "13 ";
```

```
15.        }
16.      }
17.      System.out.println(s);
18.    }
19.    static { x++; }
20. }
```

What is the result?

A. `9 10 d`

B. `8 9 10 d`

C. `9 10 10 d`

D. `9 10 10 d 13`

E. `8 9 10 10 d 13`

F. `8 9 10 9 10 10 d 13`

G. Compilation fails

9. Given:

```
 3. class Infinity { }
 4. public class Beyond extends Infinity {
 5.   static Integer i;
 6.   public static void main(String[] args) {
 7.     int sw = (int)(Math.random() * 3);
 8.     switch(sw) {
 9.       case 0: {  for(int x = 10; x > 5; x++)
10.                   if(x > 10000000) x = 10;
11.                 break; }
12.       case 1: {  int y = 7 * i;  break;  }
13.       case 2: {  Infinity inf = new Beyond();
14.                  Beyond b = (Beyond)inf;  }
15.     }
16.   }
17. }
```

And given that line 7 will assign the value 0, 1, or 2 to `sw`, which are true? (Choose all that apply.)

A. Compilation fails

B. A ClassCastException might be thrown

C. A StackOverflowError might be thrown

D. A NullPointerException might be thrown

E. An IllegalStateException might be thrown

F. The program might hang without ever completing

G. The program will always complete without exception

10. Given:

```
 3. public class Circles {
 4.   public static void main(String[] args) {
 5.     int[] ia = {1,3,5,7,9};
 6.     for(int x : ia) {
 7.       for(int j = 0; j < 3; j++) {
 8.         if(x > 4 && x < 8) continue;
 9.         System.out.print(" " + x);
10.         if(j == 1) break;
11.         continue;
12.       }
13.       continue;
14.     }
15.   }
16. }
```

What is the result?

A. `1 3 9`

B. `5 5 7 7`

C. `1 3 3 9 9`

D. `1 1 3 3 9 9`

E. `1 1 1 3 3 3 9 9 9`

F. Compilation fails

11. Given:

```
 3. public class OverAndOver {
 4.   static String s = "";
 5.   public static void main(String[] args) {
 6.     try {
 7.       s += "1";
 8.       throw new Exception();
 9.     } catch (Exception e) { s += "2";
10.     } finally { s += "3"; doStuff(); s += "4";
11.     }
```

```
12.     System.out.println(s);
13.   }
14.   static void doStuff() { int x = 0; int y = 7/x; }
15. }
```

What is the result?

A. `12`

B. `13`

C. `123`

D. `1234`

E. Compilation fails

F. `123` followed by an exception

G. `1234` followed by an exception

H. An exception is thrown with no other output

12. Given:

```
 3. public class Wind {
 4.   public static void main(String[] args) {
 5.     foreach:
 6.     for(int j=0; j<5; j++) {
 7.       for(int k=0; k< 3; k++) {
 8.         System.out.print(" " + j);
 9.         if(j==3 && k==1) break foreach;
10.         if(j==0 || j==2) break;
11.       }
12.     }
13.   }
14. }
```

What is the result?

A. `0 1 2 3`

B. `1 1 1 3 3`

C. `0 1 1 1 2 3 3`

D. `1 1 1 3 3 4 4 4`

E. `0 1 1 1 2 3 3 4 4 4`

F. Compilation fails

13. Given:

```
 3. public class Gotcha {
 4.   public static void main(String[] args) {
 5.     // insert code here
 6.
 7.   }
 8.   void go() {
 9.     go();
10.   }
11. }
```

And given the following three code fragments:

```
I.   new Gotcha().go();
II.  try { new Gotcha().go(); }
     catch (Error e) { System.out.println("ouch"); }

III. try { new Gotcha().go(); }
     catch (Exception e) { System.out.println("ouch"); }
```

When fragments I - III are added, independently, at line 5, which are true? (Choose all that apply.)

A. Some will not compile
B. They will all compile
C. All will complete normally
D. None will complete normally
E. Only one will complete normally
F. Two of them will complete normally

14. Given:

```
3. public class Clumsy {
4.   public static void main(String[] args) {
5.     int j = 7;
6.     assert(++j > 7);
7.     assert(++j > 8): "hi";
8.     assert(j > 10): j=12;
9.     assert(j==12): doStuff();
```

```
10.     assert(j==12): new Clumsy();
11.   }
12.   static void doStuff() { }
13. }
```

Which are true? (Choose all that apply.)

A. Compilation succeeds

B. Compilation fails due to an error on line 6

C. Compilation fails due to an error on line 7

D. Compilation fails due to an error on line 8

E. Compilation fails due to an error on line 9

F. Compilation fails due to an error on line 10

15. Given:

```
1. public class Frisbee {
2.   // insert code here
3.     int x = 0;
4.     System.out.println(7/x);
5.   }
6. }
```

And given the following four code fragments:

```
I.   public static void main(String[] args) {
II.  public static void main(String[] args) throws Exception {
III. public static void main(String[] args) throws IOException {
IV.  public static void main(String[] args) throws RuntimeException {
```

If the four fragments are inserted independently at line 4, which are true? (Choose all that apply.)

A. All four will compile and execute without exception

B. All four will compile and execute and throw an exception

C. Some, but not all, will compile and execute without exception

D. Some, but not all, will compile and execute and throw an exception

E. When considering fragments II, III, and IV, of those that will compile, adding a try/catch block around line 6 will cause compilation to fail

16. Given:

```
 2. class MyException extends Exception { }
 3. class Tire {
 4.   void doStuff() {  }
 5. }
 6. public class Retread extends Tire {
 7.   public static void main(String[] args) {
 8.     new Retread().doStuff();
 9.   }
10.   // insert code here
11.     System.out.println(7/0);
12.   }
13. }
```

And given the following four code fragments:

```
I.   void doStuff() {
II.  void doStuff() throws MyException {
III. void doStuff() throws RuntimeException {
IV.  void doStuff() throws ArithmeticException {
```

When fragments I - IV are added, independently, at line 10, which are true? (Choose all that apply.)

A. None will compile

B. They will all compile

C. Some, but not all, will compile

D. All of those that compile will throw an exception at runtime

E. None of those that compile will throw an exception at runtime

F. Only some of those that compile will throw an exception at runtime

SELF TEST ANSWERS

1. Given two files:

```
1. class One {
2.   public static void main(String[] args) {
3.     int assert = 0;
4.   }
5. }
1. class Two {
2.   public static void main(String[] args) {
3.     assert(false);
4.   }
5. }
```

And the four command-line invocations:

```
javac -source 1.3 One.java
javac -source 1.4 One.java
javac -source 1.3 Two.java
javac -source 1.4 Two.java
```

What is the result? (Choose all that apply.)

A. Only one compilation will succeed
B. Exactly two compilations will succeed
C. Exactly three compilations will succeed
D. All four compilations will succeed
E. No compiler warnings will be produced
F. At least one compiler warning will be produced

Answer:

☑ **B** and **F** are correct. Class One will compile (and issue a warning) using the `1.3` flag, and class Two will compile using the `1.4` flag.

☒ **A, C, D,** and **E** are incorrect based on the above. (Objective 2.3)

2. Given:

```
class Plane {
  static String s = "-";
  public static void main(String[] args) {
    new Plane().s1();
```

```
    System.out.println(s);
  }
  void s1() {
    try { s2(); }
    catch (Exception e) { s += "c"; }
  }
  void s2() throws Exception  {
    s3();  s += "2";
    s3();  s += "2b";
  }
  void s3() throws Exception {
    throw new Exception();
} }
```

What is the result?

A. `-`
B. `-c`
C. `-c2`
D. `-2c`
E. `-c22b`
F. `-2c2b`
G. `-2c2bc`
H. Compilation fails

Answer:

☑ **B** is correct. Once `s3()` throws the exception to `s2()`, `s2()` throws it to `s1()`, and no more of `s2()`'s code will be executed.

☒ **A, C, D, E, F, G,** and **H** are incorrect based on the above. (Objective 2.5)

3. Given:

```
try { int x = Integer.parseInt("two"); }
```

Which could be used to create an appropriate `catch` block? (Choose all that apply.)

A. `ClassCastException`
B. `IllegalStateException`
C. `NumberFormatException`
D. `IllegalArgumentException`

E. `ExceptionInInitializerError`

F. `ArrayIndexOutOfBoundsException`

Answer:

☑ **C** and **D** are correct. `Integer.parseInt` can throw a `NumberFormatException`, and `IllegalArgumentException` is its superclass (i.e., a broader exception).

☒ **A, B, E,** and **F** are not in `NumberFormatException`'s class hierarchy. (Objective 2.6)

4. Which are true? (Choose all that apply.)

A. It is appropriate to use assertions to validate arguments to methods marked `public`

B. It is appropriate to catch and handle assertion errors

C. It is NOT appropriate to use assertions to validate command-line arguments

D. It is appropriate to use assertions to generate alerts when you reach code that should not be reachable

E. It is NOT appropriate for assertions to change a program's state

Answer:

☑ **C, D,** and **E** are correct statements.

☒ **A** is incorrect. It *is* acceptable to use assertions to test the arguments of `private` methods. **B** is incorrect. While assertion errors can be caught, Sun discourages you from doing so. (Objective 2.3)

5. Given:

```
1. class Loopy {
2.   public static void main(String[] args) {
3.     int[] x = {7,6,5,4,3,2,1};
4.     // insert code here
5.       System.out.print(y + " ");
6.     }
7. } }
```

Which, inserted independently at line 4, compiles? (Choose all that apply.)

A. `for(int y : x) {`

B. `for(x : int y) {`

C. `int y = 0; for(y : x) {`

D. `for(int y=0, z=0; z<x.length; z++) { y = x[z];`

E. `for(int y=0, int z=0; z<x.length; z++) { y = x[z];`

F. `int y = 0; for(int z=0; z<x.length; z++) { y = x[z];`

Answer:

☑ **A, D,** and **F** are correct. **A** is an example of the enhanced `for` loop. **D** and **F** are examples of the basic `for` loop.

☒ **B** is incorrect because its operands are swapped. **C** is incorrect because the enhanced `for` must declare its first operand. **E** is incorrect syntax to declare two variables in a `for` statement. (Objective 2.2)

6. Given:

```
class Emu {
  static String s = "-";
  public static void main(String[] args) {
    try {
      throw new Exception();
    } catch (Exception e) {
        try {
          try { throw new Exception();
          } catch (Exception ex) { s += "ic "; }
          throw new Exception(); }
        catch (Exception x) { s += "mc "; }
        finally { s += "mf "; }
    } finally { s += "of "; }
    System.out.println(s);
} }
```

What is the result?

A. `-ic of`

B. `-mf of`

C. `-mc mf`

D. `-ic mf of`

E. `-ic mc mf of`

F. `-ic mc of mf`

G. Compilation fails

Answer:

☑ **E** is correct. There is no problem nesting `try / catch` blocks. As is normal, when an exception is thrown, the code in the `catch` block runs, then the code in the `finally` block runs.

☒ **A, B, C, D,** and **F** are incorrect based on the above. (Objective 2.5)

7. Given:

```
 3. class SubException extends Exception { }
 4. class SubSubException extends SubException { }
 5.
 6. public class CC { void doStuff() throws SubException { } }
 7.
 8. class CC2 extends CC { void doStuff() throws SubSubException { } }
 9.
10. class CC3 extends CC { void doStuff() throws Exception { } }
11.
12. class CC4 extends CC { void doStuff(int x) throws Exception { } }
13.
14. class CC5 extends CC {  void doStuff()  { } }
```

What is the result? (Choose all that apply.)

A. Compilation succeeds

B. Compilation fails due to an error on line 8

C. Compilation fails due to an error on line 10

D. Compilation fails due to an error on line 12

E. Compilation fails due to an error on line 14

Answer:

☑ **C** is correct. An overriding method cannot throw a broader exception than the method it's overriding. Class CC4's method is an overload, not an override.

☒ **A, B, D,** and **E** are incorrect based on the above. (Objectives 1.5, 2.4)

8. Given:

```
3. public class Ebb {
4.   static int x = 7;
5.   public static void main(String[] args) {
6.     String s = "";
```

```
 7.      for(int y = 0; y < 3; y++) {
 8.        x++;
 9.        switch(x) {
10.          case 8: s += "8 ";
11.          case 9: s += "9 ";
12.          case 10: { s+= "10 "; break; }
13.          default: s += "d ";
14.          case 13: s+= "13 ";
15.        }
16.      }
17.      System.out.println(s);
18.    }
19.    static { x++; }
20. }
```

What is the result?

A. `9 10 d`

B. `8 9 10 d`

C. `9 10 10 d`

D. `9 10 10 d 13`

E. `8 9 10 10 d 13`

F. `8 9 10 9 10 10 d 13`

G. Compilation fails

Answer:

☑ **D** is correct. Did you catch the static initializer block? Remember that switches work on "fall-thru" logic, and that fall-thru logic also applies to the default case, which is used when no other case matches.

☒ **A, B, C, E, F,** and **G** are incorrect based on the above. (Objective 2.1)

9. Given:

```
3. class Infinity { }
4. public class Beyond extends Infinity {
5.   static Integer i;
6.   public static void main(String[] args) {
7.     int sw = (int)(Math.random() * 3);
8.     switch(sw) {
9.       case 0: {  for(int x = 10; x > 5; x++)
```

```
10.                   if(x > 10000000) x = 10;
11.                 break; }
12.       case 1: {  int y = 7 * i;  break;  }
13.       case 2: {  Infinity inf = new Beyond();
14.                  Beyond b = (Beyond)inf;  }
15.     }
16.   }
17. }
```

And given that line 7 will assign the value 0, 1, or 2 to `sw`, which are true? (Choose all that apply.)

A. Compilation fails

B. A ClassCastException might be thrown

C. A StackOverflowError might be thrown

D. A NullPointerException might be thrown

E. An IllegalStateException might be thrown

F. The program might hang without ever completing

G. The program will always complete without exception

Answer:

☑ **D** and **F** are correct. Because `i` was not initialized, case 1 will throw an NPE. Case 0 will initiate an endless loop, not a stack overflow. Case 2's downcast will *not* cause an exception.

☒ **A, B, C, E,** and **G** are incorrect based on the above. (Objective 2.6)

10. Given:

```
3. public class Circles {
4.   public static void main(String[] args) {
5.     int[] ia = {1,3,5,7,9};
6.     for(int x : ia) {
7.       for(int j = 0; j < 3; j++) {
8.         if(x > 4 && x < 8) continue;
9.         System.out.print(" " + x);
10.        if(j == 1) break;
11.        continue;
12.      }
13.      continue;
14.    }
15.  }
16. }
```

What is the result?

A. `1 3 9`

B. `5 5 7 7`

C. `1 3 3 9 9`

D. `1 1 3 3 9 9`

E. `1 1 1 3 3 3 9 9 9`

F. Compilation fails

Answer:

☑ **D** is correct. The basic rule for unlabeled continue statements is that the current iteration stops early and execution jumps to the next iteration. The last two continue statements are redundant!

☒ **A, B, C, E,** and **F** are incorrect based on the above. (Objective 2.2)

11. Given:

```
 3. public class OverAndOver {
 4.   static String s = "";
 5.   public static void main(String[] args) {
 6.     try {
 7.       s += "1";
 8.       throw new Exception();
 9.     } catch (Exception e) { s += "2";
10.     } finally { s += "3"; doStuff(); s += "4";
11.     }
12.     System.out.println(s);
13.   }
14.   static void doStuff() { int x = 0; int y = 7/x; }
15. }
```

What is the result?

A. `12`

B. `13`

C. `123`

D. `1234`

E. Compilation fails

F. `123` followed by an exception

G. `1234` followed by an exception

H. An exception is thrown with no other output

Answer:

☑ **H** is correct. It's true that the value of `String s` is `123` at the time that the divide-by-zero exception is thrown, but `finally()` is *not* guaranteed to complete, and in this case `finally()` never completes, so the System.out.println (S.O.P.) never executes.

☒ **A, B, C, D, E, F,** and **G** are incorrect based on the above. (Objective 2.5)

12. Given:

```
 3. public class Wind {
 4.   public static void main(String[] args) {
 5.     foreach:
 6.     for(int j=0; j<5; j++) {
 7.       for(int k=0; k< 3; k++) {
 8.         System.out.print(" " + j);
 9.         if(j==3 && k==1) break foreach;
10.         if(j==0 || j==2) break;
11.       }
12.     }
13.   }
14. }
```

What is the result?

A. `0 1 2 3`

B. `1 1 1 3 3`

C. `0 1 1 1 2 3 3`

D. `1 1 1 3 3 4 4 4`

E. `0 1 1 1 2 3 3 4 4 4`

F. Compilation fails

Answer:

☑ **C** is correct. A `break` breaks out of the current innermost loop and continues. A labeled `break` breaks out of and terminates the current loops.

☒ **A, B, D, E,** and **F** are incorrect based on the above. (Objective 2.2)

13. Given:

```
 3. public class Gotcha {
 4.   public static void main(String[] args) {
 5.     // insert code here
 6.
 7.   }
 8.   void go() {
 9.     go();
10.   }
11. }
```

And given the following three code fragments:

```
I.   new Gotcha().go();
II.  try { new Gotcha().go(); }
     catch (Error e) { System.out.println("ouch"); }

III. try { new Gotcha().go(); }
     catch (Exception e) { System.out.println("ouch"); }
```

When fragments I - III are added, independently, at line 5, which are true? (Choose all that apply.)

A. Some will not compile

B. They will all compile

C. All will complete normally

D. None will complete normally

E. Only one will complete normally

F. Two of them will complete normally

Answer:

☑ **B** and **E** are correct. First off, `go()` is a badly designed recursive method, guaranteed to cause a StackOverflowError. Since `Exception` is not a superclass of `Error`, catching an `Exception` will not help handle an `Error`, so fragment III will not complete normally. Only fragment II will catch the `Error`.

☒ **A, C, D,** and **F** are incorrect based on the above. (Objective 2.5)

14. Given:

```
 3. public class Clumsy {
 4.   public static void main(String[] args) {
 5.     int j = 7;
 6.     assert(++j > 7);
 7.     assert(++j > 8): "hi";
 8.     assert(j > 10): j=12;
 9.     assert(j==12): doStuff();
10.     assert(j==12): new Clumsy();
11.   }
12.   static void doStuff() { }
13. }
```

Which are true? (Choose all that apply.)

A. Compilation succeeds

B. Compilation fails due to an error on line 6

C. Compilation fails due to an error on line 7

D. Compilation fails due to an error on line 8

E. Compilation fails due to an error on line 9

F. Compilation fails due to an error on line 10

Answer:

☑ **E** is correct. When an `assert` statement has two expressions, the second expression must return a value. The only two-expression `assert` statement that doesn't return a value is on line 9.

☒ **A, B, C, D,** and **F** are incorrect based on the above. (Objective 2.3)

15. Given:

```
1. public class Frisbee {
2.   // insert code here
3.     int x = 0;
4.     System.out.println(7/x);
5.   }
6. }
```

And given the following four code fragments:

```
I.   public static void main(String[] args) {
II.  public static void main(String[] args) throws Exception {
III. public static void main(String[] args) throws IOException {
IV.  public static void main(String[] args) throws RuntimeException {
```

If the four fragments are inserted independently at line 4, which are true? (Choose all that apply.)

A. All four will compile and execute without exception

B. All four will compile and execute and throw an exception

C. Some, but not all, will compile and execute without exception

D. Some, but not all, will compile and execute and throw an exception

E. When considering fragments II, III, and IV, of those that will compile, adding a try/catch block around line 6 will cause compilation to fail

Answer:

☑ **D** is correct. This is kind of sneaky, but remember that we're trying to toughen you up for the real exam. If you're going to throw an IOException, you have to import the java.io package or declare the exception with a fully qualified name.

☒ **E** is incorrect because it's okay to both handle and declare an exception. **A, B,** and **C** are incorrect based on the above. (Objective 2.4)

16. Given:

```
 2. class MyException extends Exception { }
 3. class Tire {
 4.   void doStuff() {  }
 5. }
 6. public class Retread extends Tire {
 7.   public static void main(String[] args) {
 8.     new Retread().doStuff();
 9.   }
10.   // insert code here
11.     System.out.println(7/0);
12.   }
13. }
```

And given the following four code fragments:

```
I.   void doStuff() {
II.  void doStuff() throws MyException {
III. void doStuff() throws RuntimeException {
IV.  void doStuff() throws ArithmeticException {
```

When fragments I - IV are added, independently, at line 10, which are true? (Choose all that apply.)

A. None will compile

B. They will all compile

C. Some, but not all, will compile

D. All of those that compile will throw an exception at runtime

E. None of those that compile will throw an exception at runtime

F. Only some of those that compile will throw an exception at runtime

Answer:

☑ **C** and **D** are correct. An overriding method cannot throw checked exceptions that are broader than those thrown by the overridden method. However an overriding method *can* throw RuntimeExceptions not thrown by the overridden method.

☒ **A, B, E,** and **F** are incorrect based on the above. (Objective 2.4)