SELF TEST ANSWERS

1. Given:

```
class CardBoard {
  Short story = 200;
  CardBoard go(CardBoard cb) {
    cb = null;
    return cb;
  }
  public static void main(String[] args) {
    CardBoard c1 = new CardBoard();
    CardBoard c2 = new CardBoard();
    CardBoard c3 = c1.go(c2);
    c1 = null;
    // do Stuff
} }
```

When `// doStuff` is reached, how many objects are eligible for GC?

A. 0
B. 1
C. 2
D. Compilation fails
E. It is not possible to know
F. An exception is thrown at runtime

Answer:

☑ **C** is correct. Only one `CardBoard object (c1)` is eligible, but it has an associated Short wrapper object that is also eligible.

☒ **A, B, D, E,** and **F** are incorrect based on the above. (Objective 7.4)

2. Given:

```
class Alien {
  String invade(short ships) { return "a few"; }
  String invade(short... ships) { return "many"; }
}
class Defender {
  public static void main(String [] args) {
    System.out.println(new Alien().invade(7));
} }
```

What is the result?

A. `many`
B. `a few`
C. Compilation fails
D. The output is not predictable
E. An exception is thrown at runtime

Answer:

☑ **C** is correct, compilation fails. The var-args declaration is fine, but `invade` takes a `short`, so the argument 7 needs to be cast to a `short`. With the cast, the answer is **B**, '`a few`'.

☒ **A, B, D,** and **E** are incorrect based on the above. (Objective 1.3)

3. Given:

```
1. class Dims {
2.   public static void main(String[] args) {
3.     int[][] a = {{1,2,}, {3,4}};
4.     int[] b = (int[]) a[1];
5.     Object o1 = a;
6.     int[][] a2 = (int[][]) o1;
7.     int[] b2 = (int[]) o1;
8.     System.out.println(b[1]);
9. } }
```

What is the result?

A. `2`
B. `4`
C. An exception is thrown at runtime
D. Compilation fails due to an error on line 4
E. Compilation fails due to an error on line 5
F. Compilation fails due to an error on line 6
G. Compilation fails due to an error on line 7

Answer:

☑ **C** is correct. A `ClassCastException` is thrown at line 7 because `o1` refers to an `int[][]` not an `int[]`. If line 7 was removed, the output would be 4.

☒ **A, B, D, E, F,** and **G** are incorrect based on the above. (Objective 1.3)

4. Given:

```
class Mixer {
  Mixer() { }
  Mixer(Mixer m) { m1 = m; }
  Mixer m1;
  public static void main(String[] args) {
    Mixer m2 = new Mixer();
    Mixer m3 = new Mixer(m2);  m3.go();
    Mixer m4 = m3.m1;          m4.go();
    Mixer m5 = m2.m1;          m5.go();
  }
  void go() { System.out.print("hi "); }
}
```

What is the result?

A. `hi`

B. `hi hi`

C. `hi hi hi`

D. Compilation fails

E. `hi`, followed by an exception

F. `hi hi`, followed by an exception

Answer:

☑ **F** is correct. The `m2` object's `m1` instance variable is never initialized, so when `m5` tries to use it a `NullPointerException` is thrown.

☒ **A, B, C, D,** and **E** are incorrect based on the above. (Objective 7.3)

5. Given:

```
class Fizz {
  int x = 5;
  public static void main(String[] args) {
    final Fizz f1 = new Fizz();
    Fizz f2 = new Fizz();
    Fizz f3 = FizzSwitch(f1,f2);
    System.out.println((f1 == f3) + " " + (f1.x == f3.x));
  }
  static Fizz FizzSwitch(Fizz x, Fizz y) {
    final Fizz z = x;
    z.x = 6;
    return z;
} }
```

What is the result?

A. `true true`
B. `false true`
C. `true false`
D. `false false`
E. Compilation fails
F. An exception is thrown at runtime

Answer:

☑ **A** is correct. The references `f1`, `z`, and `f3` all refer to the same instance of `Fizz`. The `final` modifier assures that a reference variable cannot be referred to a different object, but `final` doesn't keep the object's state from changing.

☒ **B, C, D, E,** and **F** are incorrect based on the above. (Objective 7.3)

6. Given:

```
class Bird {
  { System.out.print("b1 "); }
  public Bird() { System.out.print("b2 "); }
}
class Raptor extends Bird {
  static { System.out.print("r1 "); }
  public Raptor() { System.out.print("r2 "); }
  { System.out.print("r3 "); }
  static { System.out.print("r4 "); }
}
class Hawk extends Raptor {
  public static void main(String[] args) {
    System.out.print("pre ");
    new Hawk();
    System.out.println("hawk ");
  }
}
```

What is the result?

A. `pre b1 b2 r3 r2 hawk`
B. `pre b2 b1 r2 r3 hawk`
C. `pre b2 b1 r2 r3 hawk r1 r4`
D. `r1 r4 pre b1 b2 r3 r2 hawk`
E. `r1 r4 pre b2 b1 r2 r3 hawk`

F. `pre r1 r4 b1 b2 r3 r2 hawk`

G. `pre r1 r4 b2 b1 r2 r3 hawk`

H. The order of output cannot be predicted

I. Compilation fails

Answer:

☑ **D** is correct. Static init blocks are executed at class loading time, instance init blocks run right after the call to `super()` in a constructor. When multiple init blocks of a single type occur in a class, they run in order, from the top down.

☒ **A, B, C, E, F, G, H,** and **I** are incorrect based on the above. Note: you'll probably never see this many choices on the real exam! (Objective 1.3)

7. Given:

```
3. public class Bridge {
4.   public enum Suits {
5.     CLUBS(20), DIAMONDS(20), HEARTS(30), SPADES(30),
6.     NOTRUMP(40) { public int getValue(int bid) {
                       return ((bid-1)*30)+40; } };
7.     Suits(int points) {  this.points = points;  }
8.     private int points;
9.     public int getValue(int bid) { return points * bid; }
10.  }
11.  public static void main(String[] args) {
12.    System.out.println(Suits.NOTRUMP.getBidValue(3));
13.    System.out.println(Suits.SPADES + " " + Suits.SPADES.points);
14.    System.out.println(Suits.values());
15.  }
16. }
```

Which are true? (Choose all that apply.)

A. The output could contain `30`

B. The output could contain `@bf73fa`

C. The output could contain `DIAMONDS`

D. Compilation fails due to an error on line 6

E. Compilation fails due to an error on line 7

F. Compilation fails due to an error on line 8

G. Compilation fails due to an error on line 9

H. Compilation fails due to an error within lines 12 to 14

Answer:

☑ **A** and **B** are correct. The code compiles and runs without exception. The `values()` method returns an array reference, not the contents of the enum, so DIAMONDS is never printed.

☒ **C, D, E, F, G,** and **H** are incorrect based on the above. (Objective 1.3)

8. Given:

```
3. public class Ouch {
4.   static int ouch = 7;
5.   public static void main(String[] args) {
6.     new Ouch().go(ouch);
7.     System.out.print(" " + ouch);
8.   }
9.   void go(int ouch) {
10.    ouch++;
11.    for(int ouch = 3; ouch < 6; ouch++)
12.      ;
13.    System.out.print(" " + ouch);
14.  }
15. }
```

What is the result?

A. `5 7`

B. `5 8`

C. `8 7`

D. `8 8`

E. Compilation fails

F. An exception is thrown at runtime

Answer:

☑ **E** is correct. The parameter declared on line 9 is valid (although ugly), but the variable name `ouch` cannot be declared again on line 11 in the same scope as the declaration on line 9.

☒ **A, B, C, D,** and **F** are incorrect based on the above. (Objective 1.3)

9. Given:

```
3. public class Bertha {
4.   static String s = "";
5.   public static void main(String[] args) {
6.     int x = 4; Boolean y = true; short[] sa = {1,2,3};
7.     doStuff(x, y);
8.     doStuff(x);
9.     doStuff(sa, sa);
10.    System.out.println(s);
11.  }
12.  static void doStuff(Object o)        { s += "1"; }
13.  static void doStuff(Object... o)     { s += "2"; }
14.  static void doStuff(Integer... i)    { s += "3"; }
15.  static void doStuff(Long L)          { s += "4"; }
16. }
```

What is the result?

A. `212`

B. `232`

C. `234`

D. `312`

E. `332`

F. `334`

G. Compilation fails

Answer:

☑ **A** is correct. It's legal to autobox and then widen. The first call to `doStuff()` boxes the int to an Integer then passes two objects. The second call cannot widen and then box (making the `Long` method unusable), so it boxes the int to an Integer. As always, a var-args method will be chosen only if no non-var-arg method is possible. The third call is passing two objects—they are of type 'short array.'

☒ **B, C, D, E, F,** and **G** are incorrect based on the above. (Objective 3.1)

10. Given:

```
3. class Dozens {
4.   int[] dz = {1,2,3,4,5,6,7,8,9,10,11,12};
5. }
6. public class Eggs {
7.   public static void main(String[] args) {
```

```
 8.     Dozens [] da = new Dozens[3];
 9.     da[0] = new Dozens();
10.     Dozens d = new Dozens();
11.     da[1] = d;
12.     d = null;
13.     da[1] = null;
14.     // do stuff
15.   }
16. }
```

Which two are true about the objects created within `main()`, and eligible for garbage collection when line 14 is reached?

A. Three objects were created

B. Four objects were created

C. Five objects were created

D. Zero objects are eligible for GC

E. One object is eligible for GC

F. Two objects are eligible for GC

G. Three objects are eligible for GC

Answer:

☑ **C** and **F** are correct. `da` refers to an object of type "Dozens array," and each Dozens object that is created comes with its own "int array" object. When line 14 is reached, only the second Dozens object (and its "int array" object) are not reachable.

☒ **A, B, D, E,** and **G** are incorrect based on the above. (Objective 7.4)

11. Given:

```
 3. class Beta { }
 4. class Alpha {
 5.   static Beta b1;
 6.   Beta b2;
 7. }
 8. public class Tester {
 9.   public static void main(String[] args) {
10.     Beta b1 = new Beta();     Beta b2 = new Beta();
11.     Alpha a1 = new Alpha();   Alpha a2 = new Alpha();
12.     a1.b1 = b1;
13.     a1.b2 = b1;
14.     a2.b2 = b2;
15.     a1 = null;  b1 = null;  b2 = null;
```

```
16.      // do stuff
17.    }
18. }
```

When line 16 is reached, how many objects will be eligible for garbage collection?

A. 0

B. 1

C. 2

D. 3

E. 4

F. 5

Answer:

☑ **B** is correct. It should be clear that there is still a reference to the object referred to by `a2`, and that there is still a reference to the object referred to by `a2.b2`. What might be less clear is that you can still access the other Beta object through the static variable `a2.b1`—because it's static.

☒ **A, C, D, E,** and **F** are incorrect based on the above. (Objective 7.4)

12. Given:

```
3. class Box {
4.   int size;
5.   Box(int s) { size = s; }
6. }
7. public class Laser {
8.   public static void main(String[] args) {
9.     Box b1 = new Box(5);
10.    Box[] ba = go(b1, new Box(6));
11.    ba[0] = b1;
12.    for(Box b : ba) System.out.print(b.size + " ");
13.  }
14.  static Box[] go(Box b1, Box b2) {
15.     b1.size = 4;
16.     Box[] ma = {b2, b1};
17.     return ma;
18.  }
19. }
```

What is the result?

A. `4 4`

B. `5 4`

C. `6 4`

D. `4 5`

E. `5 5`

F. Compilation fails

Answer:

☑ **A** is correct. Although `main()`'s `b1` is a different reference variable than `go()`'s `b1`, they refer to the same Box object.

☒ **B, C, D, E,** and **F** are incorrect based on the above. (Objective 7.3)

13. Given:

```
3. public class Dark {
4.   int x = 3;
5.   public static void main(String[] args) {
6.     new Dark().go1();
7.   }
8.   void go1() {
9.     int x;
10.    go2(++x);
11.  }
12.  void go2(int y) {
13.    int x = ++y;
14.    System.out.println(x);
15.  }
16. }
```

What is the result?

A. 2

B. 3

C. 4

D. 5

E. Compilation fails

F. An exception is thrown at runtime

Answer:

☑ **E** is correct. In `go1()` the local variable `x` is not initialized.

☒ **A, B, C, D,** and **F** are incorrect based on the above. (Objective 1.3)