SELF TEST

The following questions will help you measure your understanding of the dynamic and life-altering material presented in this chapter. Read all of the choices carefully. Take your time. Breathe.

1. Which are true about a static nested class? (Choose all that apply.)
   A. You must have a reference to an instance of the enclosing class in order to instantiate it
   B. It does not have access to non-`static` members of the enclosing class
   C. Its variables and methods must be `static`
   D. If the outer class is named `MyOuter`, and the nested class is named `MyInner`, it can be instantiated using `new MyOuter.MyInner();`
   E. It must extend the enclosing class

2. Given:

```
class Boo {
   Boo(String s) { }
   Boo() { }
}
class Bar extends Boo {
   Bar() { }
   Bar(String s) {super(s);}
   void zoo() {
   // insert code here
   }
}
```

   Which create an anonymous inner class from within class Bar? (Choose all that apply.)
   A. `Boo f = new Boo(24) { };`
   B. `Boo f = new Bar() { };`
   C. `Boo f = new Boo() {String s; };`
   D. `Bar f = new Boo(String s) { };`
   E. `Boo f = new Boo.Bar(String s) { };`

3. Which are true about a method-local inner class? (Choose all that apply.)
   A. It must be marked `final`
   B. It can be marked `abstract`

C. It can be marked `public`
D. It can be marked `static`
E. It can access private members of the enclosing class

4. Given:

```
1. public class TestObj {
2.   public static void main(String[] args) {
3.     Object o = new Object() {
4.       public boolean equals(Object obj) {
5.         return true;
6.       }
7.     }
8.     System.out.println(o.equals("Fred"));
9.   }
10. }
```

What is the result?
A. An exception occurs at runtime
B. `true`
C. `Fred`
D. Compilation fails because of an error on line 3
E. Compilation fails because of an error on line 4
F. Compilation fails because of an error on line 8
G. Compilation fails because of an error on a line other than 3, 4, or 8

5. Given:

```
1. public class HorseTest {
2.   public static void main(String[] args) {
3.     class Horse {
4.       public String name;
5.       public Horse(String s) {
6.         name = s;
7.       }
8.     }
9.     Object obj = new Horse("Zippo");
10.    System.out.println(obj.name);
11.  }
12. }
```

What is the result?

A. An exception occurs at runtime at line 10
B. `Zippo`
C. Compilation fails because of an error on line 3
D. Compilation fails because of an error on line 9
E. Compilation fails because of an error on line 10

6. Given:

```
public abstract class AbstractTest {
   public int getNum() {
      return 45;
   }
   public abstract class Bar {
     public int getNum() {
       return 38;
     }
   }
   public static void main(String[] args) {
      AbstractTest t = new AbstractTest() {
         public int getNum() {
           return 22;
         }
      };
      AbstractTest.Bar f = t.new Bar() {
         public int getNum() {
           return 57;
         }
      };
      System.out.println(f.getNum() + " " + t.getNum());
   }
}
```

What is the result?

A. `57 22`
B. `45 38`
C. `45 57`
D. An exception occurs at runtime
E. Compilation fails

7. Given:

```
3. public class Tour {
4.   public static void main(String[] args) {
5.     Cathedral c = new Cathedral();
6.     // insert code here
7.     s.go();
8.   }
9. }
10. class Cathedral {
11.   class Sanctum {
12.     void go() { System.out.println("spooky"); }
13.   }
14. }
```

Which, inserted independently at line 6, compile and produce the output "spooky"? (Choose all that apply.)

A. `Sanctum s = c.new Sanctum();`
B. `c.Sanctum s = c.new Sanctum();`
C. `c.Sanctum s = Cathedral.new Sanctum();`
D. `Cathedral.Sanctum s = c.new Sanctum();`
E. `Cathedral.Sanctum s = Cathedral.new Sanctum();`

8. Given:

```
5. class A { void m() { System.out.println("outer"); } }
6.
7. public class TestInners {
8.   public static void main(String[] args) {
9.     new TestInners().go();
10.   }
11.   void go() {
12.     new A().m();
13.     class A { void m() { System.out.println("inner"); } }
14.   }
15.   class A { void m() { System.out.println("middle"); } }
16. }
```

What is the result?

A. `inner`
B. `outer`

C. `middle`
D. Compilation fails
E. An exception is thrown at runtime

9. Given:

```
3. public class Car {
4.   class Engine {
5.     // insert code here
6.   }
7.   public static void main(String[] args) {
8.     new Car().go();
9.   }
10.  void go() {
11.    new Engine();
12.  }
13.  void drive() { System.out.println("hi"); }
14. }
```

Which, inserted independently at line 5, produce the output "hi"? (Choose all that apply.)

A. `{ Car.drive(); }`
B. `{ this.drive(); }`
C. `{ Car.this.drive(); }`
D. `{ this.Car.this.drive(); }`
E. `Engine() { Car.drive(); }`
F. `Engine() { this.drive(); }`
G. `Engine() { Car.this.drive(); }`

10. Given:

```
3. public class City {
4.   class Manhattan {
5.     void doStuff() throws Exception { System.out.print("x "); }
6.   }
7.   class TimesSquare extends Manhattan {
8.     void doStuff() throws Exception { }
9.   }
10.  public static void main(String[] args) throws Exception {
11.    new City().go();
12.  }
13.  void go() throws Exception { new TimesSquare().doStuff(); }
14. }
```

What is the result?

A. `x`
B. `x x`
C. No output is produced
D. Compilation fails due to multiple errors
E. Compilation fails due only to an error on line 4
F. Compilation fails due only to an error on line 7
G. Compilation fails due only to an error on line 10
H. Compilation fails due only to an error on line 13

11. Given:

```
 3. public class Navel {
 4.   private int size = 7;
 5.   private static int length = 3;
 6.   public static void main(String[] args) {
 7.     new Navel().go();
 8.   }
 9.   void go() {
10.     int size = 5;
11.     System.out.println(new Gazer().adder());
12.   }
13.   class Gazer {
14.     int adder() { return size * length; }
15.   }
16. }
```

What is the result?

A. `15`
B. `21`
C. An exception is thrown at runtime
D. Compilation fails due to multiple errors
E. Compilation fails due only to an error on line 4
F. Compilation fails due only to an error on line 5

12. Given:

```
 3. import java.util.*;
 4. public class Pockets {
 5.   public static void main(String[] args) {
 6.     String[] sa = {"nickel", "button", "key", "lint"};
 7.     Sorter s = new Sorter();
 8.     for(String s2: sa) System.out.print(s2 + " ");
 9.     Arrays.sort(sa,s);
10.     System.out.println();
11.     for(String s2: sa) System.out.print(s2 + " ");
12.   }
13.   class Sorter implements Comparator<String> {
14.     public int compare(String a, String b) {
15.       return b.compareTo(a);
16.     }
17.   }
18. }
```

What is the result?

A. Compilation fails

B.
```
button key lint nickel
nickel lint key button
```

C.
```
nickel button key lint
button key lint nickel
```

D.
```
nickel button key lint
nickel button key lint
```

E.
```
nickel button key lint
nickel lint key button
```

F. An exception is thrown at runtime

SELF TEST ANSWERS

1. Which are true about a static nested class? (Choose all that apply.)
   A. You must have a reference to an instance of the enclosing class in order to instantiate it
   B. It does not have access to non-`static` members of the enclosing class
   C. Its variables and methods must be `static`
   D. If the outer class is named `MyOuter`, and the nested class is named `MyInner`, it can be instantiated using `new MyOuter.MyInner();`
   E. It must extend the enclosing class

Answer:

☑ **B** and **D. B** is correct because a static nested class is not tied to an instance of the enclosing class, and thus can't access the non-`static` members of the class (just as a `static` method can't access non-static members of a class). **D** uses the correct syntax for instantiating a static nested class.

☒ **A** is incorrect because static nested classes do not need (and can't use) a reference to an instance of the enclosing class. **C** is incorrect because static nested classes can declare and define non-static members. **E** is wrong because…it just is. There's no rule that says an inner or nested class has to extend anything.

2. Given:

```
class Boo {
   Boo(String s) { }
   Boo() { }
}
class Bar extends Boo {
   Bar() { }
   Bar(String s) {super(s);}
   void zoo() {
   // insert code here
   }
}
```

Which create an anonymous inner class from within class Bar? (Choose all that apply.)

A. `Boo f = new Boo(24) { };`
B. `Boo f = new Bar() { };`

C. `Boo f = new Boo() {String s; };`
D. `Bar f = new Boo(String s) { };`
E. `Boo f = new Boo.Bar(String s) { };`

Answer:

☑ **B** and **C. B** is correct because anonymous inner classes are no different from any other class when it comes to polymorphism. That means you are always allowed to declare a reference variable of the superclass type and have that reference variable refer to an instance of a subclass type, which in this case is an anonymous subclass of Bar. Since Bar is a subclass of Boo, it all works. **C** uses correct syntax for creating an instance of Boo.

☒ **A** is incorrect because it passes an `int` to the Boo constructor, and there is no matching constructor in the Boo class. **D** is incorrect because it violates the rules of polymorphism; you cannot refer to a superclass type using a reference variable declared as the subclass type. The superclass doesn't have everything the subclass has. **E** uses incorrect syntax.

3. Which are true about a method-local inner class? (Choose all that apply.)

A. It must be marked `final`
B. It can be marked `abstract`
C. It can be marked `public`
D. It can be marked `static`
E. It can access private members of the enclosing class

Answer:

☑ **B** and **E. B** is correct because a method-local inner class can be `abstract`, although it means a subclass of the inner class must be created if the `abstract` class is to be used (so an `abstract` method-local inner class is probably not useful). **E** is correct because a method-local inner class works like any other inner class—it has a special relationship to an instance of the enclosing class, thus it can access all members of the enclosing class.

☒ **A** is incorrect because a method-local inner class does not have to be declared `final` (although it is legal to do so). **C** and **D** are incorrect because a method-local inner class cannot be made `public` (remember—local variables can't be `public`) or `static`.

4. Given:

```
1. public class TestObj {
2.   public static void main(String[] args) {
3.     Object o = new Object() {
```

```
4.        public boolean equals(Object obj) {
5.          return true;
6.        }
7.      }
8.      System.out.println(o.equals("Fred"));
9.    }
10. }
```

What is the result?

A. An exception occurs at runtime
B. `true`
C. `fred`
D. Compilation fails because of an error on line 3
E. Compilation fails because of an error on line 4
F. Compilation fails because of an error on line 8
G. Compilation fails because of an error on a line other than 3, 4, or 8

Answer:

☑ **G**. This code would be legal if line 7 ended with a semicolon. Remember that line 3 is a statement that doesn't end until line 7, and a statement needs a closing semicolon!

☒ **A, B, C, D, E,** and **F** are incorrect based on the program logic described above. If the semicolon were added at line 7, then answer **B** would be correct—the program would print `true`, the return from the `equals()` method overridden by the anonymous subclass of `Object`.

5. Given:

```
1. public class HorseTest {
2.   public static void main(String[] args) {
3.     class Horse {
4.       public String name;
5.       public Horse(String s) {
6.         name = s;
7.       }
8.     }
9.     Object obj = new Horse("Zippo");
10.    System.out.println(obj.name);
11.  }
12. }
```

What is the result?

A. An exception occurs at runtime at line 10
B. `Zippo`
C. Compilation fails because of an error on line 3
D. Compilation fails because of an error on line 9
E. Compilation fails because of an error on line 10

Answer:

☑ **E.** If you use a reference variable of type `Object`, you can access only those members defined in class `Object`.

☒ **A, B, C,** and **D** are incorrect based on the program logic described above.

6. Given:

```
public abstract class AbstractTest {
   public int getNum() {
      return 45;
   }
   public abstract class Bar {
     public int getNum() {
       return 38;
     }
   }
   public static void main(String[] args) {
      AbstractTest t = new AbstractTest() {
         public int getNum() {
           return 22;
         }
      };
      AbstractTest.Bar f = t.new Bar() {
         public int getNum() {
           return 57;
         }
      };
      System.out.println(f.getNum() + " " + t.getNum());
} }
```

What is the result?

A. `57 22`
B. `45 38`
C. `45 57`
D. An exception occurs at runtime
E. Compilation fails

Answer:

☑ **A**. You can define an inner class as `abstract`, which means you can instantiate only concrete subclasses of the abstract inner class. The object referenced by the variable `t` is an instance of an anonymous subclass of AbstractTest, and the anonymous class overrides the `getNum()` method to return `22`. The variable referenced by `f` is an instance of an anonymous subclass of `Bar`, and the anonymous Bar subclass also overrides the `getNum()` method (to return `57`). Remember that to create a Bar instance, we need an instance of the enclosing AbstractTest class to tie to the new `Bar` inner class instance. AbstractTest can't be instantiated because it's `abstract`, so we created an anonymous subclass (non-`abstract`) and then used the instance of that anonymous subclass to tie to the new Bar subclass instance.

☒ **B, C, D,** and **E** are incorrect based on the program logic described above.

7. Given:

```
 3. public class Tour {
 4.   public static void main(String[] args) {
 5.     Cathedral c = new Cathedral();
 6.     // insert code here
 7.     s.go();
 8.   }
 9. }
10. class Cathedral {
11.   class Sanctum {
12.     void go() { System.out.println("spooky"); }
13.   }
14. }
```

Which, inserted independently at line 6, compile and produce the output "spooky"? (Choose all that apply.)

A. `Sanctum s = c.new Sanctum();`
B. `c.Sanctum s = c.new Sanctum();`
C. `c.Sanctum s = Cathedral.new Sanctum();`
D. `Cathedral.Sanctum s = c.new Sanctum();`
E. `Cathedral.Sanctum s = Cathedral.new Sanctum();`

Answer:

☑ **D** is correct. It is the only code that uses the correct inner class instantiation syntax.

☒ **A, B, C,** and **E** are incorrect based on the above. (Objective 1.1)

8. Given:

```
 5. class A { void m() { System.out.println("outer"); } }
 6.
 7. public class TestInners {
 8.   public static void main(String[] args) {
 9.     new TestInners().go();
10.   }
11.   void go() {
12.     new A().m();
13.     class A { void m() { System.out.println("inner"); } }
14.   }
15.   class A { void m() { System.out.println("middle"); } }
16. }
```

What is the result?

A. `inner`
B. `outer`
C. `middle`
D. Compilation fails
E. An exception is thrown at runtime

Answer:

☑ **C** is correct. The `"inner"` version of `class A` isn't used because its declaration comes after the instance of `class A` is created in the `go()` method.

☒ **A, B, D,** and **E** are incorrect based on the above. (Objective 1.1)

9. Given:

```
 3. public class Car {
 4.   class Engine {
 5.     // insert code here
 6.   }
 7.   public static void main(String[] args) {
 8.     new Car().go();
 9.   }
10.   void go() {
11.     new Engine();
12.   }
13.   void drive() { System.out.println("hi"); }
14. }
```

Which, inserted independently at line 5, produce the output "hi"? (Choose all that apply.)

A. `{ Car.drive(); }`
B. `{ this.drive(); }`
C. `{ Car.this.drive(); }`
D. `{ this.Car.this.drive(); }`
E. `Engine() { Car.drive(); }`
F. `Engine() { this.drive(); }`
G. `Engine() { Car.this.drive(); }`

Answer:

☑ **C** and **G** are correct. **C** is the correct syntax to access an inner class's outer instance method from an initialization block, and **G** is the correct syntax to access it from a constructor.

☒ **A, B, D, E,** and **F** are incorrect based on the above. (Objectives 1.1, 1.4)

10. Given:

```
 3. public class City {
 4.   class Manhattan {
 5.     void doStuff() throws Exception { System.out.print("x "); }
 6.   }
 7.   class TimesSquare extends Manhattan {
 8.     void doStuff() throws Exception { }
 9.   }
10.   public static void main(String[] args) throws Exception {
11.     new City().go();
12.   }
13.   void go() throws Exception { new TimesSquare().doStuff(); }
14. }
```

What is the result?

A. `x`
B. `x x`
C. No output is produced
D. Compilation fails due to multiple errors
E. Compilation fails due only to an error on line 4
F. Compilation fails due only to an error on line 7
G. Compilation fails due only to an error on line 10
H. Compilation fails due only to an error on line 13

Answer:

☑ **C** is correct. The inner classes are valid, and all the methods (including `main()`), correctly throw an Exception, given that `doStuff()` throws an Exception. The `doStuff()` in class `TimesSquare` overrides class `Manhattan`'s `doStuff()` and produces no output.

☒ **A, B, D, E, F, G,** and **H** are incorrect based on the above. (Objectives 1.1, 2.4)

11. Given:

```
 3. public class Navel {
 4.   private int size = 7;
 5.   private static int length = 3;
 6.   public static void main(String[] args) {
 7.     new Navel().go();
 8.   }
 9.   void go() {
10.     int size = 5;
11.     System.out.println(new Gazer().adder());
12.   }
13.   class Gazer {
14.     int adder() { return size * length; }
15.   }
16. }
```

What is the result?

A. `15`
B. `21`
C. An exception is thrown at runtime
D. Compilation fails due to multiple errors
E. Compilation fails due only to an error on line 4
F. Compilation fails due only to an error on line 5

Answer:

☑ **B** is correct. The inner class `Gazer` has access to `Navel`'s `private static` and private instance variables.

☒ **A, C, D, E,** and **F** are incorrect based on the above. (Objectives 1.1, 1.4)

12. Given:

```
 3. import java.util.*;
 4. public class Pockets {
 5.   public static void main(String[] args) {
 6.     String[] sa = {"nickel", "button", "key", "lint"};
 7.     Sorter s = new Sorter();
 8.     for(String s2: sa) System.out.print(s2 + " ");
 9.     Arrays.sort(sa,s);
10.     System.out.println();
11.     for(String s2: sa) System.out.print(s2 + " ");
12.   }
13.   class Sorter implements Comparator<String> {
14.     public int compare(String a, String b) {
15.       return b.compareTo(a);
16.     }
17.   }
18. }
```

What is the result?

A. Compilation fails

B. `button key lint nickel`
`nickel lint key button`

C. `nickel button key lint`
`button key lint nickel`

D. `nickel button key lint`
`nickel button key lint`

E. `nickel button key lint`
`nickel lint key button`

F. An exception is thrown at runtime

Answer:

☑ **A** is correct, the inner class `Sorter` must be declared `static` to be called from the `static` method `main()`. If `Sorter` had been `static`, answer E would be correct.

☒ **B, C, D, E,** and **F** are incorrect based on the above. (Objectives 1.1, 1.4, 6.5)