# SELF TEST

1. Given:

```
import java.util.regex.*;
class Regex2 {
  public static void main(String[] args) {
    Pattern p = Pattern.compile(args[0]);
    Matcher m = p.matcher(args[1]);
    boolean b = false;
    while(b = m.find()) {
      System.out.print(m.start() + m.group());
    }
  }
}
```

And the command line:

```
java Regex2 "\d*" ab34ef
```

What is the result?

A. `234`

B. `334`

C. `2334`

D. `0123456`

E. `01234456`

F. `12334567`

G. Compilation fails

2. Given:

```
import java.io.*;
class Player {
  Player() { System.out.print("p"); }
}
class CardPlayer extends Player implements Serializable {
  CardPlayer() { System.out.print("c"); }
```

```
  public static void main(String[] args) {
    CardPlayer c1 = new CardPlayer();
    try {
      FileOutputStream fos = new FileOutputStream("play.txt");
      ObjectOutputStream os = new ObjectOutputStream(fos);
      os.writeObject(c1);
      os.close();
      FileInputStream fis = new FileInputStream("play.txt");
      ObjectInputStream is = new ObjectInputStream(fis);
      CardPlayer c2 = (CardPlayer) is.readObject();
      is.close();
    } catch (Exception x ) { }
  }
}
```

What is the result?

A. `pc`

B. `pcc`

C. `pcp`

D. `pcpc`

E. Compilation fails

F. An exception is thrown at runtime

**3.** Given:

```
class TKO {
  public static void main(String[] args) {
    String s = "-";
    Integer x = 343;
    long L343 = 343L;
    if(x.equals(L343))  s += ".e1 ";
    if(x.equals(343))  s += ".e2 ";
    Short s1 = (short)((new Short((short)343)) / (new Short((short)49)));
    if(s1 == 7)        s += "=s ";
    if(s1 < new Integer(7+1))  s += "fly ";
    System.out.println(s);
  }
}
```

Which of the following will be included in the output String `s`? (Choose all that apply.)

A. `.e1`
B. `.e2`
C. `=s`
D. `fly`
E. None of the above
F. Compilation fails
G. An exception is thrown at runtime

**4.** Given:

```
import java.io.*;

class Keyboard { }
public class Computer implements Serializable {
  private Keyboard k = new Keyboard();
  public static void main(String[] args) {
    Computer c = new Computer();
    c.storeIt(c);
  }
  void storeIt(Computer c) {
    try {
      ObjectOutputStream os = new ObjectOutputStream(
         new FileOutputStream("myFile"));
      os.writeObject(c);
      os.close();
      System.out.println("done");
    } catch (Exception x) {System.out.println("exc"); }
  }
}
```

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

A. `exc`
B. `done`
C. Compilation fails
D. Exactly one object is serialized
E. Exactly two objects are serialized

**5.** Using the fewest **fragments** possible (and filling the fewest slots possible), complete the **code** below so that the class builds a directory named "`dir3`" and creates a file named "`file3`" inside "`dir3`". Note you can use each fragment either zero or one times.

Code:

```
import java.io._____________

class Maker {
  public static void main(String[] args) {

    __________  __________  __________

    __________  __________  __________

    __________  __________  __________

    __________  __________  __________

    __________  __________  __________

    __________  __________  __________

    __________  __________  __________
  }
}
```

Fragments:

```
File;       FileDescriptor;      FileWriter;     Directory;
try {       .createNewDir();     File dir        File
{ }         (Exception x)        ("dir3");       file
file        .createNewFile();    = new File      = new File
dir         (dir, "file3");      (dir, file);    .createFile();
} catch     ("dir3", "file3");   .mkdir();       File file
```

**6.** Given that 1119280000000L is roughly the number of milliseconds from Jan 1, 1970, to June 20, 2005, and that you want to print that date in German, using the `LONG` style such that "`June`" will be displayed as "`Juni`", complete the code using the fragments below. Note: you can use each fragment either zero or one times, and you might not need to fill all of the slots.

Code:

```
import java._____________

import java._____________

class DateTwo {
  public static void main(String[] args) {
    Date d = new Date(1119280000000L);

    DateFormat df = _________________________________

                    _________________ , _________________ );

    System.out.println(_________________
  }
}
```

Fragments:

```
io.*;      new DateFormat(                Locale.LONG
nio.*;     DateFormat.getInstance(        Locale.GERMANY
util.*;    DateFormat.getDateInstance(    DateFormat.LONG
text.*;    util.regex;                    DateFormat.GERMANY
date.*;    df.format(d));                 d.format(df));
```

**7.** Given:

```
import java.io.*;
class Directories {
  static String [] dirs = {"dir1", "dir2"};
  public static void main(String [] args) {
    for (String d : dirs) {

      // insert code 1 here

      File file = new File(path, args[0]);

      // insert code 2 here
    }
  }
}
```

and that the invocation

```
java Directories file2.txt
```

is issued from a directory that has two subdirectories, "dir1" and "dir2", and that "dir1" has a file "file1.txt" and "dir2" has a file "file2.txt", and the output is "false true"; which set(s) of code fragments must be inserted? (Choose all that apply.)

A.
```
String path = d;

System.out.print(file.exists() + " ");
```

B.
```
String path = d;

System.out.print(file.isFile() + " ");
```

C.
```
String path = File.separator + d;

System.out.print(file.exists() + " ");
```

D.
```
String path = File.separator + d;

System.out.print(file.isFile() + " ");
```

**8.** Given:

```
import java.io.*;

public class TestSer {
  public static void main(String[] args) {
    SpecialSerial s = new SpecialSerial();
    try {
      ObjectOutputStream os = new ObjectOutputStream(
         new FileOutputStream("myFile"));
      os.writeObject(s);  os.close();
      System.out.print(++s.z + " ");

      ObjectInputStream is = new ObjectInputStream(
         new FileInputStream("myFile"));
      SpecialSerial s2 = (SpecialSerial)is.readObject();
      is.close();
```

```
      System.out.println(s2.y + " " + s2.z);
    } catch (Exception x) {System.out.println("exc"); }
  }
}
class SpecialSerial implements Serializable {
  transient int y = 7;
  static int z = 9;
}
```

Which are true? (Choose all that apply.)

A. Compilation fails

B. The output is `10 0 9`

C. The output is `10 0 10`

D. The output is `10 7 9`

E. The output is `10 7 10`

F. In order to alter the standard deserialization process you would implement the `readObject()` method in `SpecialSerial`

G. In order to alter the standard deserialization process you would implement the `defaultReadObject()` method in `SpecialSerial`

**9.** Given:

```
 3. public class Theory {
 4.   public static void main(String[] args) {
 5.     String s1 = "abc";
 6.     String s2 = s1;
 7.     s1 += "d";
 8.     System.out.println(s1 + " " + s2 + " " + (s1==s2));
 9.
10.     StringBuffer sb1 = new StringBuffer("abc");
11.     StringBuffer sb2 = sb1;
12.     sb1.append("d");
13.     System.out.println(sb1 + " " + sb2 + " " + (sb1==sb2));
14.   }
15. }
```

Which are true? (Choose all that apply.)

A. Compilation fails

B. The first line of output is `abc abc true`

C. The first line of output is `abc abc false`

D. The first line of output is `abcd abc false`

E. The second line of output is `abcd abc false`

F. The second line of output is `abcd abcd true`

G. The second line of output is `abcd abcd false`

**10.** Given:

```
 3. import java.io.*;
 4. public class ReadingFor {
 5.   public static void main(String[] args) {
 6.     String s;
 7.     try {
 8.       FileReader fr = new FileReader("myfile.txt");
 9.       BufferedReader br = new BufferedReader(fr);
10.       while((s = br.readLine()) != null)
11.         System.out.println(s);
12.       br.flush();
13.     } catch (IOException e) { System.out.println("io error"); }
16.   }
17. }
```

And given that myfile.txt contains the following two lines of data:

```
ab
cd
```

What is the result?

A. `ab`

B. `abcd`

C.
```
ab
cd
```

D.
```
a
b
c
d
```

E. Compilation fails

**11.** Given:

```
 3. import java.io.*;
 4. public class Talker {
 5.   public static void main(String[] args) {
 6.     Console c = System.console();
 7.     String u = c.readLine("%s", "username: ");
 8.     System.out.println("hello " + u);
 9.     String pw;
10.     if(c != null && (pw = c.readPassword("%s", "password: ")) != null)
11.       // check for valid password
12.   }
13. }
```

If line 6 creates a valid Console object, and if the user enters *fred* as a username and *1234* as a password, what is the result? (Choose all that apply.)

A. `username:`
`password:`

B. `username: fred`
`password:`

C. `username: fred`
`password: 1234`

D. Compilation fails

E. An exception is thrown at runtime

**12.** Given:

```
 3. import java.io.*;
 4. class Vehicle { }
 5. class Wheels { }
 6. class Car extends Vehicle implements Serializable {  }
 7. class Ford extends Car { }
 8. class Dodge extends Car {
 9.   Wheels w = new Wheels();
10. }
```

Instances of which class(es) can be serialized? (Choose all that apply.)

A. Car

B. Ford

C. Dodge
D. Wheels
E. Vehicle

**13.** Given:

```
 3. import java.text.*;
 4. public class Slice {
 5.   public static void main(String[] args) {
 6.     String s = "987.123456";
 7.     double d  = 987.123456d;
 8.     NumberFormat nf = NumberFormat.getInstance();
 9.     nf.setMaximumFractionDigits(5);
10.     System.out.println(nf.format(d) + " ");
11.     try {
12.       System.out.println(nf.parse(s));
13.     } catch (Exception e) { System.out.println("got exc"); }
14.   }
15. }
```

Which are true? (Choose all that apply.)

A. The output is `987.12345 987.12345`
B. The output is `987.12346 987.12345`
C. The output is `987.12345 987.123456`
D. The output is `987.12346 987.123456`
E. The try/catch block is unnecessary
F. The code compiles and runs without exception
G. The invocation of `parse()` must be placed within a try/catch block

**14.** Given:

```
 3. import java.util.regex.*;
 4. public class Archie {
 5.   public static void main(String[] args) {
 6.     Pattern p = Pattern.compile(args[0]);
 7.     Matcher m = p.matcher(args[1]);
 8.     int count = 0;
 9.     while(m.find())
10.       count++;
```

```
11.      System.out.print(count);
12.    }
13. }
```

And given the command line invocation:

```
java Archie "\d+" ab2c4d67
```

What is the result?

A. 0

B. 3

C. 4

D. 8

E. 9

F. Compilation fails

**15.** Given:

```
 3. import java.util.*;
 4. public class Looking {
 5.   public static void main(String[] args) {
 6.     String input = "1 2 a 3 45 6";
 7.     Scanner sc = new Scanner(input);
 8.     int x = 0;
 9.     do {
10.       x = sc.nextInt();
11.       System.out.print(x + " ");
12.     } while (x!=0);
13.   }
14. }
```

What is the result?

A. `1 2`

B. `1 2 3 45 6`

C. `1 2 3 4 5 6`

D. `1 2 a 3 45 6`

E. Compilation fails

F. `1 2` followed by an exception

# SELF TEST ANSWERS

1. Given:

```
import java.util.regex.*;
class Regex2 {
  public static void main(String[] args) {
    Pattern p = Pattern.compile(args[0]);
    Matcher m = p.matcher(args[1]);
    boolean b = false;
    while(b = m.find()) {
      System.out.print(m.start() + m.group());
    }
  }
}
```

And the command line:

```
java Regex2 "\d*" ab34ef
```

What is the result?

A. `234`

B. `334`

C. `2334`

D. `0123456`

E. `01234456`

F. `12334567`

G. Compilation fails

Answer:

☑ **E** is correct. The `\d` is looking for digits. The `*` is a quantifier that looks for `0` to many occurrences of the pattern that precedes it. Because we specified `*`, the `group()` method returns empty Strings until consecutive digits are found, so the only time `group()` returns a value is when it returns 34 when the matcher finds digits starting in position 2. The `start()` method returns the starting position of the previous match because, again, we said find `0` to many occurrences.

☒ **A, B, C, D, F,** and **G** are incorrect based on the above. (Objective 3.5)

**2.** Given:

```
import java.io.*;
class Player {
  Player() { System.out.print("p"); }
}
class CardPlayer extends Player implements Serializable {
  CardPlayer() { System.out.print("c"); }
  public static void main(String[] args) {
    CardPlayer c1 = new CardPlayer();
    try {
      FileOutputStream fos = new FileOutputStream("play.txt");
      ObjectOutputStream os = new ObjectOutputStream(fos);
      os.writeObject(c1);
      os.close();
      FileInputStream fis = new FileInputStream("play.txt");
      ObjectInputStream is = new ObjectInputStream(fis);
      CardPlayer c2 = (CardPlayer) is.readObject();
      is.close();
    } catch (Exception x ) { }
  }
}
```

What is the result?

A. `pc`

B. `pcc`

C. `pcp`

D. `pcpc`

E. Compilation fails

F. An exception is thrown at runtime

Answer:

☑ **C** is correct. It's okay for a class to implement Serializable even if its superclass doesn't. However, when you deserialize such an object, the non-serializable superclass must run its constructor. Remember, constructors don't run on deserialized classes that implement Serializable.

☒ **A, B, D, E,** and **F** are incorrect based on the above. (Objective 3.3)

**3.** Given:

```
class TKO {
  public static void main(String[] args) {
    String s = "-";
    Integer x = 343;
    long L343 = 343L;
    if(x.equals(L343))  s += ".e1 ";
    if(x.equals(343))  s += ".e2 ";
    Short s1 = (short)((new Short((short)343)) / (new Short((short)49)));
    if(s1 == 7)        s += "=s ";
    if(s1 < new Integer(7+1))  s += "fly ";
    System.out.println(s);
} }
```

Which of the following will be included in the output String `s`? (Choose all that apply.)

A. `.e1`

B. `.e2`

C. `=s`

D. `fly`

E. None of the above

F. Compilation fails

G. An exception is thrown at runtime

Answer:

☑ **B, C,** and **D** are correct. Remember, that the `equals()` method for the integer wrappers will only return `true` if the two primitive types and the two values are equal. With **C**, it's okay to unbox and use `==`. For **D**, it's okay to create a wrapper object with an expression, and unbox it for comparison with a primitive.

☒ **A, E, F,** and **G** are incorrect based on the above. (Remember that **A** is using the `equals()` method to try to compare two different types.) (Objective 3.1)

**4.** Given:

```
import java.io.*;

class Keyboard { }
public class Computer implements Serializable {
```

```
  private Keyboard k = new Keyboard();
  public static void main(String[] args) {
    Computer c = new Computer();
    c.storeIt(c);
  }
  void storeIt(Computer c) {
    try {
      ObjectOutputStream os = new ObjectOutputStream(
         new FileOutputStream("myFile"));
      os.writeObject(c);
      os.close();
      System.out.println("done");
    } catch (Exception x) {System.out.println("exc"); }
  }
}
```

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

A. `exc`

B. `done`

C. Compilation fails

D. Exactly one object is serialized

E. Exactly two objects are serialized

Answer:

☑ **A** is correct. An instance of type Computer Has-a Keyboard. Because Keyboard doesn't implement Serializable, any attempt to serialize an instance of Computer will cause an exception to be thrown.

☒ **B, C, D,** and **E** are incorrect based on the above. If Keyboard did implement Serializable then two objects would have been serialized. (Objective 3.3)

**5.** Using the fewest fragments possible (and filling the fewest slots possible), complete the code below so that the class builds a directory named "`dir3`" and creates a file named "`file3`" inside "`dir3`". Note you can use each fragment either zero or one times.

Code:

```
import java.io._____________

class Maker {
  public static void main(String[] args) {

    ___________  ___________  ___________

    ___________  ___________  ___________

    ___________  ___________  ___________

    ___________  ___________  ___________

    ___________  ___________  ___________

    ___________  ___________  ___________

    ___________  ___________  ___________
} }
```

Fragments:

```
File;      FileDescriptor;     FileWriter;    Directory;
try {      .createNewDir();    File dir       File
{ }        (Exception x)       ("dir3");      file
file       .createNewFile();   = new File     = new File
dir        (dir, "file3");     (dir, file);   .createFile();
} catch    ("dir3", "file3");  .mkdir();      File file
```

Answer:

```
import java.io.File;
class Maker {
  public static void main(String[] args) {
    try {
      File dir = new File("dir3");
      dir.mkdir();
      File file = new File(dir, "file3");
      file.createNewFile();
    } catch (Exception x) { }
} }
```

Notes: The `new File` statements don't make actual files or directories, just objects. You need the `mkdir()` and `createNewFile()` methods to actually create the directory and the file. (Objective 3.2)

**6.** Given that 1119280000000L is roughly the number of milliseconds from Jan. 1, 1970, to June 20, 2005, and that you want to print that date in German, using the `LONG` style such that "`June`" will be displayed as "`Juni`", complete the code using the fragments below. Note: you can use each fragment either zero or one times, and you might not need to fill all of the slots.

Code:

```
import java._____________
import java._____________

class DateTwo {
  public static void main(String[] args) {
    Date d = new Date(1119280000000L);
    DateFormat df = _________________________________

                    _________________ , ___________________ );

    System.out.println(_________________
  }
}
```

Fragments:

```
io.*;      new DateFormat(                 Locale.LONG
nio.*;     DateFormat.getInstance(         Locale.GERMANY
util.*;    DateFormat.getDateInstance(     DateFormat.LONG
text.*;    util.regex;                     DateFormat.GERMANY
date.*;    df.format(d));                  d.format(df));
```

Answer:

```
import java.util.*;
import java.text.*;
class DateTwo {
  public static void main(String[] args) {
    Date d = new Date(1119280000000L);
  DateFormat df = DateFormat.getDateInstance(
                    DateFormat.LONG, Locale.GERMANY);
    System.out.println(df.format(d));
  }
}
```

Notes: Remember that you must build `DateFormat` objects using static methods. Also remember that you must specify a Locale for a `DateFormat` object at the time of instantiation. The `getInstance()` method does not take a Locale. (Objective 3.4)

**7.** Given:

```
import java.io.*;

class Directories {
  static String [] dirs = {"dir1", "dir2"};
  public static void main(String [] args) {
    for (String d : dirs) {

      // insert code 1 here

      File file = new File(path, args[0]);

      // insert code 2 here
    }
  }
}
```

and that the invocation

```
java Directories file2.txt
```

is issued from a directory that has two subdirectories, "`dir1`" and "`dir2`", and that "`dir1`" has a file "`file1.txt`" and "`dir2`" has a file "`file2.txt`", and the output is "`false true`", which set(s) of code fragments must be inserted? (Choose all that apply.)

A.
```
String path = d;

System.out.print(file.exists() + " ");
```

B.
```
String path = d;

System.out.print(file.isFile() + " ");
```

C.
```
String path = File.separator + d;

System.out.print(file.exists() + " ");
```

D.
```
String path = File.separator + d;

System.out.print(file.isFile() + " ");
```

Answer:

☑ **A** and **B** are correct. Because you are invoking the program from the directory whose direct subdirectories are to be searched, you don't start your path with a `File.separator` character. The `exists()` method tests for either files or directories; the `isFile()` method tests only for files. Since we're looking for a file, both methods work.

☒ **C** and **D** are incorrect based on the above. (Objective 3.2)

**8.** Given:

```
import java.io.*;

public class TestSer {
  public static void main(String[] args) {
    SpecialSerial s = new SpecialSerial();
    try {
      ObjectOutputStream os = new ObjectOutputStream(
         new FileOutputStream("myFile"));
      os.writeObject(s);  os.close();
      System.out.print(++s.z + " ");

      ObjectInputStream is = new ObjectInputStream(
         new FileInputStream("myFile"));
      SpecialSerial s2 = (SpecialSerial)is.readObject();
      is.close();
      System.out.println(s2.y + " " + s2.z);
    } catch (Exception x) {System.out.println("exc"); }
  }
}
class SpecialSerial implements Serializable {
  transient int y = 7;
  static int z = 9;
}
```

Which are true? (Choose all that apply.)

A. Compilation fails

B. The output is `10 0 9`

C. The output is `10 0 10`

D. The output is `10 7 9`

E. The output is `10 7 10`

F. In order to alter the standard deserialization process you would implement the `readObject()` method in `SpecialSerial`

G. In order to alter the standard deserialization process you would implement the `defaultReadObject()` method in `SpecialSerial`

Answer:

☑ **C** and **F** are correct. **C** is correct because `static` and `transient` variables are not serialized when an object is serialized. **F** is a valid statement.

☒ **A, B, D,** and **E** are incorrect based on the above. **G** is incorrect because you don't implement the `defaultReadObject()` method, you call it from within the `readObject()` method, along with any custom read operations your class needs. (Objective 3.3)

**9.** Given:

```
 3. public class Theory {
 4.   public static void main(String[] args) {
 5.     String s1 = "abc";
 6.     String s2 = s1;
 7.     s1 += "d";
 8.     System.out.println(s1 + " " + s2 + " " + (s1==s2));
 9.
10.     StringBuffer sb1 = new StringBuffer("abc");
11.     StringBuffer sb2 = sb1;
12.     sb1.append("d");
13.     System.out.println(sb1 + " " + sb2 + " " + (sb1==sb2));
14.   }
15. }
```

Which are true? (Choose all that apply.)

A. Compilation fails

B. The first line of output is `abc abc true`

C. The first line of output is `abc abc false`

D. The first line of output is `abcd abc false`

E. The second line of output is `abcd abc false`

F. The second line of output is `abcd abcd true`

G. The second line of output is `abcd abcd false`

Answer:

☑ **D** and **F** are correct. While String objects are immutable, references to Strings are mutable. The code `s1 += "d";` creates a new String object. StringBuffer objects are mutable, so the `append()` is changing the single StringBuffer object to which both StringBuffer references refer.

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

**10.** Given:

```
 3. import java.io.*;
 4. public class ReadingFor {
 5.   public static void main(String[] args) {
 6.     String s;
 7.     try {
 8.       FileReader fr = new FileReader("myfile.txt");
 9.       BufferedReader br = new BufferedReader(fr);
10.       while((s = br.readLine()) != null)
11.         System.out.println(s);
12.       br.flush();
13.     } catch (IOException e) { System.out.println("io error"); }
16.   }
17. }
```

And given that myfile.txt contains the following two lines of data:

```
ab
cd
```

What is the result?

A. `ab`

B. `abcd`

C.
```
ab
cd
```

D.
```
a
b
c
d
```

E. Compilation fails

Answer:

☑ **E** is correct. You need to call `flush()` only when you're writing data. Readers don't have `flush()` methods. If not for the call to `flush()`, answer C would be correct.

☒ **A, B, C,** and **D** are incorrect based on the above. (Objective 3.2)

**11.** Given:

```
 3. import java.io.*;
 4. public class Talker {
 5.   public static void main(String[] args) {
 6.     Console c = System.console();
 7.     String u = c.readLine("%s", "username: ");
 8.     System.out.println("hello " + u);
 9.     String pw;
10.     if(c != null && (pw = c.readPassword("%s", "password: ")) != null)
11.       // check for valid password
12.   }
13. }
```

If line 6 creates a valid Console object, and if the user enters *fred* as a username and *1234* as a password, what is the result? (Choose all that apply.)

A.
```
username:
password:
```

B.
```
username: fred
password:
```

C. `username: fred`
`password: 1234`

D. Compilation fails

E. An exception is thrown at runtime

Answer:

☑ **D** is correct. The `readPassword()` method returns a `char[]`. If a `char[]` were used, answer B would be correct.

☒ **A, B, C,** and **E** are incorrect based on the above. (Objective 3.2)

**12.** Given:

```
3. import java.io.*;
4. class Vehicle { }
5. class Wheels { }
6. class Car extends Vehicle implements Serializable {  }
7. class Ford extends Car { }
8. class Dodge extends Car {
9.   Wheels w = new Wheels();
10. }
```

Instances of which class(es) can be serialized? (Choose all that apply.)

A. Car

B. Ford

C. Dodge

D. Wheels

E. Vehicle

Answer:

☑ **A** and **B** are correct. Dodge instances cannot be serialized because they "have" an instance of Wheels, which is not serializable. Vehicle instances cannot be serialized even though the subclass Car can be.

☒ **C, D,** and **E** are incorrect based on the above. (Objective 3.3)

**13.** Given:

```
 3. import java.text.*;
 4. public class Slice {
 5.   public static void main(String[] args) {
 6.     String s = "987.123456";
 7.     double d  = 987.123456d;
 8.     NumberFormat nf = NumberFormat.getInstance();
 9.     nf.setMaximumFractionDigits(5);
10.     System.out.println(nf.format(d) + " ");
11.     try {
12.       System.out.println(nf.parse(s));
13.     } catch (Exception e) { System.out.println("got exc"); }
14.   }
15. }
```

Which are true? (Choose all that apply.)

A. The output is `987.12345 987.12345`

B. The output is `987.12346 987.12345`

C. The output is `987.12345 987.123456`

D. The output is `987.12346 987.123456`

E. The try/catch block is unnecessary

F. The code compiles and runs without exception

G. The invocation of `parse()` must be placed within a try/catch block

Answer:

☑ **D, F,** and **G** are correct. The `setMaximumFractionDigits()` applies to the formatting but not the parsing. The try/catch block is placed appropriately. This one might scare you into thinking that you'll need to memorize more than you really do. If you can remember that you're formatting the number and parsing the string you should be fine for the exam.

☒ **A, B, C,** and **E** are incorrect based on the above. (Objective 3.4)

**14.** Given:

```
3. import java.util.regex.*;
4. public class Archie {
5.   public static void main(String[] args) {
6.     Pattern p = Pattern.compile(args[0]);
```

```
 7.     Matcher m = p.matcher(args[1]);
 8.     int count = 0;
 9.     while(m.find())
10.       count++;
11.     System.out.print(count);
12.   }
13. }
```

And given the command line invocation:

```
java Archie "\d+" ab2c4d67
```

What is the result?

A. 0

B. 3

C. 4

D. 8

E. 9

F. Compilation fails

Answer:

☑ **B** is correct. The "\d" metacharacter looks for digits, and the + quantifier says look for "one or more" occurrences. The `find()` method will find three sets of one or more consecutive digits: 2, 4, and 67.

☒ **A, C, D, E,** and **F** are incorrect based on the above. (Objective 3.5)

**15.** Given:

```
 3. import java.util.*;
 4. public class Looking {
 5.   public static void main(String[] args) {
 6.     String input = "1 2 a 3 45 6";
 7.     Scanner sc = new Scanner(input);
 8.     int x = 0;
 9.     do {
10.       x = sc.nextInt();
11.       System.out.print(x + " ");
12.     } while (x!=0);
13.   }
14. }
```

What is the result?

A. `1 2`

B. `1 2 3 45 6`

C. `1 2 3 4 5 6`

D. `1 2 a 3 45 6`

E. Compilation fails

F. `1 2` followed by an exception

Answer:

☑ **F** is correct. The `nextXxx()` methods are typically invoked after a call to a `hasNextXxx()`, which determines whether the next token is of the correct type.

☒ **A, B, C, D,** and **E** are incorrect based on the above. (Objective 3.5)