



I

Self-Assessment Test I

MAIN TOPICS

- How Close Are You to Ready?
- Assessment Test I
- Quick Answer Key
- Assessment Test I: Answers
- Analyzing Your Results
- Creating Your Exam Log
- More Study Tips
- Common Questions and Complaints, and Some Answers
- Preparing for Self-Assessment Test 2

How Close Are You to Ready?

The 14-question assessment tests in this chapter and Chapter 2 are designed to help you answer the question: “How close am I to ready?” The best way to start answering the readiness question is to set aside 42 minutes (yes, 42!) and take the first assessment exam.

As we discussed in the introduction, many Java programmers with years of experience are surprised by the depth and breadth of topics covered in the OCP Java SE 6 Programmer exam. The assessment exams in this chapter and Chapter 2 are designed to help you determine how much more studying you’ll need to do before you take the real exam. Each of the first two chapters includes a table to help you match your assessment test results to a rough study plan. In a nutshell, for each of the assessment exams, 8 (out of 14) correct questions is on the boundary of achieving a passing score.

The exams in this book are intended primarily for candidates who feel they’ve mostly completed their studies and want additional exam practice. If your scores on the assessment exams are not near the passing mark, we recommend you do more studying before trying any of the four full practice exams included in later chapters. In other words, this book’s job is to help you put the final touches on your OCP Java SE 6 Programmer preparation. It’s NOT intended to be your primary study guide.

With all that said, set your timer to 42 minutes and dive in. We’ll see you on the other side...

ASSESSMENT TEST I

The real exam has 60 questions and you are given three hours. Since this assessment exam has only 14 questions, allow yourself only 42 minutes to complete this exam. On the real exam, and on all of the exams in this book, give yourself credit only for those questions that you answer 100 percent correctly. For instance, if a question has three correct answers and you get two of the three correct, you get zero credit. There is no partial credit. Good luck!

1. Given:

```
2. public class Bunnies {
3.     static int count = 0;
4.     Bunnies() {
5.         while(count < 10) new Bunnies(++count);
6.     }
7.     Bunnies(int x) { super(); }
8.     public static void main(String[] args) {
9.         new Bunnies();
10.        new Bunnies(count);
11.        System.out.println(count++);
12.    }
13. }
```

What is the result?

- A. 9
- B. 10
- C. 11
- D. 12
- E. Compilation fails.
- F. An exception is thrown at runtime.

2. Given:

```
2. public class Jail {
3.     private int x = 4;
4.     public static void main(String[] args) {
5.         protected int x = 6;
6.         new Jail().new Cell().slam();
7.     }
8.     class Cell {
9.         void slam() { System.out.println("throw away key " + x); }
10.    }
11. }
```

4 Chapter 1: Self-Assessment Test I

Which are true? (Choose all that apply.)

- A. Compilation succeeds.
- B. The output is "throw away key 4".
- C. The output is "throw away key 6".
- D. Compilation fails due to an error on line 5.
- E. Compilation fails due to an error on line 6.
- F. Compilation fails due to an error on line 9.

3. Given:

```
2. public class Fabric extends Thread {
3.     public static void main(String[] args) {
4.         Thread t = new Thread(new Fabric());
5.         Thread t2 = new Thread(new Fabric());
6.         t.start();
7.         t2.start();
8.     }
9.     public static void run() {
10.        for(int i = 0; i < 2; i++)
11.            System.out.print(Thread.currentThread().getName() + " ");
12.    }
13. }
```

Which are true? (Choose all that apply.)

- A. Compilation fails.
- B. No output is produced.
- C. The output could be Thread-1 Thread-3 Thread-1 Thread-2
- D. The output could be Thread-1 Thread-3 Thread-1 Thread-3
- E. The output could be Thread-1 Thread-1 Thread-2 Thread-2
- F. The output could be Thread-1 Thread-3 Thread-3 Thread-1
- G. The output could be Thread-1 Thread-3 Thread-1 Thread-1

4. Given:

```
2. class Feline { }
3. public class BarnCat2 extends Feline {
4.     public static void main(String[] args) {
5.         Feline ff = new Feline();
6.         BarnCat2 b = new BarnCat2();
7.         // insert code here
8.     }
9. }
```

Which, inserted independently at line 7, compile? (Choose all that apply.)

- A. `if(b instanceof ff) System.out.print("1 ");`
- B. `if(b instanceof (ff)) System.out.print("2 ");`
- C. `if(b instanceof Feline) System.out.print("3 ");`
- D. `if(b instanceof Feline) System.out.print("4 ");`
- E. `if(b instanceof (Feline)) System.out.print("5 ");`

5. Given:

```

2. public class Choosy {
3.     public static void main(String[] args) {
4.         String result = "";
5.         int x = 7, y = 8;
6.         if(x == 3) { result += "1"; }
7.         else if (x > 9) { result += "2"; }
8.         else if (y < 9) { result += "3"; }
9.         else if (x == 7) { result += "4"; }
10.        else { result += "5"; }
11.        System.out.println(result);
12.    }
13. }
```

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

- A. 3
- B. 34
- C. 35
- D. 345
- E. Compilation fails due to an error on line 5.
- F. Compilation fails due to errors on lines 8 and 9.
- G. Compilation fails due to errors on lines 7, 8, and 9.

6. Given:

```

1. public class Twine {
2.     public static void main(String[] args) {
3.         String s = "";
4.         StringBuffer sb1 = new StringBuffer("hi");
5.         StringBuffer sb2 = new StringBuffer("hi");
6.         StringBuffer sb3 = new StringBuffer(sb2);
7.         StringBuffer sb4 = sb3;
8.         if(sb1.equals(sb2)) s += "1 ";
9.         if(sb2.equals(sb3)) s += "2 ";
```

6 Chapter 1: Self-Assessment Test I

```
10.     if(sb3.equals(sb4)) s += "3 ";
11.     String s2 = "hi";
12.     String s3 = "hi";
13.     String s4 = s3;
14.     if(s2.equals(s3)) s += "4 ";
15.     if(s3.equals(s4)) s += "5 ";
16.     System.out.println(s);
17.     }
18. }
```

What is the result?

- A. 1 3
 - B. 1 5
 - C. 1 2 3
 - D. 1 4 5
 - E. 3 4 5
 - F. 1 3 4 5
 - G. 1 2 3 4 5
 - H. Compilation fails.
7. Which are true? (Choose all that apply.)
- A. All classes of Exception extend Error.
 - B. All classes of Error extend Exception.
 - C. All Errors must be handled or declared.
 - D. All classes of Exception extend Throwable.
 - E. All Throwables must be handled or declared.
 - F. All Exceptions must be handled or declared.
 - G. RuntimeExceptions need never be handled or declared.
8. Given:
- ```
2. import java.util.*;
3. public class Birthdays {
4. public static void main(String[] args) {
5. Map<Friends, String> hm = new HashMap<Friends, String>();
6. hm.put(new Friends("Charis"), "Summer 2009");
7. hm.put(new Friends("Draumur"), "Spring 2002");
8. Friends f = new Friends(args[0]);
9. System.out.println(hm.get(f));
10. }
11. }
```

```

12. class Friends {
13. String name;
14. Friends(String n) { name = n; }
15. }

```

And the command line invocation:

```
java Birthdays Draumur
```

What is the result?

- A. null
- B. Draumur
- C. Spring 2002
- D. Compilation fails.
- E. The output is unpredictable.
- F. An exception is thrown at runtime.
- G. Friends@XXXX (where XXXX is a representation of a hashcode)

9. Given:

```

2. import java.util.*;
3. class Cereal { }
4. public class Flakes extends Cereal {
5. public static void main(String[] args) {
6. List<Flakes> c0 = new List<Flakes>();
7. List<Cereal> c1 = new ArrayList<Cereal>();
8. List<Cereal> c2 = new ArrayList<Flakes>();
9. List<Flakes> c3 = new ArrayList<Cereal>();
10. List<Object> c4 = new ArrayList<Flakes>();
11. ArrayList<Cereal> c5 = new ArrayList<Flakes>();
12. }
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.
- G. Compilation fails due to an error on line 11.

## 8 Chapter 1: Self-Assessment Test I

### 10. Given:

```
3. public class RediMix extends Concrete {
4. RediMix() { System.out.println("r "); }
5. public static void main(String[] args) {
6. new RediMix();
7. }
8. }
9. class Concrete extends Sand {
10. Concrete() { System.out.print("c "); }
11. private Concrete(String s) { }
12. }
13. abstract class Sand {
14. Sand() { System.out.print("s "); }
15. }
```

What is the result?

- A. r
  - B. c r
  - C. r c
  - D. s c r
  - E. r c s
  - F. Compilation fails due to a single error in the code.
  - G. Compilation fails due to multiple errors in the code.
11. Which statement(s) are true? (Choose all that apply.)
- A. Coupling is the OO principle most closely associated with hiding a class's implementation details.
  - B. Coupling is the OO principle most closely associated with making sure classes know about other classes only through their APIs.
  - C. Coupling is the OO principle most closely associated with making sure a class is designed with a single, well-focused purpose.
  - D. Coupling is the OO principle most closely associated with allowing a single object to be seen as having many types.

### 12. Given:

```
2. class Mosey implements Runnable {
3. public void run() {
4. for(int i = 0; i < 1000; i++) {
5. System.out.print(Thread.currentThread().getId() + "-" + i + " ");
6. } } }
```

```

7. public class Stroll {
8. public static void main(String[] args) throws Exception {
9. Thread t1 = new Thread(new Mosey());
10. // insert code here
11. }
12. }

```

Which of the following code fragments, inserted independently at line 10, will probably run most (or all) of the main thread's `run()` method invocation before running most of the `t1` thread's `run()` method invocation? (Choose all that apply.)

- A. `t1.setPriority(1);`  
`new Mosey().run();`  
`t1.start();`
- B. `t1.setPriority(9);`  
`new Mosey().run();`  
`t1.start();`
- C. `t1.setPriority(1);`  
`t1.start();`  
`new Mosey().run();`
- D. `t1.setPriority(8);`  
`t1.start();`  
`new Mosey().run();`

13. Given:

```

37. boolean b = false;
38. int i = 7;
39. double d = 1.23;
40. float f = 4.56f;
41.
42. // insert code here

```

Which line(s) of code, inserted independently at line 42, will compile and run without exception? (Choose all that apply.)

- A. `System.out.printf(" %b", b);`
- B. `System.out.printf(" %i", i);`
- C. `System.out.format(" %d", d);`
- D. `System.out.format(" %d", i);`
- E. `System.out.format(" %f", f);`

**14.** Given:

```
1. import java.util.*;
2. public class MyPancake implements Pancake {
3. public static void main(String[] args) {
4. List<String> x = new ArrayList<String>();
5. x.add("3"); x.add("7"); x.add("5");
6. List<String> y = new MyPancake().doStuff(x);
7. y.add("1");
8. System.out.println(x);
9. }
10. List<String> doStuff(List<String> z) {
11. z.add("9");
12. return z;
13. }
14. }
15. interface Pancake {
16. List<String> doStuff(List<String> s);
17. }
```

What is the most likely result?

- A. [3, 7, 5]
- B. [3, 7, 5, 9]
- C. [3, 7, 5, 9, 1]
- D. Compilation fails.
- E. An exception is thrown at runtime.

# QUICK ANSWER KEY



- 1. B
- 2. D
- 3. A
- 4. C
- 5. A

- 6. E
- 7. D, G
- 8. A
- 9. B, D, E, F, G
- 10. D

- 11. B
- 12. A, B, C
- 13. A, D, E
- 14. D

## ASSESSMENT TEST I: ANSWERS

1. Given:

```
2. public class Bunnies {
3. static int count = 0;
4. Bunnies() {
5. while(count < 10) new Bunnies(++count);
6. }
7. Bunnies(int x) { super(); }
8. public static void main(String[] args) {
9. new Bunnies();
10. new Bunnies(count);
11. System.out.println(count++);
12. }
13. }
```

What is the result?

- A. 9
- B. 10
- C. 11
- D. 12
- E. Compilation fails.
- F. An exception is thrown at runtime.

Answer (for Objective 5.4):

- B** is correct. It's legal to invoke "new" from within a constructor, and it's legal to call `super()` on a class with no explicit superclass. On the real exam, it's important to watch out for pre- and post-incrementing.
- A, C, D, E, and F** are incorrect based on the above.

2. Given:

```
2. public class Jail {
3. private int x = 4;
4. public static void main(String[] args) {
5. protected int x = 6;
6. new Jail().new Cell().slam();
7. }
8. class Cell {
9. void slam() { System.out.println("throw away key " + x); }
10. }
11. }
```

Which are true? (Choose all that apply.)

- A. Compilation succeeds.
- B. The output is "throw away key 4".
- C. The output is "throw away key 6".
- D. Compilation fails due to an error on line 5.
- E. Compilation fails due to an error on line 6.
- F. Compilation fails due to an error on line 9.

Answer (for Objective 1.3):

- D** is correct. Line 5 is declaring local variable "x", and local variables cannot have access modifiers. If line 5 read "int x = 6", the code would compile and the result would be "throw away key 4". Line 5 creates an anonymous Jail object, an anonymous Cell object, and invokes slam(). Inner classes have access to their enclosing class's private variables.
- A, B, C, E, and F** are incorrect based on the above.

3. Given:

```

2. public class Fabric extends Thread {
3. public static void main(String[] args) {
4. Thread t = new Thread(new Fabric());
5. Thread t2 = new Thread(new Fabric());
6. t.start();
7. t2.start();
8. }
9. public static void run() {
10. for(int i = 0; i < 2; i++)
11. System.out.print(Thread.currentThread().getName() + " ");
12. }
13. }

```

Which are true? (Choose all that apply.)

- A. Compilation fails.
- B. No output is produced.
- C. The output could be Thread-1 Thread-3 Thread-1 Thread-2
- D. The output could be Thread-1 Thread-3 Thread-1 Thread-3
- E. The output could be Thread-1 Thread-1 Thread-2 Thread-2

## 14 Chapter 1: Self-Assessment Test 1

- F. The output could be Thread-1 Thread-3 Thread-3 Thread-1
- G. The output could be Thread-1 Thread-3 Thread-1 Thread-1

Answer (for Objective 4.1):

- A** is correct. Fabric does not correctly extend Thread because the run() method cannot be static. If run() was correctly implemented, then **D**, **E**, and **F** would have been correct. Thread names do *NOT* have to be sequentially assigned.
- C** is wrong even if run() is correct because only two threads are involved. **G** is wrong even if run() is correct, because run() is called only once per thread.

### 4. Given:

```
2. class Feline { }
3. public class BarnCat2 extends Feline {
4. public static void main(String[] args) {
5. Feline ff = new Feline();
6. BarnCat2 b = new BarnCat2();
7. // insert code here
8. }
9. }
```

Which, inserted independently at line 7, compile? (Choose all that apply.)

- A. `if (b instanceof ff) System.out.print("1 ");`
- B. `if (b instanceof (ff)) System.out.print("2 ");`
- C. `if (b instanceof Feline) System.out.print("3 ");`
- D. `if (b instanceofOf Feline) System.out.print("4 ");`
- E. `if (b instanceof (Feline)) System.out.print("5 ");`

Answer (for Objective 7.6):

- C** is the correct syntax.
- A**, **B**, **D**, and **E** all use incorrect syntax for the instanceof operator.

### 5. Given:

```
2. public class Choosy {
3. public static void main(String[] args) {
4. String result = "";
5. int x = 7, y = 8;
```

```

6. if(x == 3) { result += "1"; }
7. else if (x > 9) { result += "2"; }
8. else if (y < 9) { result += "3"; }
9. else if (x == 7) { result += "4"; }
10. else { result += "5"; }
11. System.out.println(result);
12. }
13. }

```

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

- A. 3
- B. 34
- C. 35
- D. 345
- E. Compilation fails due to an error on line 5.
- F. Compilation fails due to errors on lines 8 and 9.
- G. Compilation fails due to errors on lines 7, 8, and 9.

Answer (for Objective 2.1):

- A is correct. It's legal to declare several variables on a single line, and it's legal to have multiple `else-if` statements. Once an `else-if` succeeds, the remaining `else-if` and `else` statements in the block are ignored.
- B, C, D, E, F, and G are incorrect based on the above.

6. Given:

```

1. public class Twine {
2. public static void main(String[] args) {
3. String s = "";
4. StringBuffer sb1 = new StringBuffer("hi");
5. StringBuffer sb2 = new StringBuffer("hi");
6. StringBuffer sb3 = new StringBuffer(sb2);
7. StringBuffer sb4 = sb3;
8. if(sb1.equals(sb2)) s += "1 ";
9. if(sb2.equals(sb3)) s += "2 ";
10. if(sb3.equals(sb4)) s += "3 ";
11. String s2 = "hi";
12. String s3 = "hi";
13. String s4 = s3;

```

## 16 Chapter 1: Self-Assessment Test 1

```
14. if(s2.equals(s3)) s += "4 ";
15. if(s3.equals(s4)) s += "5 ";
16. System.out.println(s);
17. }
18. }
```

What is the result?

- A. 1 3
- B. 1 5
- C. 1 2 3
- D. 1 4 5
- E. 3 4 5
- F. 1 3 4 5
- G. 1 2 3 4 5
- H. Compilation fails.

Answer (for Objective 3.1):

- E is correct. The `StringBuffer` class doesn't override the `equals()` method, so two different `StringBuffer` objects with the same value will not be equal according to the `equals()` method. On the other hand, the `String` class's `equals()` method has been overridden so that two different `String` objects with the same value will be considered equal according to the `equals()` method.
- A, B, C, D, F, G, and H are incorrect based on the above.

7. Which are true? (Choose all that apply.)
- A. All classes of Exception extend Error.
  - B. All classes of Error extend Exception.
  - C. All Errors must be handled or declared.
  - D. All classes of Exception extend Throwable.
  - E. All Throwables must be handled or declared.
  - F. All Exceptions must be handled or declared.
  - G. RuntimeExceptions need never be handled or declared.

Answer (for Objective 2.5):

- D** and **G** are correct. While it's true that this is a strict memorization question, some facts are so essential that you just have to burn them into your brain. The class hierarchy relationships between `Throwable`, `Error`, `Exception`, and `RuntimeException` fall into that “gotta know ‘em” category.
- A**, **B**, **C**, **E**, and **F** are incorrect statements.

8. Given:

```

2. import java.util.*;
3. public class Birthdays {
4. public static void main(String[] args) {
5. Map<Friends, String> hm = new HashMap<Friends, String>();
6. hm.put(new Friends("Charis"), "Summer 2009");
7. hm.put(new Friends("Draumur"), "Spring 2002");
8. Friends f = new Friends(args[0]);
9. System.out.println(hm.get(f));
10. }
11. }
12. class Friends {
13. String name;
14. Friends(String n) { name = n; }
15. }

```

And the command line invocation:

```
java Birthdays Draumur
```

What is the result?

- A. `null`
- B. `Draumur`
- C. `Spring 2002`
- D. Compilation fails.
- E. The output is unpredictable.
- F. An exception is thrown at runtime.
- G. `Friends@XXXX` (where `XXXX` is a representation of a hashcode)

Answer (for Objective 6.2):

- A** is correct. The `Friends` class doesn't override `equals()` and `hashCode()`, so the key to the `HashMap` is a specific instance of `Friends`, not the value of a given `Friends` instance's name.
- B, C, D, E, F, and G** are incorrect based on the above.

9. Given:

```
2. import java.util.*;
3. class Cereal { }
4. public class Flakes extends Cereal {
5. public static void main(String[] args) {
6. List<Flakes> c0 = new List<Flakes>();
7. List<Cereal> c1 = new ArrayList<Cereal>();
8. List<Cereal> c2 = new ArrayList<Flakes>();
9. List<Flakes> c3 = new ArrayList<Cereal>();
10. List<Object> c4 = new ArrayList<Flakes>();
11. ArrayList<Cereal> c5 = new ArrayList<Flakes>();
12. }
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.
- G.** Compilation fails due to an error on line 11.

Answer (for Objective 6.3):

- B, D, E, F, and G** are correct because those lines of code will *NOT* compile. **B**, (line 6), is incorrect because `List` is abstract. **D, E, F, and G** are all incorrect because polymorphic assignments can't be applied to the generic type parameter.
- A** is incorrect based on the above. **C** is incorrect because line 7 uses legal syntax.

10. Given:

```

3. public class RediMix extends Concrete {
4. RediMix() { System.out.println("r "); }
5. public static void main(String[] args) {
6. new RediMix();
7. }
8. }
9. class Concrete extends Sand {
10. Concrete() { System.out.print("c "); }
11. private Concrete(String s) { }
12. }
13. abstract class Sand {
14. Sand() { System.out.print("s "); }
15. }

```

What is the result?

- A. r
- B. c r
- C. r c
- D. s c r
- E. r c s
- F. Compilation fails due to a single error in the code.
- G. Compilation fails due to multiple errors in the code.

Answer (for Objective 1.5):

- D is correct. It's legal for abstract classes to have constructors, and it's legal for a constructor to be private. Normal constructor chaining is the result of this code.
- A, B, C, E, F, and G are incorrect based on the above.

11. Which statement(s) are true? (Choose all that apply.)

- A. Coupling is the OO principle most closely associated with hiding a class's implementation details.
- B. Coupling is the OO principle most closely associated with making sure classes know about other classes only through their APIs.
- C. Coupling is the OO principle most closely associated with making sure a class is designed with a single, well-focused purpose.
- D. Coupling is the OO principle most closely associated with allowing a single object to be seen as having many types.

Answer (for Objective 5.1):

- B** is correct.
- A** refers to encapsulation, **C** refers to cohesion, and **D** refers to polymorphism.

12. Given:

```

2. class Mosey implements Runnable {
3. public void run() {
4. for(int i = 0; i < 1000; i++) {
5. System.out.print(Thread.currentThread().getId() + "-" + i + " ");
6. } } }
7. public class Stroll {
8. public static void main(String[] args) throws Exception {
9. Thread t1 = new Thread(new Mosey());
10. // insert code here
11. }
12. }
```

Which of the following code fragments, inserted independently at line 10, will probably run most (or all) of the main thread's `run()` method invocation before running most of the `t1` thread's `run()` method invocation? (Choose all that apply.)

- A. `t1.setPriority(1);`  
`new Mosey().run();`  
`t1.start();`
- B. `t1.setPriority(9);`  
`new Mosey().run();`  
`t1.start();`
- C. `t1.setPriority(1);`  
`t1.start();`  
`new Mosey().run();`
- D. `t1.setPriority(8);`  
`t1.start();`  
`new Mosey().run();`

Answer (for Objective 4.2):

- A, B, and C** are correct. For **A** and **B**, the main thread executes the `run()` method before it starts `t1`. **C** is correct because `t1` is set to a low priority, giving the main thread scheduling priority.
- D** is incorrect because by setting `t1`'s priority to 8, the `t1` thread will tend to execute mostly before the main thread.

13. Given:

```

37. boolean b = false;
38. int i = 7;
39. double d = 1.23;
40. float f = 4.56f;
41.
42. // insert code here

```

Which line(s) of code, inserted independently at line 42, will compile and run without exception? (Choose all that apply.)

- A. `System.out.printf(" %b", b);`
- B. `System.out.printf(" %i", i);`
- C. `System.out.format(" %d", d);`
- D. `System.out.format(" %d", i);`
- E. `System.out.format(" %f", f);`

Answer (for Objective 3.4):

- A, D, and E** have the correct conversion characters for their respective argument. Remember that `printf()` and `format()` have the same functionality.
- B** is incorrect because (as we can see with answer **D**), integers should use `%d`. **C** is incorrect because both floats and doubles should use `%f`.

14. Given:

```

1. import java.util.*;
2. public class MyPancake implements Pancake {
3. public static void main(String[] args) {
4. List<String> x = new ArrayList<String>();
5. x.add("3"); x.add("7"); x.add("5");
6. List<String> y = new MyPancake().doStuff(x);
7. y.add("1");
8. System.out.println(x);
9. }
10. List<String> doStuff(List<String> z) {
11. z.add("9");
12. return z;
13. }
14. }
15. interface Pancake {
16. List<String> doStuff(List<String> s);
17. }

```

## 22 Chapter 1: Self-Assessment Test 1

What is the most likely result?

- A. [3, 7, 5]
- B. [3, 7, 5, 9]
- C. [3, 7, 5, 9, 1]
- D. Compilation fails.
- E. An exception is thrown at runtime.

Answer (for Objective 7.1):

- D is correct. `MyPancake.doStuff()` must be marked `public`. If it is, then C would be correct.
- A, B, C, and E are incorrect based on the above.

## Analyzing Your Results

Now that you've taken this book's first assessment exam, it's time to look at your results and figure out what they mean and what to do next.

As of this writing, a passing score on the OCP Java SE 6 Programmer exam is 58.33 percent (35 out of 60 questions). Of course, this chapter's assessment exam had only 14 questions, so if you got 8.1 of the 14 questions correct, then in theory you passed!

Now for the bad news... We picked 14 of the easiest questions in the book to be on this first assessment exam. Based on the fact that we thought these were easier than average questions, Table 1-1 is a rough guide to where you are in your studies:

**TABLE 1-1** What Your Score Means

| Number of Correct Answers | Recommended Plan                                                                                                                                               |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0–5                       | You should do a LOT of studying before taking more of the exams in this book.                                                                                  |
| 6–8                       | You should do a little more studying before taking more of the exams in this book.                                                                             |
| 9–11                      | You're right on the "passing" boundary; the next assessment test will tell you more.                                                                           |
| 12–14                     | You're probably ready to use the five remaining exams in this book to polish your skills, but start with the second assessment exam to verify this conclusion. |

## Creating Your Exam Log

The more you can pinpoint your weaknesses, the more focused and efficient your studies will be. We recommend you create a written log for each of the exams in this book, and for any other practice exams you take. Once you've taken a practice exam and scored it, go back through the exam and make a log. We like the rough format shown in Table 1-2, but you should modify it to suit your learning style. Note that we filled in several example rows to illustrate how you might fill in your logs.

Table 1-2 is an example of what a partial exam log might look like after you've filled it in. The more honest you are with yourself, the more useful these logs will be. For example, if you guessed on a question, even a little bit, it will be helpful to you to acknowledge that guess in your log. Another way to be honest with yourself is to reflect on whether you knew a certain topic thoroughly, or whether the question just happened to hit on your strong point.

TABLE 1-2 Sample Exam Log

| Test Name: <i>Self-Assessment Test 1</i> |                                           |                         |            |                                           |
|------------------------------------------|-------------------------------------------|-------------------------|------------|-------------------------------------------|
| Q#                                       | Topic                                     | Objective               | Correct    | Notes                                     |
| 1                                        | <i>“new” and super() in a constructor</i> | <i>5.4-constructors</i> | <i>no</i>  | <i>Confused super() with overloading.</i> |
| 2                                        | <i>else ifs</i>                           | <i>2.1-if/ switch</i>   | <i>yes</i> | <i>I guessed on the nested part.</i>      |
| 3                                        | <i>formatting characters</i>              | <i>3.5-regex</i>        | <i>no</i>  | <i>Wow, I don’t have these memorized!</i> |
| 4                                        | <i>threads, run() signature</i>           | <i>4.1-threads</i>      | <i>yes</i> | <i>This one seemed easy.</i>              |
| ...                                      |                                           |                         |            |                                           |

In addition, it will help you to fill out ALL the columns in the log. The very act of thinking through them and then writing down your own summary of each question will be a great learning exercise. Be precise with your terms! Here are some examples:

- Are you referring to an instance of a class or a class?
- Call them reference variables, not pointers.
- Think in terms of a method’s “signature.”
- Call them “chained constructors.”
- Use the phrase “handle or declare,” and so on.

The most successful OCP Java SE 6 Programmer candidates we’ve seen take the time to learn the phrases and terms that experienced Java programmers use. When you find yourself thinking in terms of that vernacular, know that you’re in good company—so fill out your logs.

## More Study Tips

Some of the following ideas were covered in the introduction and bear repeating. Some of these ideas, however, are new.

**Write Lots of Code** Almost every test question in this book has the hidden benefit of being a great starting point for small coding projects. It’s almost universally

true that the candidates who score well on the OCP Java SE 6 Programmer exam wrote lots and lots of small Java programs during their studies. We wrote lots of programs as we wrote this book. Take the code you see in a question, type it in, compile it, and run it. Then, tweak the code, compile it, and run it again. Add `System.out.println` statements. Try to break stuff.

**Don't Use an IDE** The team that created the actual exam wrote their code using a basic text editor, and they compiled and ran their code from the command line. Once you've got your basics covered, IDEs are great tools. While you're studying for the exam, they're not. *Usually* an IDE will give you the same result as you get from the command line, but not always! And the trick is in knowing when your results will vary.

**Get a Good Study Guide** We're fond of the study guide written by a couple of hackers named Kathy and Bert, but several good study guides are available. We recommend you visit "the bunkhouse" at JavaRanch.com. It has tons of excellent book reviews, and based on those reviews and comments you can get a good sense of the styles and quality of the books you're considering.

We understand that these study guides can be expensive, but consider the value of your time. A good study guide can easily save you hundreds of hours of poking around the Internet trying to find the documentation you need. Sure, you could probably find most of the information for "free" on the Web, but that takes time. A good study guide represents a single source compendium of all that good info that's scattered around the Internet.

**Make and Revise Flash Cards** Earlier in the chapter, we recommended you create exam logs and "Topics to Study" lists. Based on those documents, you can make flash cards as described in the introduction. Your exam logs and "Topics to Study" lists should help you decide which of your existing flash cards you can retire, and which new ones you should make.

**Focus Your API Studies** The Java APIs are huge. Even the few packages and classes listed in the OCP Java SE 6 Programmer objectives make a daunting list. The good news is that the API-related questions in this book will tend to focus you on the parts of the API you need to know. Similarly, the better study guides and practice exams will also do a good job of focusing on those parts of the API that you really need to know. As mentioned earlier, read the reviews! Some practice exams are known to be well focused, and some less so. If you're really focused on the OCP Java SE 6 Programmer exam, stick to those practice exams that are more focused.

## Common Questions and Complaints, and Some Answers

Over the years, we've heard a lot of questions and complaints from candidates. Here are some of the most common concerns we hear, and (we hope) some decent answers.

### Why Is the Code Formatting so Terrible?

Remember, when you go to a test center to take the official test, you're going to a center that more or less has to be replicated on a world-wide basis. In other words, the hardware and test engine you use is more or less the same as you would discover in any test center in the world. That means your computer monitor probably won't be a high-resolution monitor, and you may have to scroll up and down to see all the code you need to review for a given question. That's life. In order to minimize the pain associated with low-res monitors, the exam creators often jam a LOT of code into small spaces. So when you see code like this

```

3. public class VLA implements Comparator<VLA> {
4. int dishSize;
5. public static void main(String[] args) {
6. VLA[] va = {new VLA(40), new VLA(200), new VLA(60)};
7. for(VLA v: va) System.out.print(v.dishSize + " ");
8. } }

```

understand that the exam team knew this was horrible formatting (and not an example of “best practices coding”). They were trying to help you do less scrolling.

### Why Is There so Much Memorization?

Let's break this question down into two parts: the API and the language.

#### API Memorization

We often hear programmers say, “In the real world, when I have to use something from the API, I'm going to look it up anyway, rather than just trust my memory. So why do I have to memorize API stuff for the exam?”

A couple of reasons. First, an OCP Java SE 6 Programmer should know *how* to use the API in general. Second, if you've studied the commonly used packages in the API, you'll remember their basic capabilities even if you don't remember the exact details. This knowledge will make you a much better programmer. For instance, when we wrote this book, we had forgotten some of the API details that we needed to write the questions. However, we remembered where the "gotchas" were, and we knew what basic capabilities existed, so we were able to use the APIs quickly. That's part of being an OCP Java SE 6 Programmer and part of being a good programmer.

### Language Memorization

We hear similar complaints about having to memorize language details. You should know that in the current version(s) of the exam, great care was taken to remove questions that focused on seldom-used "corner cases" in the language. The exam team's strategy was to write questions that test the kinds of constructs you're likely to encounter in the real world when looking at someone else's code. The second reason is similar to the API discussion. You might not, for instance, remember the exact syntax for using a switch statement, but you will remember its capabilities—and when it's the right tool to use, you'll be able to use it quickly and correctly. The bottom line is this: after studying for this exam, you *will* be a better Java programmer, and you *will* use the language more like it was intended.

## exam

### Watch

**Although we said this in the introduction, it bears repeating. In the real exam, EVERY multiple choice question will ALWAYS tell you how many correct answers there are. In other words, the real exam will never say, "Choose all that apply." When you're taking the real exam, if you get to a question you're not quite sure of, knowing**

**the number of correct answers can help you narrow down the choices and make a better guess. (And guessing wrong DOES NOT hurt your score!) But in this book, we want to toughen you up. We want you to be extra-prepared, so we often say, "Choose all that apply."**

## exam

### Watch

*The real exam has two kinds of questions: (1) multiple choice and (2) drag and drop. On the real exam, you should expect 15 to 20 percent of your questions to be drag-and-drop style. In this book, we've created a few questions (more like 4 percent) that attempt to emulate the*

*kinds of drag-and-drop questions you'll get on the real exam.*

*On both the real exam and in our drag-and-drop questions, there may be more than one correct answer. In both cases, you get full credit for any correct answer.*

## Preparing for Self-Assessment Test 2

When you feel you've done the appropriate amount of preparation, head to the next chapter and take a whack at the second assessment exam.