Chapter 5: Enhancing Classes
Solutions

Multiple Choice Solutions

1. a
2. c
3. a
4. b
5. e
6. c
7. e
8. d
9. e
10. d

Short Answer Solutions

5.1 What is output by the following code?

```
String s1 = “hello “;
String s2 = s1;
s2 = s2 + “there “;
System.out.println(s1);
System.out.println(s2);

hello
hello there
```

5.2 Discuss how Java passes parameters to a method. Is this technique the same for primitive types and objects? Explain.

`Java passes all parameters by` *`value`*`. This means that the current value of the actual parameter is copied into the formal parameter in the method header. This technique is consistent between primitive types and objects because object` *`references`* `rather than objects themselves are passed. When an object (actually, an object reference) is passed, the current value of the reference is copied into the corresponding formal parameter in the method header.`

5.3 Explain why a static method cannot refer to an instance variable.

`A static method is invoked through a class rather than through an object of the class. No object of the class needs to be instantiated in order to invoke a static method. If no object is instantiated, no instance variable exists. Hence, a static method cannot refer to an instance variable.`

5.4 Can a class implement two interfaces that each contain the same method signature? Explain.

`Yes, a class can implement two interfaces each of which contains the same method signature. The class which implements an interface provides method implementations for each of the abstract methods defined in the interface. In satisfying the requirements for a method of one interface, it simultaneously satisfies the requirements for a method with the same signature in another interface.`

5.5 Create an interface called `Visible` that includes two methods: `makeVisible` and `makeInvisible`. Both methods should take no parameters and should return a `boolean` result. Describe how a class might implement this interface.

```
public interface Visible
{
      public boolean makeVisible ();
      public boolean makeInvisible ();
}
```

```
A class implementing Visible would begin with

      public class Icon implements Visible

and would contain at least two methods (makeVisible and makeInvisible), which take
no parameters and return a boolean value indicating the success of the method.
```

5.6 Create an interface called `VCR` with methods that represent what a video cassette recorder does (play, stop, etc.). Define the method signatures any way you want. Describe how a class might implement this interface.

```
public interface VCR
{
      public void play ();
      public void pause ();
      public void stop ();
      public void rewind ();
      public void fastforward ();
}
```

```
A class implementing VCR would include "implements VCR" in the class header and
would implement each of the five methods from the VCR interface. The methods would
behave as indicted by their name. For example, fastforward causes the VCR to start
fast-forwarding. This keeps going until the stop method is called or another
method such as play or rewind is called.
```

5.7 Given the `Num` and `ParameterTester` classes listed earlier in the chapter, what is the result of executing the following lines of code?

```
ParameterTester myTester = new ParameterTester();
int anInteger = 27;
Num aNum = new Num(38);
Num anotherNum = new Num(49);
myTester.changeValues(anInteger, aNum, anotherNum);
System.out.println("anInteger: " + anInteger);
System.out.println("aNum: " + aNum);
System.out.println("anotherNum: " + anotherNum);
```

```
Before changing the values:
f1      f2      f3
27      38      49

After changing the values:
f1      f2      f3
999     888     777

anInteger: 27
aNum: 888
anotherNum: 49
```

AP-Style Multiple Choice Solutions

1. D
2. B
3. B
4. A
5. E
6. A

AP-Style Free Response Solution

5.1

a.

```
implements Comparable
```

b.

```
public String getFirst()
{
   return first;
}
public String getLast()
{
   return last;
}

public int compareTo (Object otherName)
{
   if (this.last.compareTo(((Name)otherName).getLast()) == 0)
      return this.first.compareTo(((Name)otherName).getFirst());
   else
      return this.last.compareTo(((Name)otherName).getLast());
}
```

c.

```
public String toString ()
{
   return first + " " + last;
}
```