Chapter 9 Inheritance and Polymorphism

1. (a) The printout is

```
A's no-arg constructor is invoked
```

(b) The default constructor of B attempts to invoke the default of constructor of A, but class A's default constructor is not defined.

2. All false.
(1) A subclass is an extension of a superclass and normally contains more details information than its superclass.
(2) If a subclass's constructor explicitly invoke a superclass's constructor, the superclass's no-arg constructor is not invoked.
(3) You can only override accessible instance methods.
(4) You can only override accessible instance methods.

3. The following lines are erroneous:

```
{
  radius = radius; // Must use this.radius = radius
}

class B extends Circle (missing extends)

{
  Circle(radius);  // Must use super(radius)
  length = length; // Must use this.length = length
}

public double getArea()
{
  return getArea()*length; // super.getArea()
}
```

4. Method overloading defines methods of the same name in a class. Method overriding modifies the methods that are defined in the superclasses.

5. Yes, because these two methods are defined in the Object class; therefore, they are available to all Java classes. The subclasses usually override these methods to provide specific information for these methods.

The toString() method returns a string representation of the object; the equals() method compares the contents of two objects to determine whether they are the same.

6. `B's constructor is invoked`

A's constructor is invoked

The default constructor of Object is invoked, when new A(3) is invoked. The Object's constructor is invoked before any statements in B's constructor are executed.

7. (a)

(circle instanceof GeometricObject1) => true
(object1 instanceof GeometricObject1) => true
(circle instanceof Circle1) => true
(object1 instanceof Circle1) => false

(b)
Yes, because you can always cast from subclass to superclass.

(c)
Causing a runtime exception (ClassCastExcpetion)

8.

Is fruit instanceof Orange true? false

Is fruit instanceof Apple true? true

Is fruit instanceof GoldDelicious true? true

Is fruit instanceof Macintosh true? false

Is orange instanceof Orange true? true

Is orange instanceof Fruit true? true

Is orange instanceof Apple true? false

Suppose the method makeApple is defined in the Apple class. Can fruit invoke this method? Yes

Can orange invoke this method? No

Suppose the method makeOrangeJuice is defined in the Orange class. Can orange invoke this method? Yes.

Can fruit invoke this method? No.

9.

Object apple = (Apple)fruit;
Causes a runtime ClassCastingException.

10.

How do you create an ArrayList?
Use ArrayList list = new ArrayList();

How do you append an object to a list?
```
list.add(object);
```

How do you insert an object at the beginning of a list?
```
list.add(0, object);
```

How do you find out the number of objects in a list?
```
list.size();
```

How do you remove a given object from a list?
```
list.remove(object);
```

How do you remove the last object from the list?
```
list.remove(list.size() - 1);
```

How do you check whether a given object is in a list?
```
list.contains(object);
```

How do you retrieve an object at a specified index from a list?
```
list.get(index);
```

11.

```
String city = list.get(0);
```

Should be written as

```
String city = (String)list.get(0);
```

12. default visibility modifier.

13. protected.

14. If the question marks are replaced by blanks, can class B be compiled? Yes.
If the question marks are replaced by private, can class B be compiled? No.
If the question marks are replaced by protected, can class B be compiled? Yes.

15. If the question marks are replaced by blanks, can class B be compiled? No.
If the question marks are replaced by private, can class B be compiled? No.
If the question marks are replaced by protected, can class B be compiled? Yes.

16. The output is false if the Circle class in (A) is used. The Circle class has two **overloaded** methods: equals(Circle circle) defined in the Circle class and equals(Object

circle) defined in the Object class, inherited by the Circle class. At compilation time, circle1.equals(circle2) is matched to equals(Circle circle), because equals(Circle circle) is more specific than equals(Object circle).

The output is true if the Circle class in (B) is used. The Circle class overrides the equals(Object circle) method defined in the Object class. At compilation time, circle1.equals(circle2) is matched to equals(Circle circle) and at runtime, the equals(Object circle) method implemented in the Circle class is invoked.

17. The equals, hashCode, finalize, clone, and getClass methods are defined in the Object class. The finalize method is invoked on the object when the object is destroyed.

18. The clone method is defined in Object as protected. So this method cannot be invoked from every object. To invoke it the Class for the object must implement it and make it public and also the class must implement the Cloneable interface. See Chapter 9 for discussions on interfaces.

19. The return type of the getClass method is java.lang.Class. If two objects o1 and o2 have the same class type, o1.getClass() is same as o2.getClass().

20.

```
2 // because b's declared type is the class B.
4 // because b's declared type is the class B.
1 // because a's declared type is the class A.
3 // because a's declared type is the class A.
1 // because a's actually an object of B, so the m()
     method in B is invoked.
7 // because a's declared type is A, so the static
     method m1() in A is invoked.
```

21. If Line 3 is replaced by B x = new B(), the output is:

```
(1) x.i is 2
(2) (B)x.i is 2
(3) x.j is 12
(4) ((B)x).j is 12
(5) x.m1() is B's static m1
(6) ((B)x).m1() is B's static m1
(7) x.m2() is B's instance m2
(8) x.m3() is A's instance m3
```

If Line 3 is replaced by a x = new a(), you cannot cast to (B) in Lines 7, 11, and 15.

22. The output is the following for (A):

```
j is 2 (from Line 16)
i is 1 (from Line 12)
```

The static initialization block (Lines 15-17) is executed when class A is loaded. The instance non-static initialization block (Line 11-13) is executed when A's default constructor is invoked.

The output is the following for (B)

```
i is 0 (from Line 17)
j is 4 (from Line 18)
i is 5 (from Line 12)
j is 4 (from Line 13)
```

When new A() is invoked (Line 3), A's superclass's default constructor is executed. So, m() in B's constrcutor is executed. Since m() is defined in B (Lines 27-28), but overridden in A (Lines 11-14), the body of m() defined in A is invoked by polymorphism. The statement System.out.println in Line 17 is now executed. At this time, 5 has not been assigned to i yet. Therefore, the printout from Line 17 is (1) i is 0. The printout from Line 18 is (2) j is 4, since j is a static variable and it is initialized when class A is loaded. 5 will be assigned to i (Line 8) after A's superclass constructor is invoked. When the System.out.println statement in Line 12 is executed, i is 5. Therefore, the printout from Line 12 is i is 5. The printout form Line 13 is (4) j is 4.

23. If Line 3 is deleted, the output is

(1) InitializationBlockDemo's static initialization block is invoked

If Line 3 is is replaced by Bird bird = new Parrot(), the output is

(1) InitializationBlockDemo's static initialization block is invoked
(3) Bird's static initialization block is invoked
(4) Parrot's static initialization block is invoked
(5) Bird's instance initialization block is invoked
(6) Bird's constructor is invoked
(7) Parrot's instance initialization block is invoked
(8) Parrot's constructor is invoked

24. Find these terms in this chapter.

25. Indicate true or false for the following statements:

1. True.

2. False. (But yes in a subclass that extends the class where the protected datum is defined.)

3. True.

4. A final class can have instances.
   Answer: True

5. A final class can be extended.
   Answer: False

6. A final method can be overridden.
   Answer: False

7. You can always successfully cast a subclass to a superclass.
   Answer: True

8. You can always successfully cast a superclass to a subclass.
   Answer: False

9. The order in which modifiers appear before a method is important.
   Answer: False

26. See the NOTE in Section 8.6.

27. See the NOTE in Section 8.6.