# HERBERT SCHILDT'S *Java: A Beginner's Guide*, 6TH ED.

*Chapter 7 Self Test Answers*

1. **Does a superclass have access to the members of a subclass? Does a subclass have access to the members of a superclass?**

   A superclass has no knowledge of its subclasses. Subclasses can access all the members of a superclass other than those declared *private*. Private members can be accessed via getter and setter methods (assuming they exist).

2. **Create a subclass of *TwoDShape* called *Circle*. Include an *area()* method that computes the area of the circle and a constructor that uses *super* to initialize the *TwoDShape* portion.**

   The following program meets the specified criteria:

```
class TwoDShape {
  private double width;
  private double height;

  TwoDShape(double w, double h){
    width = w;
    height = h;
  }

  double getWidth(){
    return width;
  }

  double getHeight(){
    return height;
  }

  void setWidth(double w){
    width = w;
  }

  void setHeight(double h){
    height = h;
  }

  void showDim(){
    System.out.println("Width and height are "
                  + width + " and " + height);
  }
} // TwoDShape

class Circle extends TwoDShape{

  private double area;
  private double radius;

  Circle (double w, double h, double r){
    super(w, h);
    radius = r;
  }

  double area(){
    area = 2 * 3.14159 * radius;
    return area;
  }

  public static void main(String[] args){

    Circle circle1 = new Circle(5, 5, 12.37823431);

    System.out.println();

    System.out.printf("showDim() method from TwoDShape: ");

    circle1.showDim();
```

```
        System.out.println("The area of a circle with a radius of "
                            + circle1.radius + " is "
                            + circle1.area() + ". \n");

    } // end main
} // end Circle
```

3. **How do you prevent a subclass from having access to a member of the superclass?**

   Declaring a member of a superclass as *private* and not providing getter or setter methods will prevent a subclass from accessing it. Inheriting a class does not override the *private* access restriction.

4. **Describe the purpose and use of the two versions of *super* described in this chapter.**

   There are two uses of *super* in Java. The first is to call superclass constructors. In a subclass constructor by inserting the word *super* as the first line of the constructor (and any appropriate parameters). This will call the constructor of the superclass, and then any additional information is added in by the constructor of the subclass. An example would be the following:

```
class Dog extends Animal{
  Dog (int numberOfLegs, int weight, String color){
    super(numberOfLegs, weight);  // call superclass constructor
    dogColor = color;
  }
}
```

   The second use of the keyword *super* is when the subclass and the superclass both have members with the same names. In this case, the subclass member will mask the superclass member and thereby disallow access to the superclass member via typical access methods. In order to remedy this, simply insert the keyword *super* in front of the member name when accessing it. For example, if the subclass and superclass had a member named *ambiguous*, then they could be accessed separately via:

```
System.out.println(super.ambiguous);  // print the superclass
                                      // member named ambiguous
System.out.println(ambiguous); // print the subclass member named ambiguous
```

5. **Given the following hierarchy, in what order do the constructors for these classes complete their execution when a *Gamma* object is instantiated?**

```
class Alpha {...}
class Beta extends Alpha{...}
class Gamma extends Beta {...}
```

   With Java, the constructors are created based on the superclass(es) first. This means when a Gamma object is created, the order of execution in the given hierarchy would be Alpha, Beta, Gamma.

6. **A superclass reference can refer to a subclass object. Explain why this is important as it relates to method overriding.**

   Overridden methods allow Java to support run-time polymorphism. It is the type of the object being referred to that determines which version of an overridden method will be executed. This is a very important principle in Java and is known as *dynamic method dispatch*.

7. **What is an abstract class?**

   An abstract method is one which is defined in the superclass, but has no body and is therefore not implemented by the superclass. It must be overridden by the subclass. Abstract methods cannot be *static*. Abstract methods are used to create an outline for subclasses within the superclass, but are not defined in any meaningful way in the superclass. Subclasses implement the abstract methods provided by the superclass, thus giving meaning to the “blueprint” provided by the superclass.

8. **How to you prevent a method from being overridden? How to you prevent a class from being inherited?**

   Methods cannot be overridden if they are declared *final*. The *final* keyword also prevents classes from being inherited. An example of a method that cannot be overridden is as follows:

```
class Example {
  final void noOverrideMethod{
    System.out.println("This method cannot be overridden.");
  }
}
```

9. **Explain how inheritance, method overriding, and abstract classes are used to support polymorphism.**

   Polymorphism is essential to object-oriented programming because it allows a general class to specify the methods that will be common to all of its derivatives, while allowing all subclasses to define the specific implementation of some or all of these methods (Schildt, p. 255). Inheritance, method overriding and abstract classes all support different methods of implementing polymorphism. Inheritance allows subclasses to inherit from superclasses; method overriding allows methods with the same signature to react differently based on the context in the code; abstract classes make it easy to create blueprint superclasses that are actually implemented in subsequent subclasses.

10. **What class is a superclass of every other class?**

    In Java, the ultimate superclass is known as *Object*. All other classes are a subclass of *Object*, even without an explicit inheritance declaration. It is simply built into the Java language.

11. **A class that contains at least one abstract method must, itself, be declared abstract. True or False?**

    True. If the class contains an abstract method, it is by definition abstract and an object of this class cannot be instantiated.

12. **What keyword is used to create a named constant?**

    A named constant is created using the keyword *final*. This is a very useful programming concept - by using a named constant when writing code, it is easy for the programmer to modify that constant by only changing one variable instead of going through the code and changing it throughout. The amount of interest on a loan, for example, is something that is often declared as *final*. For example:

    ```
    private final int LOAN_INTEREST = 2;
    ```