Group 31

Brennon Francis - bqfranci - bqfranci@asu.edu
Andronick Martusheff - aamartus - aamartus@asu.edu

HW2

3. Objects, Classes, and Inheritance

3.1
It is not required. Only when using the variable outside of its class do we have to provide an accessor/mutator method.

3.2
B & C from the given statements are legal. X cannot be set to Y because a superclass cannot refer to a subclass object, but the opposite is possible. Y can be set to X because the subclass can hold the extended properties of its superclass.

3.3
True

3.4
False

3.5
//C1 variables - C1 method
1. x1, x2, and x3 are all directly accessible because they are in the same class.
2. x1, x2, and x3
3. x1, x2, and x3

//C1 variables - C2 method

4. x1 and x2
5. x1 and x2 - protected method
6. x1 and x2 - private method

//C1 variables - C3 method

7. x1 and x2 - public method
8. x1 and x2 - protected method
9. x1 and x2 - private method

//C2 variables - C1 method

10. y1 and y2 - public method
11. y1 and y2 - protected method

12. y1 and y2 - private method
    //C3 variables - C1 method
13. z1 and z2 - public method
14. z1 and z2 - protected method
15. z1 and z2 - private method
    //C1 Methods called to C2 Methods
16. c1Method1() and c1Method2() - called from a public method
17. c1Method1() and c1Method2() - called from a protected method
18. c1Method1() and c1Method2() - called from a private method
    //C1 Methods called to C3 Methods
19. c1Method1() and c1Method2() - called from a public method
20. c1Method1() and c1Method2() - called from a protected method
21. c1Method1() and c1Method2() - called from a private method
    //C2 Methods called to C1 Methods
22. c2Method1() and c2Method2() - called from a public method
23. c2Method1() and c2Method2() - called from a protected method
24. c2Method1() and c2Method2() - called from a private method

25. 3
26. 5
27. 7

3.6

Overloaded Methods:
Method overloading allows for classes to have multiple methods of the same name as long as the input parameters are different.

```
public class H2TestCode {
   public static void main(String[] args) {
       System.out.println(add(1,2));
       System.out.println(add(3,4,9));
       System.out.println(add("Hello", "World")); }

   public static int add(int a, int b){
       return a + b; }
   public static int add(int a, int b, int c){
       return a + b + c; }
   public static String add(String a, String b){
       return a + " " + b; }}
```

3.7

Overridden Methods:

A derived class (or subclass) inherits methods from a base/parent class (superclass), and they may sometimes not do exactly what you would like them to do. The method could be overridden, and held specific to the subclass it was overridden inside of. It is recommended that when overriding a method you use the “@override” annotation.

// Main
```
public class Demo {
   public static void main(String[] args) {
       Monster m = new Monster("Monster", 100);
       Skeleton s = new Skeleton("Skeleton", 50, 200);
       m.stats();
       s.stats(); } }
```
// Monster
```
public class Monster {
   private String name;
   private int attack;

   public void stats() {
       System.out.println(name + ": Attack - " + attack); }

   public Monster() {
       name = "unknown type";
       attack = 0; }

   public Monster(String name, int attack){
       this.name = name;
       this.attack = attack; }

   public int getAttack(){
       return this.attack; }
   public String getName(){
       return this.name; }}
```

// Skeleton
```
public class Skeleton extends Monster {
   private int range;

   public Skeleton() {
       super();
       range = 0; }

   public Skeleton(String name, int attack, int range){
       super(name, attack);
       this.range = range; }

   @Override
   public void stats(){
       System.out.println(getName() + ": Attack - " + getAttack() + ", Range
- " + range); } }
```

3.8

Preventing overloaded methods?

Method overloading is a way of creating multiple methods with the same name but different input parameters. (Ex. 3.6) In Java, you are able to prevent methods from being overwritten (by making them final), but you cannot prevent methods from being overloaded. Depending on your IDE, when hovering over the input field for a method, you may see different parameter options. This would be the best way to make sure you are inputting the correct information into the correct method.

3.9

How to call overridden superclass method in overridden child class method:

If you've overridden a superclass method in a child class, you can still access the original method by calling it as follows:

```
@Override
public void stats(){
   super.stats();
   System.out.println(getName() + ": Attack - " + getAttack() + ", Range - " +
range); } }
```

If you compare this overridden method within the Skeleton class to the one seen in section 3.7, you'll see that the superclass method was accessed by calling it with `super.stats();`. In our case, the name and attack of our Skeleton will output twice, but the range will only output once. This is because the original output (as called in `super.stats();`) only accounted for the name and attack parameter.

3.10

In order to save space, the solution to section 3.6 & 3.10 were merged in section 3.6. There you will find the method "add" declared 3 times, each with their own sets of parameters. The methods are also called in the main block. This code can be copied, pasted, and tested.

Is it legal to write a method in a class which overloads another method declared in the same class?

Yes, it is! This is basically the definition of method overloading. Multiple methods can share the same name (be overloaded) as so long as each method has its own input parameters.

3.11

Is it legal to write a method in a class which overrides another method declared in the same class?

No! Overridden methods are reserved for subclasses.

3.12

Is it legal in a subclass to write a method which overloads a method declared in the superclass?

Yes! Because the subclass and superclass have an inheritance relationship, the subclass inherits methods from the superclass. So methods could be overloaded just the same as if they were all in one single class. If the same input parameters are given, the subclass will overwrite the superclass method.

3.13

Is it legal to write a method in a subclass which overrides a method declared in the superclass?
Yes! Having a method in the subclass change a method from the superclass is the purpose of overriding.

3.14

This exercise was removed from the assignment.

3.15

Is it legal to write a method in a superclass which overrides a method declared in a subclass?
No. The subclass inherits the public fields of the superclass. If you were looking to create methods/fields that are specific to the superclass, they could be written in private fields and won't be inherited. Also, if you wanted a subclass to have access to a particular private superclass method, you could use a protected method to access it.

3.16

When in a subclass constructor, the default constructor of the superclass is called before the subclass constructor begins executing. If you wanted to call a different constructor of the superclass, you would do so by immediately calling the super class super(<input>);. By doing this, you bypass the default constructor and create an object of the superclass with the parameters specified with your super(); call.

3.17

In Java, you can consider a concrete class to have some level of implementation/object creation. An abstract class relies on inheritance. If we had a class 'Shape' with subclasses 'Circle' and 'Rectangle', where 'Shape' is dependent on subclasses 'Circle' and 'Rectangle', 'Shape' could handle the shapes 'Circle' and 'Rectangle' constructed within it, but is not itself a constructor of either of the shapes.

4. Objects, Classes, Polymorphism, Interfaces

4.1

The makeSound() method is the method that is polymorphically called because it is being used in all three super classes and there are different sounds for each critter in the array list.

4.2 - 4.5

File submitted separately.

5. GUI Programming

5.1-5.3

File submitted separately.

6. Nested Classes

6.1

An inner class is a class that is declared within another class (outer class). It is able to access outer class variables.

6.2

Local classes are a type of inner class. Unlike inner classes, local classes are declared within a “block”. For example it can be declared in a method or in “if-else” statements. Since they are created in a block they do not have to be created along with an object. Local classes are not accessible from outside the blocks in which they are declared. Although, they have access to the instance variables of the outer class as well as final variables in the block. They cannot access any local variables that were declared in the block.

6.3

Anonymous classes are a type of inner class. They do not have a name assigned to it. Unlike local classes, anonymous classes and an object are both created at the same time. It is not a class declaration like the others, but it is an expression (like an expression you would find in math). Since it is an expression, anonymous classes can be a part of a larger expression, such as a method call.