***CSC 3020***

***Java Programming***

**Assignment 06**

**40 points**

**Due 11/13/2023 (11:45 A.M.)**

Assignment Objectives:

■■ To define a subclass from a superclass through inheritance.

■■ To invoke the superclass’s constructors and methods using the **super** keyword.

■■ To override instance methods in the subclass.

■■ To distinguish differences between overriding and overloading.

■■ To explore the **toString()** method in the **Object** class.

■■ To discover polymorphism and dynamic binding.

■■ To describe casting and explain why explicit downcasting is necessary.

■■ To explore the **equals** method in the **Object** class.

■■ To store, retrieve, and manipulate objects in an **ArrayList**.

■■ To construct an array list from an array to sort and shuffle a list, and to obtain max and min element from a list.

■■ To enable data and methods in a superclass accessible from subclasses using the **protected** visibility modifier.

■■ To prevent class extending and method overriding using the **final** modifier.

**Solution to this assignment will not be posted on Canvas; however, any question can be discussed in the class upon request of a student.**

All assignments must be submitted by the Canvas. **No email or hard copy** is accepted. You must follow the following format:

1. For non-programming questions, use a word file to type your answers. Don’t use the text box on the Canvas to answer the questions or to write comments, we will not read it.
2. State your answer clearly.
3. For programming questions, include only the source file for each problem.
4. Submit your file to the Canvas. You must submit your assignment on time; otherwise, you will receive zero. In addition, you cannot submit your file more than one time.
5. There will be several folders on the Canvas. You need to upload your file(s) using the correct folder on the Canvas.
6. Name each file: “Assignment Number(Question number(s))”.
7. To upload your file(s):

* In Course Navigation, click the Assignments link.
* Click the title of the assignment.
* Click the **Submit** Assignment button.
* Add **File**. ...
* Add Another **File**. ...
* **Submit** Assignment. ...
* View **Submission**.

**It is your responsibility to make sure that each file is uploaded correctly. If you uploaded a wrong file, you receive zero; files will not be accepted after due date even if you have a prove that the file is created before the due date.**

**Make sure you review the Cheating & Plagiarism policy on Canvas.**

**Answer questions 1 to 10 in a separate single .txt file; include only your answers with questions numbers. Write a program for each of Q.11 - Q.12; save each program in a .txt file. Submit total of 3 .txt files by the due date.**

**Question 01 (6 points)**

Answer the following questions? Provide clear full answer as discussed in the class.

1. How does inheritance promote software reusability?

Software becomes more reusable using inheritance in java because it allows using methods and variables defined in parent classes. Thus instead of rewriting code, one can simply extend a parent class.

1. Explain protected member access.

Protected member access allows access by subclasses but not through public classes.

1. Explain the difference between composition (i.e., the *has-a* relationship) and inheritance (i.e., the *is-a* relationship).

Composition is a relationship between an instance of one class and a class, while inheritance is a relationship between a class and its superclass.

1. Explain how to invoke a superclass method from a subclass method for the case in which the subclass method overrides a superclass method and the case in which the subclass method does not override a superclass method.  
   To access a super class method with the same name in the subclass, simply use the super keyword followed by ‘.’ and the class name. The super keyword is not needed otherwise if the name is not shared in the superclass and subclass.
2. What is single inheritance? What is multiple inheritance? Does Java support multiple inheritance?  
   Single inheritance is inheriting attributes from only one parent class. Multiple inheritance is inheriting attributes from multiple parent classes. Java does NOT support multiple inheritance.
3. Explain the difference between method overloading and method overriding.  
   Method overriding is defining multiple methods with the same name, but different lists of parameters. Method overriding is when a subclass shares the same name and parameters as a method in the superclass.
4. What is the benefit of using the @Override annotation?  
   The @Override annotation helps check during compilation to ensure that the method with the annotation is an override.
5. How do you prevent a class from being extended? How do you prevent a method from being overridden?  
   Putting the final keyword in front of a class declaration prevents it from being extended. Placing the final keyword in front of a method declaration similarly prevents it from being extended.
6. What is dynamic binding?  
   Dynamic binding is when the method implementation to be invoked is determined during runtime, instead of during compile time.
7. Does every object have a toString method? Where does it come from? How is it used? Is it appropriate to override this method?  
   Every object in java has a toString method from the object class which is the basis of all java classes. This method returns a string of the class name followed by @ and the object’s hashcode. It is appropriate to override this method, it is clunky and does not necessarily provide a meaningful string when you want it to.
8. What modifier should you use on a class so that a class in the same package can access it, but a class in a different package cannot access it?  
   Private.
9. What modifier should you use so that a class in a different package cannot access the class, but its subclasses in any package can access it?

Protected.

**Question 02 (3 points)**

Indicate true or false for the following statements:

1. A protected datum or method can be accessed by any class in the same package.   
   True.
2. A protected datum or method can be accessed by any class in different packages.   
   False
3. A protected datum or method can be accessed by its subclasses in any package.   
   True.
4. A final class can be extended.   
   False.
5. A final method can be overridden.  
   False.
6. You can override a private and static methods defined in a superclass.  
   False.

**Question 03 (2 points)**

What is the output of running the class C in (a)?:  
“Error, class C is public, should be declared in a file named C.java”.  
However, in a more general sense, C creates an instance of B, which is an extension of A. B doesn’t have a constructor, so it uses A, which prints “A’s no arg- constructor is invoked”.

What problem arises in compiling the program in (b)?  
C creates an instance of B, which is an extension of A. B however, lacks a constructor, and in its creation it is created without any parameters. Therefore when B is created, it calls the constructor in A, however A lacks a no-argument constructor, and therefore it fails to compile.

(a)

**class** A {

**public** A() {

System.out.println("A's no-arg constructor is invoked");

}

}

**class** B **extends** A {

}

**public** **class** C {

**public** **static** **void** main(String[] args) {

B b = **new** B();

}

}

(b)

**class** A {

**public** A(**int** x) {

}

}

**class** B **extends** A {

**public** B() {

}

}

**public** **class** C {

**public** **static** **void** main(String[] args) {

B b = **new** B();

}

}

**Question 04 (2 points)**

Identify the problems in the following code:

//this is the circle class thing. Would be nice if in the future you didn’t include the numbers, it makes loading it into an online compiler very difficult to see the compilation errors.  
 1 **public** **class** Circle {

2 **private** **double** radius;

3

4 **public** Circle(**double** radius) {

5 radius = radius; //use this.radius instead to assign radius to the instance of the class. Are all circles ever in existence being changed? No – that would be silly.

6 }

7

8 **public** **double** getRadius() {

9 **return** radius;

10 }

11

12 **public** **double** getArea() {

13 **return** radius \* radius \* Math.PI; //this looks fine.

14 }

15 }

16

//this is the extension of the circle class.

17 **class** B **extends** Circle {

18 **private** **double** length;

19

20 B(**double** radius, **double** length) {

21 Circle(radius);//what circle? CIRCLE ISN’T REAL – ONLY SUPER IS. Use super(radius) – it defines the radius in the super class of this instance.

22 length = length; //this should be this.length

23 }

24

25 @Override

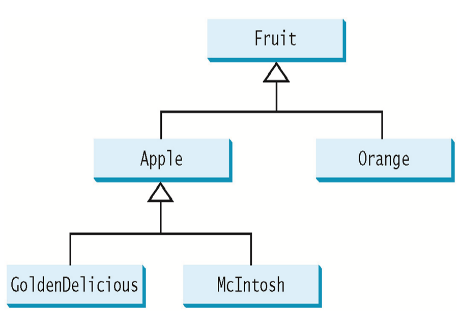
26 **public** **double** getArea() {

27 **return** getArea() \* length; //I assume this is trying to call the super class. Use super.getArea()\*length to get the actual length.

28 }

29 }

**Question 05 (7 points)**

Suppose that Fruit, Apple, Orange, GoldenDelicious, and McIntosh are defined in the following inheritance hierarchy:   
   
Assume that the following code is given:

Fruit fruit = **new** GoldenDelicious();

Orange orange = **new** Orange();

Answer the following questions: 

1. Is fruit instanceof Fruit?   
   Yes
2. Is fruit instanceof Orange?   
   No.
3. Is fruit instanceof Apple?   
   Yes.
4. Is fruit instanceof GoldenDelicious?

No.

1. Is fruit instanceof McIntosh?

No.

1. Is orange instanceof Orange?

Yes.

1. Is orange instanceof Fruit?   
   Yes.
2. Is orange instanceof Apple?   
   No.
3. Suppose the method makeAppleCider is defined in the Apple class. Can fruit invoke this method? Can orange invoke this method?   
   Fruit can invoke this method, orange cannot.
4. Suppose the method makeOrangeJuice is defined in the Orange class. Can orange invoke this method? Can fruit invoke this method?   
   orange can invoke this method, fruit cannol.
5. Is the statement Orange p = new Apple() legal?   
   No.
6. Is the statement McIntosh p = new Apple() legal?   
   No.
7. Is the statement Apple p = new McIntosh() legal?   
   Yes.
8. True or False: You can always successfully cast an instance of a subclass to a superclass and cast an instance of a superclass to a subclass.  
   False.

**Question 06 (2 points)**

Show the output of the following code and explain the output:

(a)

**public** **class** Test {

**public** **static** **void** main(String[] args) {

**new** Person().printPerson();

**new** Student().printPerson();

}

}

**class** Student **extends** Person {

@Override

**public** String getInfo() {

**return** "Student";

}

}

**class** Person {

**public** String getInfo() {

**return** "Person";

}

**public** **void** printPerson() {

System.out.println(getInfo());

}

}

//the above code.PrintPerson is a method in the parent class of Student, it runs the getInfo method which returns “Person”. However, in the subclass Person, ‘getinfo’ has been defined to override its parent class’s method, which instead returns “Student”. Therefore when an instance of Person is created, it returns ‘person’ but when an instance of Student is created, it returns ‘Student. The above code generates the following result:  
//Person  
//Student  
//

(b)

**public** **class** Test {

**public** **static** **void** main(String[] args) {

**new** Person().printPerson();

**new** Student().printPerson();

}

}

**class** Student **extends** Person {

**private** String getInfo() {

**return** "Student";

}

}

**class** Person {

**private** String getInfo() {

**return** "Person";

}

**public** **void** printPerson() {

System.out.println(getInfo());

}

}

//Remember reading that boring answer in question a, well bad news, class Student has used the private keyword – which means that the printPerson function can only be accessed within the class Student – because the PrintPerson() function is within class Person, it is not accessed by the printPerson() method. The output is as follows:  
//Person  
//Person

**Question 07 (3 points)**

Show the output of following program:

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Apple a = **new** Apple();

System.out.println(a);

System.out.println("---------------");

GoldenDelicious g = **new** GoldenDelicious(7);

System.out.println(g);

System.out.println("---------------");

Apple c = **new** GoldenDelicious(8);

System.out.println(c);

}

}

**class** Apple {

**double** weight;

**public** Apple() {

**this**(1);

System.out.println("Apple no-arg constructor");

}

**public** Apple(**double** weight) {

**this**.weight = weight;

System.out.println("Apple constructor with weight");

}

@Override

**public** String toString() {

**return** "Apple: " + weight;

}

}

**class** GoldenDelicious **extends** Apple {

**public** GoldenDelicious() {

**this**(5);

System.out.println("GoldenDelicious non-arg constructor");

}

**public** GoldenDelicious(**double** weight) {

**super**(weight);

**this**.weight = weight;

System.out.println("GoldenDelicious constructor with weight");

}

@Override

**public** String toString() {

**return** "GoldenDelicious: " + weight;

}

}  
  
//Here is the output of the above code:  
/\*  
Apple constructor with weight

Apple no-arg constructor

Apple: 1.0

---------------

Apple constructor with weight

GoldenDelicious constructor with weight

GoldenDelicious: 7.0

---------------

Apple constructor with weight

GoldenDelicious constructor with weight

GoldenDelicious: 8.0

\*/

**Question 08 (3 points)**

In the following code, the classes A and B are in the same package. If the question marks in (a) are replaced by blanks, can class B be compiled? If the question marks are replaced by private, can class B be compiled? If the question marks are replaced by protected, can class B be compiled?

If the question marks are replaced by blanks, Class B can be compiled. If the question marks are replaced by private, Class B cannot be compiled. If the question mark are replaced by protected, class B can be compiled.

**package** p1;

**public** **class** A {

? **int** i;

? **void** m() {

...

}

}

**package** p1;

**public** **class** B **extends** A {

**public** **void** m1(String[] args) {

System.out.println(i);

m();

}

}

**Question 09 (3 points)**

In the following code, the classes A and B are in different packages. If the question marks in class A are replaced by blanks, can class B be compiled? If the question marks are replaced by private, can class B be compiled? If the question marks are replaced by protected, can class B be compiled?  
Blanks: cannot compile B  
Private: cannot compile B  
Protected: CAN compile B.

**package** p1;

**public** **class** A {

? **int** i;

? **void** m() {

...

}

}

**package** p2;

**public** **class** B **extends** A {

**public** **void** m1(String[] args) {

System.out.println(i);

m();

}

}

**Question 10 (2 points)**

In the following code, the classes A, B, and Main are in the same package. Can the Main class be compiled?

Yes.

**class** A {

**protected** **void** m() {

}

}

**class** B **extends** A {

}

**class** Main {

**public** **void** p() {

B b = **new** B();

b.m();

}

}

**Question 11 (3 points)**

Write a method that returns the union of two array lists of integers using the following header:

**public static** ArrayList<Integer> union(ArrayList<Integer> list1, ArrayList<Integer> list2)

For example, the addition of two array lists {2, 3, 1, 5} and {3, 4, 6} is {2, 3, 1, 5, 3, 4, 6}. Write a test program that prompts the user to enter two lists, each with five integers, and displays their union. The numbers are separated by exactly one space. Here is a sample run:

|  |
| --- |
| Enter five integers for list1: 3 5 45 4 3  Enter five integers for list2: 33 51 5 4 13  The combined list is 3 5 45 4 3 33 51 5 4 13 |

**Question 12 (4 points)**

Create a base class called Vehicle that has private data fields: the manufacturer’s name (type String) and number of cylinders in the engine (type int). Then create a class called Truck that is derived from Vehicle and has additional property: the load capacity in tons (type Double).

Give your classes no argument and argument constructors (do not create mutators or accessors methods). Override equals and toString methods in each class. Write a driver program that create two objects of class Truck, invoke method toString in each object, and compare both objects.

|  |
| --- |
| Vehicle: manufacturer= Ford, cylinders=5, load= 1.5  Vehicle: manufacturer= Honda, cylinders=6, load= 1.75  truck1 and truck2 are not the same  Vehicle: manufacturer= Ford, cylinders=5, load= 1.5  Vehicle: manufacturer= Ford, cylinders=5, load= 1.5  truck3 and truck4 are the same |