***CSC 3020***

***Java Programming***

**Assignment 05**

**50 points**

**Due 03/25/2021 (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 question 1 to 10 on a word file; write a program for Q.11.

**Question 01 (13 points)**

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

1. How does inheritance promote software reusability?
   1. Inhertinance allows for objects to acquire the methods of the parent class thus allowing them to be used withot having to be rewritten.
2. Explain protected member access.
   1. Protected member access allows for access from its own class or from a subclass, but not from the world.
3. Explain the difference between composition (i.e., the *has-a* relationship) and inheritance (i.e., the *is-a* relationship).
   1. Inheritance describes a class that **is-an** extension of another class, where as composition describes a class which **has** objects from other classes.
4. 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.
   1. you can invoke a superclass method from a subclass method by using the super keyword.
   2. Or in the case of not overriding it you can create an object of the subclass to access it.
5. What is single inheritance? What is multiple inheritance? Does Java support multiple inheritance?
   1. when a subclass is only an extension of one superclass.
   2. when a subclass is an extension of multiple classes.
   3. This is not supported by Java.
6. Explain the difference between method overloading and method overriding.
   1. overloading is when you have more than one method with the same name, and method overriding is when you are modifing a method that has been implemented in the superclass.
7. What is the benefit of using the @Override annotation?
   1. it reduces compiler errors by making the instructor know that you are overriding and not creating a new method.
8. How do you prevent a class from being extended? How do you prevent a method from being overridden?
   1. You can use the word final for both cases.
9. How does polymorphism enable you to program “in the general” rather than “in the specific”?
   1. It allows you to use the same methods and implement them in different ways.
10. What is dynamic binding?
    1. Dynamic binding is when a method is created in multiple inherited classes.
11. Does every object have a toString method? Where does it come from? How is it used? Is it appropriate to override this method?
    1. yes, they comes from the objcet class. toString creates a string out of a given object and equals will compare two objects. Yes, it is appropriate to override them.
12. 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?
    1. By using the default modifier.
13. 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?
    1. You should use protected.

**Question 02 (3.5 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 method defined in a superclass. false
7. You can override a static method defined in a superclass. false

**Question 03 (4 points)**

What is the output of running the class C in (a)?

What problem arises in compiling the program in (b)?

(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();

}

}

**Output:**

A’s no-arg constructor is invoked

(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();

}

}

A does not have a default constructor for B to inherit.

**Question 04 (3 points)**

Identify the problems in the following code:

1 **public** **class** Circle {

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

3

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

5 radius = radius;

6 }

7

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

9 **return** radius;

10 }

11

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

13 **return** radius \* radius \* Math.PI;

14 }

15 }

16

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

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

19

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

21 Circle(radius);

22 length = length;

23 }

24

25 @Override

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

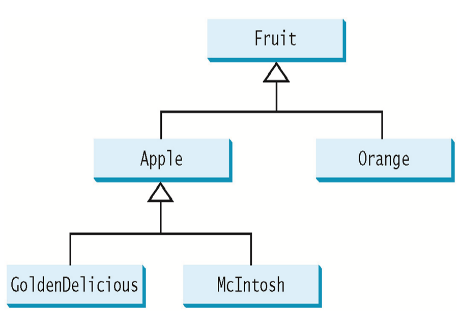
27 **return** getArea() \* length;

28 }

29 }

Should use this.radius on line 5, and this.length on line 22. Circle does not have a default constructor, Line 21, should be super(radius). Line 27 should be super.getArea().

**Question 05 (7.5 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? yes
5. Is fruit instanceof McIntosh? no
6. Is orange instanceof Orange? yes
7. Is orange instanceof Fruit? yes
8. Is orange instanceof Apple? no
9. Suppose the method makeAppleCider is defined in the Apple class. Can fruit invoke this method? Can orange invoke this method?
   1. fruit cannot invoke this class, Orange cannot invoke this method either.
10. Suppose the method makeOrangeJuice is defined in the Orange class. Can orange invoke this method? Can fruit invoke this method?
    1. Orange can invoke this method, fruit cannot invoke this method as it is a parent of this class.
11. Is the statement Orange p = new Apple() legal? No, no inheritance.
12. Is the statement McIntosh p = new Apple() legal?  No, McIntosh is a subclass of Apple.
13. Is the statement Apple p = new McIntosh() legal? Yes, Apple is a superclass of McIntosh.
14. True or False: You can always successfully cast an instance of a subclass to a superclass.
    1. True
15. True or False: You can always successfully cast an instance of a superclass to a subclass.
    1. 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());

}

}

**Output:**

Person

Student

**Reason:**

This is modifying the getInfo method with @Override

(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());

}

}

**Output:**

Person

Person

**Reason:**

Does not modify method so it calls method in Person class.

**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;

}

}

**Output:**

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?

**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();

}

}

Replacing with blanks yes, private no, protected yes they can.

**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?

**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();

}

}

Replacing with blanks no, private no, protected yes they can.

**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?

**class** A {

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

}

}

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

}

**class** Main {

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

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

b.m();

}

}

Yes, because m() is a protected method in class A, it can be accessed by object b as class B is an extension of class A.

**Question 11 (6 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:

