CSUS
COLLEGE OF ENGINEERING AND COMPUTER SCIENCE
Department of Computer Science

CSc 133 — Object-Oriented Graphics Programming (Spring 2019)

Name TALAL JAWAID

MIDTERM EXAM

1. Write your name in the space above.
2. The exam is closed book, closed notes, except that you may use a *single sheet* of *hand-written* notes. If you use a note sheet you must put your name on it and turn it in with your exam.
3. There are 100 total points; you have 60 minutes to work on it — budget your time accordingly.
4. Absolutely NO use of ANY electronic devices is allowed during the exam. This includes cell phones, tablets, laptops, or any other communications device.
5. Please be neat — I cannot give credit to answers I cannot read.
6. The exam has 13 pages (including the work space), counting this cover page. Make sure you have all the pages.

Problem	Points	Possible
1	34	50
2	7	15
3	5	15
4	10	20
Total	56	100

1. Multiple Choice/Selection, Short Answers. Write the letter of the best answer in the blank to the left. (50 points)

A Process that involves recognizing and focusing on the important characteristics of a situation or object is known as:

A. abstraction B. encapsulation C. inheritance D. overloading

B Which statement is true about accessibility of members?

A. Private members are always accessible from within the same package.

B. Private members can only be accessed by code from within the class of the member

C. A member with default accessibility can be accessed by any subclass of the class in which it is defined

D. Private members cannot be accessed at all

D Which of the following class relationships best fits the composite pattern?

A. Zoo contains a Set, an Exhibit contains a Set, and an Animal contains a number of properties about that individual animal. To get information about a particular Animal, a client would write something such as:
Zoo.getExhibit("Penguins").getPenguin("Tux").getAge();

B. Dalmatian is a subclass of Dog, which is a subclass of Mammal, which is a subclass of Animal. Each subclass overrides some methods while using the inherited version of others, for some shared behavior and some distinct behavior.

C. GeometricShape is an interface implemented by Square, Circle, Sphere, and Dodecahedron. Though they have the same public interface and can all be used anywhere a GeometricShape is required, they otherwise have no relationship and do not depend on each other.

D. The class Food is implemented by PeanutButterAndJellySandwich, which contains objects of type Bread, PeanutButter, and Jelly. Bread contains Flour and Salt, and Jelly contains Fruit and Sugar. All of these objects are Food objects themselves.

E. None of the above

B A certain Java/CN1 class named "B" extends another class named "A". Class B defines a method named "C" with the same signature as that of a method named "C" in Class A. Method C does not contain the keyword "super". A program constructs an instance of B and invokes method "C" in that object. The code which will be executed as a result of this invocation is

A. the code in A.C
B. the code in B.C
C. the code in A.C followed by the code in B.C
D. the code in B.C followed by the code in A.C
E. it depends on the code in A.C
F. None of the above

B The Output of the following code is correct:

```
class Vehicle {
    public void applyBrakes() {
        System.out.printf ("Applying vehicle brakes\n");
    }
}

class Airplane extends Vehicle {
    public void applyBrakes() {
        System.out.printf("Applying Airplane brakes...\n");
    }
}
class Tank extends Vehicle {
    public void applyBrakes() {
        System.out.printf("Applying Tank brakes...\n");
    }
}

class Test {
    public static void main (String [] args){
        Vehicle v ;
        Airplane a = new Airplane();
        Tank t = new Tank();
        v = a;
        t = (Tank) v;
        t.applyBrakes();
    }
}
```

Output

```
Applying Tank brakes...
```

A. True.
B. False.

C When would you use a private constructor?

A. When you get bored with public
B. If you want to disallow instantiation of that class from outside that class
C. If you want to protect your class's members from outside modification
D. Never, it's not allowed

A If a Java/CN1 program contains a declaration such as "class A {...}", where "..." represents the code defining the class, then

A. A has no parent class
B. A is its own parent
C. A is a superclass of Object
D. A is an abstraction of Object
E. A is a subclass of Object

D In object-oriented programming, new classes can be defined by extending existing classes. This is an example of.

A. Composition
B. Encapsulation
C. Ambiguous
(D.) Inheritance

_N/A_ Explain what is Polymorphism in context of CN1/Java? How this concept is being utilized in your assignment 1? Motivate your answer with a short of snippet code.

Polymorphism: Polymorphism in context of CN1/Java is when the parent and child classes have methods w/ same signature but different implementations. So method call on child object calls method from child class instead of parent.

Example: In A1, enemy Missile extends Missile. Both classes have fire() methods with same signature but different implementations.

```
EnemyMissile eMissile = new EnemyMissile();
eMissile.fire(); // calls fire() from EnemyMissile, not fire() from Missile class
```

A A certain Java/CN1 class named **`Sphere`** contains a method named **`getColor()`** which returns the color of the **`Sphere`** object. This method is an example of a (an)

(A.) accessor B. mutator C. aggregation D. design pattern E. abstraction

C Which of the following describes the Singleton pattern correctly?

A. This pattern creates object without exposing the creation logic to the client and refer to newly created object using a common interface.
B. In this pattern an interface is responsible for creating a factory of related objects without explicitly specifying their classes.
(C.) This pattern involves a single class which is responsible to create an object while making sure that only single object gets created.
D. This pattern is used when we want to pass data with multiple attributes in one shot from client to server.

B Which of the following pattern is used when there is one-to-many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically?

X
A. Iterator.
(B.) Factory.
C. Observer.
D. None of the above.

A What must a non-abstract child do about an abstract method in its parent class?

(A.) A child must override an abstract method inherited from its parent by defining a method with the same signature and same return type.

B. A child must define an additional method similar to the one inherited from its parent by defining a method with the same signature and different return type.

C. A child must not define any method with the same signature as the parent's abstract method.

D. A non-abstract child must define an abstract method with the same signature and same return type as the parent's abstract method.

C If the child object can exist beyond the lifetime of its parent, then the relationship is:

A. Generalization.
B. Composition.
C. Aggregation.
D. None of the above.

C What relationship is appropriate for a Book and its Chapters?

A. Association.
B. Aggregation.
C. Composition.
D. Dependency.
E. None of the above.

B In Java, declaring a class abstract is useful

A. To prevent developers from further extending the class.
B. When it doesn't make sense to have objects of that class.
C. So that it cannot be inherited from.
D. Because it has no abstract methods.
E. None of the above.

B The wrapping up of data and functions into a single unit is called

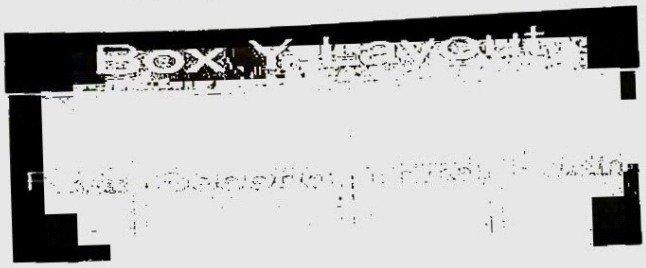
A. Abstraction
B. Encapsulation
C. Data Hiding
D. Capsule

A The following code:

TRUE

```
Form h1 = new Form("Box Y Layout", new BoxLayout(BoxLayout.Y_AXIS));
h1.add(new Label("First")).
   add(new Label("Second")).
   add(new Label("Third")).
   add(new Label("Fourth")).
   add(new Label("Fifth"));
```

Which results in this:



A. True.
B. False.

B The following is referred to as "FlowLayout".

A. True.
B. False.

D The concept of multiple inheritances is implemented in Java by:
I. Extending two or more classes.
II. Extending one class and implementing one or more interfaces.
III. Implementing two or more interfaces.

A. Only (II)
B. (I) and (II)
C. (II) and (III)
D. Only (III)

A Which of the following describes the Structural pattern correctly?

A. This type of patterns provides a way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator.
B. This type of patterns concerns class and object composition. Concept of inheritance is used to compose interfaces and define ways to compose objects to obtain new functionalities.
C. This type of pattern is specifically concerned with communication between objects.
D. This type of pattern is specifically concerned with the presentation tier.

B Which of the following characteristics of an object-oriented programming language restricts behavior so that an object can only perform actions that are defined for its class?

A. Dynamic Binding
B. Polymorphism
C. Inheritance
D. Encapsulation

N/A _ Provide the expected output for the following code:

```
class Animal {
   public void move() {
      System.out.println("Animals can move");
   }
}

class Dog extends Animal {
   public void move() {
      System.out.println("Dogs can walk and run");
   }
   public void bark() {
      System.out.println("Dogs can bark");
   }
}

public class TestDog {

   public static void main(String args[]) {
      Animal a = new Animal();   // Animal reference and object
      Animal b = new Dog();   // Animal reference but Dog object

      a.move();   // runs the method in Animal class
      b.move();   // runs the method in Dog class
      b.bark();
   }
}
```

Output:

Animals can move
Dogs can walk and run
Dogs can Bark

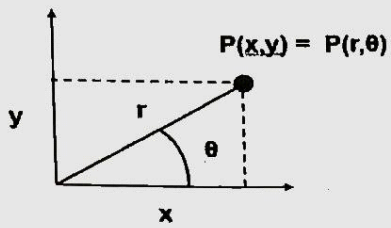
Compiler error.
No output!

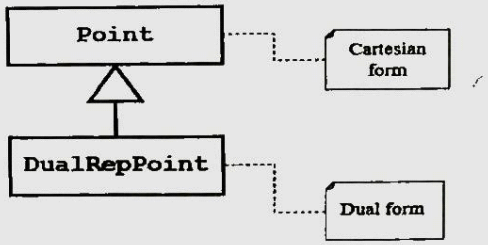
Selection: Place the following selection(s) (use Letter(s) Only, i.e A, B) into appropriate column(s). Use only features below Java 8 (Unchecked as in Codename One – Project)

Java Abstract	Java Interface
F	B
A	C
E x	D x
	No Points lost

A. can have both abstract and concrete methods
B. supports multiple inheritance
C. can only have public static final (constant) variable
D. there is a clear inheritance hierarchy to be defined
E. if various implementations only share method signatures then it is better to use
F. contains constructor

2. **Inheritance**: Consider the following DualRepPoint Class. It supports both Cartesian and Polar Coordinates. (15 points)





You are given the following Classes:

```
public class Point {
        private double x, y ;
        public Point (){
                x = 0.0 ;
                y = 0.0 ;
        }

        public double getX() { return x ; }
        public double getY() { return y ; }
        public void setX (double newX) {
                x = newX ;
        }
        public void setY (double newY) {
                y = newY ;
        }
}
```

```
/** This class maintains a point representation in both Polar and
Rectangular form and protects against inconsistent changes in the local
fields */

public class DualRepPoint extends Point {
        private double radius, angle ;

        public DualRepPoint () {
                radius = 2.0 ;
                angle = 45.0;
                updateRectangularValues();
        }
        public double getRadius() { return radius ; }
        public double getAngle() { return angle ; }
        public void setRadius(double theRadius) {
                radius = theRadius ;
                updateRectangularValues();
        }
        public void setAngle(double angleInDegrees) {
                angle = angleInDegrees;
                updateRectangularValues();
        }
        // force the Cartesian values (inherited from Point)
        // to be consistent
        private void updateRectangularValues() {
                x = radius * Math.cos(Math.toRadians(angle));
                y = radius * Math.sin(Math.toRadians(angle));
        }
}
```

Here is a client class who would like to use the DualRepPoint :

```
public class SomeClientClass {
        private DualRepPoint myDRPoint ;
        public SomeClientClass() {        // client constructor
                myDRPoint = new DualRepPoint(); // create a private DualRepPoint
                myDRPoint.setX(2.2) ;
                myDRPoint.setY(7.7) ;
        }
}
```

Please analyze the above classes. (1) If there is an issue(s) with any of the class(es) above, please state the reason(s) and describe how to fix it (them). Or (2) Otherwise, if there is no issue with these classes, please provide your justifications.

updateRectangularValues cannot directly modify x, y values. ✓
It must use mutator methods to modify x, y values. ✓ nice!
Can fix by using setX() and setY() instead of x= ... y= ...

-8 second issue not found.

3. **Design Pattern and UML class diagram**: This pattern is one of the most used design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object. Here, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.

Study the following code:

```
public interface Shape {
   void draw();
}
```

```
public class Rectangle implements Shape {
   @Override
   public void draw() {
      System.out.println("In Rectangle draw() method.");
   }
}
```

```
public class Circle implements Shape {
   @Override
   public void draw() {
      System.out.println("In Circle draw() method.");
   }
}
```

```
public class ShapeMisteryPattern {
   //use getShape method to get object of type shape
   public Shape getShape(String shapeType){
      if(shapeType == null){
         return null;
      }
      if(shapeType.equalsIgnoreCase("CIRCLE")){
         return new Circle();
      } else if(shapeType.equalsIgnoreCase("RECTANGLE")){
         return new Rectangle();
      }
      return null;
   }
}
```

```
public class MisteryPatternDemo {
   public static void main(String[] args) {
      ShapeMisteryPattern myShape = new ShapeMisteryPattern();
      //get an object of Circle and call its draw method.
      Shape shape1 = myShape.getShape("CIRCLE");
      //call draw method of Circle
      shape1.draw();
      //get an object of Rectangle and call its draw method.
      Shape shape2 = myShape.getShape("RECTANGLE");
      //call draw method of Rectangle
      shape2.draw();
   }
}
```

a) Provide the name (ONLY ONE) of a design pattern that most fit by the above description, in the space below: (5 points)

Structural ✓ ✗

b) Draw a UML class diagram depicting the associations between the elements of this program. (10 points)