CS401 MPP Midterm - *Solutions*

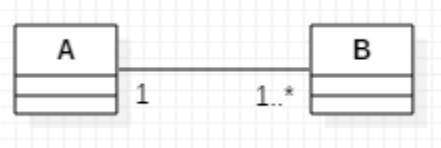
Corazza

Name:_____ StudentId:_____

Part I (16)	Part II.1 (10)	Part II.2 (8)	Part II.3 (12)	Part II.4 (8)	Part III SCI (3)

Part I: Short Answer (2 points each)

_F_ 1. (T/F) To implement the class diagram below in code, a list of type A objects must be placed inside the class B.



_F_2. (T/F) If the class diagram below has been implemented in code, the following must be true at runtime: When an instance of class A is created, it keeps a reference to class B.

_T_3. (T/F) A Sequence Diagram shows the flow of communication between the running objects of the system, driven by the use cases of the system.

_C_4. What happens when the main method in the following code is executed?

```
public class Base extends Extension {
    public static void main(String[] args) {
        Extension s = new Base();
        s.print();
    }
    public void print() {
        System.out.println("From Base");
    }
}
```

```
public class Extension {
    void print() {
        System.out.println("From Extension");
    }
}
```

A. There is a compiler error.

B. There is a runtime error.

C. "From Base" is printed to the console.

D. "From Extension" is printed to the console.

_A_ 5. In the following code, which of the following is correct regarding the relationship between Employer and Gardener? Circle one letter.

A. There is a dependency from Employer to Gardener

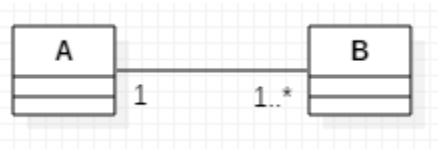
B. There is a one-way association from Employer to Gardener

C. There is a two-way association between Employer and Gardener

D. Not possible to determine from the code shown

```
public class Employer {
    public void employ() {
        Gardener gardener = new Gardener();
        gardener.garden();
    }
}

public class Gardener {
    public void garden() {
        //do gardening
    }
}
```

B, C, D 6. Consider the following class diagram:



Which of the following statements is (are) correct? Circle all that are correct.

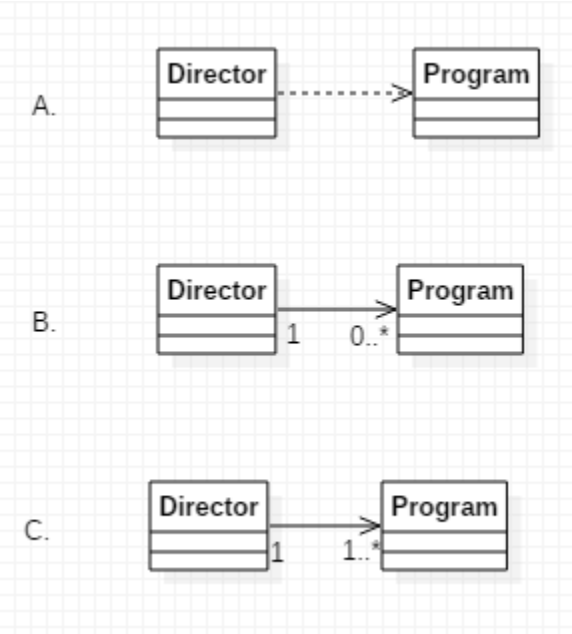
A. Each instance of the class B contains a list of instances of A.

B. Each instance of the class A contains a list of instances of B.

C. A is a property of B.

D. If an instance of A has been created, at least one instance of B has also been created.

_C_7. Which of the following UML diagrams correctly models the relationship between Director and Program? The code for Director and Program is shown below.




```
public class Program {
    private String programId;

    Program(String id) {
        programId = id;
    }
}
```

```
public class Director {
    private String name;
    private List<Program> programs = new ArrayList<>();
    public Director(String name, String progId) {
        this.name = name;
        addProgram(progId);
    }
    public void addProgram(String progId) {
        programs.add(new Program(progId));
    }
}
```

8. Consider the following Customer class. It contains instance variables of type Account, LocalDate, and List<Double>. (The Account class is also shown.)

```
public class Customer {
    private Account checkingAccount;
    private LocalDate birthdate;
    private List<Double> thisYearsSalaries;
    public Customer(Account checking, LocalDate bdate,
            List<Double> salaries) {
        this.checkingAccount = checking;
        this.birthdate = bdate;
        this.thisYearsSalaries = salaries;
    }
}

public class Account {
    private double balance;
}
```

If this Customer class is modeled in a class diagram:

a. Which instance variables should be modeled as *attributes*?
birthdate, thisYearsSalaries

b. Which instance variables should be modeled as *associations?*
checkingAccount

Explain your answer.

checkingAccount has an internal structure with its own attributes and behavior so it needs to be modeled with an association

The internal structures of date and listOfSalaries are not being modeled – we do not need to represent them with associations

Part II: Skill Questions

1. [10 pts] A rectangle can be specified by specifying two sides joined at an endpoint (in other words, length and width), but it can also be specified by specifying one side and a diagonal.

   A. (5 points) The following code attempts to implement a Rectangle class and provide support for
   the two ways of constructing a Rectangle. The code does not compile. Why is there a compiler error? (Write your answer below.)

```
public class Rectangle {
    double side1, side2, diagonal;
    public Rectangle(double s1, double s2) {
        this.side1 = s1   ;
        this.side2 = s2   ;
        diagonal = Math.sqrt(side1 * side1 + side2 * side2);
    }
    public Rectangle(double s1, double diagonal) {
        this.side1 = s1
        this.diagonal = diagonal;
        side2 = Math.sqrt(diagonal * diagonal - side1 * side1);
    }
    public double computeArea() {
        return side1 * side2;
    }
}
```

Your Explanation:

Java syntax does not allow two constructors with the same signature

Grading:
Student's answer must be essentially the same as the above answer.

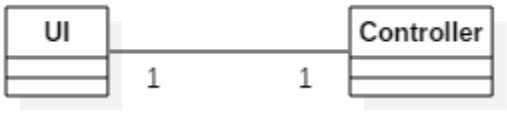
B. (5 points) In the space provided below, rewrite the code for Rectangle (from Part A) so that it supports the two ways of constructing a rectangle. Use a technique described in the course.

```
public class Rectangle {
    double length, width, diagonal;
    public static Rectangle createRectangleByLengthWidth(double len, double width) {
        Rectangle r = new Rectangle();
        r.length = len;
        r.width = width;
        r.diagonal = Math.sqrt(r.length * r.length + width * width);
        return r;
    }
    public static Rectangle createRectangleByWidthDiag(double width, double diag) {
        Rectangle r = new Rectangle();
        r.width = width;
        r.diagonal = diag;
        r.length = Math.sqrt(r.diagonal * r.diagonal - width * width);
        return r;
    }
    public double computeArea() {
        return length * width;
    }
}
```

Grading.

- 2.5 points for showing two public static factory methods which call the Rectangle constructor
- 2.5 points for setting instance variables correctly (could be done another way so be careful)

2. [8 pts] The diagram below shows that (for a particular application) there is a one-one bidirectional association between a UI class and a Controller class.



In the space provided below, write Java code that implements this diagram. Assume that UI and Controller are the only classes in a particular package. Your code must meet the following requirements:

a. The UI class owns the relationship, so it should not be possible to create an instance of Controller independently of an already existing UI class
b. The code must show relevant instance variables, constructor implementations, and methods, sufficient to implement this model. (Show only those instance variables that are implied by the diagram. Getters for instance variables should be provided.) Note: Using this diagram, the only instance variables will be those that are implied by the bidirectional 1-1 relationship.
c. All classes, properties, methods, and constructors must be given appropriate visibility qualifiers (private, protected, public, or package level).

```
public class UI {
    private Controller controller;
    public UI() {
        controller = new Controller(this);
    }
}
public class Controller {
    private UI ui;
    //package level access
    Controller(UI ui) {
        if(ui == null)
            throw new NullPointerException(
                    "UI argument must not be null!");
        this.ui = ui;

    }
}
```

not necessary

Grading:
2 pts for having instance variables of each class in the other class
1 pt for calling Controller constructor inside UI constructor
1.5 pts for passing "this" in "new Controller" call inside UI().
1.5 pts for setting ui variable from within Controller constructor
1 pt for package level access of Controller constructor

3. [12 pts] In the problem description below, a properties management system is described. As a first step in analysis for providing a solution to this problem, a very simple class diagram is given below. For this problem, develop the class diagram further using inheritance and include associations (with multiplicities) and some operations for you classes. Your new diagram should use the new class Property for the purpose of inheritance.

   The code provided gives implementations of all the classes in the diagram shown below. Your objective is to update those implementations so that they correspond to the new version of the class diagram. Note that the Admin and Driver classes that have been provided for you have implementations that produce correct outputs, but the method computeTotalRent in Admin performs its computation by checking the types of different rental properties. You need to refactor the implementation of computeTotalRent so that the inheritance you have introduced in your new diagram is used and computation is performed using polymorphism. To do this you will need to implement and make use of the (unimplemented) class Property.

   Note: This problem is asking for 4 things:

   (i) An improved class diagram that shows associations, multiplicities, inheritance (and uses the Property class)

   (ii) Refactored code in Admin and Driver; in particular, a rewritten version of computeTotalRent(). (Note: You do not need to implement the functionality of listing all properties in a certain city that is mentioned in the Problem Description below.)

   (iii) Implementation of the Property class

   (iv) Updates of the other classes provided so that they correspond to your new class diagram

   The Admin and Driver classes must be completely rewritten and an implementation of Property needs to be done (use the space provided for each of these). For the other classes, just make small modifications of the classes provided rather than rewriting the code for these.

Problem Description:

A landlord owns several types of properties: houses, condominiums, and trailers. A house has an address and a lot size. Rent for a house is computed by

rent = 0.1 * lot size

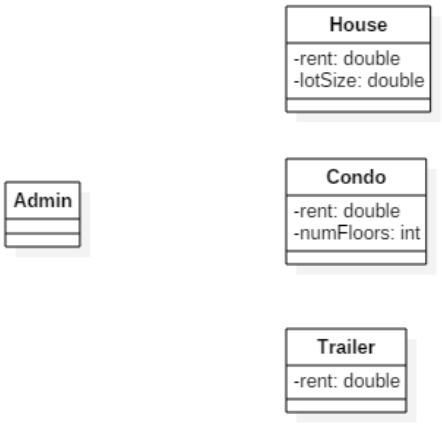
A condominium has an address and a certain number of floors (1 floor, 2 floors, or 3 floors). Rent for a condominium is computed by

rent = 400 * number of floors

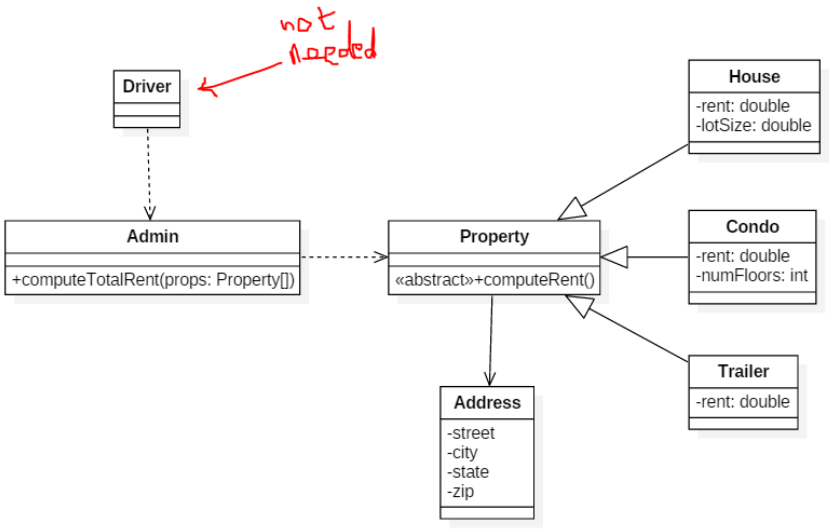
A trailer belongs to a particular trailer park (specified by the trailer park address). The rent for a trailer is always $500.

The property managemet software is required to have an Admin module that supports various functions. One of these functions is to compute total rent for all the properties registered in the system. Another function is to list all properties in the system that are in a specified city.

//Simple class diagram for this problem – you must create a new class diagram (next page)
// showing associations, multiplicities, a few operations, and a new inheritance relationship



//Your new class diagram should be drawn on this page



Grading: 6 points

Admin -> Property is a dependency

Three inheritance relationships

Property -> address association

//Your code for Problem 3 should begin here
//Implement Admin and Driver from scratch and give a new implementation of Property

```
public class Admin {
    public static double computeTotalRent(Property[] properties) {
        double totalRent = 0;
        for(Property p: properties) {
            totalRent += p.computeRent();
        }
        return totalRent;
    }
}
```

```
public class Driver {

    public static void main(String[] args) {

        Property[] objects = { new House(9000), new Condo(2),
                new Trailer(new Address("111 Main", "Fairfield", "IA", "52556")) };
        double totalRent = Admin.computeTotalRent(objects);
        System.out.println(totalRent);
    }
}
```

```
abstract public class Property {
    abstract double computeRent();
}
```

//Non-OO versions of Driver and Admin are shown here – your code
//above should refactor these

```
public class Admin {
    public static double computeTotalRent(Object[] properties) {
        double totalRent = 0;
        for (Object o : properties) {
            if (o instanceof House) {
                House h = (House) o;
                totalRent += h.computeRent();
            }
            else if (o instanceof Condo) {
                Condo h = (Condo) o;
                totalRent += h.computeRent();
            }
            else if (o instanceof Trailer) {
                Trailer h = (Trailer) o;
                totalRent += h.computeRent();
            }
        }
        return totalRent;
    }
}
```

```
public class Driver {

    public static void main(String[] args) {

        Object[] objects = { new House(9000),
                new Condo(2), new Trailer() };
        double totalRent = Admin.computeTotalRent(objects);
        System.out.println(totalRent);
    }
}
```

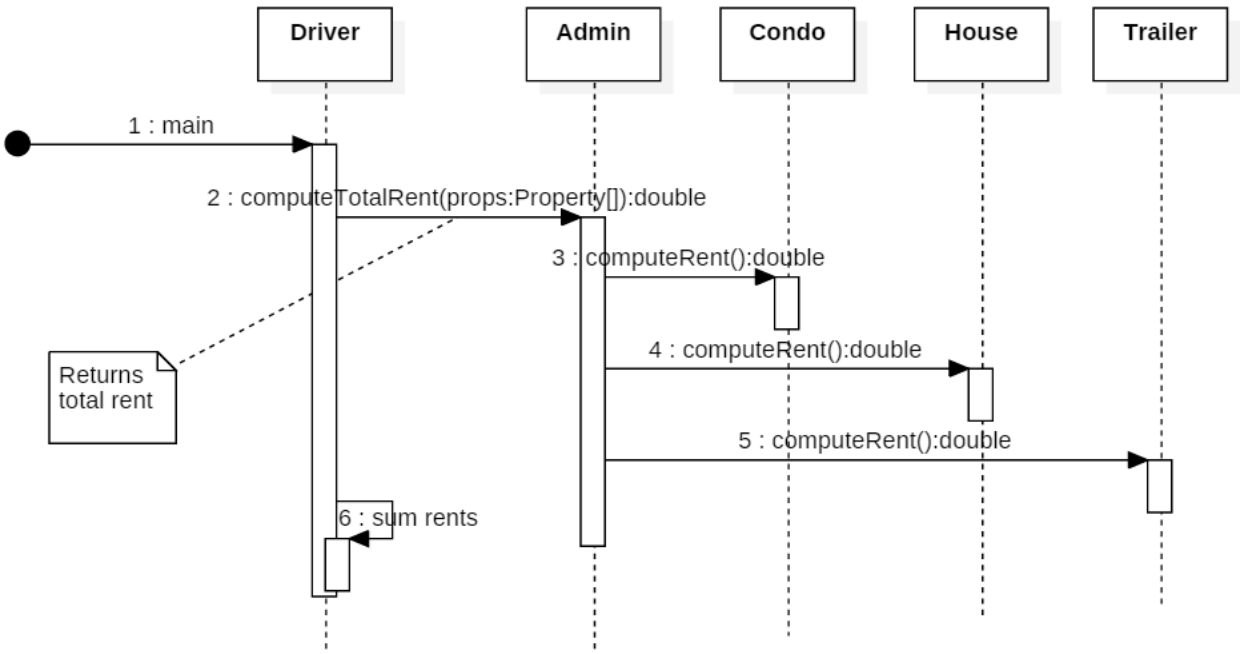
//Implementations of other classes are shown below. These
//must be updated as necessary so that they match your new class diagram

```java
public class Condo extends Property {

    private int numberOfFloors;

    public Condo(int numberOfFloors) {
        this.numberOfFloors = numberOfFloors;
    }

    public double computeRent(){
        return 500 * numberOfFloors;
    }
}
public class House extends Property {

    private double lotSize;


    public House(double lotSize) {
        this.lotSize = lotSize;
    }


    public double computeRent(){
        return 0.1 * lotSize;
    }
}


public class Trailer extends Property {
    private static final double RENT = 500;
    private Address address;

    public Trailer(Address address) {
        this.address = address;
    }
    public double computeRent(){
        return RENT;
    }
    public Address getAddress() {
        return address;
    }
}
```

Grading: 6 points

- 1.5 coding Admin
- 1.5 coding Driver
- 1.5 coding Property
- 1.5 filling in "extends" in the three other classes

4. [8 points] Create a sequence diagram to model the dynamics of the problem given in Problem 3. Provide a diagram only for the main flow. In this case, the main flow will be the flow in which three properties – one house, one condominium, and one trailer – are passed to the computeTotalRent method in Admin (as shown in the Driver class implementation). Remember: sequence diagrams are concerned with run-time objects only.



Grading:
1. All five objects should be shown
2. Array param for computeTotalRent should be shown
3. computeTotalRent should call computeRent for each Property
4. Sum of rents should be computed
5. Numbering and activation bars should be shown

Part III: SCI. [3 pts] In an ancient text, one reads the following:

Know that by which all this is known.

1. What does this expression mean? What is it saying? Is it some kind of SCI point?
2. Does this expression illuminate any aspect of the software engineering discipline discussed in class? Explain.