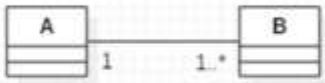
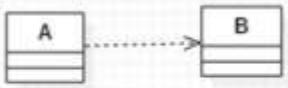
MPP MID EXAM – SOLUTIONS - COROZA 2021

_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.

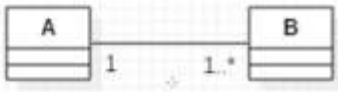
C. "From Base" is printed to the console.

What type of relationship do those classes have? **Dependency**

```java
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.

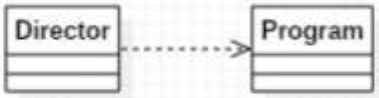
B. Each instance of the class A contains a list of instances of B.

C. A is a property of B.

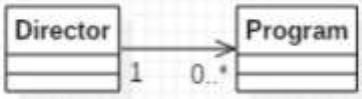
D. If an instance of A has been created, at least one instance of B has also been created.

_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.

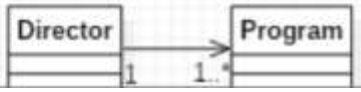
A.



B.



C.



Correction: al the relationships are Bidirectional

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

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 = Matn.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;
    }
```

```
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;
    }
}
```

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;

    }
}
```

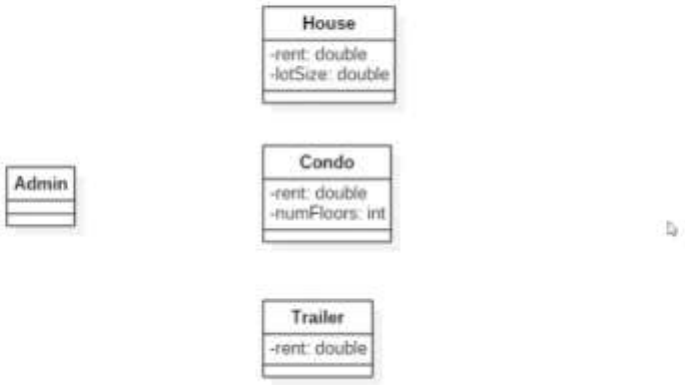
not necessary

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.

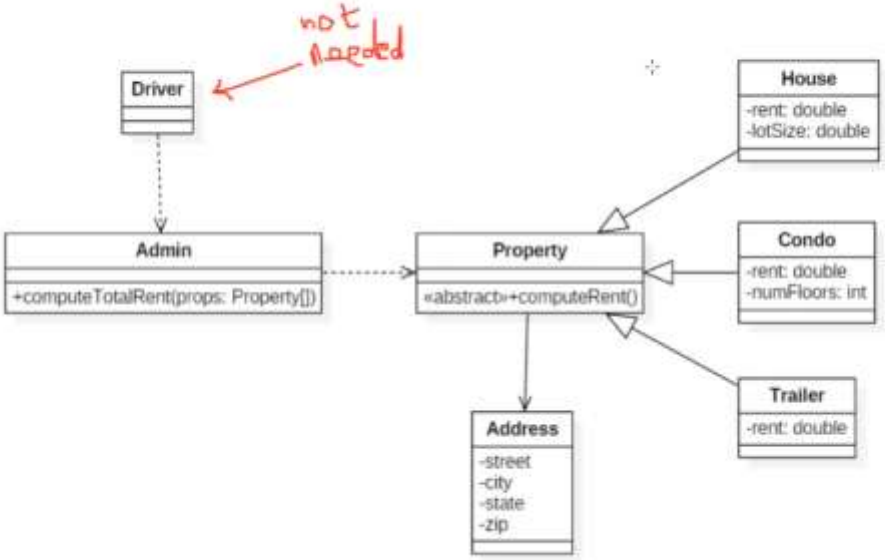
   The code provided gives implementations of all the classes in the diagram shown below. Your objective is to update those implementations so that they correspond to the new version of the class diagram. Note that the Admin and Driver classes that have been provided for you have implementations that produce 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.

> 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



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

```java
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;
    }
}
```

```java
public class Driver {

    public static void main(String[] args) {

        Object[] objects = { new House(9000),
                new Condo(3), 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

//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();
}
```

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.