CalPolyPomona

CS 3560 – Assignment #2

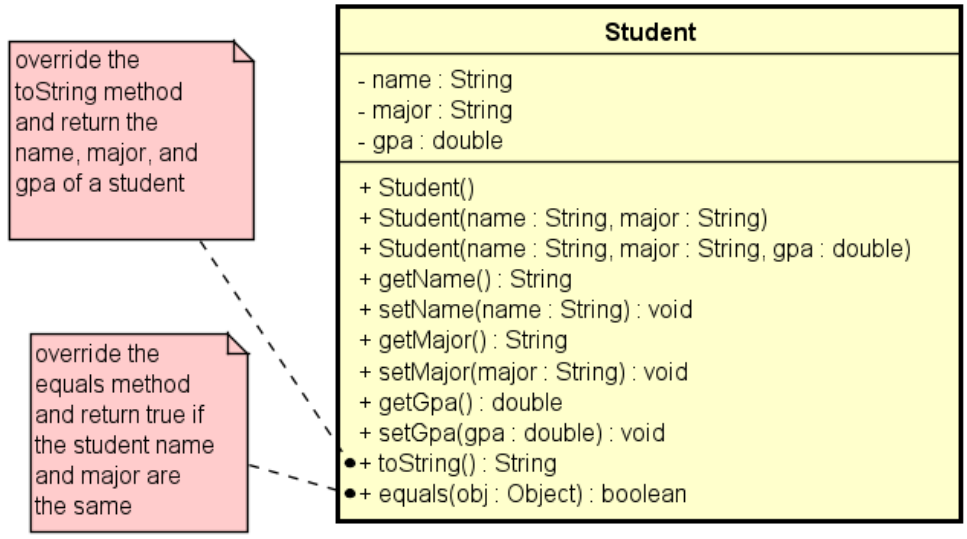
Maximum Points: 100 pts.

Bronco ID: |_0_|_1_|_4_|_2_|_3_|_6_|_2_|_2_|_2_|

Last Name: _____Hussain_____ First Name: ____Farhan_____

Note 1: Your submission header must have the format as shown in the above-enclosed rounded rectangle.
Note 2: Homework is to be done individually. You may discuss the homework problems with your fellow students, but you are NOT allowed to copy – either in part or in whole – anyone else's answers.
Note 3: All submitted materials must be legible. Figures/diagrams must follow the given instructions.
Note 4: Your deliverable should be a .pdf file submitted through Gradescope by the deadline. Do not forget to assign a page to each of your answers when making a submission. In addition, source code (.java files) should be added to an online repository (e.g., GitHub) to be downloaded and executed later.
Note 5: Please use and check the Canvas discussion for further instructions, questions, answers, and hints. The bold words/sentences provide information for a complete or accurate answer.

1. [15 points] Given the corresponding UML class below, answer the following questions.



a) [10 points] Write the corresponding Java code for the `Student` class. Make sure to include the **expected code** inside the **methods** and **constructor(s)** when appropriate.

```
class Student
{
    private String name;
    private String major;
    private double gpa;

    Student()
    {

    }

    Student(String name, String major)
    {
        this.name = name;
        this.major = name;
    }




    Student(String name, String major, double gpa)
    {
        this.name = name;
        this.major = major;
        this.gpa = gpa;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public String getMajor()
    {
        return major;
    }

    public void setMajor(String major)
    {
        this.major = major;
    }
```

```
    public double getGpa()
    {
        return gpa;
    }

    public void setGpa(double gpa)
    {
        this.gpa = gpa;
    }

    @Override
    public String toString()
    {
        return String.format("Student info: %s, %s, %f", name, major,
gpa);
    }









    @Override
    public boolean equals(Object obj)
    {
        if (this == obj)
            return true;

        if (obj instanceof Student)
        {
            Student other = (Student) obj;
            return this.name.equals(other.name) &&
this.major.equals(other.major);
        }

        return false;
    }
}
```

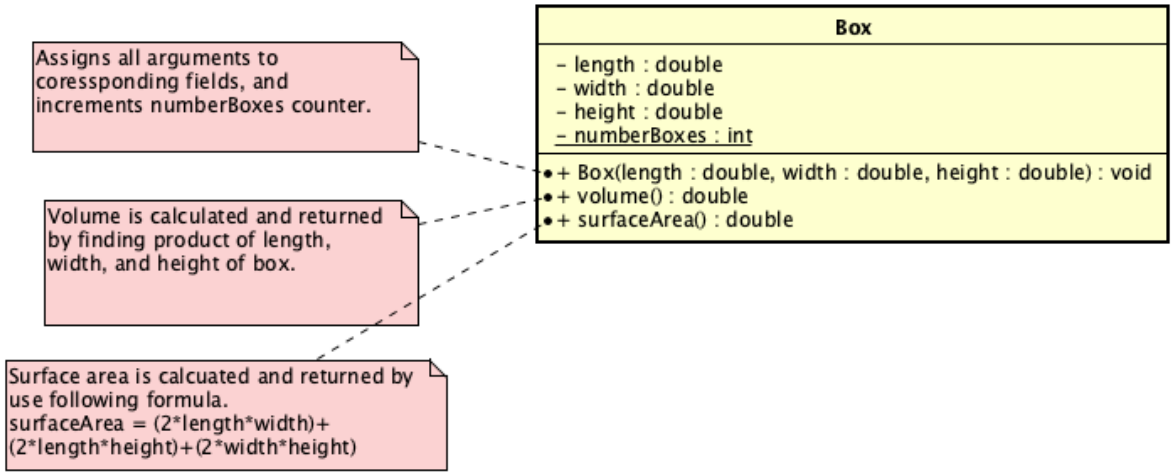
b) [5 points] Write a driver class called `StudentTest` that instantiates 2 students inside its main() method. The first student {"John", "CS", 3.5} is created by using a **parameterized** constructor while the second student {"Mary Ann", "CE", 3.3} is created by using the **no-arg constructor**. After creating both students and updating their name, major, and gpa, **print** their states by using the `toString()` method.

```
public class StudentTest
{
    public static void main(String[] args)
    {
        Student s1 = new Student("John", "CS", 3.5);

        Student s2 = new Student();
        s2.setName("Mary Ann");
        s2.setMajor("CE");
        s2.setGpa(3.3);

        System.out.println(s1.toString());
        System.out.println(s2.toString());

    }
}
```

2. [15 points] Given the Java code below, answer the following questions.

```
public class Box {
private double length, width, height;
private static int numberBoxes;
public Box(double length, double width, double height) {

            this.length = length;
            this.width = width;
            this.height = height;
            numberBoxes ++;
         }
         public double volume() {
            return (length * width * height);
         }

public double surfaceArea() {
return ((2 * length * width) + (2 * length * height) + (2 *

                                                        width * height));

} }
```

a) [10 points] Draw the corresponding UML class diagram **(astah must be used)**. Add some notes to explain what the `Box()`, `volume()`, and `surfaceArea()` methods should do/return.



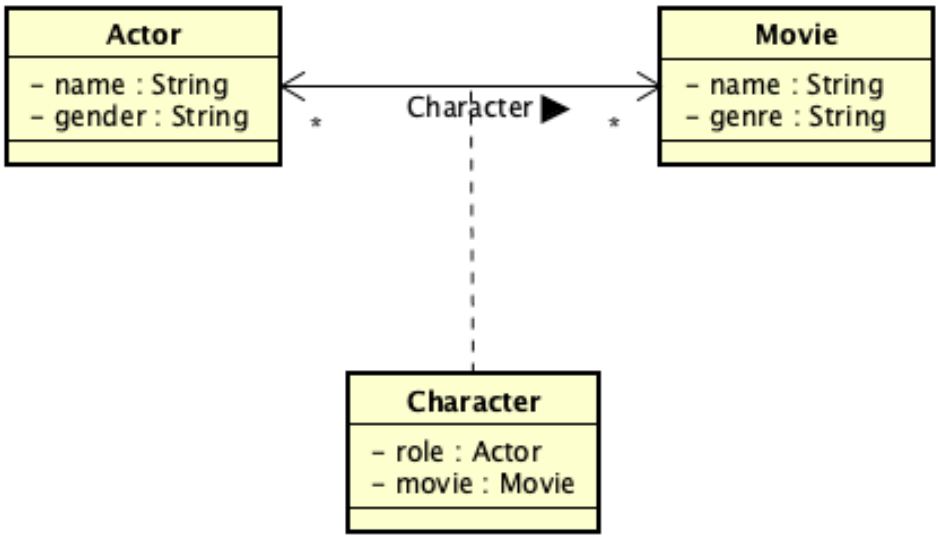
b) [5 points] Explain the **practical implications** of making `length`, `width`, and `height` as instance fields while `numberBoxes` is defined as a static field.

The practical implications of making length, width, and height as instance fields is that accessing an instances dimensions is easy since each instances stores its unique dimensions within itself.

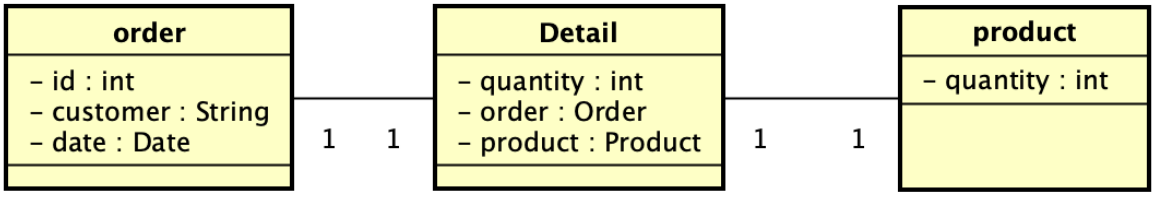
The practical implications of making numberBoxes defined as a static field is that any instance or class itself can easily find the number of boxes that have been created. This piece of data isn't unique to one particular object so it makes sense to have it be shared with all instances of the class by having the data belong to the class itself. By doing this, data is being stored in one place and shared among all instances rather than being stored in each individual instance. That would be redundant and amking changes to the data would be harder, thus its pragmatic to have numberBoxes be shared.

3. [10 points] Draw the UML class diagrams for the proposed scenarios below **(astah must be used)**. Include the attributes shown in the {} inside your classes by choosing appropriate data types. No methods need to be included.

   a) [5 points] A given actor {name, gender} can act in multiple movies {name, genre}, and for each movie, we need to register the **multiple characters** {role} played by the actor.

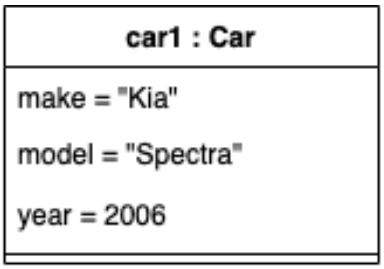


b) [5 points] An order {id, customer, date} can have multiple products {name, price} and for each product, we need to register its **details** {quantity} in the order.

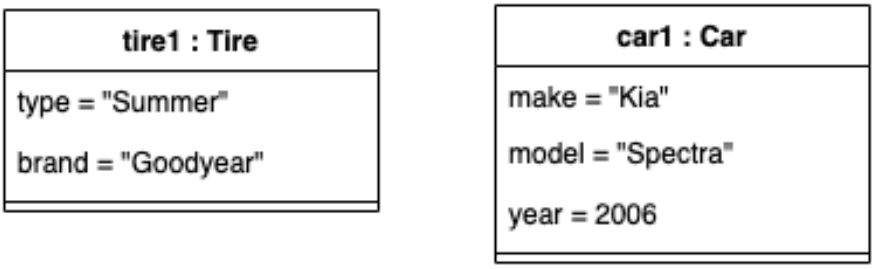


4. [10 points] Based on the UML class diagram below, draw the appropriate object diagrams that will correspond to the proposed system states. **Include all attributes** inside your objects.
    a) [5 points] **One** tire has been **created** {Summer, Goodyear} and **added** to the car {Kia, Spectra, 2006}.

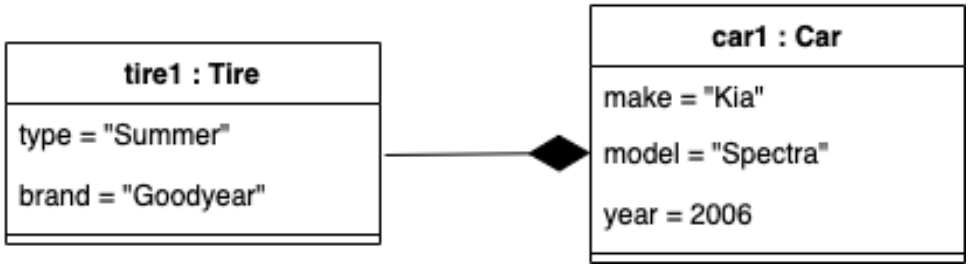
(1)



(2)

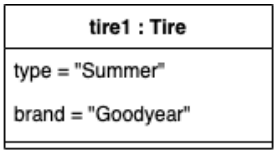


(3)

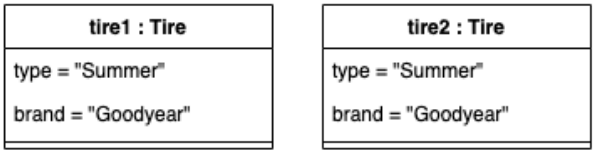


b) [5 points] **Four** tires have been **created** {Summer, Goodyear}, {Summer, Goodyear}, {Summer, Firestone}, and {Summer, Firestone} but **only** the **Goodyear's** ones have been added to the car {Kia, Spectra, 2006}.

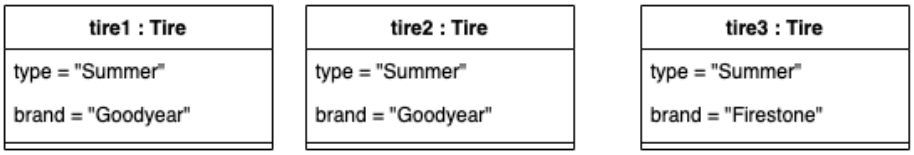
(1)



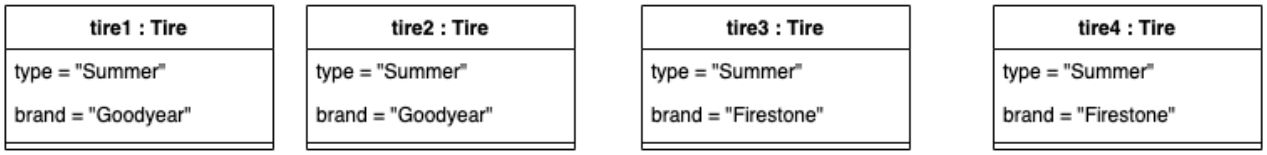
(2)



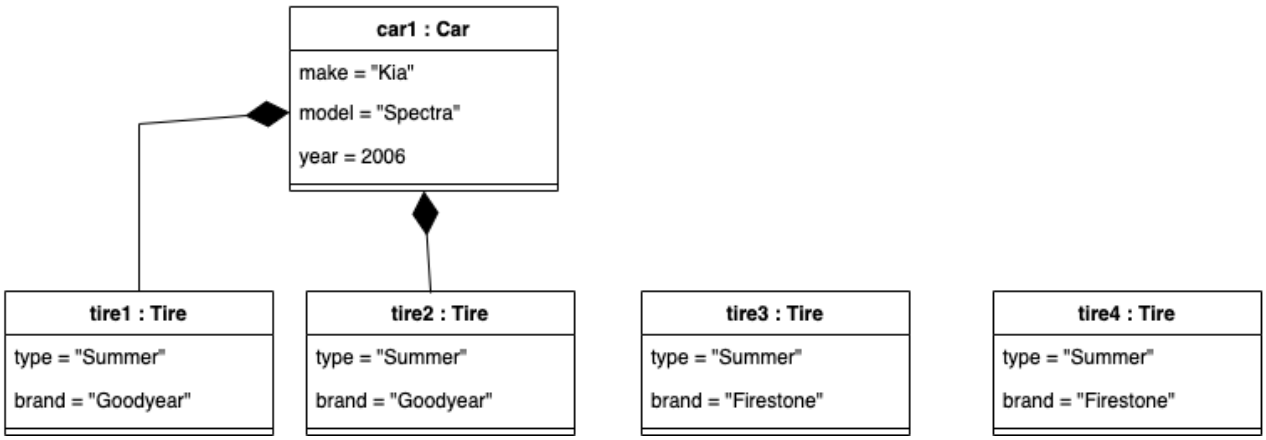
(3)



(4)



(5)



5. [50 points] Based on the relationships shown in the UML class diagrams below, write the corresponding Java code. Include all **fields**, **constructors**, and **other extra methods** as guided by the instructions (getters and setters not needed). Also, include the expected behavior of each of those methods.
    a) [5 points]. Student and Transcript: fully parameterized constructors.

```
class Student2
{
    private String name;
    private Transcript transcript;

    Student2(String name, Transcript transcript)
    {
        this.name = name;
        this.transcript = transcript;
    }

    @Override
    public String toString()
    {
        return String.format("Student2[%s, %s]", name,
transcript.toString());
    }
}
class Transcript
{
    private String course;
    private double grade;

    Transcript(String course, double grade)
    {
        this.course = course;
        this.grade = grade;
    }
```

```
    @Override
    public String toString()
    {
        return String.format("transcript[%s, %.2f]", course,
grade);
    }
}
```

Driver Method

```
private static void five_a()
{
    System.out.println("5 a) Unidirectional association between Student and Transcript.");
    Transcript t = new Transcript("CS 3560", 89.99);
    Student2 s = new Student2("Bob", t);

    System.out.println("Transcript obj info:   "+t.toString());
    System.out.println("Student obj info:   "+s.toString());
    System.out.println("\n");
}
```

b) [5 points]. Course: fully parameterized constructor. Book: no-arg constructor.

```
import java.util.ArrayList;

class Course
{
    private int code;
    private ArrayList<Book> books;

    Course(int code, Book book)
    {
        this.books = new ArrayList<Book>();
        this.code = code;
        this.books.add(book);
    }

    public void addBook(Book book)
    {
        books.add(book);
    }

    public String toString()
    {
        return String.format("Course[%s, numOfBooks=%d]", code,
books.size());
    }
}
```

```
class Book
{
    private String name;
    private String author;
    private Course course;

    public void setName(String name)
```

```
    {
        this.name = name;
    }
    public void setAuthor(String author)
    {
        this.author = author;
    }
    public void setCourse(Course course)
    {
        this.course = course;
    }

    public String toString()
    {
        return String.format("Book[%s, %s, %s]", name, author,
course);
    }
}
```

Driver Method

```
private static void five_b()
{
    System.out.println("5 b) Bidirectional association between Course & book.");

    Book b = new Book();
    b.setName("Calculus");
    b.setAuthor("James Stuawart");

    Course c = new Course(123, b);
    b.setCourse(c);

    System.out.println("book obj info: "+b.toString());
    System.out.println("course obj info: "+c.toString());

    System.out.println("\n");
}
```

c) [5 points]. Team: no-arg constructor. Player: fully parameterized constructor.

```
import java.util.ArrayList;

class Team
{
    private int code;
    private ArrayList<Player> players;

    Team()
    {
        this.players = new ArrayList<Player>();
    }

    public void addPlayer(Player player)
    {

        this.players.add(player);
    }

    public void setCode(int code)
    {
        this.code = code;
    }

    @Override
    public String toString()
    {
        return String.format("team[%d, %s]", code,
players.toString());
    }
}
```

```
class Player
{
    private String name;
    private boolean expert;

    Player(String name, boolean expert)
    {
        this.name = name;
        this.expert = expert;
    }

    @Override
    public String toString()
    {
        return String.format("team[%s, %b]", name, expert);
    }
}
```

Driver Method

```
private static void five_c()
{
    System.out.println("5 c) Team aggregated of Players");
    Player p1 = new Player("p1", true);
    Player p2 = new Player("p2", false);
    Player p3 = new Player("p3", false);
    Player p4 = new Player("p4", false);
    Player p5 = new Player("p5", false);
    Team team = new Team();
    team.setCode(123);
    team.addPlayer(p1);
    team.addPlayer(p2);
    team.addPlayer(p3);
    team.addPlayer(p4);
    team.addPlayer(p5);

    System.out.println("team:  "+team);

    System.out.println("\n");
}
```

d) [5 points]. Dog: no-arg constructor. Paw: fully parameterized constructor.

```
class Dog
{
    private String breed;
    private String name;
    private Paw frontLeftPaw;
    private Paw frontRightPaw;
    private Paw backLeftPaw;
    private Paw backRightPaw;

    Dog()
    {
        frontLeftPaw = new Paw(0);
        frontRightPaw = new Paw(1);
        backLeftPaw = new Paw(2);
        backRightPaw = new Paw(3);
    }

    public void setBreed(String breed)
    {
        this.breed = breed;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public void setFrontLeftPaw(Paw paw)
    {
        frontLeftPaw = paw;
    }
    public void setFrontRightPaw(Paw paw)
    {
```

```
        frontRightPaw = paw;
    }
    public void setBackLeftPaw(Paw paw)
    {
        frontLeftPaw = paw;
    }
    public void setBackRightPaw(Paw paw)
    {
        backRightPaw = paw;
    }

    @Override
    public String toString()
    {
        return String.format("Dog[%s, %s, %s, %s, %s, %s]",
breed, name, frontLeftPaw, frontRightPaw, backLeftPaw,
backRightPaw);
    }
}

class Paw
{
    private int position;

    Paw(int position)
    {
        this.position = position;
    }

    @Override
    public String toString()
    {
        return String.format("Paw[%d]", position);
    }
```

Driver Method

```
private static void five_d()
{
    System.out.println("5 d) Dog composed of Paw objects.");
    Dog dog = new Dog();

    dog.setBreed("Huskey");
    dog.setName("Bob");

    System.out.println("dog:  "+dog);

    System.out.println("\n");
}
```

e) [8 points]. Employee, Professor, and Staff: fully parameterized constructors.

```
class Employee
{
    private String name;
    private int hours;

    Employee(String name, int hours)
    {
        this.name = name;
        this.hours = hours;
    }

    public double calculateSalary()
    {
        return hours * 20;
    }

    public int getHours()
    {
        return hours;
    }

    @Override
    public String toString()
    {
        return String.format("Employee[%s, %d]", name, hours);
    }
}

class Professor extends Employee
{
```

```
    private String field;

    Professor(String name, int hours, String field)
    {
        super(name, hours);
        this.field = field;
    }

    public double calculateSalary()
    {
        return getHours() * 30;
    }

    @Override
    public String toString()
    {
        return String.format("Employee[%s]", field);
    }
}

class Staff extends Employee
{
    private int role;

    Staff(String name, int hours, int role)
    {
        super(name, hours);
        this.role = role;
    }

    @Override
    public String toString()
    {
        return String.format("Employee[%d]", role);
    }
}
```

Driver Method

```
private static void five_e()
{
    System.out.println("5 e) Inheritance relationship where Professor & Staff extend from Employee.");
    Employee e = new Employee("Bob", 1);
    Professor p = new Professor("Bill", 1, "CS");
    Staff s = new Staff("Mark", 1, 1);

    System.out.println("employee:  "+e);
    System.out.println("professor:  "+p);
    System.out.println("staff:  "+s);


    System.out.println("\n");
}
```

f) [8 points]. Magazine and Ticket: no-arg constructors.

```
/// Interface is essentially like an abstract class
/// but where allmethods are implicitly public and abstract
/// and where all fiellds are implicityl public, static, and
final
interface Salableitem
{
    // 'public abstract' is implicitly added
    void sellCopy();
}
```

```
class Magizine
{
    public void sellingCopy()
    {
        System.out.println("Selling a Magazine.\n");
    }
}

class Ticket
{
    public void sellingCopy()
    {
        System.out.printf("Selling a Ticket.\n");
    }
}
```

Driver Method

```
private static void five_f()
{
    System.out.println("5 f) Magazine & Ticket extending from Salableitem interface.\n");
    Magizine m = new Magizine();
    Ticket t = new Ticket();

    m.sellingCopy();
    t.sellingCopy();

    System.out.println("\n");
}
```

g) [8 points]. Person, Movie, and Watch: fully parameterized constructors.

```
class Person
{
    private String name;

    Person(String name)
    {
        this.name = name;
    }

    @Override
    public String toString()
    {
        return String.format("%s", name);
    }
}

class Movie
{
    private String name;
    private String genre;

    Movie(String name, String genre)
    {
        this.name = name;
        this.genre = genre;
    }

    @Override
    public String toString()
```

```
    {
        return String.format("%s", name);
    }
}

class Watch
{
    private int rating;
    private Person person;
    private Movie movie;

    Watch(int rating, Movie movie, Person person)
    {
        this.rating = rating;
        this.movie = movie;
        this.person = person;
    }

    @Override
    public String toString()
    {
        return String.format("(%s, %s, %d)", person, movie,
rating);
    }
}
```

Driver Method

```
private static void five_g()
{
    System.out.println("5 g) Person & Movie classes with Watch as the Association Class");
    ArrayList<Watch> watchList = new ArrayList<Watch>();

    // Create two people
    Person p1 = new Person("John Doe");
    Person p2 = new Person("Jane Doe");

    // Create two movies
    Movie m1 = new Movie("Home Alone", "comdey");
    Movie m2 = new Movie("Avengers", "Action");

    watchList.add( new Watch(10, m1, p1) );
    watchList.add( new Watch(9, m2, p2) );
    watchList.add( new Watch(8, m1, p2) );
    watchList.add( new Watch(7, m2, p1) );

    for (Watch w : watchList)
    {
        System.out.println(w);
    }

    System.out.println("\n");
}
```

h) [6 points]. Payroll: no-arg constructor. Worker: fully parameterized constructor.

```
class Payroll
{
    public void processPayments(Worker worker)
    {
        System.out.printf("Payment processed for worker %s",
worker);
    }
}

class Worker
{
    private String name;
    private double hourlyRate;

    Worker(String name, double hourlyRate)
    {
        this.name = name;
        this.hourlyRate = hourlyRate;
    }

    public String getName()
    {
        return name;
    }

    public String toString()
    {
        return String.format("%s", name);
    }
```

```
}
```

Driver Method

```
private static void five_h()
{
    System.out.println("5 h) Payroll class has dependency with Worker Class");

    Payroll payroll = new Payroll();
    Worker worker = new Worker("Bob", 25);

    // Payroll obj depends on Worker class
    // because payroll obj uses worker obj
    payroll.processPayments(worker);

    System.out.println("\n");
}
```

Important Note: Answers to all questions should be written clearly, concisely, and unmistakably delineated. You may resubmit multiple times until the deadline (the last submission will be considered).

NO LATE ASSIGNMENTS WILL BE ACCEPTED. ALWAYS SUBMIT WHATEVER YOU HAVE COMPLETED FOR PARTIAL CREDIT BEFORE THE DEADLINE!