OBJECT-ORIENTED PROGRAMMING
LAB 6: INHERITANCE

I. Objective

After completing this tutorial, you can:

- Understand *inheritance* in OOP.

II. Definition

Inheritance is the process in which one class acquires the properties (methods and fields) of another. With the use of inheritance, the information is made manageable in a hierarchical order.

The class which inherits the properties of another class is known as a *subclass* (derived class, child class) and the class whose properties are inherited is known as a *superclass* (base class, parent class).

1. extends keyword

The *extends* keyword is used to inherit the variables and methods of a class (except the constructors, private variables, and private methods).

```java
public class Super {
    //...
}

public class Sub extends Super {
    //...
}
```

2. super keyword

The *super* keyword in Java is a reference variable that refers to its parent class object. The usage of the *super* keyword:

- To refer to parent class instance variable.
- To invoke the parent class method.
- The `super()` can be used to invoke the parent class constructor.

If a class is inheriting the properties of another class. And if the members of the subclass have the names same as the superclass, to differentiate these variables we use the **super** keyword as follows:

- For variable: `super.variableName`
- For method: `super.methodName()`

3. Override

The subclass containing the method has the same signature as the method declared in the superclass that is method overriding. If a class wants to override the method in the other, it must be in the "is-a" relationship (Inheritance).

The annotation **@Override** indicates that a method declaration is intended to override a method declaration in a supertype. This annotation is optional, when using this annotation, it makes your code cleaner. If a method is annotated with this annotation but is not an overriding method, there is an error will be generated.

Employee.java

```java
public class Employee {
    protected String name = "";
    public Employee(String name) {
        this.name = name;
    }
    public String title() {
        return "Employee";
    }
    @Override
    public String toString() {
        return this.name + ": " + title();
    }
}
```

Developer.java

```java
public class Developer extends Employee {
    public Developer(String name) {
        super(name);
    }
    @Override
    public String title() {
        return "Developer";
    }
}
```

TestOverride.java

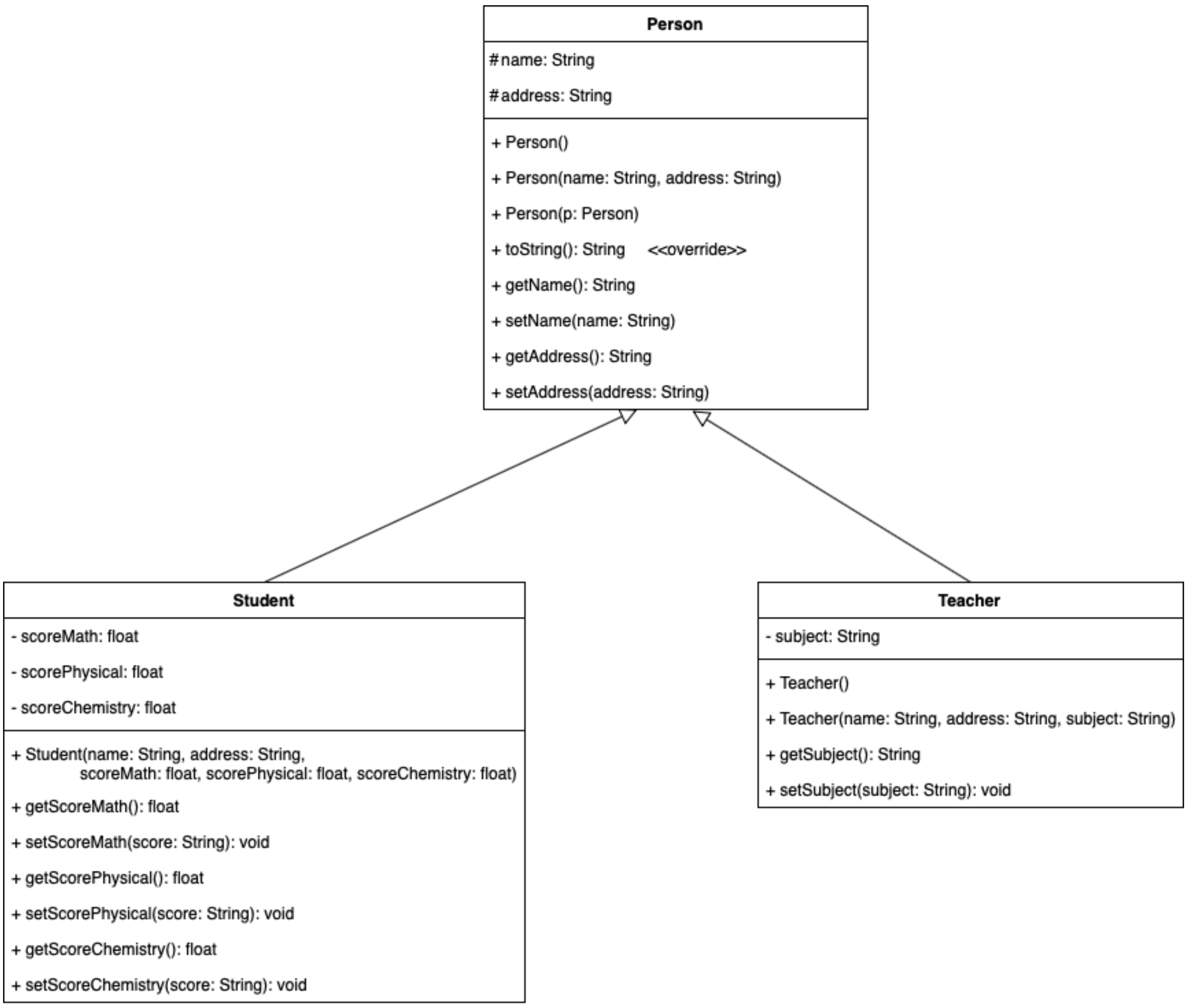
```java
public class TestOverride {
    public static void main(String[] args) {
        Employee emp = new Employee("Bob");
        Developer dev = new Developer("Alice");
        Employee emp1 = new Developer("Trudy"); // We will discuss this later.
        System.out.println(emp);
        System.out.println(dev);
        System.out.println(emp1);
    }
}
```

Result:

```
Bob: Employee
Alice: Developer
Trudy: Developer
```

III. Sample program

To demonstrate how to implement inheritance in a Java program, we take an example as shown below diagram:


Person
#name: String
#address: String
+ Person()
+ Person(name: String, address: String)
+ Person(p: Person)
+ toString(): String <<override>>
+ getName(): String
+ setName(name: String)
+ getAddress(): String
+ setAddress(address: String)
Student
- scoreMath: float
- scorePhysical: float
- scoreChemistry: float
+ Student(name: String, address: String, scoreMath: float, scorePhysical: float, scoreChemistry: float)
+ getScoreMath(): float
+ setScoreMath(score: String): void
+ getScorePhysical(): float
+ setScorePhysical(score: String): void
+ getScoreChemistry(): float
+ setScoreChemistry(score: String): void
Teacher
- subject: String
+ Teacher()
+ Teacher(name: String, address: String, subject: String)
+ getSubject(): String
+ setSubject(subject: String): void

Person.java

```
public class Person {
    protected String name;
    protected String address;

    public Person() {
        this.name = "";
        this.address = "";
    }

    public Person(String name, String address) {
        this.name = name;
        this.address = address;
    }

    public Person(Person person) {
        this.name = person.name;
        this.address = person.address;
    }

    @Override
    public String toString() {
        return "Person{" + "name='" + name + "\'" + ", address='" + address + "\'" + "}";
    }

    public String getName() {
        return this.name;
    }

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

    public String getAddress() {
        return this.address;
    }

    public void setAddress(String address) {
        this.address = address;
    }
}
```

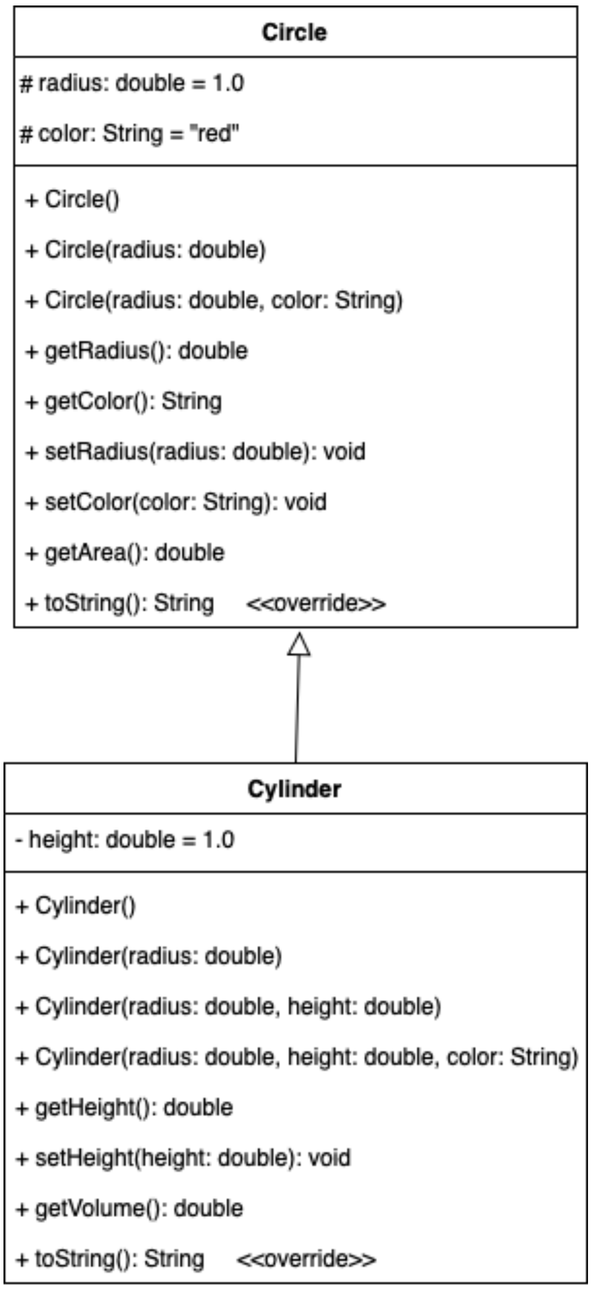
Teacher.java

```
public class Teacher extends Person {
    private String subject;

    public Teacher() {
        super();
        this.subject = "";
    }

    public Teacher(String name, String address, String subject) {
        super(name, address);
        this.subject = subject;
    }

    public String getSubject() {
        return this.subject;
    }

    public void setSubject(String subject) {
        this.subject = subject;
    }
}
```

Now, you can proceed to the next pages and complete the exercises to gain a better understanding of inheritance in Java.

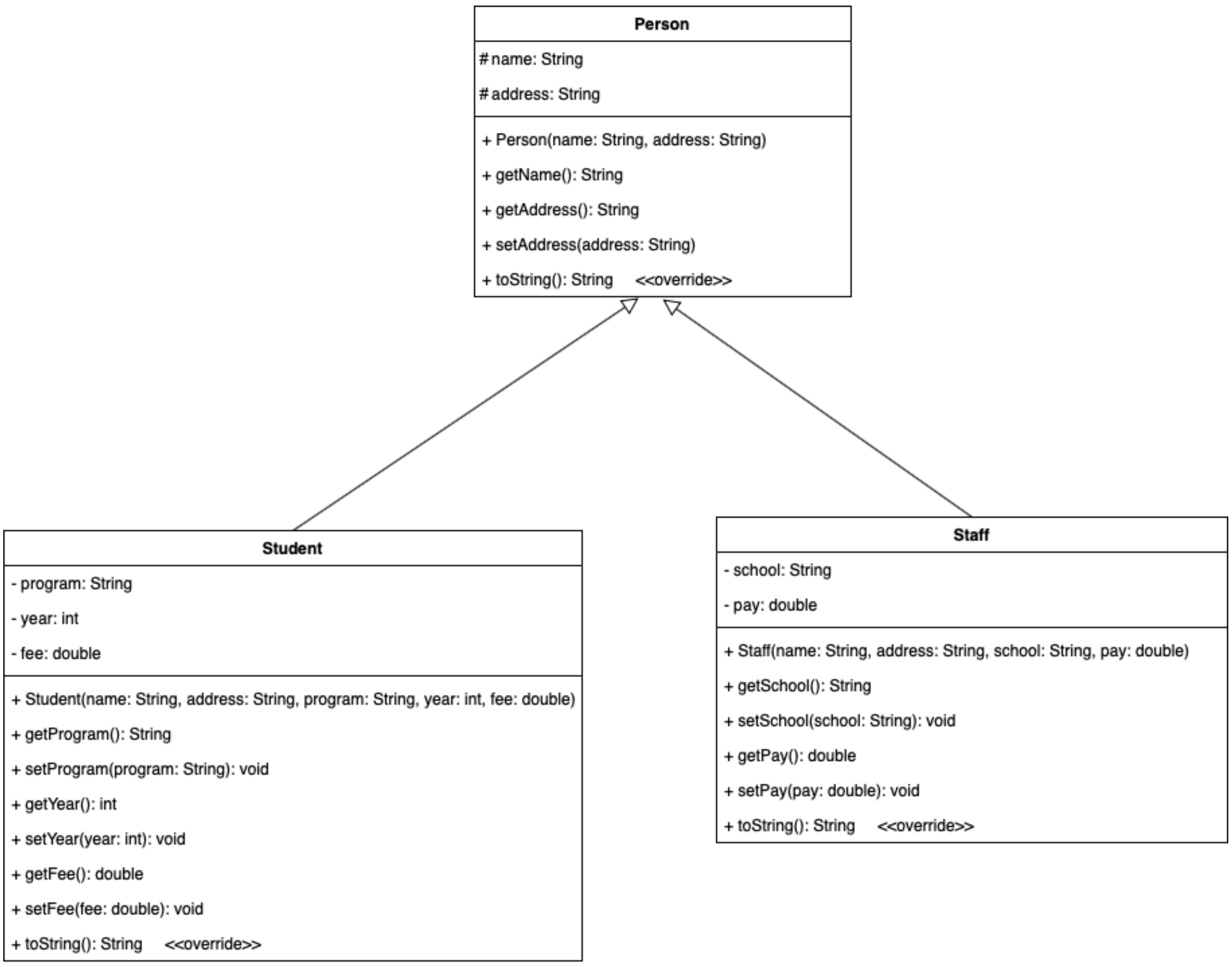
IV. Exercise

1. Giving the Circle class and the Cylinder class.


Circle
# radius: double = 1.0
# color: String = "red"
+ Circle()
+ Circle(radius: double)
+ Circle(radius: double, color: String)
+ getRadius(): double
+ getColor(): String
+ setRadius(radius: double): void
+ setColor(color: String): void
+ getArea(): double
+ toString(): String <<override>>
Cylinder
- height: double = 1.0
+ Cylinder()
+ Cylinder(radius: double)
+ Cylinder(radius: double, height: double)
+ Cylinder(radius: double, height: double, color: String)
+ getHeight(): double
+ setHeight(height: double): void
+ getVolume(): double
+ toString(): String <<override>>

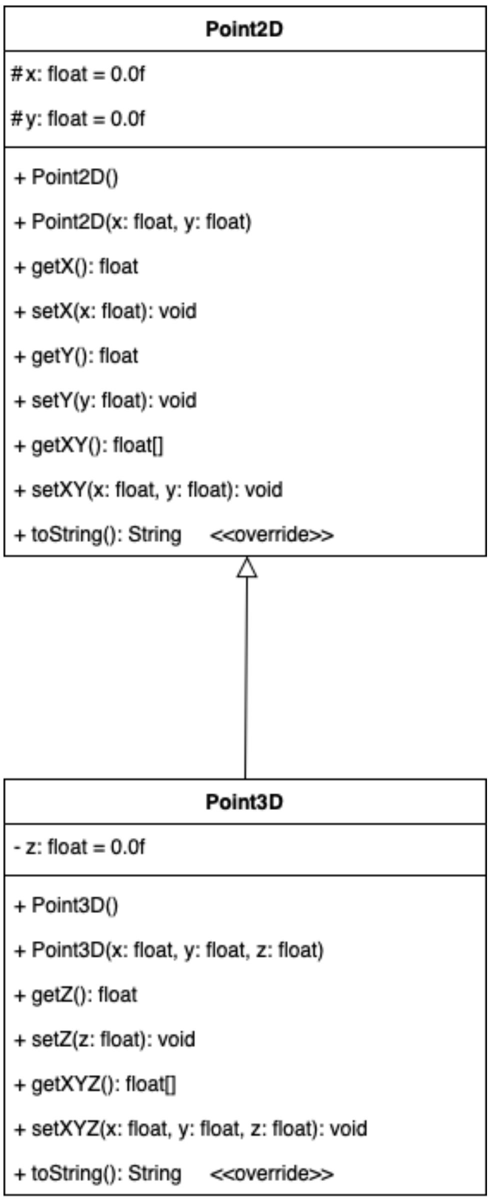
Implement the Java program based on the above diagram.

2. Giving superclass **Person** and subclasses.


Person
# name: String
# address: String
+ Person(name: String, address: String)
+ getName(): String
+ getAddress(): String
+ setAddress(address: String)
+ toString(): String <<override>>
Student
- program: String
- year: int
- fee: double
+ Student(name: String, address: String, program: String, year: int, fee: double)
+ getProgram(): String
+ setProgram(program: String): void
+ getYear(): int
+ setYear(year: int): void
+ getFee(): double
+ setFee(fee: double): void
+ toString(): String <<override>>
Staff
- school: String
- pay: double
+ Staff(name: String, address: String, school: String, pay: double)
+ getSchool(): String
+ setSchool(school: String): void
+ getPay(): double
+ setPay(pay: double): void
+ toString(): String <<override>>

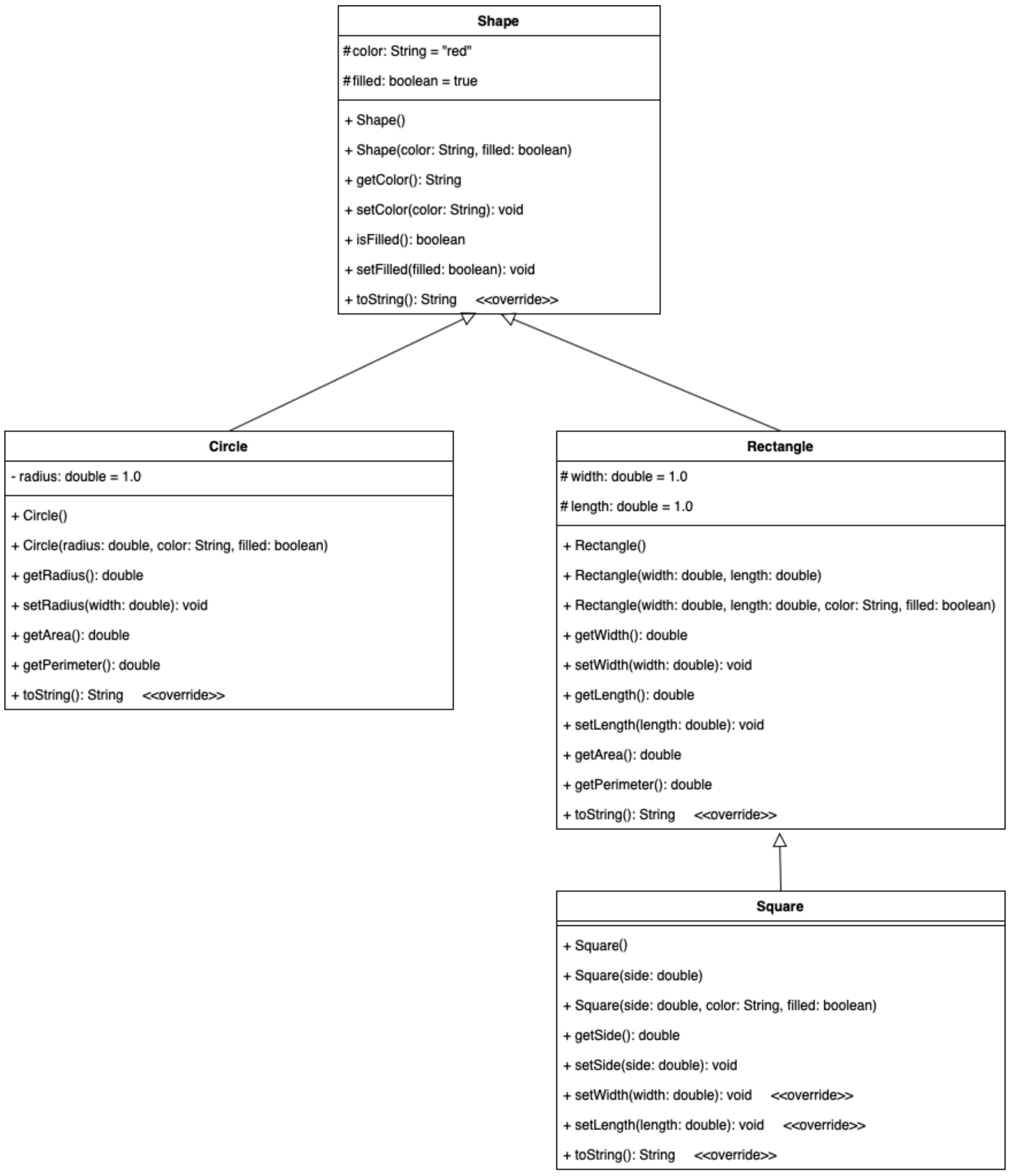
Implement the Java program based on the above diagram.

3. Giving the Point2D class and the Pointed3D class.



Implement the Java program based on the above diagram.

4. Giving superclass Shape and its subclasses.


Shape
#color: String = "red"
#filled: boolean = true
+ Shape()
+ Shape(color: String, filled: boolean)
+ getColor(): String
+ setColor(color: String): void
+ isFilled(): boolean
+ setFilled(filled: boolean): void
+ toString(): String <<override>>
Circle
- radius: double = 1.0
+ Circle()
+ Circle(radius: double, color: String, filled: boolean)
+ getRadius(): double
+ setRadius(width: double): void
+ getArea(): double
+ getPerimeter(): double
+ toString(): String <<override>>
Rectangle
# width: double = 1.0
# length: double = 1.0
+ Rectangle()
+ Rectangle(width: double, length: double)
+ Rectangle(width: double, length: double, color: String, filled: boolean)
+ getWidth(): double
+ setWidth(width: double): void
+ getLength(): double
+ setLength(length: double): void
+ getArea(): double
+ getPerimeter(): double
+ toString(): String <<override>>
Square
+ Square()
+ Square(side: double)
+ Square(side: double, color: String, filled: boolean)
+ getSide(): double
+ setSide(side: double): void
+ setWidth(width: double): void <<override>>
+ setLength(length: double): void <<override>>
+ toString(): String <<override>>

The format for the *toString()* method is: **ClassName[attribute1, attribute2, …]**. For example, the *toString()* method of the **Circle** class is expected to return the string: **Circle[<color>, <filled>, <radius>]**.

Implement the Java program based on the above diagram.

5. Implement the Employee class to store the information of employees in the manufacturing company ABC.

   Attributes:

   - **ID**: String

   - **fullName**: String

   - **yearJoined**: int

   - **coefficientsSalary**: double

   - **numDaysOff**: int (number of days off in the month)

   Constructors:

   - Constructor with no parameter **Employee()** (ID = 0, fullName = ””, yearJoined = 2020, coefficientsSalary = 1.0, numDaysOff = 0)

   - Constructor with parameter **Employee(ID: String, fullName: String, coefficientsSalary: double)** (yearJoined = 2020, numDaysOff = 0)

   - Constructor with full parameters.

   Methods:

   - **public double getSenioritySalary():** calculating seniority salary of employees: Know that if an employee works for 5 years or more, the seniority salary is calculated according to the following formula:

   seniority salary = years of work * basic salary / 100

   - **public String considerEmulation():** write a method to evaluate employee emulation.

   If the number of *holidays* <= 1 is graded A.

   If 2 <= the number of holidays <= 3 is graded B.

   If the number of holidays > 3 is graded C.

   - **public double getSalary():** write a method for calculating salaries for employees. Know that *salary* is calculated using the following formula with *basic salary = 1150*:

   *salary = basic salary + basic salary * (salary coefficient + emulation coefficient) + seniority salary*

   • If rated A: emulation coefficient = 1.0

   • If rated B: emulation coefficient = 0.75

   • If rated C: emulation coefficient = 0.5

   ABC Company also has a management team called Managers to manage all the company's activities. Let's build a Manager class to let ABC know that managers are also employees of the company. However, due to the role and function, each manager will have a corresponding position, department, and salary coefficient by position. The manager is also an employee, we will let the Manager class inherit from the Employee class and add some necessary attributes.

Attributes:

- The additional attributes include *position*, *department*, and *salary coefficient* by position.

Constructors:

- Constructor with no parameter **Manager()**: write a default constructor that creates a manager like an employee but has the position of head of the administrative office and a coefficient salary of 5.0.

- Constructor with parameter **Manager(ID: String, fullName: String, coefficientsSalary: double, position: String, salaryCoefficientPosition: double)** (yearJoined = 2024, numDaysOff = 0).

- Constructor with full parameters.

Methods:

- **public String considerEmulation()**: override the method to evaluate employee emulation and know that the manager is *always rated A*.

- **public double bonusByPosition():** calculating bonus using the following formula:

*position bonus = basic salary * salary coefficient by position*

- **public double getSalary()**: override the method for calculating salaries for employees. Know that the manager's salary is calculated using the following formula:

*salary = basic salary + basic salary * (salary coefficient + emulation coefficient) + seniority salary + position bonus*

Write a class with the *main* method to test the classes above.

6. In this exercise, students will develop a **basic order management system** for a Seven-Eleven convenience store.

   You are tasked with designing a **Seven-Eleven order management system** that can:

   - Manage **inventory** of food and beverage items.
   - Calculate **final prices**, including promotions and surcharges.
   - Process **customer orders** and update stock levels.
   - Automatically **apply store promotions** where applicable.

   Your program should simulate a real store experience, handling **at least six products**, processing orders, and generating an invoice.

   Your solution should include the following **five Java classes**:

a. Product (Base Class)

This is the parent class representing a general product sold at the store.

Attributes (Protected for Inheritance)

- *name*: The name of the product.
- *basePrice*: The standard price before any discounts or surcharges.
- *stock*: The number of available units.
- *promotionDiscount*: The percentage discount applied (if applicable).

Methods

- **calculateFinalPrice()**: Returns the final price after applying the promotion discount.
- **sellItem(int quantity)**: Decreases stock if enough items are available, returning **true** if successful, otherwise **false**.
- **toString()**: Give the string of values of product's attributes.

b. FoodItem (Subclass of Product)

Represents **food items** sold at Seven-Eleven.

Additional Attribute

- *isHot*: A boolean value indicating whether the food is served hot.

Business Rule

- If *isHot* is true, a **surcharge of 5,000 VND** is added to the final price.

Overrides

- **calculateFinalPrice()**: Adjusts the final price based on the isHot condition.

c. BeverageItem (Subclass of Product)

Represents **beverage items** sold at Seven-Eleven.

Additional Attribute

- *size*: A character (S, M, or L) representing small, medium, or large cup sizes.

Business Rule

- M size adds **3,000 VND**.
- L size adds **6,000 VND**.

Overrides

- **calculateFinalPrice()**: Adjusts the final price based on size.

d. OrderManager (Handles Sales Transactions)

Manages store inventory and processes customer purchases.

Responsibilities

- Maintain an **array** of available products.
- **findProduct(String name):** Searches for a product by name and returns it if found, otherwise returns **null**.
- **processOrder(String[] orderNames, int[] quantities):**
  - Loops through the order list, deducting stock and calculating total cost. If a product is out of stock, skip it.
  - Applies the "Buy 1 Ham Sandwich, Get 1 Free Vietnamese Iced Coffee" promotion. This promotion only applies to 1 coffee per order. (Check the quantity of Vietnamese Iced Coffee carefully before applying the promotion. If the order include Vietnamese Iced Coffee, this item will be free).
  - Returns the total order cost.

e. SevenElevenOrderManager (Main Class for Execution)

This class serves as the entry point for the program.

Responsibilities

- Initialize at least **six products** in the store inventory.
- Simulate customer transaction.
- Print out the total cost of the transaction and the stock after order completion.

Given the example:
Inventory Information:
The store has the following products available in stock:

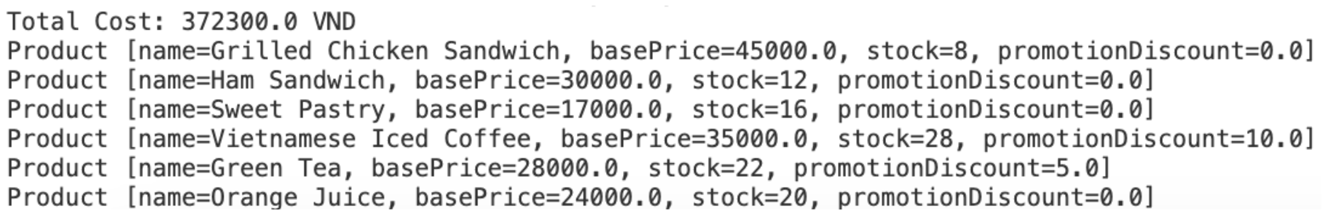
Product Name	Type	Base Price (VND)	Stock	Discount (%)	Extra Charge Rule
Grilled Chicken Sandwich	Food	45000	10	0	Hot
Ham Sandwich	Food	30000	15	0	None
Sweet Pastry	Food	17000	20	0	None
Vietnamese Iced Coffee	Beverage	35000	30	10	Size M
Green Tea	Beverage	28000	25	5	None
Orange Juice	Beverage	24000	20	0	Size L

Example Order Input:
A customer orders the following items:

- Grilled Chicken Sandwich × 2
- Ham Sandwich × 3
- Sweet Pastry × 4

- Vietnamese Iced Coffee × 2
- Green Tea x 3

The example of output:



```
Total Cost: 372300.0 VND
Product [name=Grilled Chicken Sandwich, basePrice=45000.0, stock=8, promotionDiscount=0.0]
Product [name=Ham Sandwich, basePrice=30000.0, stock=12, promotionDiscount=0.0]
Product [name=Sweet Pastry, basePrice=17000.0, stock=16, promotionDiscount=0.0]
Product [name=Vietnamese Iced Coffee, basePrice=35000.0, stock=28, promotionDiscount=10.0]
Product [name=Green Tea, basePrice=28000.0, stock=22, promotionDiscount=5.0]
Product [name=Orange Juice, basePrice=24000.0, stock=20, promotionDiscount=0.0]
```

--- THE END ---