

**Problem 1**

Write a program to price public transportation tickets. Create a class **Ticket** with the private static field **basePrice** (and the corresponding getter and setter) and a non-static method **getPrice** that returns this price. Also, add a static function **getSum** which calculates and returns the total price of all tickets passed to the function as an array of type **Ticket[]**. MT

Create a class **ReducedTicket** inheriting from **Ticket**. Override the method **getPrice** to return 50% of the base price.

Create a class **SeniorTicket** inheriting from **Ticket**. Override the method **getPrice** to return the price reduced by a fixed amount of 5 PLN (but not less than 0 PLN).

In the **main** function, create an array of type **Ticket[]**. Fill it with different tickets and use the **Ticket.getSum** function to calculate the total price for all the tickets from the array.

For example, the following program

[download Tickets.java](#)

```
public class Main {
    public static void main(String[] args) {
        Ticket[] tickets = {
            new Ticket(), new ReducedTicket(),
            new Ticket(), new SeniorTicket()
        };

        Ticket.setBasePrice(4.50);
        System.out.printf("Base price = %4.2f; Sum = %5.2f\n",
            Ticket.getBasePrice(), Ticket.getSum(tickets));

        Ticket.setBasePrice(6.60);
        System.out.printf("Base price = %4.2f; Sum = %5.2f\n",
            Ticket.getBasePrice(), Ticket.getSum(tickets));
    }
}
```

should print

```
Base price = 4,50; Sum = 11,25
Base price = 6,60; Sum = 18,10
```

**Problem 2**

Write a program which defines a **Strategy** class

```
class Strategy {
    public double getMean(double[] data) {
```

```

        throw new UnsupportedOperationException();
    }
}

```

and then classes inheriting from it that override the **getMean** method: **StrategyArithmetic**, **StrategyGeometric**, and **StrategyHarmonic**.

The **Mean** class has a field of type **Strategy** and an array of numbers (of type **double[]**) which is passed to the constructor. The **setStrategy** method allows changing the method (strategy) of calculating the mean value for the elements  $v_1, v_2, \dots, v_n$  of the array. The **getMean** method of this class delegates the necessary calculations to the strategy object, which in our case, calculates one of the so called Pythagorean means:

- arithmetic:  $\frac{v_1 + v_2 + \dots + v_n}{n}$
- geometric:  $\sqrt[n]{v_1 \cdot v_2 \cdot \dots \cdot v_n}$
- or harmonic:  $\frac{n}{\frac{1}{v_1} + \frac{1}{v_2} + \dots + \frac{1}{v_n}}$

The following program

[download Strategies.java](#)

```

public class Strategies {
    public static void main(String[] args) {
        double[] data = {2, 3, 4, 5, 6, 7, 8, 9};
        Mean mean = new Mean(data);

        double    meanVal;

        mean.setStrategy(new StrategyArithmetic());
        meanVal  = mean.getMean();
        System.out.printf("%10s strategy: mean = %5.3f\n",
                           "Arithmetic", meanVal);

        mean.setStrategy(new StrategyGeometric());
        meanVal  = mean.getMean();
        System.out.printf("%10s strategy: mean = %5.3f\n",
                           "Geometric", meanVal);

        mean.setStrategy(new StrategyHarmonic());
        meanVal  = mean.getMean();
        System.out.printf("%10s strategy: mean = %5.3f\n",
                           "Harmonic", meanVal);
    }
}

```

should print

```
Strategy arithmetic: mean = 5,500
Strategy  geometric: mean = 4,954
Strategy  harmonic: mean = 4,374
```

This simple program illustrates important programming concepts: the Strategy pattern, composition instead of inheritance, dynamically changing an object's behavior during program execution, and the Open/Closed Principle: classes, modules, and functions should be open for extension but closed for modification (you can add new functionality without altering existing code).

### Problem 3

---

Define classes from the program below:

KB

[download Patients.java](#)

```
public class Patients {
    public static void main(String[] args) {
        Patient[] patients = {
            new IllHead("Johnny"),
            new IllLeg("Eddy"),
            new IllDyspepsia("Manny")
        };
        for (Patient p : patients) {
            System.out.println("Patient:  " +
                p.name() + '\n' + "Illness:  " +
                p.illness() + '\n' + "Treatment: " +
                p.treatment() + "\n"
            );
        }
    }
}
```

in such a way that it produces the following output:

```
Patient:  Johnny
Illness:  head
Treatment: aspirin

Patient:  Eddy
Illness:  leg
Treatment: plaster

Patient:  Manny
Illness:  dyspepsia
Treatment: coal
```

NOTE: The **Patients** class must not be modified.

---