ЛАБОРАТОРНО УПРАЖНЕНИЕ №1

Основи на програмния език Java

Цел: Реализиране на Java конзолни приложения, които са свързани като тематика с практическото използване на операторите на езика, конструкции за управление на изпълнението, програмни класове, обекти и интерфейси и обработка на изключения.

Задача 1: Напишете Java конзолно приложение, което създава масив от N на брой случайни цели числа в диапазона [0, N], където N е четно число. Напишете програмния код, чрез който на конзолата се разпечатва позицията на всички числа със стойност M=N/2.

Стойността на N трябва да се въвежда от клавиатурата и трябва да се валидира – да е четно число, по-голямо от 0. Как да получим случайно число в Java? Може да използвате статичен метод random от клас Math. Той връща случайно число в интервала [0, 1]. Трябва да преобразувате този интервал до [0, N], например: int value = (int) (Math.random() * N); В случая се налага кастване на стойността, върната от метод random() до цяло число.

На Фиг. 1 е показан програмния код чрез който може да се получи и валидира стойността на броя на елементите в масива N. Реализирайте програмната реализация на Задача 1 като създадете масив с N елемента и проверите дали в него има елементи със стойност M=N / 2.

```
boolean status;
String warning
        = String.format("N трябва да е четно число по-голямо от 0");

Scanner input = new Scanner(System.in);
do {
    status = false;
    System.out.print("Моля, въведете броя на елементите в масива N = ");
    try {
        N = input.nextInt();
        if (N < 1 || N % 2 == 1) {
            System.out.println(warning);
        } else {
            status = true;
        }
    } catch (InputMismatchException e) {
        input.next();
        System.out.println(warning);
    }
} while (!status);
```

Фиг. 1. Програмен код за въвеждане и валидиране на стойността на N

Задача 2: Да се напише Java конзолно приложение, което описва програмно двумерни геометрични фигури (правоъгълник, прост полигон и окръжност). Да се предостави възможност за изчисляване на периметъра и площта на всяка от фигурите.

Анализирайте условието на задачата и преценете колко програмни класа ще използвате, ще използвате ли наследяване и/или програмен интерфейс.

Помощ: Всички фигури, без окръжността, се описват като последователност от точки в Декартова координатна система. Следователно, трябва да декларирате клас, който описва точка. Всяка една от фигурите (правоъгълник, полигон и окръжност) е геометрична фигура. Можем да декларираме клас Shape, който да е родителски за класове Rectangle, Polygon и Circle. В този клас трябва да декларираме всички свойства и действия, които са общи за всички дъщерни класове. За всяка фигура трябва да се декларират следните полета: name, perimeter и area, както и методи calculateArea() и calculatePerimeter(). За да накараме програмиста, който наследява Shape, да не забрави да напише кода на тези методи ще ги направим абстрактни. Това автоматично налага клас Shape да бъде също абстрактен. Ще използваме и един интерфейс – ShapeInterface – чрез който ще се реализира извличане на информация за произволна фигура. Това ще реализираме чрез методи getName(), setName(), getPerimeter(), getArea(), showInfo() и getInfo(). Последните два метода ще разпишем направо в тялото на интерфейса. За целта те трябва да се обявят като default. Останалите методи се декларират абстрактни, без да е необходима ключова дума abstract. И така, всеки клас наследник на Shape и реализиращ ShapeInterface трябва задължително да има декларация на техните методи.

Всеки правоъгълник или полигон ще описваме като последователност от точки в Декартова координатна система. За целта ще използваме клас Point. За да определим периметъра на полигона и правоъгълника ще ни трябва разстоянието между две точки. Ще използваме статичен метод distance, деклариран в тялото на клас ShapeUnit. Ако ви потрябват други общи методи, свързани с изчисления, декларирайте ги в този клас. На Фиг. 2 е показан клас Point и метод чрез който се изчислява площта на полигон.

```
public class Point {
    public int x, y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
private double calculateArea(Point[] points) {
    double sum = 0;
    int n = points.length;
    for (int i = 0; i < n - 1; i++) {
        sum += (points[i].x * points[i+1].y - points[i].y * points[i+1].x);
    }
    return 0.5 * Math.abs(sum);
}
```

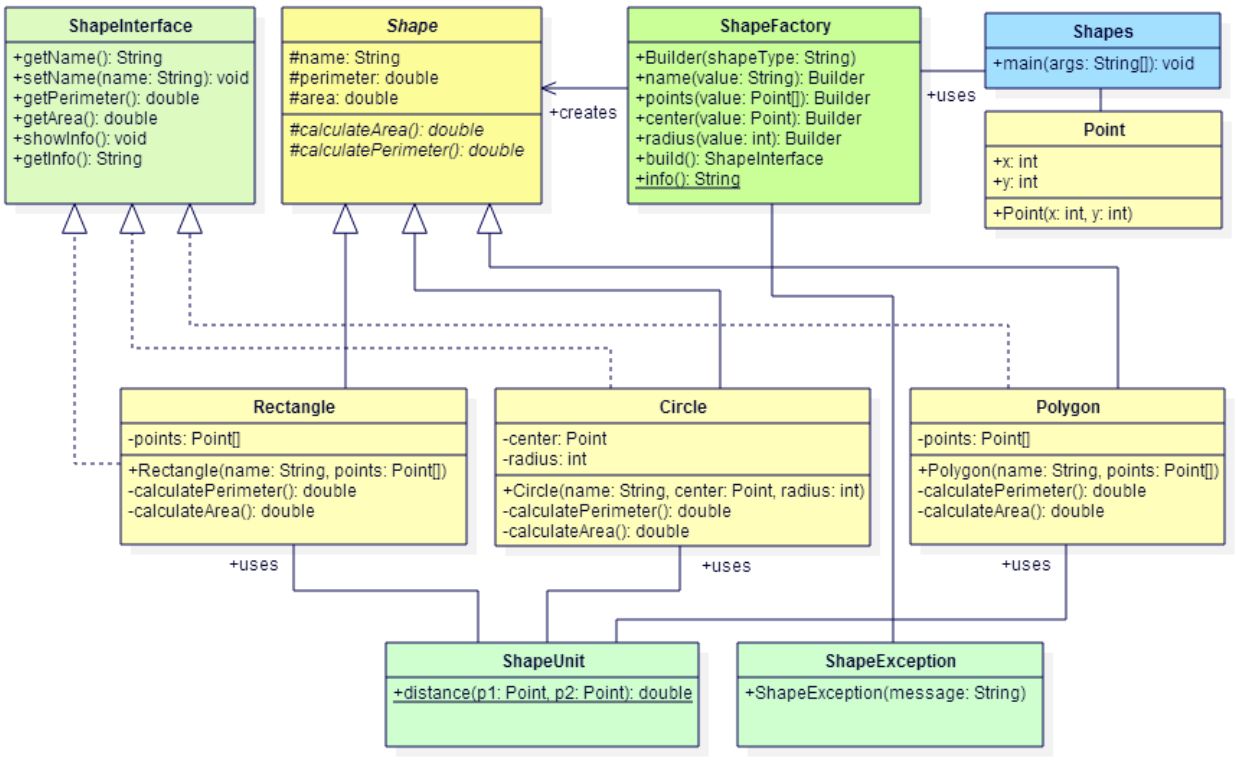
Фиг. 2. Клас Point и метод calculateArea за получаване на площта на прост полигон

Когато създавате обекти е добре да предоставите интерфейс на потребителя за това. Не е добра идея директно да създавате всеки обект чрез директно викане на конструктора му. Би било добре да не се налага потребителят на вашите класове да помни точно как се създава обект от вашия клас, какви аргументи и в каква последователност се очаква да спазвате. Освен това, грешките, получени при валидиране на стойностите на аргументите към конструкторите, трябва да се реализира за всеки клас по отделно. По-доброто решение е да се предостави унифициран интерфейс, който ще отговаря за създаването на обекти от нашите класове и ще обработва всички грешки само чрез този интерфейс. В конкретният случай този интерфейс ще реализираме чрез шаблон за дизайн Factory. Той се реализира чрез клас ShapeFactory. Този клас съдържа статичен клас Builder чрез който можем да създаваме произволна фигура (метод build). Следва пример за използване на ShapeFactory, за да се създаде правоъгълник:

```
ShapeInterface myRectangle =
    new ShapeFactory.Builder(ShapeType.RECTANGLE)
            .name("Правоъгълник-1").points(points1).build();
if (myRectangle != null) {
    // можем да използваме обекта
}
```

Ако обектът myRectangle е създаден успешно, то неговата инстанция не е null. Винаги използвайте интерфейс за създаване на своите обекти, тъй като така кодът ви е по-разбираем, лесно се модифицира, а обработката на грешки може да се съсредоточи на едно място.

На Фиг. 3 е показана структурата на проекта под формата на UML диаграма на класовете и връзките между тях.



Фиг. 3. UML диаграма на класовете и интерфейсите и връзките между тях

Задача 3: Да се напише Java конзолно приложение, което очаква въвеждане на низ от клавиатурата. Да се напише програмен клас Unit, който съдържа метод, който връща колко пъти се среща в низа зададен като аргумент символ (без значение дали е малка или главна буква, ако символът е буква):

```
int calculateOccuranceOfChar(String input, char character)
```

Да се валидира входа от клавиатурата. На конзолата да се разпечата кой символ се търси и колко пъти се среща в низа.

Задачата може да се реши по много начини:

1. Използвайте цикъл в който проверявате всеки символ от низа дали не съвпада по стойност със зададения. Използвайте метод charAt(i), за да получите кода на символа на позиция i от началото на низа.
2. Може да използвате регулярен израз чрез който да укажете какво търсите в низа. Когато търсите символ може да използвате следните регулярни изрази:

```
String regex1 = "['c']";
String regex2 = "(?i)['c']";
```

При regex1 се търсят всички символи 'c'. При regex2 се търсят всички символи 'c' или 'C' (case in-sensitive search). Използвайте метод compile() от Pattern, за да получите шаблон за търсене. След това приложете метод matcher() към този шаблон, за да получите желания резултат. Използвайте метод find(), за да проверите колко на брой са съвпаденията.

3. Може да използвате възможностите на Java 8 за филтриране на потоци. Преобразувайте низа в поток от символи чрез метод chars(). След това приложете метод filter() към така получения поток. Метод filter() получава като аргумент функция, която съдържа логиката на филтриране. Накрая, получете броя на съвпаденията чрез метод count(), който трябва да приложите към резултата, получен след изпълнение на filter(), например:

```
long counter = input.chars().filter(c -> c == CHARECTER).count();
```