Obiektowe wizytówki

Naszym zadaniem będzie stworzenie programu do tworzenia i drukowania wizytówek. Wizytówka zawiera zazwyczaj kilka cennych informacji, np:

- imię
- nazwisko
- numer telefonu

Informacje te przedstawione są na wizytówce w schludny sposób, otoczone ramką:

```
******************
* Piotr Budynek  *
* tel. 123456789 *
******************
```

Zaproponuj **obiektową** reprezentację wizytówki, która będzie umożliwiała:

- tworzenie wizytówek z podaniem wszystkich niezbędnych informacji
- tworzenie wizytówek bez podania informacji - taka wizytówka powinna ustawić swoje wartości na domyślne (z przykładu powyżej)
- ustawianie/pobieranie każdej z informacji zawartej na wizytówce (imię, nazwisko, telefon)
- pobranie znaku, reprezentującego ramkę (`*`) bez możliwości jego zmiany
- wydrukowanie (wypisanie) wizytówki

Stwórz także klasę testującą rozwiązanie (zawierającą funkcję `main`), w której stworzysz kilka przykładowych wizytówek i spróbujesz je wydrukować.

Do realizacji ostatniego punktu wykorzystaj poniższy kod (będzie nam potrzebny w kolejnym zadaniu):

```
public void print() {
    ArrayList<String> lines = getLines();

    int maxLength = getMaxLength(lines);

    String borderLine = "";
    for (int i = 0; i < maxLength + 2; i++) {
        borderLine += getBorderChar();
    }

    System.out.println(borderLine);
```

```
        for (String line : lines) {
            int lengthDiff = maxLength - line.length();

            for (int i = 0; i < lengthDiff; i++) {
                line += " ";
            }

            line = getBorderChar() + line + getBorderChar();

            System.out.println(line);
        }

        System.out.println(borderLine);
    }


    private int getMaxLength(ArrayList<String> lines) {
        int maxLength = 0;

        for (String line : lines) {
            int lineLength = line.length();

            if (lineLength > maxLength) {
                maxLength = lineLength;
            }
        }

        return maxLength;
    }

    protected ArrayList<String> getLines() {
        ArrayList<String> lines = new ArrayList<>();

        lines.add(name + " " + surname);
        lines.add("tel. " + phone);

        return lines;
    }
```

Przydatne informacje

Atrybuty, metody i konstruktory

Jak dotąd nie tworzyliśmy obiektów własnych klas i omijaliśmy mechanizmy obiektowe w Javie. W rzeczywistości każda klasa, którą tworzymy to pełnoprawny typ danych, który określa swoje zachowanie i rodzaje danych, które może przechowywać.

Wewnątrz klasy mogą znajdować się elementy trzech rodzajów:

- atrybuty
- metody
- konstruktory

Atrybuty (pola) to zmienne lub stałe reprezentujące dane, które może przechowywać klasa. Obiekt danej klasy posiada konkretne wartości swoich atrybutów. Jeśli na przykład stworzymy klasę `Point`, która ma posłużyć do reprezentowania współrzędnych (x,y) w dwuwymiarowej przestrzeni, definicja takiej klasy może wyglądać np. tak:

```
public class Point {
   public double x;
   public double y;
}
```

Oznacza to, że każdy punkt składa się z dwóch atrybutów typu `double`: `x` oraz `y`. Gdy utworzymy obiekt takiego typu, możemy odwoływać się do wartości tych pól przy pomocy operatora `.`:

```
Point myPoint = new Point();
myPoint.x = 3.0;
myPoint.y = 4.0;
```

Oprócz atrybutów każda klasa może także definiować różne zachowania. Służą do tego **metody**. W poprzednich zadaniach pisaliśmy już własne metody, ale w nieco inny sposób (tworzyliśmy jedynie metody statyczne). Załóżmy, że nasz punkt udostępnia metodę `distance()`, która umożliwia zmierzenie odległości punktu od początku układu współrzędnych (punktu (0,0)). Nasza klasa będzie teraz wyglądać następująco:

```
public class Point {
   public double x;
   public double y;
```

```
    public double distance() {
        return Math.sqrt(x*x + y*y);
    }
}
```

Zauważmy, że metoda może operować na atrybutach danej klasy. Tak stworzoną metodę możemy oczywiście wywołać na obiekcie naszej klasy:

```
Point myPoint = new Point();
myPoint.x = 3.0;
myPoint.y = 4.0;

double d = myPoint.distance(); // po wywołaniu metody zmienna d będzie miała wartość 5
```

Domyślnie każdy atrybut w nowoutworzonym obiekcie posiada jakąś wartość: dla prymitywnych typów liczbowych jest to 0, dla typu `boolean` wartość `false`, a dla typów obiektowych pusta wartość `null`.
Jeśli chcemy już w trakcie tworzenia obiektu zainicjować wartości pól lub wykonać jakiekolwiek inne operacje, powinniśmy zdefiniować tzw. **konstruktor**. Jest to specjalna metoda, która musi spełniać dwa warunki:

- nazywać się tak jak klasa, wewnątrz której ją definiujemy
- nie posiadać typu zwracanego (nawet `void` !)

Konstruktory mogą za to posiadać parametry tak jak normalne metody i może być ich kilka - muszą jednak różnić się liczbą lub typem parametrów. W naszym przykładzie możemy zdefinować konstrutkor, który przyjmuje wartości `x` i `y`:

```
public class Point {
    public double x;
    public double y;

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public double distance() {
        return Math.sqrt(x*x + y*y);
    }
}
```

Pojawia się tu słowo kluczowe `this` - jest to po prostu referencja na obiekt, na którym wywoływana jest metoda ("ja sam"). `this` możemy używać w każdej metodzie w obrębie danej klasy - najczęściej służy nam do odróżnienia atrybutu klasy od zmiennej lokalnej (parametru metody), tak jak w powyższym przykładzie. Do konstruktora zostają przekazane wartości zapisane w parametrach `x` i `y`, a następnie wartości tych parametrów są przypisywane do atrybutów `this.x` i `this.y`. Zauważmy, że w metodzie `distance` pominęliśmy `this` - nie było ono konieczne, gdyż w tej sytuacji nie mamy niejednoznaczności.

Po modyfikacjach punkty będą tworzone w nieco inny sposób:

```
Point myPoint = new Point(3.0, 4.0);
myPoint.x = 6.0; // wciąż możemy zmieniać wartości atrybutów
```

Jak widać, konstruktory są wykorzystywane wraz ze słowem `new`. Jeśli nie zdefiniowaliśmy w klasie żadnego konstruktora (jak na początku tego przykładu), posiada ona konstruktor domyślny, bez żadnych parametrów. Gdy zdefiniujemy jakikolwiek inny konstruktor, domyślny przestaje być dostępny - chyba że jego również zdefiniujemy:

```
public Point() {
   this(0.0, 0.0);
}
```

Ponownie wykorzystaliśmy tu `this`, tym razem jednak jako odwołanie do innego konstruktora - tego typu wywołania zawsze musi być pierwszą linijką konstruktora (lub nie pojawić się w ogóle)

Modyfikatory dostępu

Do tej pory wszystkie metody, które pisaliśmy poprzedzaliśmy modyfikatorami `public static`. Słowo kluczowe `static` oznacza, że metoda (lub atrybut) jest statyczna, a więc niezwiązana z konkretnym obiektem danej klasy. Przykładem takiej metody jest `main`, które rozpoczyna każdy program napisany w Javie. `main` nie jest metodą wołaną na jakimkolwiek obiekcie, wiąże się jedynie z samą klasą, w której ją zamieścimy.

Słowo kluczowe `public` to jeden z tzw. **modyfikatorów dostępu**. Możemy w ten sposób określać które atrybuty lub metody będą widoczne dla innych klas i w jakim zakresie. Istnieją cztery rodzaje widoczności:

- `public` - element jest widoczny dla wszystkich klas i można go używać na zewnątrz, niezależnie od miejsca, z którego go wywołamy.
- `private` - element jest widoczny tylko dla klasy, która go deklaruje. Może być używany wewnątrz tej klasy i nigdzie poza nią.

- `protected` - element jest widoczny tylko dla klasy, która go deklaruje oraz dla klas rozszerzających i znajdujących się w tym samym pakiecie. O pakietach i klasach rozszerzających powiemy w dalszej części zajęć.
- brak modyfikatora - element jest widoczny tylko dla klasy, która go deklaruje oraz dla klas znajdujących się w tym samym pakiecie.

Jedną z podstawowych zasad programowania obiektowego jest tzw. **enkapsulacja**. Przyjmuje się, że atrybuty danej klasy powinny być prywatne, a dostęp do nich kontrolowany przez publiczne metody.

Niektóre metody mogą być również prywatne, jeśli pełnią rolę pomocniczych w danej klasie i nie powinny być dostępne na zewnątrz. W poprzednim przykładzie atrybuty klasy `Point` były publiczne. To dlatego mogliśmy je modyfikować bezpośrednio:

```
myPoint.x = 3.0;
myPoint.y = 4.0;
```

Jeśli zmienimy ich modyfikatory na `private` powyższa operacja będzie niemożliwa. Jak zatem modyfikować wartości atrybutów? Jeśli jest taka potrzeba, możemy zawsze stworzyć metodę, która zwraca lub ustawia wartość danego atrybuty. Tego typu metody nazywamy odpwiednio **akcesorami** i **mutatorami**. W tym przypadku jest to jedynie konwencja, metody te nie mają żadnego specjalnego znaczenia z punktu widzenia składni Javy:

```
public class Point {
   private double x;
   private double y;

   public Point(double x, double y) {
      this.x = x;
      this.y = y;
   }

   public double getX() {
      return x;
   }

   public void setX(double x) {
      this.x = x;
   }

   public double getY() {
      return y;
   }
```

```
    public void setY(double y) {
        this.y = y;
    }

    public double distance() {
        return Math.sqrt(x*x + y*y);
    }
}
```

Od tej pory wartości punktu możemy modyfikować używając metod typu `set`:

```
myPoint.setX(3.0);
myPoint.setY(4.0);
```