Java - wprowadzenie do programowania obiektowego

Definiowanie klasy

- Przygotujemy klasę `Person`, zawierającą pola `firstName` i `lastName`:

```
package tpsi.labs.lab2;

public class Person {

    private String firstName;
    private String lastName;

}
```

- Dodajmy do klasy konstruktor inicjujący pola `firstName` i `lastName`:

```
public Person(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
}
```

- Dodajmy metody umożliwiające dostęp do pól `firstName` i `lastName` - zgodnie z konwencją, nazwiemy je `getFirstName()` i `getLastName()`:

```
public String getFirstName() {
    return firstName;
}

public String getLastName() {
    return lastName;
}
```

- Przydadzą nam się także metody pozwalające na ustawianie wartości obu pól - odpowiednio `setFirstName()` i `setLastName()`:

```
public String getFirstName() {
    return firstName;
}

public String getLastName() {
    return lastName;
}

public void setFirstName(String firstName) {
    this.firstName = firstName;
}
```

```
public void setLastName(String lastName) {
    this.lastName = lastName;
}
```

- Cały kod klasy powinien w tej chwili wyglądać tak:

```
package tpsi.labs.lab2;

public class Person {

    private String firstName;
    private String lastName;

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```

Tworzenie obiektów

- Jeśli nie masz w projekcie klasy głównej z metodą `main()`, dodaj ją teraz - nazwijmy ją `Main`, umieśćmy w niej statyczną metodę `main()`:

```
public static void main(String[] args) {

}
```

- Wypróbujmy klasę `Person` - w metodzie `main()` utworzymy parę obiektów tej klasy:

```
public static void main(String[] args) {
```

```
    Person p1 = new Person("Bill", "Gates");
    Person p2 = new Person("Gal", "Anonim");

    System.out.println(p1);
    System.out.println(p2);
}
```

- Co zostało wyświetlone na ekranie po wywołaniu `System.out.println(p1)`? Czy tego oczekiwaliśmy?
- Aby móc konwertować obiekty na `String` (np. w celu wyświetlenia na ekranie lub zapisania w logu), musimy dodać w klasie metodę `toString()`. Do klasy `Person` dodaj kod:

```
@Override
public String toString() {
    return firstName + " " + lastName;
}
```

- Sprawdź działanie - teraz powinno być lepiej...

Dziedziczenie

- Dodajmy do projektu klasę `Student`.
- `Student` powinien dziedziczyć z klasy `Person`:

```
package tpsi.labs.lab2;

public class Student extends Person {

}
```

- Widzimy w tym momencie błąd kompilacji - dlaczego?

minuta na zastanowienie....

i jeszcze minutka...

- W klasie `Student` brakuje konstruktora kompatybilnego z klasą nadrzedną `Person`. Dodajmy taki konstruktor i wywołajmy w nim konstruktor klasy bazowej:

```
public Student(String firstName, String lastName) {
    super(firstName, lastName);
}
```

- W tej chwili kod już powinien się kompilować.
- Dodajmy studentowi pole z identyfikatorem grupy:

```
private String groupId;
```

- Dodajmy do konstruktora nowy parametr, inicjujący grupę:

```
public Student(String firstName, String lastName, String groupId) {
    super(firstName, lastName);
    this.groupId = groupId;
}
```

- Dodajmy też getter i setter:

```
public String getGroupId() {
    return groupId;
}

public void setGroupId(String groupId) {
    this.groupId = groupId;
}
```

- Kompletny kod klasy `Student` powinien w tej chwili wyglądać tak:

```
package tpsi.labs.lab2;

public class Student extends Person {

    private String groupId;

    public Student(String firstName, String lastName, String groupId) {
        super(firstName, lastName);
        this.groupId = groupId;
    }

    public String getGroupId() {
        return groupId;
    }

    public void setGroupId(String groupId) {
        this.groupId = groupId;
    }
}
```

Zadanie 1

Dodaj do projektu klasę `Teacher`. `Teacher` dziedziczy z `Person` i ma dodatkowo pole `courseName` z nazwą uczonego przedmiotu. Pole `courseName` powinno być inicjowane w konstruktorze, będziemy też potrzebować metod akcesorów (getter i setter).

Definiowanie interfejsu

- Zdefiniujemy teraz interfejs `EmailRecipient`. Klasy implementujące ten interfejs będą reprezentowały adresatów emaila - czyli dowolny obiekt, który może być odbiorcą emaila (a zatem dowolna osoba, ale także na przykład instytucja).
- Dodaj do projektu interfejs (podobnie jak dodawałeś klasy), nazwijmy go `EmailRecipient`:

```
package tpsi.labs.lab2;

public interface EmailRecipient {

}
```

- Interfejs będzie posiadał jedną metodę `getEmailAddress()`:

```
package tpsi.labs.lab2;

public interface EmailRecipient {

    String getEmailAddress();

}
```

- Klasy implementujące intefejs `EmailRecipient` będą musiały posiadać metodę `getEmailAddress()`.
- Wróćmy do klasy `Person` - niech klasa ta implementuje interfejs `EmailRecipient`:

```
public class Person implements EmailRecipient {
    ...
}
```

- Powinieneś zobaczyć w tym momencie błąd kompilacji. Dlaczego?

i znowu minutka na zastanowienie...

albo nawet dwie...

już? wiesz?

na pewno?

- Skoro klasa `Person` implementuje interfejs `EmailRecipient`, musi posiadać zadklarowane w nim metody - w tym przypadku jedną metodę `getEmailAddress()`.

Zadanie 2

Dokończ implementację interfejsu `EmailRecipient` w klasie `Person`:

- dodaj pole `emailAddress`;
- dodaj metodę wymaganą przez interfejs, zwróć w niej wartość pola `emailAddress`;
- dodaj nowy argument w konstruktorze, inicjujący wartość pola `emailAddress`.

W tym momencie klasa `Person` powinna się kompilować i działać prawidło - ale w klasach `Student` i `Teacher` zobaczymy znowu błąd kompilacji. Dlaczego? Spróbuj naprawić klasy `Student` i `Teacher`.

Interfesjy - c.d.

- Jak już mówiliśmy, nie tylko osoba może być adresatem emaila. Dodajmy do projektu klasę `University`. Uniwersytet opisany będzie nazwą i adresem email. Klasa implementować będzie interfejs `EmailRecipient`:

```
package tpsi.labs.lab2;

public class University implements EmailRecipient {

    private String emailAddress;
    private String name;

    public University(String name, String emailAddress) {
        this.name = name;
        this.emailAddress = emailAddress;
    }

    public String getName() {
        return name;
    }

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

    @Override
    public String getEmailAddress() {
        return emailAddress;
    }
}
```

- Zobaczmy teraz, jak możemy skorzystać w kodzie z faktu, że klasy implementują pewien interfejs. Załóżmy, że piszemy ogólny kod do rozsyłania spamu - ten fragment kodu nie jest zainteresowany tym, czy wysyła email do osoby, instytucji czy psa: ważne, żeby obiekt posiadał adres email, czyli implementował interfejs `EmailRecipient`.

- W klasie `Main` stwórzmy kilka obiektów róznych typów i przygotujmy listę odbiorców dla naszego spammera:

```
University zut = new University("Zachodniopomorski Uniwersystet Technologiczny w Szczec
University wsks = new University("Wyższa Szkoła Kosmetologii Stosowanej w Koluszkach",

Person s1 = new Student("Jan", "Kowalski", "jkowalski@wi.zut.edu.pl", "32A");
Person s2 = new Student("Hermenegilda", "Nowak", "henowak@wi.zut.edu.pl", "32A");
Person t1 = new Teacher("Bjarne", "Stroustrup", "bjarne@fake.org", "Introduction to C++

EmailRecipient[] spamList = new EmailRecipient[5];
spamList[0] = zut;
spamList[1] = wsks;
spamList[2] = s1;
spamList[3] = s2;
spamList[4] = t1;
```

- Zauważ, że do tablicy zadklarowanej jako `EmailRecipient[]` możemy dodać obiekty dowolnych klas implementujących dany interfejs. Mogą to być klasy z różnych hierarchii dziedziczenia (a więc nie tylko `Person`/`Student`/`Teacher`, ale także `University`).
- Chcąc rozesłać spam iterujemy po liście odbiorców - ten kod nie musi wiedzieć, z jaką konkretnie klasą ma do czynienia:

```
for(EmailRecipient recipient : spamList) {
    String email = recipient.getEmailAddress();
    System.out.println(email);
}
```

Wprowadzenie do kolekcji: listy

- Załóżmy teraz, że chcemy przechować w obiektach klasy `Student` informacje o ocenach:
  - Ocena niech będzie reprezentowana przez wartość typu `double`.
  - Student może mieć wiele ocen, nie wiemy z góry ile - nie możemy zatem użyć tablicy.
  - Użyjemy listy - lista może przechowywać dowolną liczbę obiektów i dynamicznie zmieniać rozmiar.
- Przykład użycia klasy `ArrayList`:

```
List<String> kolory = new ArrayList<>();

kolory.add("czerwony");
kolory.add("niebieski");
kolory.add("zielony");
```

```
for(Sting kolor : kolory) {
    System.out.println(kolor);
}
```

- W powyższym kodzie widzimy deklarację `List<String>` - oznacza ona, że w deklarowanej zmiennej przechowywać będziemy obiekty klasy `String`.
- Na liście nie można przechowywać typów prostych (np. `int`, `double`), tylko obiekty - dlatego lista ocen w klasie `Student` przechowywać będzie obiekty klasy `Double`.
- Do klasy `Student` dodajmy nowe pole:

  ```
  private List<Double> grades = new ArrayList<>();
  ```

- Po utworzeniu obiektu lista ocen będzie pusta. Dodajmy zatem metody `getGrades()` i `addGrade()`, odpowiednio do pobrania listy ocen i dodania nowej oceny do listy:

  ```
  public List<Double> getGrades() {
      return grades;
  }

  public void addGrade(double grade) {
      grades.add(grade);
  }
  ```

Zadanie 3

Dodaj do klasy `Student` metodę `getGradesAverage()` obliczającą średnią ocen. Wypróbuj tę metodę w klasie `Main`.

Zadanie 4

Dodaj do projektu klasę `Faculty` reprezentującą wydział uniwersytetu. Uniwersytet powinien posiadać listę wydziałów (pole `faculties`) i metody `getFaculties()` i `addFaculty()`. Wydział powinien posiadać listę nauczycieli i metodę `addTeacher()`. Wydział powinien też implementować interfejs `EmailRecipient`.