PGO - Ćwiczenia 7-8

inż. Samsel Dariusz dsamsel@pjwstk.edu.pl
mgr inż. Gago Piotr pgago@pjwstk.edu.pl
mgr inż. Kubica Maksymilian maksous@pjwstk.edu.pl
inż. Anna Voitenkowa lacrit@pjwstk.edu.pl
mgr inż. Daniel Obrębski dobrebski@pjwstk.edu.pl

Metody statyczne

Z metodami statycznymi tak naprawdę mieliśmy już doświadczenia. Wykorzystywaliśmy praktycznie jedną taką metodę na każdych ćwiczeniach.

```
package pl.edu.pjwstk;

public class Main {

    // W momencie uruchomienia wywoływane jest Main.main() z tablicą argumentów
    public static void main(String[] args) {
        // Main start = new Main(); -- aby wywołać metodę
        // nie musieliśmy tworzyć obiektu Main
    }
}
```

Czym metoda statyczna się charakteryzuje?

1. Może być wywołana bez tworzenia obiektu danej klasy – odwołujemy się do niej za pomocą nazwy klasy.
2. Taka metoda ma tylko dostęp do atrybutów i metod statycznych danej klasy. Czyli w metodzie statycznej możemy wywoływać inne metody statyczne i korzystać z atrybutów statycznych.
3. Najczęściej służy jako metoda pomocnicza, do często powtarzających się zadań.

Przyjmijmy, że mamy klasę *Calculator*, która ma metodę pozwalającą na dodawanie dwóch elementów.

```
package pl.edu.pjwstk;

public class Calculator {

    public static int add(int a, int b) {
        return a + b;
    }
}
```

Wówczas z klasy *Main* możemy wywołać metodę *add* bez tworzenia obiektu *Calculator*:

```
package pl.edu.pjwstk;

public class Main {

    public static void main(String[] args) {
        Calculator.add(1, 2);
    }
}
```

Jak wspomnieliśmy wcześniej metody statyczne mogą korzystać tylko z atrybutów statycznych i mogą wywoływać inne metody statyczne.

```
public class Calculator {
    private static double PI = 3.14;

    private static double square(double d) {
        return d * d;
    }

    public static double getCircleSurfaceArea(double radius) {
        return PI * square(radius);
    }
}
```

W przykładzie powyżej. Mamy jedną publiczną metodę, która zwraca pole powierzchni koła. Korzysta ona ze zmiennej statycznej PI oraz z metody square, której celem jest podniesienie danej wartości do kwadratu, wywołanie:

```
package pl.edu.pjwstk;

public class Main {

    public static void main(String[] args) {
        Calculator.getCircleSurfaceArea(4.5);
    }
}
```

Niepoprawnym jest korzystanie z atrybutów niestatycznych klasy, poniższy kod nie skompiluje się:

```
package pl.edu.pjwstk;

public class Calculator {
    private double PI = 3.14;
    private int d = 3;

    private double square() {
        return d * d;
    }

    public static double getCircleSurfaceArea(double radius) {
        return PI * square();
    }
}
```

Błąd nastąpi z powodów:
`Non-static field ‘PI’ cannot be referenced from a static context.`
`Non-static method ‘square()’ cannot be referenced from a static context.`

Przypomnienie!
Metody statyczne także mogą być **przeciążane** – dlatego niepoprawnym jest poniższy kod:

```
package pl.edu.pjwstk;

public class Calculator {

    public int add(int a, int b) {
        return a +b;
    }

    public static int add(int a, int b) {
        return a + b;
    }
}
```

Powyższy kod się nie skompiluje, ponieważ są dwie metody *add*, które przyjmują takie same parametry.

Metody statyczne też mają **modyfikatory dostępu** w wypadku, gdy metoda *add* będzie prywatna:

```
package pl.edu.pjwstk;

public class Calculator {

    private static int add(int a, int b) {
        return a + b;
    }
}
```

Kod w klasie *Main* będzie się nie kompilował z powodu:
`'add(int, int)' has private access in 'pl.edu.pjwstk.Calculator'`

W naszym przypadku, ponieważ klasa Main jest w tym samym pakiecie co Calculator, metoda add i klasa Calculator nie powinny mieć żadnego modyfikatora dostępu:

```
package pl.edu.pjwstk;

class Calculator {

    static int add(int a, int b) {
        return a + b;
    }
}
```

Pola statyczne

W przykładzie powyżej mieliśmy pole statyczne PI. Pole statyczne to takie pole, które jest współdzielone przez wszystkie obiekty danej klasy. Pamięć ładowana jest tylko raz. Możemy wyobrazić sobie klasę User, która będzie przetrzymywała informacje na temat liczby zalogowanych aktualnie użytkowników.

```
package pl.edu.pjwstk;

class User {
    public static int loggedInUsers = 0;

    private String name;

    User(String name) {
        this.name = name;
    }

    void logIn() {
        loggedInUsers++;
    }

    void logOut() {
        loggedInUsers--;
    }
}
```

Jak widać zmienne statyczne mogą być wykorzystywane w niestatycznych metodach.

Do takiego parametru odwołać się zarówno po nazwie obiektu danej klasy jak i samej nazwie klasy.

```
package pl.edu.pjwstk;

public class Main {

    public static void main(String[] args) {
        int i = User.loggedInUsers;
        User u = new User("Stefan");
        int j = u.loggedInUsers;
    }
}
```

Wartości statyczne mogą być „inicjalizowane” bez podania wartości, wówczas przyjmują one wartość domyślną:

- Dla liczb całkowitych to jest 0.
- Dla liczb rzeczywistych to jest 0.0
- Dla boolean jest to false
- Dla obiektów jest to null

Dla przykładu poniższy zapis jest równoznaczny:

```
public static int loggedInUsers = 0;
public static int loggedInUsers;
```

Atrybuty typu final

W javie metody, pola, parametry oraz zmienne możemy oznaczać słowem kluczowym typu final.

W skrócie w przypadku pól, parametrów oraz zmiennych, gdy dodamy słowo final wartość możemy nadać tylko raz i będzie ona ostateczna.

Zmienna typu final

Poniższy kod się nie skompiluje, ponieważ próbujemy nadpisać „finalnego" x wartością 4.

```
public class Main {

    public static void main(String[] args) {
        final int x = 3;
        x = 4;
    }
}
```

Atrybut klasy typu final

Jeśli atrybut klasy jest typu final to musimy ustawić jego wartość w konstruktorze:

```
package pl.edu.pjwstk;

class User {
    private final String name;
    private String email;

    User(String name) {
        this.name = name;
    }

    User(String email) {
        //this.name = "Test"; -- musi się pojawić w każdym konstuktorze ustawienie wartości
name
        this.email = email;
    }
}
```

Każdy z konstruktorów musi ustawiać wartość *name*. Dlatego aby powyższy kod był poprawny konstruktor przyjmujący wartość dla *email* musi ustawiać wartość dla *name*.

Parametr typu final

Jeśli parametr jest typu final to oznacza, że nie możemy go nadpisać w danej metodzie.

```
package pl.edu.pjwstk;

class User {

    private void multiplyAge(final int age, int multiplier) {
        age = age * multiplier;
    }

}
```

W powyższym przykładzie nie możemy napisać wieku, ponieważ jest ono oznaczone słowem kluczowym final.

Pola statyczne – finalne (stałe)
Pole statyczne finale to w Javie inaczej stałe. Stałe muszą być zainicjalizowane od razu jakąś wartością. Stałe są niezmienne w momencie działania całego programu. Nie można zmienić ich wartości oraz są dostępne tak samo jak pola statyczne. W przykładzie z liczeniem pola koła użyliśmy zmiennej statycznej PI:

```
package pl.edu.pjwstk;

public class Calculator {
    private double PI = 3.14;
}
```

PI jest liczbą stałą i jej wartość nie powinna się zmieniać – najbardziej poprawnym zapisem jest wówczas:

```
package pl.edu.pjwstk;

public class Calculator {
    private static double PI = 3.14;
}
```

UWAGA – Dobre praktyki!

1. Nazwy stałych powinny być pisane z WIELKICH_LITER, słowa powinny być oddzielone _.
2. Słowo kluczowe static powinno być po modyfikatorze dostępu (private, protected public, przed określeniem typu)

Zadanie 1 (2 pkt). Utwórz klasę Furniture reprezentującą mebel. Klasa powinna przechowywać nazwę oraz cenę mebla, oraz stawkę procentową VAT która jest taka sama dla każdego mebla. Klasa powinna posiadać konstruktor oraz metody:

- getPrice zwracającą cenę mebla bez vat

- getPrice zwracającą cenę mebla z vat.

Utwórz klasę Room reprezentującą pokój. Klasa powinna przechowywać informacje o powierzchni pokoju oraz jego meblach (w postaci listy). Klasa powinna zawierać konstruktor oraz metodę pozwalającą zwrócić cenę z vat wszystkich mebli znajdujących się w pokoju.

Zadanie 2 (2 pkt). Utwórz klasę House reprezentującą dom. Klasa powinna przechowywać listę pokoi, zawierać prywatny konstruktor, który będzie używany w publicznej statycznej metodzie createHouse, która będzie przyjmowała adres (String) jako parametr i zwracała obiekt domu. Klasa powinna zawierać trzy metody:

- addRoom pozwalającą dodać pokój do domu

- addRome pozwalającą dodać listę pokoi do domu

- getRoomCount zwracającą ilość pokoi w danym domu.

Liczba pokoi w domu powinna być przechowywana jako atrybut statyczny.