WYŻSZA SZKOŁA BANKOWA
Warszawa

Programowanie aplikacji w Java

Maciej Gowin

Zjazd 1 - dzień 2

Linki

Opis

https://maciejgowin.github.io/wsb-java/

Kod źródłowy przykładów oraz zadań

https://github.com/MaciejGowin/wsb-programowanie-aplikacji-java

Język Java: tablice

Tablica jest kolekcją elementów danego typu. Pozwala na przechowywanie większej ilości danych w uporządkowany sposób. Dla zmiennej typ jej jest definiowany przy pomocy składni `type[] variable`. Dla przykładu tablica przechowująca typ `int`:

```
int[] arrayOfInt;
```

Powyższe wyrażenie definiuje tylko zmienną bez przypisania wartości. Kolejnym krokiem jest stworzenie obiektu przez inicjalizację zmiennej. Podczas inicjalizacji tablicy musimy zdefiniować jej rozmiar, związane jest to z alokacją przestrzeni w pamięci.

```
int[] arrayIfInt = new int[10];
```

O obiektach oraz inicjalizacji będziemy mówić w kolejnych rozdziałach.

Programowanie: przykład 04

Tablica po zainicjalizowaniu zawiera wartości domyślne. W przykładzie użyliśmy pętli `for...each`, która pozwala na sprawdzenie wszystkich elementów tablicy (tzw. iteracja). Pętla ta będzie omawiana w szczegółach w kolejnych rozdziałach.

Programowanie: przykład 04

```
public class Main {

    public static void main(String[] args) {
        int[] intItems = new int[2];
        for (int intItem: intItems) {
            System.out.println(intItem);
        }

        double[] doubleItems = new double[2];
        for (double doubleItem: doubleItems) {
            System.out.println(doubleItem);
        }

        String[] stringItems = new String[2];
        for (String stringItem: stringItems) {
            System.out.println(stringItem);
        }
    }
}
```

Język Java: tablice

Następnym etapem jest przypisanie wartości do kolejnych elementów tablicy. Każdy element tabeli ma przyporządkowany numer szeregowy, który nazywamy `indeksem`. Przypisanie i odczyt odbywa się analogicznie przy użyciu indeksu danego elementu:

```
int[] intItems = new int[2];
intItems[0] = 2;
intItems[1] = 4;

System.out.println(intItems[0]);
System.out.println(intItems[1]);
```

Język Java: tablice

> Indeksy tabel są dość wyjątkowe. Pierwszy element tabeli znajduje się pod indeksem 0.

Jeżeli pierwszym indeksem jest 0, możemy wywnioskować, że dla tabeli o rozmiarze 10 (uogólniając `n`):

- pierwszym indeksem będzie 0
- ostatnim indeksem będzie 9 (uogólniając `n - 1`)

> W przypadku próby dostępu do nieistniejącego indeksu tablicy (np. n + 1) zostanie zgłoszony wyjątek `ArrayIndexOutOfBoundsException`.

Język Java: tablice

Możemy też zainicjalizować tablicę wraz z przypisaniem wartości podczas deklaracji. Poniższe formy są równoznaczne:

```
int[] intItems = {1, 2, 3};
int[] intItems = new int[] {1, 2, 3};
```

W przypadku inicjalizacji podczas deklaracji rozmiar tablicy nie jest definiowany i zostaje automatycznie obliczony przez kompilator. Rozmiar możemy pobrać, używając właściwości (property) tabeli: `length` :

```
int[] intItems = {1, 2, 3}; int length = intItems.length;
```

Programowanie: zadanie 06

Zdefiniuj tabelę z 10 elementami składającą się z liczb od 2 do 20. Używając:

- pętli for...each
- operatorów arytmetycznych + oraz /
- długości tabeli
  oblicz średnią wartość wszystkich elementów w tabeli. Wynik wypisz na ekran.

Język Java: tablice

Oczywiście elementami tablic mogą być też tablice. W taki sposób powstają tablice wielowymiarowe.

```
int[][] items = { {11}, {21, 22}, {31, 32, 33} };
```

Bezpośredni dostęp do elementów jest analogiczny do tego znanego z tablic jednoelementowy.

```
int[] subItems = items[1];
int subItem = items[1][1];
```

Programowanie: przykład 05

Tablica wielowymiarowa oraz sposób iteracji po jej elementach z zagnieżdżoną pętlą `for...each`.

```
public class Main {

    public static void main(String[] args) {
        int[][] items = { {11}, {21, 22}, {31, 32, 33} };

        for (int[] subItems: items) {
            for (int subItem: subItems) {
                System.out.println("item: " + subItem);
            }
        }
    }
}
```

Język Java: metody

Do czynienia z metodą mieliśmy już przy okazji każdego programu zawierającego definicję `main`. Ma ona specjalne znaczenie, ponieważ jest ona domyślną metodą wywoływaną podczas uruchamiania programu.

Metodę możemy opisać jako blok kodu zawierający wyrażenia i czasami zwracający wartość. Główną ich zaletą jest podział kodu na mniejsze części wykonujące konkretną funkcjonalność. Tak wydzielona funkcjonalność może później zostać u żyta wielokrotnie.

Język Java: metody

Aby lepiej zrozumieć działanie metody, wyobraźmy sobie blok kodu pozwalający na obliczenie pola trójkąta:

```
double triangleField(double a, double h) {
    return a * h / 2;
}
```

Metoda składa się z następujących części:

- nazwy, w przykładzie: `triangleField`
- parametrów, w przykładzie: `a` oraz `h`
- typu wartości zwracanej, w przykładzie: `double`

Język Java: metody

Uogólniając:

```
typWartościZwracanej nazwaMetody(parametry) {
    return wartośćZwracana
}
```

Programowanie: przykład 06

Definicja oraz wywołanie metody obliczającej pole trójkąta.

```
public class Main {

    public static void main(String[] args) {
        double triangle1 = triangleField(2.0, 3.0);
        System.out.println("Triangle 1: " + triangle1);
        System.out.println("Triangle 2: " + triangleField(4.0, 6.0));
    }

    public static double triangleField(double a, double h) {
        return a * h / 2;
    }
}
```

Język Java: metody

Wywołanie metody polega na użyciu jej nazwy oraz podaniu parametrów. Wartość zwracana może zostać przypisana lub używa bezpośrednio. Słowa kluczowe `public` oraz `static` zostaną omówione w dalszej części.

Metody nie zawsze zwracają wartości oraz nie muszą też przyjmować żadnych parametrów. Jeżeli metoda nie zwraca wartości podczas definicji typ zwracany zastępujemy słowem kluczowym `void`. Przy braku parametrów po prostu je pomijamy.

Programowanie: przykład 07

Porównanie metod z parametrami i wartościami zwracanymi.

```
public class Main {

    public static void main(String[] args) {
        parametersReturns(2);
        parametersNoReturn(2);
        noneParametersReturns();
        noneParametersNoneReturn();
    }

    public static int parametersReturns(int parameter) {
        int result = parameter * 10;
        System.out.printf("parameters: %s, returns: %s\n", parameter, result);
        return result;
    }

    public static void parametersNoReturn(int parameter) {
        System.out.printf("parameters: %s, returns: %s\n", parameter, "none");
    }

    public static int noneParametersReturns() {
        int result = 1;
        System.out.printf("parameters: %s, returns: %s\n", "none", result);
        return result;
    }

    public static void noneParametersNoneReturn() {
        System.out.printf("parameters: %s, returns: %s\n", "none", "none");
    }
}
```

Język Java: metody

Do tej pory definiowaliśmy własne metody, ale też wywoływaliśmy metody wbudowane w język Java i dostarczane wraz z JRE jako biblioteki standardowe. Przykładem takich metod są: `System.out.println`, `System.out.printf` oraz `System.out.print`. Kolejne będziemy pozwać wraz z przykładami. Istotą programowania jest umiejętne poruszanie się po bibliotekach oraz metodach, które realizują pewne funkcjonalności oraz rozwiązują dane problemy.

Programowanie: zadanie 07

Stwórz program służący do konwersji walut z jedną metodą pozwalającą na konwersje EUR do PLN.

Język Java: klasa i obiekt

Java jest językiem obiektowym. Możemy z tego wywnioskować, że obiekty odgrywają w nim znaczącą rolę. Obiekt to byt posiadający stan oraz zachowanie. Możemy to odnieść do świata rzeczywistego na przykładzie samochodu:

- ma stan: jest uruchomiony, stoi, jest czerwony, jest marki Opel
- ma zachowanie: jechanie, hamowanie, parkowanie

Obiekty tworzymy na podstawie definicji klasy. Klasa jest niejako prototypem opisującym, jak dany obiekt może wyglądać oraz jakie zachowania są z nim związane. Dla jednej definicji klasy może istnieć nieskończenie wiele jej instancji, czyli obiektów.

Język Java: klasa i obiekt

Klasę definiujemy używając słowa kluczowego `class` wraz z jej:

- polami (odpowiedzialnymi za stan) oraz
- metodami (odpowiedzialnymi za zachowanie)

```
class NazwaKlasy {
    // Pola
    // Metody
}
```

Język Java: klasa i obiekt

Korzystając z przykładu samochodu:

```
class Car {
    String brand = "Unknown";

    void startEngine() {
        System.out.println("Starting engine");
    }
}
```

Zdefiniowana klasa ustala również domyślną markę każdego nowo utworzonego samochodu. Tworzenie obiektów danej klasy odbywa się przy pomocy słowa kluczowego `new` oraz z udziałem konstruktora klasy. Tak stworzona wartość jest przypisywana do zmiennej opisanej nazwą klasy, podobnie jak to miało miejsce podczas definicji zmiennych dla typów prymitywnych.

Język Java: klasa i obiekt

Obiekty nazywamy również instancjami danej klasy.

```
Car mercedes = new Car();
Car bmw = new Car();
```

Przy tak utworzonym obiekcie możemy się odwołać do jego pól i metod.

```
Car mercedes = new Car();
String brand = mercedes.brand;
mercedes.startEngine();
```

Jak już wiemy, metody mogą przyjmować parametry oraz zwracać wartość. Uzyskujemy do nich dostęp poprzez `.` oraz zmienną, do której przypisany jest obiekt. Metody klasy operują też na polach oraz mogąc zmieniać ich wartość.

Język Java: null

Do zdefiniowanego zmiennej danej klasy możemy przypisać instancję lub też literał `null`. Możemy go rozumieć jako brak przypisanej wartości, coż co nie istnieje. Odwołanie się do zmiennej, do której przypisany jest `null` będzie skutkowało błędem.

```
Car car1; // will not compaline as not defined
Car car2 = null; // works fine
Car car3 = new Car(); // works fine
car2.run(); // throws NullPointerException
car3.run(); // works fine
```

Język Java: null

> Przypisanie `null` do zmiennej nie jest tym samym co zadeklarowanie zmiennej przez przypisania w obrębie funkcji. W obrębie własności klasy wszystkie wartości są automatycznie zainicjalizowane przy użyciu `null`.

Programowanie: przykład 08

```
class Car {
    String brand = "Unknown";
    boolean engineStarted = false;

    void startEngine() {
        engineStarted = true;
    }

    void stopEngine() {
        engineStarted = false;
    }
}

public class Main {

    public static void main(String[] args) {
        Car unknown = new Car();

        Car bmw = new Car();
        bmw.brand = "BMW";
        bmw.startEngine();

        Car mercedes = new Car();
        mercedes.brand = "Mercedes";
        mercedes.startEngine();
        mercedes.stopEngine();

        System.out.print("\nSummary:\n");
        System.out.printf("Car %s with engine %s\n", unknown.brand, onOrOff(unknown.engineStarted));
        System.out.printf("Car %s with engine %s\n", bmw.brand, onOrOff(bmw.engineStarted));
        System.out.printf("Car %s with engine %s\n", mercedes.brand, onOrOff(mercedes.engineStarted));
    }

    public static String onOrOff(boolean isStarted) {
        if (isStarted) {
            return "on";
        } else {
            return "off";
        }
    }
}
```

Język Java: metody i pola statyczne

Metody statyczne

W poprzednich rozdziałach mówiliśmy o metodach. Każda z nich zdefiniowana była w kontekście klasy. Możemy jednak zauważyć małą różnicę w ich definicji w porównaniu z metodami zdefiniowanymi w kolejnych przykładach:

```
    public static String onOrOff(boolean isStarted) {
        if (isStarted) {
            return "on";
        } else {
            return "off";
        }
    }
```

```
    void stopEngine() {
        engineStarted = false;
    }
```

Język Java: metody i pola statyczne

Metody statyczne

Mowa o słowie kluczowym `static`. Jaka jest zatem różnica pomiędzy nimi? Każda metoda zdefiniowana ze słowem kluczowym `static` może zostać wywołana bez konieczności stworzenia obiektu. Innymi słowy, działa ona w kontekście klasy. Co za tym idzie, z poziomu metody statycznej nie mamy dostępu do pól obiektów. Wykorzystywane są one najczęściej w kontekście klas narzędziowych (ang. utility classes).

Język Java: metody i pola statyczne

Przykładem klasy narzędziowej może być wbudowana klasa `Math`. Definiuje metody statyczne, do których mamy dostęp z poziomu klasy.

```
double sqrtOf2 = Math.sqrt(2);
int max = Math.max(1, 2);
```

Oczywiście metodę statyczną możemy też wywołać z poziomu obiektu. Jednak nie jest możliwe wywołanie metody niestatycznej z poziomu klasy.

Programowanie: zadanie 08

Stwórz klasę `Car`, która zawiera definicję marki samochodu w polu `brand`. Stwórz klasę `CarCalculator` posiadającą statyczną metodę `countBrand` pozwalającą na zliczenie samochodów danej marki na podstawie 2 parametrów: `brand` oraz `cars`. W metodzie głównej `main` zainicjalizuj kilka obiektów samochodu oraz dokonaj obliczeń ilości samochodów danej marki. Wyniki obliczeń wypisz na ekran.

Przetestuj użycie metody `countBrand` przy użyciu wywołania metody z poziomu klasy `CarCalculator` oraz jej instancji.

Język Java: metody i pola statyczne

Pola statyczne

W analogiczny sposób do metod statycznych możemy zdefiniować pola statyczne. Również ich znaczenie jest podobne do metod statycznych. Odwołują się one do klasy oraz są wspólne dla wszystkich obiektów.

```
class Car {
    static int speedLimit = 180;
    String brand = "Mercedes";
}
```

Jako że pole statyczne jest zdefiniowane na poziomie klasy, jakakolwiek zmiana jego wartości będzie widoczna w każdym obiekcie.

Programowanie: przykład 09

Definicja pól statycznych oraz dostęp z poziomu klasy i obiektu.

```
class Car {
    static int speedLimit = 180;
}

public class Main {

    public static void main(String[] args) {
        Car bmw = new Car();
        Car mercedes = new Car();

        System.out.printf("Car speed limit: %s\n", Car.speedLimit);
        System.out.printf("Car speed limit: %s\n", bmw.speedLimit);
        System.out.printf("Car speed limit: %s\n", mercedes.speedLimit);

        Car.speedLimit = 190;

        System.out.printf("Car speed limit: %s\n", Car.speedLimit);
        System.out.printf("Car speed limit: %s\n", bmw.speedLimit);
        System.out.printf("Car speed limit: %s\n", mercedes.speedLimit);


        mercedes.speedLimit = 200;

        System.out.printf("Car speed limit: %s\n", Car.speedLimit);
        System.out.printf("Car speed limit: %s\n", bmw.speedLimit);
        System.out.printf("Car speed limit: %s\n", mercedes.speedLimit);
    }
}
```

Język Java: przeciążanie metod

Przeciążanie metod (ang. method overloading) to nic innego jak używanie tej samej nazwy dla kilku metod w obrębie jednej klasy. Jest ono możliwe, jeżeli:

- każda z tych metod ma różną liczbę parametrów
- typy parametrów są różne
- te zachodzą dwa powyższe warunki.

Dla przykładu wszystkie z poniższych są metodami przeciążonymi:

```
int someFunction(int i) { ... }
double someFunction(double d) { ... }
double someFunction(double d1, double d2) { ... }
```

Język Java: przeciążanie metod

Co istotne typ zwracany nie wpływa na przeciążanie, dlatego też poniższe nie jest możliwe i spowoduje błąd kompilacji:

```
int someFunction(int i) { ... }
double someFunction(int i) { ... }
```

Przeciążanie metod jest istotne z punktu widzenia czytelności kodu. Wyobraźmy sobie metodę sumującą liczby. Gdybyśmy musieli stworzyć osobną nazwę dla każdej z funkcji, mogłoby to wyglądać następująco:

```
int sumOf2Ints(int a, int b) { ... }
int sumOf3Ints(int a, int b, int c) { ... }
double sumOf2Doubles(double a, double b) { ... }
```

Język Java: przeciążanie metod

Oczywiste jest to, że funkcje te mają podobne działanie, jednak ich nazwy stają się nieczytelne. Ujednolicenie nazwy pozwoli na lepsze ich zrozumienie, czytelność kodu, jak i łatwiejsze użycie przez programistę, który będzie z nich w przyszłości korzystał.

Programowanie: przykład 10

Przeciążanie metod na podstawie kalkulatora obliczającego sumę liczb.

```
class Calculator {

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

    static int sum(int a, int b, int c) {
        return sum(a, b) + c;
    }

    static double sum(double a, double b) {
        return a + b;
    }
}

public class Main {

    public static void main(String[] args) {

        System.out.println(Calculator.sum(1, 2));
        System.out.println(Calculator.sum(1, 2, 3));
        System.out.println(Calculator.sum(1.0, 2.0));
    }
}
```

Język Java: podział programu

Do tej pory wszystkie klasy definiowaliśmy w jednym pliku. Nie jest to problematyczne przy prostych programach. Jednak umieszczenie większej ilości klas w jednym pliku sprawiłoby, że kod byłby nieczytelny. Pierwszym etapem podziały programu jest zdefiniowanie klas w osobnych plikach. Należy pamiętać, że każdy z plików musi zawierać przynajmniej jedną klasę publiczną (modyfikator dostępu, o którym mowa w dalszej części) tożsamą z nazwą pliku.

Język Java: podział programu

> Car.java

```
public class Car {
}
```

> Main.java

```
public class Main {

    public static void main(String[] args) {
        Car car = new Car();
    }
}
```

W przykładzie klasa `Main` używa klasy `Car` zdefiniowanej w innym pliku. Jest to możliwe, gdyż pliki te są umieszczone w tym samym katalogu. Podczas kompilacji (JDK) oraz uruchomienia programu (JRE) procesy używa obecnego katalogu (`.`) do wyszukiwania definicji klas.

Język Java: pakiety

Przy bardziej skompilowanych programach przydatne jest grupowanie klas w `pakiety`.
Tak jak klasom odpowiadają pliki, tak za podział na pakiety odpowiedzialne są katalogi.
Nazwa pakietu wynika z nazw katalogów, w jakich znajdują się pliki.

```
Ścieżka katalogów: {root}/my/first/package
Pakiet: my.first.package
```

Język Java: pakiety

Głównymi zaletami używania pakietów są:

- lepsza czytelność kodu
- unikanie kolizji nazewniczych (szczególnie istotne przy definiowaniu bibliotek)

Z kolizją mamy do czynienia w przypadki dwóch klas o tych samych nazwach. Dzięki pakietom klasy te są zdefiniowane w plikach o różnej lokalizacji, dzięki czemu może się do nich odnosić bezpośrednio przy użyciu pełnej nazwy.

Programowanie: przykład 11

> vehicles/Car.java

```
package vehicles;

public class Car {
}
```

> vehicles/Bus.java

```
package vehicles;

public class Bus {
}
```

Programowanie: przykład 11

> Main.java

```
import vehicles.Car;

public class Main {

    public static void main(String[] args) {
        Car car = new Car();
        vehicles.Bus bus = new vehicles.Bus();
    }
}
```

Język Java: pakiety

Każda z klas, która zlokalizowana jest w danym katalogu musi posiadać definicję `package` o nazwie równoznacznej z nazwą katalogu. Jeżeli natomiast chcemy użyć użyć klasy z innego pakietu musimy go załadować poprzez instrukcję `import` lub użyć jej pełnej ścieżki dostępowej. W poprzednich przykładach wszystkie z plików znajdowały się z tym samym katalogu (pakiet domyślny) lub też w tym samym pliku. Klasy z tego samego pakietu nie importujemy explicite.

Importowanie klas jest istotne z perspektywy kompilatora oraz środowiska uruchomieniowego. Dzięki temu narzędzia te otrzymują informacje, w jakim miejscu dana klasa jest zlokalizowana i pozwala na rozszerzenie przestrzeni wyszukiwań.

Jeżeli chcemy zaimportować wszystkie klasy zdanego pakietu, możemy użyć konstrukcji:

```
import vehicles.*;
```

Język Java: pakiety

Język Java dostarcza biblioteki zawierające wbudowane pakiety i klasy. Mogą one być użyte w naszym kodzie i też podlegają procesowi importowania. Wyjątkiem są klasy z pakietu `java.lang`, które są używane z pominięciem importowania. Bibliotekom poświęcony będzie osobny z rozdziałów.

```
import java.time.LocalDate;
import java.util.ArrayList;
```

Programowanie: zadanie 09

Zmodyfikuj program z zadania 08, dzieląc go na osobne pliki, dla każdej z klas oraz wprowadzając pakiet `cars` dla klas `Car` oraz `CarCalculator`.