Java – Pierwsze zajęcia

Anna Gogolińska

Konwencja nazewnictwa

- Nazwenictwo CamelCase:
  - Klasy: pierwsze słowo z dużej litery, kolejne słowa z dużej litery, pisane łącznie, np. FirstClass, MyFileReader, Observer
  - Metody: pierwsze słowo z małej litery, dalej tak samo jak klasy, np. run(), processText()
  - Zmienne: tak samo jak metody, np. a, b, prevHeight
  - Stałe: dużymi literami, słowa łączone podreśleniem, np. MAX_VALUE
- Zalecane nazewnictwo w języku angielskim.
- Nazwy powinny opisywać co robi dana klasa/metoda/zmienna (zazwyczaj).

Pierwszy program

- W notatniku stworzyć pierwszy program i uruchomić go z wiersza poleceń:

```
public class MyFirstClass {

    public static void main(String[] args) {

        System.out.println("Witaj swiecie");
    }

}
```

Argumenty wiersza poleceń

- Każdy argument to *String*, *args* to tablica *Stringów*.
- *args.length* – długość tablicy
- *args[i]* – i-ty element
- Argumenty podaje się w linii poleceń lub w specjalnym polu jeśli używamy IDE.

Podstawy

- Typy podstawowe jak w C++: int, char, byte, long, short, float, double, boolean.
- Operatory: jak w C++.
- Dla każdego typu podstawowego istnieje klasa „opakowująca”, np. Integer, Double, ... – można jej używać na przykład do konwersji.
  - *Integer.parseInt(s)* – zwraca *int* na podstawie *s*
  - *Double.parseDouble(s)* - zwraca *double* na podstawie *s*
- Łańcuchy znaków: String.

Podstawowe instrukcje

- Tworzenie tablic:
  - *Typ[] nazwa = new Typ[wielość]*:
    - *int[] tab = new int[n];*
    - *int[] tab2 = wyrażenie_zwracające_tablicę;*
- Instrukcje warunkowe i pętle – takie jak w C++;
- Pętla *for ... each:*
  - Bardziej optymalna, ale nie można zmieniać elementów w kolekcji po której iterujemy (np. w tablicy)
  - Składnia: *for(typ zmienna : kolekcja_danych)*:

```
for(int a : tab) {
        s = s + a;
        System.out.println(a);
}
```

Obsługa we/wy

- Wypisywanie na ekran:
  - *System.out.println(a);* - wypisuje z przejściem do nowej linii
  - *System.out.print("jakis tekst");* - wypisuje bez przejścia do nowej linii
- Wczytywanie – Scanner posiada więcej metod (sprawdź domunetację):

```
Scanner odczyt = new Scanner(System.in);
while(s.hasNext() ) {
        String s = odczyt.nextLine();
}
```

Parę słów o Stringach

- Konkatencja napisów: *+*
- Każdy obiekt ma metodę *toString()* która zwraca jego tekstową reprezentację.
- Łączenie tekstu i zmiannych poprzez *+* (konkatenację).
- Łączenie większej ilości napisów/zmiennych w dłuższy napis przy użyciu *+* jest niewydajne.
- Użycie *StringBuilder*
  - Metoda *append()* – dodaje tekst do tworzonego napisu
  - Metoda *toString()* – zwraca stworzony napis

```
public class MyFirstClass {

    public static void main(String[] args) {
        String a = "tekst1 tekst2";
        int c = 10;
        String e = a + c;
        String f = "" + c;
        System.out.println(e);
        System.out.println(a + c);
        StringBuilder sb = new StringBuilder();
        sb.append(a);
        sb.append(" jakis tekst ");
        sb.append(c);
        String s = sb.toString();
        System.out.println(sb.toString());
    }

}
```

Klasy

- Wskazane jest, aby każda klasa znajdowała się w osobnym pliku (są od tego wyjątki).
- Tworzymy jedną klasę, która nazywa się jak projekt i która zawiera metodę *main()* – w IDE może ona zostać stworzona automatycznie.
- Pozostałe klasy tworzymy według schematu opisanego na kolejnych slajdach.

Klasy

- Wszystkie pola ustawiamy jako *private/protected*.
- Dla pól tworzymy metody, które umożliwią jego modyfikację i pobieranie wartości (tzw. settery i gettery, IDE tworzą je automatycznie).
- Tworzymy konstruktor/konstruktory (w zależności od potrzeb, IDE tworzą je automatycznie):
  - z pustą listą argumentów (tzw. pusty konkstuktor) – może on zawierać przypisanie polom zerowych/pustych wartości bądź być pusty.
  - konstruktor z argumentami, w którym przypisujemy polom wartości z argumentów.

```
public class TestClass {
   private int firstValue;
   private boolean boolValue;

   public TestClass() {
        this.firstValue = 0;
        boolValue = false;
   }

   public void setFirstValue(int firstValue) {
        this.firstValue = firstValue;
   }

   public int getFirstValue() {
        return firstValue;
   }

   public void setBoolValue(boolean boolValue) {
        this. boolValue = boolValue;
   }

   public boolean isBoolValue() {
        return boolValue;
   }

}
```

Przysłanianie i przeciążanie

- Jeśli pola i argumenty/zmienne nazywają się tak samo to argument/zmienna przysłania pole (używając ich nazwy odwołujemy się do argumentu/zmiennej, a nie pola). Aby wskazać, że chodzi nam o pole używamy słowa kluczowego *this*. ***this*** reprezentuje aktualny obiekt.
- Kilka metod może mieć te same nazwy, ale muszą mieć różne argumenty (typ i ilość). Na podstawie dopasowania typów i ilości argumentów wywoływana jest odpowiednia metoda – przeciążanie.

Dziedziczenie

- Słowo ***extends*** w definicji klasy (po nazwie klasy). Można dziedziczyć tylko z jednej klasy***.***
- Zasady dziedziczenia – podobnie jak w C++.
- W klasie potomnej można się odwołać do klasy nadrzędnej poprzez słowo ***super,*** np. super() – wywoła bezargumentowego konstruktora z klasy bazowej – używane często w konstruktorach klasy potomnej, super.method() – wywyła metodę z klasy bazowej.