Podstawy Java SE

Hello!

Tomasz Lisowski

Software developer, JIT Solutions
IT trainer

Agenda

- wprowadzenie
- typy danych
- instrukcje sterujące
- równość obiektów
- kolekcje
- modyfikatory dostępu
- interfejsy
- wyjątki
- strumienie
- wątki

Java SE
Wprowadzenie

- aktualna wersja to Java8
- aplikacje pisane w Javie są kompilowane do bytecodu, a następnie uruchamiane na maszynie wirtualnej (JVM)
- Java jest multiplatformowa
- Język obiektowy
- Automatyczne zarządzanie pamięcią

Java SE
Wprowadzenie

Dystrubycje javy:

- **Java SE** – Standard Edition
  - podstawowa dystrybucja
- **Java EE** – Enterprise Edition
  - do wytwarzania aplikacji webowych
- **Java ME** – Micro Edition
  - aplikacje na urządzenia mobilne (coraz mniej popularna)

Java SE
Wprowadzenie

- **JVM** – wirtualna maszyna javy
  - “procesor” wykonujący skompilowany kod Javy
- **JRE** – Java Runtime Environment
  - zawiera JVM oraz klasy niezbędne do uruchomienia programów Java
- **JDK** – Java Development Kit
  - zawiera JRE oraz narzędzia do implementacji I kompilacji

Java SE
Klasa

- typ danych
- konkretna definicja
- .. zawierająca pola
- .. i metody

Java SE
Klasa

WŁAŚCIWOŚCI/CECHY/DANE:

- marka
- model
- numer
- kontakty
- kolor
- ...

CZYNNOŚCI:

- dzwonienie
- odbieranie połączeń
- wysyłanie SMS
- odbieranie SMS
- wyszukiwanie kontaktów
- dodawanie kontaktów
- ...

POLA:

- marka
- model
- numer
- kontakty
- kolor
- ...

METODY:

- zadzwon()
- odbierzPołączenie()
- wyslijSMS()
- odbierzSMS()
- wyszukajKontakt()
- dodajKontaktów()
- ...

Java SE
Obiekt

- instancja klasy
- konkretny obiekt
  na podstawie definicji klasy

```
class Samochod
{
    public string marka;
    public int kolor;
    public int przebieg;
}
```

```
Samochod auto = new Samochod();
```

Java SE
Ćwiczenie

- stwórz klasę o nazwie Menu
- dodaj 2 pola
  - *int number;*
  - *String text;*
- w klasie Main stwórz obiekt typu Menu
- wypisz wartości pól *number* oraz *text*

Java SE
Konstruktor

- metoda tworząca obiekt
- domyślny konstruktor
- konstruktor parametrowy
- słowo kluczowe *this*
  - zwraca aktualny obiekt

```
public Samochod(string marka)
{
    Random rand = new Random();
    kolor = rand.Next(); //Lakierujemy na losowy kolor
    przebieg = 0; //Dopiero wyjechaliśmy z fabryki
    this.marka = marka; //Ustawiamy markę naszego samochodu na podaną jako parametr
}
```

Java SE
Ćwiczenie

- stwórz parametrowy konstruktor klasy Menu
- niech przyjmuje 2 parametry tego typu co pola klasy
- przypisz wartości dla pól
- wypisz wartości pól *number* oraz *text* dla klasy stworzonej poprzez parametryzowany konstruktor

Java SE
Metody

- funkcje, które dana klasa udostępnia
- prywatne lub publiczne
- mogą posiadać parametry

```
public class Tablice{
    public void metoda1(){
        System.out.println("Ta metoda nic nie zwraca");
    }

    public int trzy(){
        return 3; // ta metoda zwraca liczbę 3
    }

    public int suma(int a, int b){
        return a+b;  //ta metoda zwraca sumę dwóch parametrów
    }
}
```

Java SE
Ćwiczenie

- stwórz 2 metody w klasie Menu:
  - jedna wypisująca na konsolę wartość *“* number – text *”*
  - druga powinna wypisać typ pola *text* (wykorzystanie metody getClass())

Java SE
Zasięg

- pola klasy dla wszystkich metod
- ..lub na zewnątrz
- pola w metodzie widoczne tylko w niej
- tworzony obiekt wewnątrz metody “żyje” tylko w niej

Java SE
OOP

- **polimorfizm**
  - “samochód jest pojazdem”
  - dany typ, może rozszerzać inny, a obiekt udostępniać metody obydwu typów

```
public class Main {
    public static void main(String[] args) {
        String str1 = new String("Siabada1");
        Object str2 = new String("Siabada2");
        System.out.println(str1);
        System.out.println(str2);
    }
}
```

- **dziedziczenie**
  - współdzielenie funkcjonalności między klasami
  - oprócz własnych atrybutów posiada te pochodzące z klasy nadrzędnej/bazowej

Java SE
OOP

- **abstrakcja**
  - obiekt jako model “wykonawcy”
  - wykonanie pracy, bez ujawniania implementacji

  np. pojazd.jedz()

- **hermetyzacja**
  - ukrywanie implementacji przez obiekt
  - ukrywanie pewnych składowych (pól, metod) tak, aby były dostępne tylko metodom wewnętrznym klasy
  - “wszystkie pola są prywatne”

Typy danych

Java SE
Co to są typy?

- *wszystko jest obiektem*
- typy tekstowe, liczbowe, zmiennoprzecinkowe
- daty
- każda klasa jest typem

Java SE
Typy proste

- typy proste nie są instancjami obiektów
- reprezentują podstawowe typy

```
int liczbaCalkowita;
long duzaLiczbaCalkowita;
double liczbaZmiennoPrzecinkowa; //64bit
float kolejnaLiczbaZmiennoPrzecinkowa; //32bit
boolean prawdaFalsz;
char znak;
```

Java SE
Typy obiektowe

- konkretne instancje
- możemy tworzyć swoje typy
- mogą mieć dowolne zachowanie (metody)

```
Integer liczbaCalkowita;
Long duzaLiczbaCalkowita;
Double liczbaZmiennoPrzecinkowa;
Float kolejnaLiczbaZmiennoPrzecinkowa;
Boolean prawdaFalsz;
String napis;
```

Java SE
Typy wyliczeniowe

- enum
- konkretny (ograniczony) zbiór możliwych wartości

```
private enum Marka {
    MERCEDES,
    AUDI,
    OPEL
}
```

```
Marka mojSamochod = Marka.AUDI;

if (mojSamochod.equals(Marka.MERCEDES)) {
    return false;
}
```

Java SE
Ćwiczenie

- stwórz własny typ wyliczeniowy
- dodaj pole tego typu do klasy Menu
- wypisz domyślną wartość pola

Java SE
Rzutowanie

- zmiana typu danych na inny
- np. dzielenie dwóch liczb całkowitych

```
int liczbaA = 10;
int liczbaB = 3;

//wynik nie jest liczbą całkowitą

double wynik = liczbaA/liczbaB;
```

Instrukcje sterujące

Java SE
if else

- podstawowa operacja – instrukcja wyboru
- if = jeżeli
- jeżeli warunek jest spełniony, to wykonaj instrukcje

```
double wynik = liczbaA/liczbaB;

if (wynik > 0) {
    return "Liczba dodatnia";
}
```

Java SE
if else

- warunek *if* można łączyć z *else*
- *else* wykonywane gdy pierwszy warunek nie jest spełniony
- można zagnieżdżać i łączyć instrukcje *if* - *else*

```
if (wynik > 0) {
    return "Liczba dodatnia";
}
else if (wynik == 0) {
    return "Liczba 0";
}
else {
    return "Liczba ujemna";
}
```

Java SE
Ćwiczenie

- stwórz obiekt menu1 z wartością *number* = 1
- stwórz obiekt menu2 z wartością *number* = 2
- stwórz warunek, przypisujący enuma zależnie od wartości *number*
- wykonaj warunek dla obydwu obiektów
- wypisz wartość enuma dla obydwu obiektów

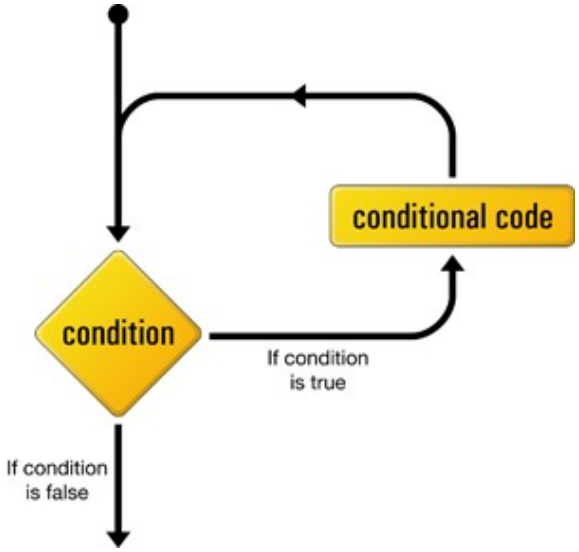
Java SE
switch

- “wielowarunkowy if”
- switch pobiera parametr i sprawdza dowolną liczbę warunków

```
switch(liczba){
  case 1:
    jakieś_instrukcje_1;
    break;
  case 2:
    jakieś_instrukcje_2;
    break;
  ...
  default:
     instrukcje, gdy nie znaleziono żadnego pasującego przypadku
}
```

Java SE
Pętle

- podstawowa operacja – cykliczne wykonanie danych instrukcji
- niewiadoma ilość wykonań
- .. lub ściśle określona
- można przerwać lub pominąć dany obieg



Java SE
pętla while

- wykorzystywana, gdy nie znamy ilość obiegów pętli
- .. ale znamy warunek jej zakończenia
- pętla *while* może wykonać się nieskończenie wiele razy
- albo wcale, gdy warunek już na starcie nie jest spełniony

```
int liczba = -5;
while(liczba < 0) {
    liczba++; //liczba = liczba + 1;
}
```

Java SE
pętla do..while

- rozbudowana wersja pętli *while*
- pętla *do..while* zawsze wykona się conajmniej jeden raz

```
int liczba = 5;
do {
    liczba++; //liczba = liczba + 1;
} while(liczba < 0);
```

Java SE
pętla for

- zazwyczaj znamy liczbę iteracji w pętli
- 3 parametry:
  - wyrażenie początkowe → *int i = 0*
  - warunek → *i < 5*
  - modyfikator → *i++*

```
for (int i = 0; i < 5; i++) {
    System.out.println("i: " + i);
}
```

Java SE
break – continue

- instrukcje manipulujące działaniem pętli
- *break*
  → przerwanie pętli
- *continue*
  → ominięcie danej iteracji

```
int liczba = -5;
while(liczba < 0) {
    if (liczba == 2) {
        continue;
    }

    if (liczba == 3) {
        break;
    }

    liczba++; //liczba = liczba + 1;
}
```

Java SE
Pętle

```
//Grab odd numbers from array
for(int i=0;i<Array.Length;i++)
{
if(i == 1){
Console.Write(i);
}
if(i == 3){
Console.Write(i);
}
if(i == 5){
Console.Write(i);
}
if(i == 7){
Console.Write(i);
}
if(i == 9){
Console.Write(i);
}
if(i == 11){
Console.Write(i);
}
if(i == 13){
Console.Write(i);
}
if(i == 15){
Console.Write(i);
}
}
```

Pobranie danych z klawiatury

Java SE
Scanner

- podstawowe pobranie danych od użytkownika
- obiekt korzysta ze strumienia wejściowego:
  Scanner scanner = new Scanner(System.in);
- popularne metody:
  - nextLine()
  - nextInt()
  - nextDouble()

Java SE
Ćwiczenie

- stwórz dwie zmienne typu Double
- pobierz je za pomocą klasy Scanner
- dodaj je do siebie
- wyświetl wynik

Java SE
Ćwiczenie

- wyświetl informację o dostępnych opcjach (enum)
- pobierz opcję z klawiatury
- ustaw odpowiednią wartość enum w zależności od podanej liczby
- jeżeli błędna wartość to wyświetl informację i spróbuj pobrać ponownie
- podanie wartości 0 przerywa działanie

Czy obiekty są równe?

Java SE
== vs equals

- instrukcje porównania
- == porównuje **referencję** (przestrzeń pamięci)
- *equals()* porównuje **wartość** dwóch pól

```
String tekstA = "tekst";
String tekstB = "tekst";

if (tekstA == tekstB) {
    System.out.println("warunek == prawdziwy");
}

if (tekstA.equals(tekstB)) {
    System.out.println("warunek equals prawdziwy");
}
```

Java SE
== vs equals

- equals() to metoda klasy Object
- wykorzystuje hashCode obiektu

- *jeśli obiekty są równe to muszą mieć ten sam hashCode*
- *jeśli obiekty mają ten sam hashCode to nie muszą być równe*

- nadpisanie metody hashCode()
- kontrakt hashCode() ↔ equals()

Java SE
Ćwiczenie

- stwórz 2 stringi o takiej samej wartości
- porównaj je za pomocą instrukcji *if* i operatorów:
  - ==
  - equals()
- wypisz ich hashCode

Podstawowe kolekcje

Java SE
set

- elementy nie mają przyporządkowanego indeksu
- dostęp za pomocą iteratora
- obiekty w zbiorze **nie mogą** się powtarzać

- **HashSet** – podstawowa implementacja, wykorzystuje hashCode()
- **TreeSet** – przechowuje elementy w postaci drzewa

Java SE
set

```
Set<String> zbior = new HashSet<String>();
zbior.add("pierwszy");
zbior.add("drugi");
for (String ciagZnakow : zbior) {
    System.out.println(ciagZnakow);
}
```

Java SE
Ćwiczenie

- stwórz kilka obiektów, w tym 2 równe
- dodaj je do kolekcji Set
- wypisz wszystkie elementy tej kolekcji

Java SE
lista

- każdy element ma przyporządkowany indeks
- obiekty mogą się powtarzać
- możemy odwołać się do konkretnego elementu po indeksie
- podstawowe operacje:
  add()
  get(indeks)

Java SE
lista

- **ArrayList** – przechowuje dane wewnątrz tablicy, wydajna gdy znamy ilość elementów lub wykonujemy mało operacji dodawania
- **LinkedList** – przchowuje dane w postaci powiązanej, wydajniejsza gdy dodajemy dużo elementów

```
List<String> lista = new ArrayList<String>();
lista.add("pierwszy");
lista.add("drugi");
System.out.println(lista.get(1)); //wypisze "drugi"
```

Java SE
Ćwiczenie

- stwórz kilka obiektów, w tym 2 równe
- dodaj je do kolekcji List
- wypisz wszystkie elementy tej kolekcji
- wypisz konkretny element za pomocą indeksu

Java SE
mapa

- formalnie nie są kolekcjami (nie są typu Collection)
- przechowują parę klucz-wartość
- do elementów odwołujemy się po kluczu
- .. który wskazuje na wartość
- klucz jest obiektem
- klucze muszą być unikalne

Java SE
mapa

- **HashMap** – właściwości podobne do HashSet, kolejność i przechowywanie wynika z implementacji hashCode()
- **TreeMap** – elementy przechowywane w formie posortowanej (wg klucza)

```
Map<String, Integer> mapa = new HashMap<String, Integer>();
mapa.put("pierwszy", 1);
mapa.put("drugi", 2);
System.out.println(map.get("pierwszy")); //wypisze "1"
```

Java SE
Ćwiczenie

- stwórz kilka obiektów, w tym 2 równe (klucze)
- stwórz kilka obiektów, w tym 2 równe (wartości)
- wypisz wszystkie elementy tej kolekcji:
  - klucze
  - wartości
  - pary klucz - wartość

Modyfikatory dostępu

Java SE
pakiety

- klasy pogrupowane w pakiety
- struktura hierarchiczna
- pakiety – katalogi, klasy – pliki
- implementacja klasy znajduje się w jakimś pakiecie
- informuje o tym instrukcja *package*
  np. klasa znajduje się w pakiecie *java*, który znajduje się w pakiecie *pl*

  package pl.java

Java SE
modyfikatory dostępu

- słowa kluczowe określające poziom dostępności pól/metod innym klasom
- **public** – dostęp do elementu dla wszystkich klas
- **protected** – dostęp tylko dla klas dziedziczących lub z tego samego pakietu
- **private** – brak widoczności elementów poza klasą
- **default** – dostęp pakietowy, nie istnieje takie słowo kluczowe
- dobra praktyka – wszystkie pola prywatne

Java SE
Ćwiczenie

- stwórz katalog
- przenieś do niego swoje klasy (poza Main)
- spróbuj w Main skorzystać z pola myMenu.text
- zmień modyfikator na public

Java SE
Ćwiczenie

- stwórz klasę (1) z metodami publicznymi i prywatnymi
- stwórz inną klasę (2) w innym pakiecie
- stwórz w klasie 2 obiekt typu 1
- zobacz jakie metody są dostępne

Thanks!

Q&A

tomasz.lisowski@protonmail.ch