Podstawy Java SE

Hello!

Tomasz Lisowski

Software developer, JIT Solutions
IT trainer

Agenda

- powtórka
- równość obiektów
- typy tekstowe
- kolekcje
- modyfikatory dostępu
- interfejsy i klasy abstrakcyjne
- wyjątki

Java SE
Metody

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

```
public class Methods {

    public void method1() {
        System.out.println("Ta metoda nie zwraca nic!");
    }

    public int getNumberTwo() {
        return 2;    //ta metoda zwraca liczbę całkowitą 2
    }

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

Java SE
String

- obiektowy typ tekstowy
- *immutable* – nie można go zmienić, zmiana powoduje stworzenie nowej instancji
- posiada zestaw metod do operacji na tekście, np. *compare(), concat(), split(), length(), replace(), substring()*

```
char[] chars = {'i', 'n', 'f', 'o', 'S', 'h', 'a', 'r', 'e'};
String s = new String(chars);

String s2 = "infoShare";
```

Java SE
String

- modyfikacje na Stringach wymagają przypisania

```
String s = "infoShare";
s.concat("Academy");
System.out.println(s);
```

```
infoShare
```

```
String s = "infoShare";
s = s.concat("Academy");
System.out.println(s);
```

```
infoShareAcademy
```

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
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
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 co najmniej 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;
}
```

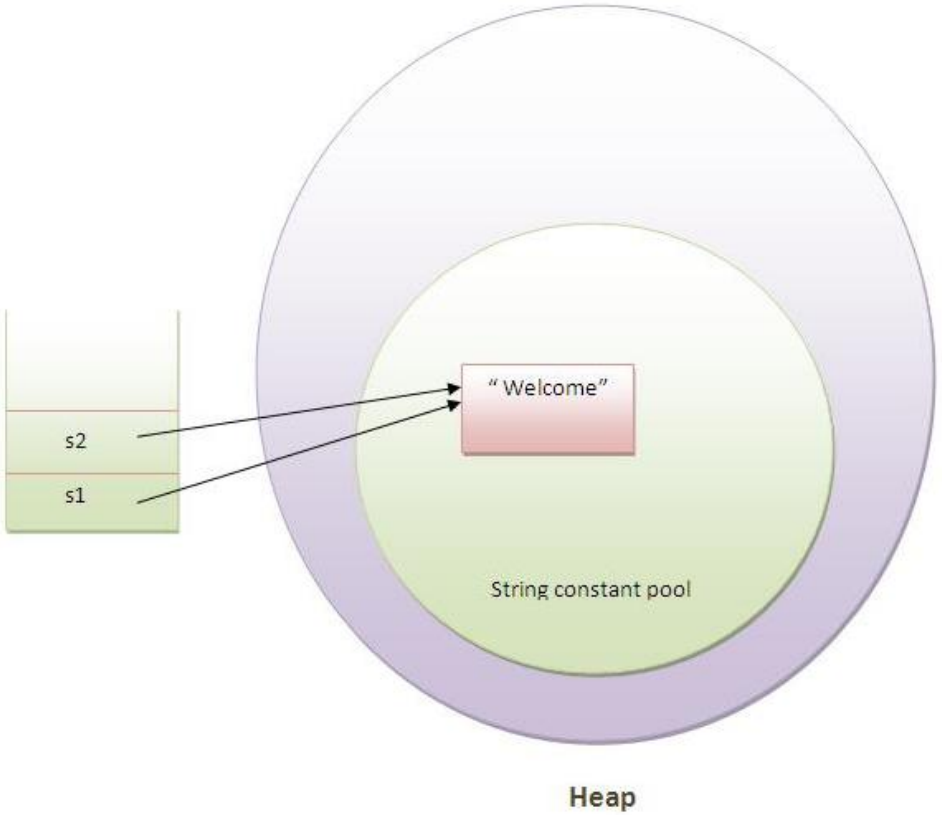
Czy obiekty są równe?

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



Java SE
== vs equals

```
String tekstA = new String( original: "tekst");
String tekstB = new String( original: "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*

Operacje na typach tekstowych

Java SE
String

- tworzenie Stringów

```
String s1="java";//creating string by java string literal
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);//converting char array to string
String s3=new String("example");//creating java string by new keyword
```

Java SE
String

- porównywanie Stringów

```
String s1 = "INFOShare";
String s2 = new String( original: "infoShare");

System.out.println(s1.equals(s2));
System.out.println(s1.equalsIgnoreCase(s2));
System.out.println(s1 == s2);
System.out.println(s1.compareTo(s2));
System.out.println(s1.compareToIgnoreCase(s2));
System.out.println(s2.compareTo(s1));
```

Java SE
StringBuilder

- specjalny builder do tworzenia Stringów
- kolejne Stringi dodajemy za pomocą metody *append()*

```
String s1 = "info" + "Share" + "Academy";

StringBuilder stringBuilder = new StringBuilder();
stringBuilder.append("info");
stringBuilder.append("Share");
stringBuilder.append("Academy");

String s2 = stringBuilder.toString();
String s3 = stringBuilder.reverse().toString();
```

Java SE
String - metody

```
String s = "  InfoShare  ";

s.toUpperCase(); //  INFOSHARE
s.toLowerCase(); //  infoshare
s.trim(); //InfoShare
s.startsWith("In"); //false
s.endsWith("are"); //false
s.charAt(5); //o
s.length(); //13
String.valueOf(10); //10
s.replace( target: "Share",  replacement: "Academy"); //  InfoAcademy
```

Java SE
String - metody

```
String s = "string:separate:by:colons";
String[] sArray = s.split( regex: ":");

System.out.println(sArray.length);
for (int i = 0; i < sArray.length; i++) {
    System.out.println(sArray[i]);
}
```

Podstawowe kolekcje

Java SE
tablica

- struktura gromadząca uporządkowane dane
- tablice jedno lub wielowymiarowe
- ich wielkość jest stała
- odwołanie do elementu po indeksie

typ[] nazwaTablicy = new typ[liczbaElemenetów]

```
int[] tab = new int[3];
int[] tab2 = {1,2,3};
```

Java SE
tablica

- struktura gromadząca uporządkowane dane

```
int[] tab = new int[3];
int[] tab2 = {1,2,3};

int number = tab[1];
int number2 = tab2[1];
```

Java SE
Ćwiczenie

- stwórz metodę przyjmującą parametr typu int
- wewnątrz metody stwórz tablicę 10-elementową typu int
- uzupełnij tablicę kolejnymi liczbami całkowitymi, zaczynając od tej podanej w parametrze
- wypisz wszystkie elementy tablicy

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 dwa 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** – przechowuje 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 dwa równe
- dodaj je do kolekcji List
- wypisz wszystkie elementy tej kolekcji

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 (Integer), w tym dwa równe (klucze)
- stwórz kilka obiektów (String), w tym dwa równe (wartości)
- dodaj elementy do mapy
- 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

Dwa słowa o OOP

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. dbDriver.connectToDB()

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


I ARE PROGRAMMER
I MAKE COMPUTER BEEP BOOP
BEEP BEEP BOOP
quickmeme.com

Java SE
dziedziczenie

- podstawowy mechanizm programowania obiektowego
- przekazanie cech innym klasom
- klasa potomna ma cechy klasy bazowej + swoje własne
- klasa potomna może rozszerzać tylko jedną klasę

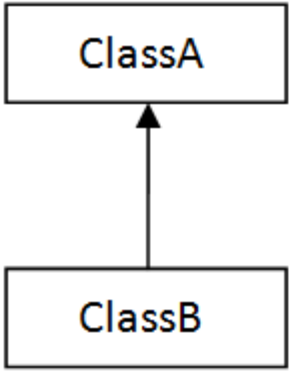
```
class Class extends OtherClass {

}
```

Java SE
dziedziczenie

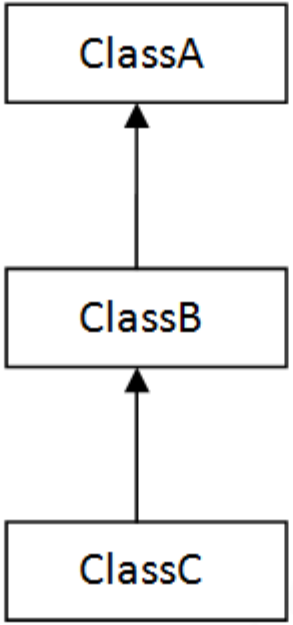
```
class Vehicle {
    String name;
    Date productionDate;
}

class Car extends Vehicle {
    int numberOfWheels;
    int enginePower;
}
```

```
Car myCar = new Car();
myCar.setName("name");
myCar.setEnginePower(9001);
```

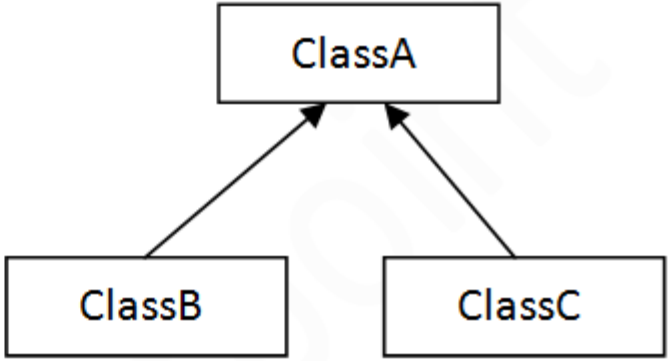
Java SE
dziedziczenie



1) Single



2) Multilevel



3) Hierarchical

Java SE
Ćwiczenie

- stwórz klasę nadrzędną Animal, która posiada metodę eat()
- stwórz dwie klasy potomne, reprezentujące konkretne zwierzęta
- wykonaj na obiektach powyższych typów metodę klasy Animal

Java SE
przeciążanie

- ang. **overloading**
- mechanizm pozwalający na tworzenie metod o tej samej nazwie
- ..ale różniących się typem lub ilością parametrów
- konstruktory również mogą być przeciążane
- pułapka automatycznego rzutowania

Java SE
przeciążanie

```
public class Calculator {

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

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

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

    public double add(double a, double b, double c) {
        return add(a, b) + c;
    }
```

Java SE
przesłanianie

- ang. **overriding**
- inaczej *nadpisanie*
- mechanizm pozwalający modyfikować metodę klasy bazowej
- używany w celu stworzenia specyficznej implementacji
- przeładowane metody muszą mieć taką samą strukturę jak bazowe
- oraz posiadać adnotację @Override

Java SE
przesłanianie

- metody *hashCode()*, *equals()*, *toString()* są metodami klasy Object i mogą być nadpisane w każdej innej klasie
- nie można przesłaniać metod statycznych

```
@Override
public String toString() {
    return "someString";
}
```

Java SE
Ćwiczenie

- stwórz enum, który oznacza dostępne opcje (np. dodawanie)
- wyświetl informację o dostępnych opcjach (enum)
- pobierz opcję z klawiatury (numer opcji)
- ustaw odpowiednią wartość enum w zależności od podanej liczby
- jeżeli błędna wartość to wyświetl informację i ponownie
- podanie wartości 0 przerywa działanie

Thanks!

Q&A

tomasz.lisowski@protonmail.ch