Politechnika Wrocławska

Platformy Programistyczne
Podstawy języka Java

Agata Migalska

6 maja 2014

Plan wykładu

1. Sztuka wysławiania się w języku Java
2. Cały świat jest obiektem
3. Kolekcje
4. Zmienne i metody statyczne
5. Słowo kluczowe final

Sztuka wysławiania się w języku Java

Wyrażenia

```
int liczbaStatkow = 5;
String imie = "Kapitan␣Hak";
liczbaStatkow = liczbaStatkow + 7;
System.out.print(imie + "␣zatopil␣");
System.out.println(liczbaStatkow + "␣statkow");
// komentarz
/*
    dluzszy komentarz
*/
```

Wyrażenia

```
int liczbaStatkow = 5;
String imie = "Kapitan␣Hak";
liczbaStatkow = liczbaStatkow + 7;
System.out.print(imie + "␣zatopil␣");
System.out.println(liczbaStatkow + "␣statkow");
// komentarz
/*
    dluzszy komentarz
*/
```

Wynik

```
Kapitan Hak zatopil 7 statkow
```

Operatory

`=`	przypisanie
`==`	równość bitów
`.equals(Object)`	równość obiektów
`!`	nieprawda
`<, >`	porównanie

Operatory

`=`	przypisanie
`==`	równość bitów
`.equals(Object)`	równość obiektów
`!`	nieprawda
`<, >`	porównanie

Przykłady

```
int numer = 7;
if (statek != null) { ... }
boolean niepusta = !tablica.isEmpty();
```

Pętle while i do-while

```
Statek[] statki = new Statek[5];
int liczbaStatkow = statki.length;

while (liczbaStatkow  > 0) {
    liczbaStatkow = liczbaStatkow - 1;
}

do {
    liczbaStatkow = liczbaStatkow - 1;
} while (liczbaStatkow > 0);
```

Pętle for i for-each

```
Statek[] statki = new Statek[5];

for (int i=0; i < statki.length; i = i+1) {
    System.out.println(statki[i].getName());
}

for (Statek statek : statki) {
    System.out.println(statek.getName());
}
```

Instrukcje warunkowe

```
if (pole == "statek" && zatopiony) {
    // zatopiony == true
    liczbaStatkow = liczbaStatkow - 1;
    // liczbaStatkow--;
    System.out.println("Trafiony␣zatopiony!");
} else if (pole == "statek" && !zatopiony) {
    // zatopiony == false
    System.out.println("Trafiony␣niezatopiony!");
} else {
    System.out.println("Pudlo!");
}
```

Operatory logiczne

Operator AND

```
String pole = null;
if (pole != null && pole.equals("statek")) {
    ...
}
```

Bitowy operator AND

```
if (pole != null & pole.equals("statek")) {
    ...
}
```

Operatory logiczne

Operator AND

```
String pole = null;
if (pole != null && pole.equals("statek")) {
      jezeli pierwszy warunek nie jest spelniony,
       drugi nie bedzie sprawdzany
}
```

Bitowy operator AND

```
if (pole != null & pole.equals("statek")) {
      NullPointerException
      - oba warunki zostana sprawdzone
}
```

Operatory logiczne

Operator OR

```
String pole = null;
if (pole == null || pole.isEmpty()) {
    ...
}
```

Bitowy operator OR

```
if (pole == null | pole.isEmpty()) {
    ...
}
```

Operatory logiczne

Operator OR

```
String pole = null;
if (pole == null || pole.isEmpty()) {
      jezeli pierwszy warunek jest spelniony,
       drugi nie bedzie sprawdzany
}
```

Bitowy operator OR

```
if (pole == null | pole.isEmpty()) {
      NullPointerException
      - oba warunki zostana sprawdzone
}
```

Cały świat jest obiektem

Cały świat jest obiektem

public class Object

Klasa Object reprezentuje obiekty w Javie.

Każdy obiekt (niejawnie) dziedziczy po klasie Object.

Cały świat jest obiektem

public class Object

Klasa Object reprezentuje obiekty w Javie.

Każdy obiekt (niejawnie) dziedziczy po klasie Object.

ale ...

Cały świat jest obiektem

public class Object

Klasa Object reprezentuje obiekty w Javie.

Każdy obiekt (niejawnie) dziedziczy po klasie Object.

ale ...

Istnieją typy proste (które nie są obiektami):

`byte`, `short`, `int`, `long`, `float`, `double`, `char`, `boolean`

Klasy i obiekty

Klasa

Wzorzec wg którego jvm tworzy obiekty.

Klasa jest zawsze jedna.

Klasy i obiekty

Klasa

Wzorzec wg którego jvm tworzy obiekty.

Klasa jest zawsze jedna.

Obiekt

Egzemplarz klasy.

Klasy i obiekty

```
public class MojaKlasa {

}
```

Konstruktor

Konstruktor domyślny

1. bezargumentowy
2. dodawany przez kompilator jeżeli:
   1. inny konstruktor nie jest zdefiniowany w klasie
   2. nadklasa posiada konstruktor bezargumentowy

Klasy i obiekty

```
public class MojaKlasa {
  public MojaKlasa() {
    // konstruktor bezargumentowy
  }

  public MojaKlasa(int jakisNumer) {
    // konstruktor jedno-argumentowy
  }
}
```

Typy zmiennych

Typy

- Typy prymitywne: int, short, byte, long, double, float, boolean, char

  przekazywane przez wartość
- Typy referencyjne: Integer, String, int[], Object, Object[]

  przekazywane przez referencję

Referencja

Bity, które reprezentują sposób dotarcia do obiektu.

Nazwy zmiennych

Nazwa zmiennej

1. zaczyna się od litery, podkreślnika _ lub dolara $
2. nie może zaczynać się od cyfry
3. nie może być słowem zarezerwowanym

Zmienne egzemplarza (instance variables)

```
public class Ship {
    private int numberOfMasts;
    private String name;
    private Coordinates coords;
}
```

- podczas tworzenia nowego obiektu zmiennym egzemplarza są przypisywane wartości domyślne
  - obiekty - null
  - typy prymitywne - 0, false

Zmienne egzemplarza

```
public class Ship {
    private int numberOfMasts; // = 0
    private String name; // = null
    private Coordinates coords; // = null
}
```

Zmienne lokalne

Zmiennymi lokalnymi nazywamy...

- parametry metod
- zmienne zadeklarowane w metodzie

Widoczność zmiennych lokalnych

Zmienne są widoczne tylko w metodzie, w której zostały zadeklarowane.

Jeżeli zmienna została zadeklarowana w bloku wewnątrz metody, jest widoczna tylko w tym bloku.

Zmienne lokalne

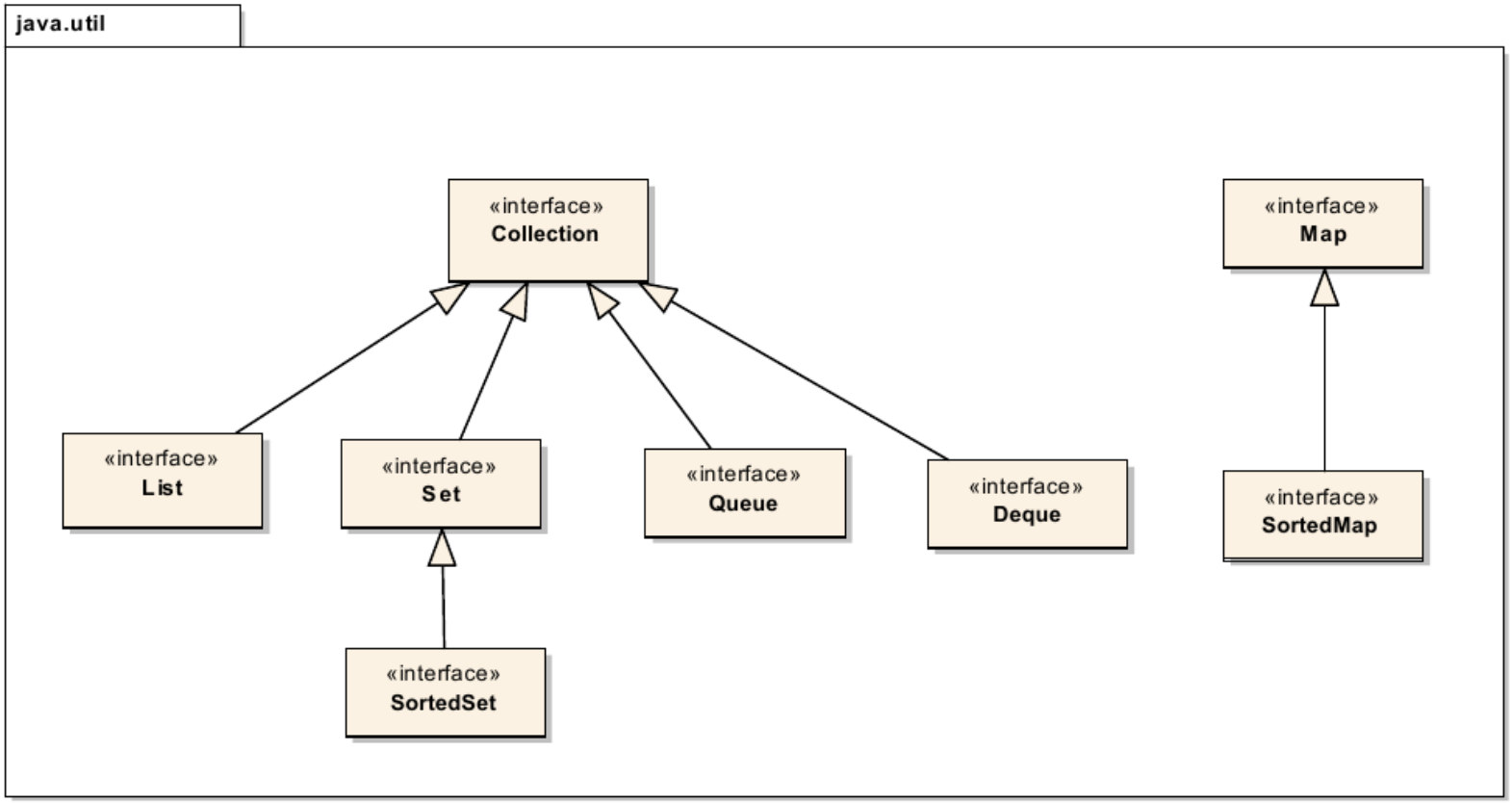
```
public class Ship() {
  public int countMasts(Coordinates coords) {
    int x;
    if (x > 1) {
      ...
    }
    for (int i=0; i<10; i++) {
        String tempString = "cokolwiek";
        ...
    }
    System.out.println("" + i);
  }
}
```

Zmienne lokalne

```
public class Ship() {
  public int countMasts(Coordinates coords) {
    int x;
    if (x > 1) {  blad kompilatora
     zmienna x nie jest zainicjowana
    }
    for (int i=0; i<10; i++) {
        String tempString = "cokolwiek";
        ...
    }
    System.out.println("" + i);
     blad kompilatora - nieznana zmienna i
  }
}
```

Kolekcje

Kolekcje


java.util
«interface»
Collection
«interface»
Map
«interface»
List
«interface»
Set
«interface»
Queue
«interface»
Deque
«interface»
SortedMap
«interface»
SortedSet

h

Iterowanie elementów kolekcji

1. Iterator
2. `for-each`
3. operacje agregujące (JDK 8)

Iterator

```
public interface Iterator<E> {
    boolean hasNext();
    E next();
    void remove(); // optional
}
```

Iterator

```
public void print(Collection<Object> collection) {

  Iterator<Object> iterator
    = collection.iterator();

  while (iterator.hasNext()) {
    System.out.println(iterator.next());
  }

}
```

Iterator

```
public void print(Collection<Object> collection) {

  for (Iterator<Object> iterator
    = collection.iterator(); iterator.hasNext(); ) {
    System.out.println(iterator.next());
  }

}
```

for-each

```
public void print(Collection<Object> collection) {

  for (Object element : collection) {
    System.out.println(element);
  }

}
```

Operacje agregujące

```
public void print(Collection<Object> collection) {

  collection.forEach(e -> System.out.println(e));

}
```

```
public void sort(Collection<Object> collection) {

  collection.sort((o1, o2) -> o1.compareTo(o2));

}
```

Klasa Collections

Klasa Collections

Zawiera użyteczne metody operujące na kolekcjach np.

- sort
- shuffle
- reverse
- swap
- ...

Zmienne i metody statyczne

Zmienne i metody statyczne

```
public class Utils {

  private static String EMAIL_PATTERN
    = "^\\S+@(\\S+\.)+\\S{2,3}$";

  private Utils() {
    // opcjonalne, ale logiczne
    // nie mozna stworzyc obiektu tej klasy
  }
  public static boolean
      validateEmail( String email ) {
    return email.matches(EMAIL_PATTERN);
  }
}
```

Słowo kluczowe final

Po klasach finalnych nie można dziedziczyć

```
public final class Oracle {}
public class ExtendedOracle extends Oracle {}
 blad kompilatora
 po klasie finalnej nie mozna dziedziczyc
```

Metod finalnych nie można nadpisywać

```
public class Oracle {
  public final Verdict getVerdict() {
    return Verdict.POSITIVE;
  }
}
public class ExtendedOracle extends Oracle {
  @Override
  public Verdict getVerdict() {
   blad kompilatora
   metod finalnych nie mozna nadpisywac
  }
}
```

Zmiennym finalnym nie można zmieniać wartości

```
public class Oracle {
  // stala VERDICTS_PER_DAY
  private final static int VERDICTS_PER_DAY = 7;

  public Verdict getVerdict() {
    // zmienna finalna typu prostego
    final int magicNumber = getMagicNumber();
    magicNumber = 5;
    blad kompilatora - nie mozna zmienic wartosci

    return Verdict.POSITIVE;
  }
}
```

Zmiennym finalnym nie można zmieniać wartości

```
public class Oracle {
  public Verdict getVerdict() {
    // zmienna finalna typu referencyjnego
    final WeatherForecast weatherForecast
      = getWeatherForcast();
    // modyfikacja stanu obiektu jest dozwolona
    weatherForecast.setTemperature(20.4);

    weatherForecast = new WeatherForecast();
     blad kompilatora
       - nie mozna zmienic referencji

    return Verdict.POSITIVE;
  }
}
```