Programowanie obiektowe

Wykład 3:
Tworzenie i usuwanie obiektów

Deklaracje pól klasy

```
class  Klasa1 {
    int i;
    Klasa2 k = new Klasa2();
    char c='x';
    byte h=0xA5;
    …
    }
```

Deklaracje metod klasy

```
class  Klasa1 {
    …
    int metoda1(char p1, int p2) {
        int i;
        …
       return i;
    }
    void metoda2(float p1) {
        …
    }
}
```


Pola i metody statyczne

static – element (pole lub metoda) jest elementem klasy (a nie obiektu)

Przykład:

```
class Klasa1 {
  int  poleObiektu=0;
  int metodaObiektu (){
    return poleObiektu;
  }
  static int poleKlasy=0;
  static int metodaKlasy(){
    return poleKlasy;
  }
}
```

```
Klasa1 o1=new Klasa1();
Klasa1 o2=new Klasa1();

o1.poleObiektu++;
o1.metodaObiektu(); ⟶ 1
o2.poleObiektu++;
o2.metodaObiektu(); ⟶ 1

o1.poleKlasy++;
o1.metodaKlasy(); ⟶ 1
o2.poleKlasy++;
o2.metodaKlasy(): ⟶ 2

Klasa1.poleKlasy++;
Klasa1.metodaKlasy(); ⟶ 3
```

Zasady stosowania elementów statycznych

- Metody i pola statyczne można wywoływać z klasy,
- Metody statyczne mogą się odwoływać tylko do pól i metod statycznych,
- Pola statyczne są wspólne dla wszystkich instancji danej klasy,
- Pola statyczne są inicjowane wcześniej niż inne pola (przy pierwszym odwołaniu do klasy).

Konstruktor

- Metoda wywoływana przy tworzeniu każdego obiektu:
  - nazwa identyczna jak nazwa klasy
  - nie zwraca żadnej wartości
  - konstruktor domyślny – bez parametrów

Przykład

```
class Tree {
   int size;
   Tree(int i) {
   size=i;
   }
}
```

```
new Tree(5);
new Tree();
new Tree(true);
```


Po co konstruktor?

```
class Data {
        int  rok, miesiac, dzien;
        Data(int r,m,d) {
           rok=r;
           miesiac = m
           dzien = d;
        }
        …
        }
```

```
Data  dd = new Data(2010, 3, 8);
Data  dd = new Data(2010, 2, 30);  ???
```

Po co konstruktor?

```
class Data {
        int  rok, miesiac, dzien;
        Data(int r,m,d) {
           if (r>0) rok=r; else blad();
           if (m>=1 && m<=12) miesiac = m; else blad();
           if (d>=1)
             if (m==2)
               if( przestepny(r) && d<=29) || d<=28 ) dzien=d; else blad();
             else if (d<=30) dzien =d;
                  else if ((m==1 || m==3 || m==5 || m==7 || m==8 || m==10 ||
                            m==12 ) && d<=31) dzien = d;
                       else blad();
           else blad()
        }
        …
        }
```

Data dd = new Data(2010, 2, 30); blad() !!!
Ale: dd.dzien=29; ???

Przeciążanie nazw metod (Overloading)

- Kilka metod o takiej samej nazwie, różniących się:
  - liczbą argumentów
  - typem argumentów
  - ~~typem zwracanym~~

Np.

int m1(int i) ← m1(5)

int m1(long i) ← m1(5L)

Dotyczy również konstruktorów!!!

this

- „adres” obiektu w którym jest użyte

Np.

```
public class Leaf{
  int i=0;
  Leaf increment() {
        i++;
        return this;
  }
}
```


Przykład 1

```
public class Kwiat {
 int liczbaPlatkow = 0;
 String s = new String("");
 Kwiat(int platki) {
  liczbaPlatkow = platki;
  System.out.println(
  "Konstruktor 1:  LiczbaPlatkow="
    + liczbaPlatkow);
 }
 Kwiat(String ss) {
  System.out.println(
    "Konstruktor 2: s=" + ss);
  s = ss;
 }
```

```
 Kwiat(String s, int platki) {
    this(platki);
    this.s = s;
    System.out.println("Konstruktor 3");
 }
 Kwiat() {
    this("aster", 47);
    System.out.println(
      "Konstruktor domyslny");
 }
 void print() {
    System.out.println(
      "LiczbaPlatkow = " + liczbaPlatkow + " s = "+ s);
 }
 public static void main(String[] args) {
    Kwiat x = new Kwiat();
    x.print();
 }
}
```

Przykład 2

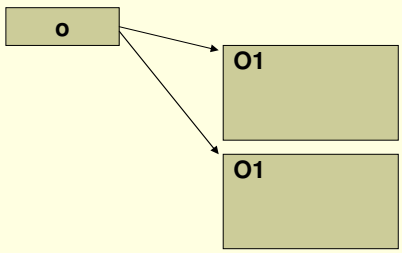
```
class K1 {
   static int  i;
   static m1(int i) {
      this.i=i;
   }
}
```


Usuwanie obiektów

- Garbage collection: System.gc()
- metoda finalize(): System.runFinalization()

Przykład:

```
O1 o;
o=new O1();
o=new O1();
o=null;
```



Przykład

```
class Book {
  boolean checkedOut = false;
  Book(boolean checkOut) {
    checkedOut = checkOut;
  }
  void checkIn() {
    checkedOut = false;
  }
  public void finalize() {
    if(checkedOut)
      System.out.println("Error:
      checked out");
  }
}
```

```
public class TerminationCondition {
  public static void main(String[ ] args) {
    Book novel = new Book(true);
   …
    novel.checkIn();
    // Teraz można usunąć

    novel= new Book(true);
    novel=null;
    // Teraz będzie sygnalizowany błąd:
    System.gc();
  }
}
```


Inicjalizacja pól

- Inicjalizacja domyślna:
  - pola liczbowe: 0, 0.0
  - boolean: false
  - char: '\0'
  - zmienne obiektowe: null
- Inicjalizacja w deklaracjach:

```
class Cinit {
    int i=5;
    int j=f(i);
    Obj   o=new Obj();
```

Podsumowanie

- Tworzenie obiektów
  - Rola konstruktora: kontrola + inicjalizacja
- Inicjalizacja
  - Kolejność inicjalizacji: domyślna, w deklaracjach, w konstruktorze
  - Najpierw pola statyczne
  - Nie są inicjalizowane zmienne lokalne
- Usuwanie obiektów
  - Nie ma destruktora w Javie!

Pytania

1. Omówić znaczenie zmiennych i metod statycznych.
2. Omówić rolę konstruktora w klasach.
3. Na czym polega i czego dotyczy przeciążenie nazw.
4. Omówić sposoby inicjalizacji pól w obiektach.
5. Omówić zasady inicjalizacji obiektów.
6. Co oznacza słowo *this* . Podać przykłady zastosowania.
7. Omówić cel i zasady stosowania metody *finalize()*.

Koniec