Klasy abstrakcyjne, interfejsy i polimorfizm

Programowanie obiektowe

Janusz Jabłonowski

12 kwietnia 2011

Organizacyjne

- Klasówka będzie 20 IV 2011.
- Sale jeszcze są pertraktowane.
- Materiał do wyjątków włącznie.
- Można mieć swoje materiały nieelektroniczne.

Wywołanie z super może nie być intuicyjne

```
1
2  public class A {
3    public void m1(){System.out.println("A");}
4  }
5
6  public class B extends A{
7   public void m2(){
8     System.out.print(" super.m1() -> ");
9     super.m1();
10    System.out.print(" this.m1() -> ");
11    this.m1();
12  }
13
14  @Override
15  public void m1(){System.out.println("B");}
16 }
```

Wywołanie z super może nie być intuicyjne

```
1
2  public class C extends B{
3    public void m3(){
4          m2();  // Czy wypisze A czy B?
5    }
6    @Override
7    public void m1(){System.out.println("C");}
8  }
```

Wywołanie z super może nie być intuicyjne

```
1 B b = new B();
2 b.m2();
3 C c = new C();
4 c.m2();
5 c.m3();
```

I wynik:

```
super.m1() -> A
this.m1() -> B
super.m1() -> A
this.m1() -> C
super.m1() -> A
this.m1() -> C
```

Klasy abstrakcyjne

- Nie mają obiektów (bezpośrednio swojej klasy).
- Kompilator odrzuci próbę utworzenia obiektu takiej klasy.
- Są bardzo ważne!
- Pozwalają wyabstrahować wspólne cechy wielu pojęć (klas).
- Zapewniają wspólny interfejs dla swoich podklas.
- Zapewniają jednocześnie wspólną implementację.
- Typowe zastosowaie - wiele implementacji tego samego pojęcia.
- Klasy abstrakcyjne - klasy konkretne.

Składnia klas abstrakcyjnych

- Klasa abstrakcyjna jest deklarowana ze słowem **abstract**.
- Klasa abstrakcyjna nie musi mieć metod abstrakcyjnych, choć zwykle ma.
- Metoda abstrakcyjna jest deklarowana ze słowem **abstract** i nie może mieć treści.
- Musi być podmieniona na konkretną w konkretnych podklasach.
- Klasa, która nie podmieni choć jednej odziedziczonej metody abstrakcyjnej lub ma własną metode abstrakcyjną musi być zadeklarowana jako abstrakcyjna.
- Można wywoływać metody abstrakcyjne (dlaczego nie ma w tym nic gorszącego?).
- Można też podmienić metodę konkretną na abstrakcyjną!

Przykład klasy abstrakcyjnej

```
1  abstract class Pojemnik {
2    public abstract void dodaj(int elt);
3    // Powyższe jest ciekawe - wymusza implementację w podklasach
4
5    public void dodajTab(int[] tab){
6      for(int i: tab)
7        dodaj(i);
8    }
9    // Powyższe jest ciekawe - konkretna metoda w abstrakcyjnej klasie
10   // wywołuje abstrakcyjną metodę
11
12   @Override
13   public abstract String toString();
14   // Powyższe jest ciekawe - toString było zdefiniowane w Object
15 }
16
17 // Całe powyższe jest zatem ciekawe QED.
```

Przykład podklasy klasy abstrakcyjnej

```
1  public class PojemnikTablicowy extends Pojemnik {
2      // Niezm.: dane są w tab[0..ile-1]
3
4      PojemnikTablicowy(){
5          tab = new int[1];  // 1024 byłoby bardziej naturalne
6      }
7
8      private int[] tab;
9      private int ile=0;
```

Przykład podklasy klasy abstrakcyjnej cd

```
1   @Override
2   public void dodaj(int elt){
3     if(ile >= tab.length){
4       int[] pom = new int[tab.length*2];
5       for(int i=0; i<ile; i++)
6         pom[i] = tab[i];
7       // System.arraycopy(tab, 0, pom, 0, ile);
8       tab = pom;    // nie ma to jak automatyczne odśmiecanie :)
9     }
10    // teraz już na pewno jest miejsce
11    tab[ile++] = elt;
12  }
```

Przykład podklasy klasy abstrakcyjnej cd

```
1     @Override
2     public String toString(){
3       String wyn="["; // Powinien być StringBuilder
4       for(int i=0; i<ile-1; i++)
5         wyn += tab[i] + ", ";
6       if(ile>0)
7         wyn+=tab[ile-1];
8       wyn+="]";
9       return wyn;
10    } // toString
11  } // PojemnikTablicowy
```

Przykład użycia tej hierarchii klas

```
1  public static void test(){
2      int[] dane = {1, 3, 8, 2, 5, 9, 8};
3      Pojemnik p = new PojemnikTablicowy();
4      System.out.println("p = " + p);
5      for(int elt: dane)
6          p.dodaj(elt);
7      System.out.println("p = " + p);
8      p.dodajTab(dane);
9      System.out.println("p = " + p);
10 }
```

I wynik:

```
p = []
p = [1, 3, 8, 2, 5, 9, 8]
p = [1, 3, 8, 2, 5, 9, 8, 1, 3, 8, 2, 5, 9, 8]
```

Przykład dziwnej hierarchii klas

```
1  public abstract class AbstrakcyjnaA {
2    public abstract void m();
3  }
4
5  public abstract class AbstrakcyjnaB extends AbstrakcyjnaA{
6    @Override
7    public abstract void m();
8    // można podmienić abstrakcyjną metodę na abstrakcyjną, tylko po co?
9  }
10
11 public class KonkretnaC extends AbstrakcyjnaB {
12   @Override
13   public void m() {
14     System.out.println("AbstrakcyjnaC.m()");
15   }
16 }
```

Przykład dziwnej hierarchii klas

```
1  public abstract class AbstrakcyjnaD extends KonkretnaC {
2      // klasa abstrakcyjna może dziedziczyć po konkretnej
3  }
4
5  public class KonkretnaE extends AbstrakcyjnaD{
6  }
7
8  AbstrakcyjnaA a = new KonkretnaE();
9  a.m();
```

I wynik:

```
AbstrakcyjnaC.m()
```

Interfejsy

- Klasy abstrakcyjne bez żadnej implementacji.
- Jeszcze lepsza postać kontraktu.
- Nie da się utworzyć obiektu z interfejsu (ale można obiekt klasy impelmentującej interfejs).
- Można (i bardzo często tak się robi) zadeklarować zmienną o typie będącym interfejsem.
- Interfejs może dziedziczyć po interfejsie (używa się wtedy słowa **extends**).
- Interfejs nie może dziedziczyć po klasie.
- Tak klasa jak i interfejs mogą dziedziczyć po dowolnie wielu interfejsach.
- Interfejsy dają więc namiastkę wielodziedziczenia.

Składnia interfejsów

- Domyślnie wszystkie składowe są rozumiane jako publiczne (i słowa **public** nie pisze się.
- Stałe interfejsu to atrybuty z **public**, **static** **final**. Te modyfikatory są przyjmowane domyślnie.
- Nie mają składowych klasowych (tj. ze słowem **static**) poza stałymi.
- Nie mają składowych będących atrybutami (poza stałymi).
- Klasa dziedzicząca po interfejsie albo implementuje wszystkie jego metody (i wtedy może być konkretna lub może być abstrakcyjna), albo nie wszystkie (w szczególności żadnej) i wtedy musi być abstrakcyjna.

Składnia interfejsów

- Każdy interfejs niedziedziczący po innym niejawnie ma dodawane nagłówki publicznych metod klasy Object — wszystko (prawie) w Javie jest obiektem.
- Interfejs może zawierać tylko:
  - stałe (pola),
  - metody,
  - zagnieżdżone klasy i interfejsy.

Składnia interfejsów - interfejs Pojemnik

```
1 public interface Pojemnik{
2   void wstaw(int i);
3   int pobierz();
4   boolean pusty();
5 }
```