Objektovo orientované programovanie

(rozhrania)

4. prednáška (1.časť)

Vladislav Novák
FEI STU v Bratislave
7.10.2014
(úprava 24.11.2014 na str. 1, 2, 4)

Obsah

Rozhrania (Interfaces)

Rozhrania slúžia na definíciu rozhrania objektov. Definujú metódy, ktoré objekt musí mať implementované.

Napr. dvaja výrobcovia sa môžu dohodnúť na spoločnom rozhraní. Jeden výrobca bude vytvárať triedy, ktorých inštancie budú vedieť vykonávať metódy definované v rozhraní. Druhý výrobca bude vedieť tieto inštancie používať vďaka definovanému rozhraniu.

Rozhranie je referenčným typom, podobne ako trieda.
Môže obsahovať:

- konštanty
- abstraktné metódy (bez implementácie)
- default-né metódy (s implementáciou)
- statické metódy (s implementáciou)
- vnorené typy (budeme preberať neskôr)

Abstraktná metóda je metóda, ktorá nemá definovanú implementáciu. Uvádza sa kľúčovým slovom `abstract`, v rozhraniach to ale nie je potrebné. Za deklaráciou abstraktnej metódy je bodkočiarka. Implementácia metódy je definovaná v podtype (v prípade rozhraní je to trieda implementujúca rozhranie, príklad bude uvedený ďalej).

V rozhraniach nie je potrebné pred deklaráciou abstraktnej metódy uvádzať kľúčové slovo `abstract`, pretože všetky metódy, ktoré nie sú default-né ani statické, sú implicitne abstraktné.

Default-né a statické metódy majú definovanú implementáciu.

Nemožno vytvoriť inštanciu rozhrania.

Rozhranie je možné implementovať v triede, alebo ho rozšíriť v inom rozhraní.

Rozhranie sa uvádza kľúčovým slovom `interface`.

príklad definície rozhrania:

```
public interface NazovRozhrania {
    //konštanty
    int KONSTANTA1 = 10;
    String KONSTANTA2 = "konstanta2";
    Integer KONSTANTA3 = new Integer(100);

    //abstraktné metódy (nie je potrebné uvádzať kľúčové slovo abstract)
    double metoda1(int parameter1, double parameter2);
    String metoda2();
    void metoda3();

    //default-né metódy
    default int defaultMetoda1() { /*.....*/ }
    default void defaultMetoda2(int p1, byte p2) { /*.....*/}

    //statické metódy
    static void statickaMetoda1(int parameter) { /*.....*/}
    static double statickaMetoda2(String p1, int p2) { /*.....*/}

    //vnorené typy
    enum VnorenyEnum {/*.....*/ }
    interface VnorenyInterface { /*.....*/ }
    class VnutornaTrieda { /*.....*/ }
}
```

Za deklaráciou abstraktnej metódy je bodkočiarka.

Modifikátory prístupu

Modifikátory prístupu pred definíciou rozhrania (pred kľúčovým slovom `interface`)

- `public` – rozhranie možno používať na ľubovoľnom mieste
- bez uvedenia modifikátora (*private-package*) – rozhranie možno používať iba v balíku, v ktorom je uvedené

Modifikátory prístupu v tele rozhrania

- Všetky konštanty definované v rozhraní sú implicitne typu `public static final`. Preto možno tieto modifikátory vynechať.
- Všetky metódy definované v rozhraní sú implicitne `public`. Takže možno tento modifikátor vynechať.

(Iné modifikátory (`private`, `protected`) nemožno uviesť)

Použitie rozhrania = implementácia rozhrania triedou:

Ak trieda implementuje rozhranie, tak sa názov rozhrania uvádza za kľúčovým slovom `implements`.
Trieda, ktorá implementuje rozhranie, musí implementovať všetky abstraktné metódy rozhrania (toto neplatí pre abstraktné triedy, ktorými sa budeme zaoberať neskôr).

príklad:

```
public class TriedaImplRozhranie implements NazovRozhrania{

    @Override
    public double metoda1(int parameter1, double parameter2){
      return 1;
    }

    @Override
    public String metoda2() {
        return konstanta2; //return NazovRozhrania.konstanta2;
    }

    @Override
    public void metoda3() {
        //.....
    }
}
```

Zápis `@Override` je anotácia, ktorá označuje, že ide o metódu definovanú v implementovanom rozhraní (anotácia `@Override` sa používa aj na označenie prekrývania metód pri použití mechanizmu dedičnosti, ktorý budeme preberať neskôr).

Implementácia viacerých rozhraní

Trieda môže implementovať viacero rozhraní. Vtedy sú za kľúčovým slovom `implements` uvedené názvy implementovaných rozhraní oddelené čiarkou.

príklad

```
public class Trieda implements Rozhranie1, Rozhranie2{

    @Override
    public double metoda1(int parameter1, double parameter2) {
      //.....
    }
}
```

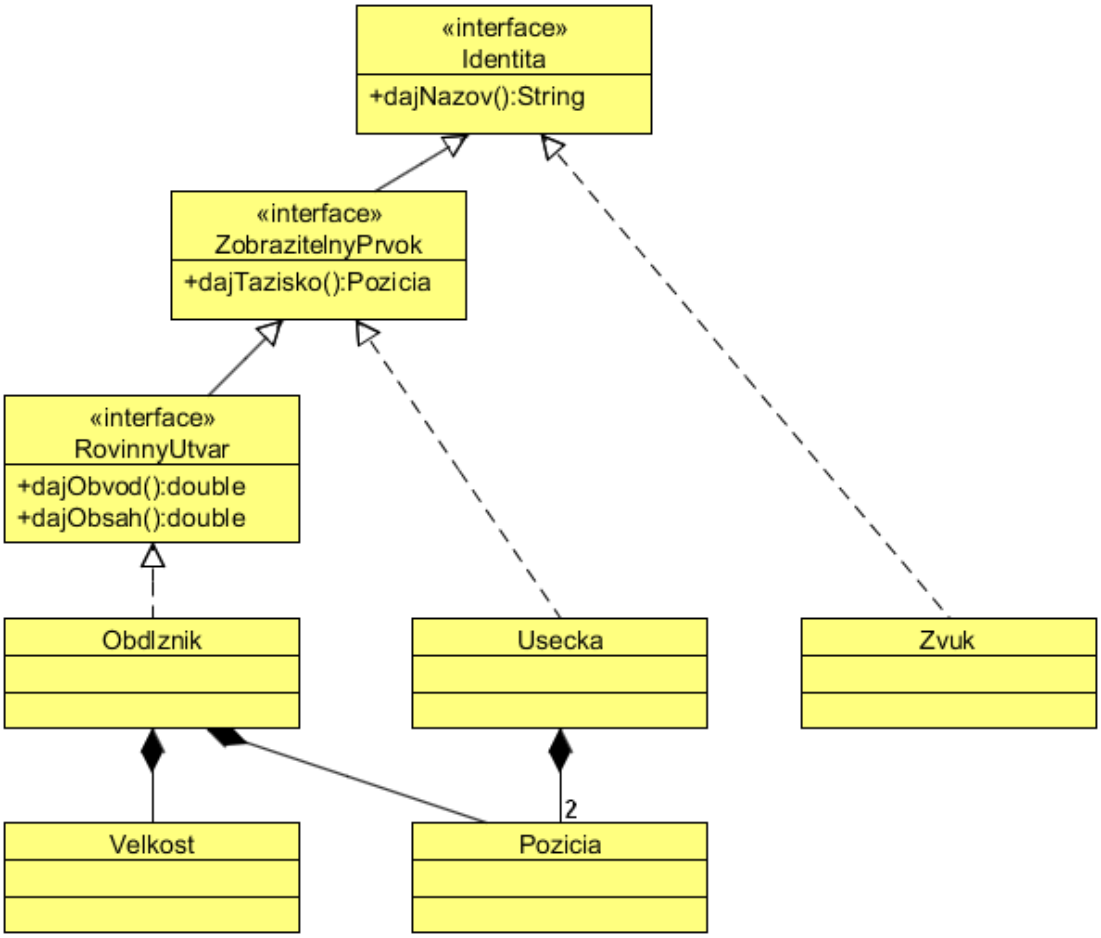
Rozhranie rozširujúce iné rozhrania

Rozhranie môže byť doplnené (rozšírené) iným rozhraním. V definícii rozhrania, ktoré rozširuje (dopĺňa) iné rozhranie sa rozširované rozhranie uvádza za kľúčovým slovom `extends`.
Ak rozhranie rozširuje viacero rozhraní, tak sú tiež vymenované za kľúčovým slovom `extends`. Názvy rozhraní za slovom `extends` sú oddelené čiarkou.

príklad:

```
public interface NoveRozhranie extends Rozhranie1, Rozhranie2{
    //konštanty
    //metódy
    //vnorené typy
}
```

Príklad



Identita.java

```
public interface Identita {
    String dajNazov();
}
```

ZobrazitelnyPrvok.java

```
public interface ZobrazitelnyPrvok extends Identita {
    Pozicia dajTazisko();
}
```

RovinnyUtvar.java

```
public interface RovinnyUtvar extends ZobrazitelnyPrvok {
    double dajObvod();
    double dajObsah();
}
```

Zvuk.java

```
public class Zvuk implements Identita {
    private String nazov;

    public Zvuk(String nazov) {
        this.nazov = nazov; //pozn.: instancie String su nemenne
    }

    @Override
    public String dajNazov() {
        return nazov; //pozn.: instancie String su nemenne
    }

    public void start() {
        //.....
    }

    public void stop() {
        //.....
    }
}
```

Usecka.java

```
public class Usecka implements ZobrazitelnyPrvok {
    private String nazov;
    private Pozicia zaciatok; //zaciatocny bod usecky
    private Pozicia koniec;   //koncovy bod usecky

    public Usecka(String nazov, double x1, double y1,
                                double x2, double y2) {
        this.nazov = nazov;
        zaciatok = new Pozicia(x1, y1);
        koniec = new Pozicia(x2, y2);
    }

    @Override
    public String dajNazov() {
        return nazov;
    }

    @Override
    public Pozicia dajTazisko() {
        double x = priemer(zaciatok.dajX(), koniec.dajX());
        double y = priemer(zaciatok.dajY(), koniec.dajY());
        return new Pozicia(x, y);
    }

    public Pozicia dajZaciatok() {
        return new Pozicia(zaciatok.dajX(), zaciatok.dajY());
    }

    public Pozicia dajKoniec() {
        return new Pozicia(koniec.dajX(), koniec.dajY());
    }
```

```
    private double priemer(double a, double b) {
        return (a + b) / 2;
    }
}
```

Obdlznik.java

```
public class Obdlznik implements RovinnyUtvar {
    private String nazov;
    private Pozicia lavyHorny; //pozicia laveho horneho bodu obdl.
    private Velkost velkost;   //velkost = sirka a vyska obdlznika

    public Obdlznik(String nazov, double lavy, double horny,
                                  double sirka, double vyska) {
        this.nazov = nazov;
        lavyHorny = new Pozicia(lavy, horny);
        velkost = new Velkost(sirka, vyska);
    }

    @Override
    public String dajNazov() {
        return nazov;
    }

    @Override
    public Pozicia dajTazisko() {
        double x = lavyHorny.dajX() + (velkost.dajSirku() / 2);
        double y = lavyHorny.dajY() + (velkost.dajVysku() / 2);
        return new Pozicia(x, y);
    }

    @Override
    public double dajObvod() {
        return 2 * (velkost.dajSirku() + velkost.dajVysku());
    }

    @Override
    public double dajObsah() {
        return velkost.dajSirku() * velkost.dajVysku();
    }

    public Pozicia dajLavyHorny() {
        return new Pozicia(lavyHorny.dajX(), lavyHorny.dajY());
    }

    public Velkost dajVelkost() {
        return new Velkost(velkost.dajSirku(), velkost.dajVysku());
    }

    public boolean jeStvorcom() {
        return velkost.dajSirku() == velkost.dajVysku();
    }
}
```

Pozicia.java

```
public class Pozicia {

    private double x;
    private double y;

    public Pozicia(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public double dajX() {
        return x;
    }

    public void nastavX(double x) {
        this.x = x;
    }

    public double dajY() {
        return y;
    }

    public void nastavY(double y) {
        this.y = y;
    }
}
```

Velkost.java

```
public class Velkost {
    private double sirka;
    private double vyska;

    public Velkost(double sirka, double vyska) {
        this.sirka = sirka;
        this.vyska = vyska;
    }

    public double dajSirku() {
        return sirka;
    }

    public double dajVysku() {
        return vyska;
    }

    public void nastavSirku(double sirka) {
        this.sirka = sirka;
    }

    public void nastavVysku(double vyska) {
        this.vyska = vyska;
    }
}
```

Použitie rozhrania ako typu

Rozhrania patria medzi referenčné typy. Názov rozhrania možno použiť všade tam, kde je možné použiť názov ľubovoľného typu.
Možno definovať referenčnú premennú typu rozhranie. Takejto premennej môžeme priradiť objekt, ktorý je inštanciou triedy implementujúcej príslušné rozhranie.
Pomocou premennej typu rozhranie možno nad objektom volať metódy, ktoré sú v rozhraní definované (iné metódy sa volať nedajú).

príklad (pokračovanie):

```
public static void main(String[] args) {

  Obdlznik o = new Obdlznik("obdlznik1", 40, 50, 100, 110);
  Usecka u = new Usecka("usecka1", 10, 15, 20, 25);
  Zvuk z = new Zvuk("zvuk1");

  //Ak je implicitné pretypovanie kompilovatelné,
  //tak sa za behu vždy podarí.

  Identita io = o; //OK (Obdlznik implementuje rozhranie Identita)
  Identita iu = u; //OK (Usecka implementuje rozhranie Identita)
  Identita iz = z; //OK (Zvuk implementuje rozhranie Identita)

  ZobrazitelnyPrvok zo = o; //OK (Obdlznik impl. rozhr. Zobr.)
  ZobrazitelnyPrvok zu = u; //OK (Usecka impl. rozhr. Zobr.)
  //ZobrazitelnyPrvok zz = z; CHYBA (Zvuk neimpl. rozhr. Zobr.)

  RovinnyUtvar ro = o; //OK (Obdlznik impl. rozhr. RovinnyUtvar)
  //RovinnyUtvar ru = u; CHYBA (Usecka neimpl. rozhr. RovinnyUtvar)
  //RovinnyUtvar rz = z; CHYBA (Zvuk neimpl. rozhr. RovinnyUtvar)


  //volanie metód

  io.dajNazov();   //OK
  //io.dajTazisko(); CHYBA(metoda nie je v rozhrani definovana)
  //io.dajObsah();   CHYBA(metoda nie je v rozhrani definovana)
  //io.jeStvorcom(); CHYBA(metoda nie je v rozhrani definovana)

  zo.dajNazov();  //OK
  zo.dajTazisko();//OK
  //zo.dajObsah();  CHYBA(metoda nie je v rozhrani definovana)
  //zo.jeStvorcom();CHYBA(metoda nie je v rozhrani definovana)

  ro.dajNazov();  //OK
  ro.dajTazisko();//OK
  ro.dajObsah();  //OK
  //ro.jeStvorcom();CHYBA(metoda nie je v rozhrani definovana)

  o.dajNazov();   //OK
  o.dajTazisko(); //OK
  o.dajObsah();   //OK
  o.jeStvorcom(); //OK
```

```
  //Ak je explicitné pretypovanie kompilovatelné
  //môže za behu nastať chyba
  //Explicitné pretypovanie znamená vloženie kontroly pretypovania,
  //ktoré sa vykoná za behu
  //(podľa typu objektu na ktorý je v premennej referencia za behu).

  ZobrazitelnyPrvok zp;
  //Aky bude rozdiel pri pouziti jedneho z dvoch nasledujucich
  //priradeni "zp = ....."?
  //Obidve je mozne skompilovat, ale neskor za behu
  //pri explicitnom pretypovani nastane alebo nenastane chyba
  //(podla typu objektu na ktory je v „zp" referencia za behu)

  zp = o; //ak vlozime toto priradenie, tak NEnastane chyba za behu
          //  pretoze „zp" bude obsahovat referenciu na obdlznik
          //  (explicitné pretypovanie sa podari)
  zp = u; //ak vlozime toto pretypovanie, nastane chyba za behu
          //  pretoze „zp" bude obsahovat referenciu na usecku
          //  (explicitné pretypovanie sa NEpodari)

  Identita io2 = zp; //OK (instancia implementujuca rozranie
                //ZobrazitelnyPrvok implementuje aj rozhr. Identita)

  //RovinnyUtvar ro2 = zp; CHYBA (zp nemusi za behu obsahovat
                  //referenciu na objekt implementujuci RovinnyUtvar)
  RovinnyUtvar ro2 = (RovinnyUtvar) zp; //explicitne pretypovanie
                    //explicitne pretypovanie -> kontrola za behu
                    //KOMPILACIA OK, ALE ZA BEHU MOZE NASTAT CHYBA

  //Obdlznik o2 = zp; //CHYBA (zp nemusi za behu obsahovat obdlznik)
  Obdlznik o2 = (Obdlznik) zp; //explicitne pretypovanie
                       // explicitne pretypovanie -> kontrola za behu
                       //KOMPILACIA OK, ALE ZA BEHU MOZE NASTAT CHYBA
}
```