Programování pro přírodovědce

Dedenie a polymorfizmus

Juraj Jašík

juraj.jasik@gmail.com
Univerzita Karlova v Praze
Přírodovědecká fakulta
Katedra organické a jaderné chemie

December 16, 2012

- objekty danej triedy zdieľajú rovnakú štruktúru a správanie (vnútorné premenné a metódy)
- *dedenie (inheritance)* - *podtrieda* (*subclass*) zdedí štruktúru a správanie po svojom predkovi a
- *polymorfizmus* - podtrieda môže modifikovať určité správanie zdedené po predkovi

Rozšírenie existujúcej triedy

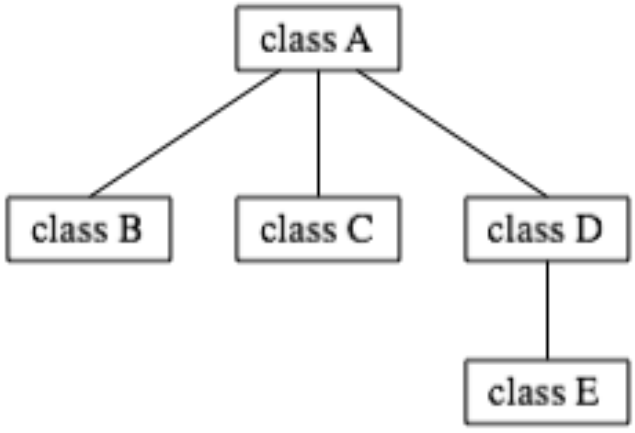
- máme nejakú triedu a chceme ju modifikovať pomocou niekoľkých zmien alebo chceme niečo pridať

```
public class subclass-name extends existing-class-name
{
    .
    .
    // Changes and additions
    .
    .
}
```

Modifikátory prístupu (access modifiers)

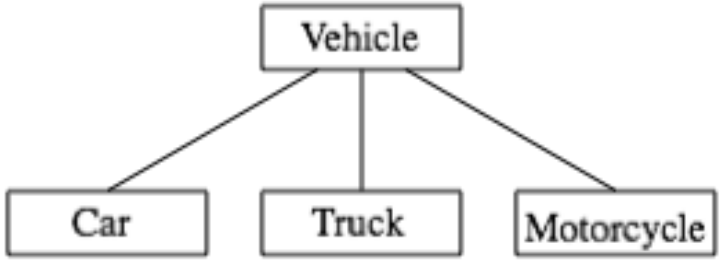
- `public` - prístupné z ľubovoľného miesta programu, aj mimo triedy
- `private` - prístupné len v rámci triedy
- `protected` - prístupné len v rámci triedy alebo jej podtriedy

Dedenie a hierarchia tried



- `B`, `C` a `D` - podtriedy (potomci) triedy `A`
- `A` - nadtrieda (predok) tried `B`, `C` a `D`
- `E` - potomok triedy `D` čiže aj potomok triedy `A`

Príklad - Vehicles




```
class Vehicle {
    int registrationNumber;
    Person owner;
    void transferOwnership(Person newOwner) {
        . . .
    }
    . . .
}
```

```
class Car extends Vehicle {
    int numberOfDoors;
    . . .
}
```

```
class Truck extends Vehicle {
    int numberOfAxles;
    . . .
}
```

```
class Motorcycle extends Vehicle {
    boolean hasSidecar;
    . . .
}
```

```
Car myCar = new Car();
```

- prístup k vlastným premenným objektu
  - `myCar.numberOfDoors`
- prístup k premenným predka
  - `myCar.registrationNumber`
  - `myCar.owner`
  - `myCar.transferOwnership()`

- `Car` alebo `Truck` alebo `Motorcycle` sú automaticky aj objekty typu `Vehicle`:
  `Vehicle myVehicle = myCar;`
  ale aj
  `Vehicle myVehicle = new Car();`

premenná, ktorá môže držať referenciu na objekt triedy A môže tiež držať referenciu na objekt ľubovoľnej podtriedy triedy A

```
Vehicle myVehicle = new Car();
```

- objekt, na ktorý ukazuje `myVehicle` si pamätá, že je `Car` a nie len `Vehicle`
- môžeme otestovať, či objekt patrí k danej triede:

```
if (myVehicle instanceof Car) ...
```

- ale nemôžme napísať

~~`myCar = myVehicle;`~~

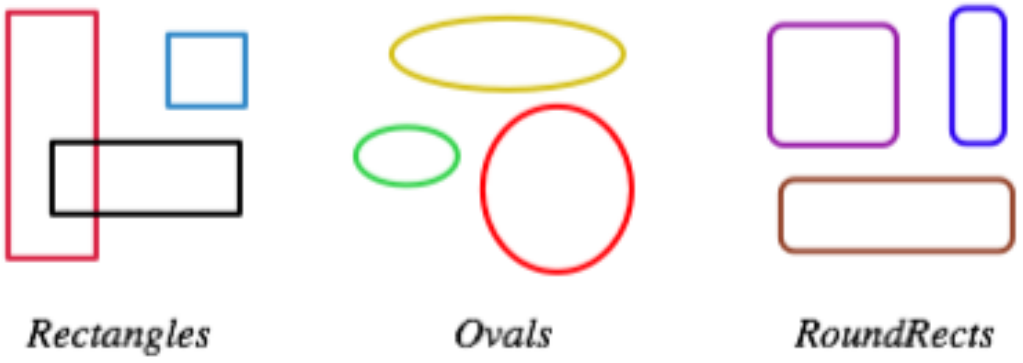
- `myVehicle` môže ukazovať nielen na `Car` aj na iné podtriedy `Vehicle`
- ale ak vieme, že `myVehicle` ukazuje na `Car`, môžme napísať:

```
myCar = (Car)myVehicle;
```

- alebo aj

```
((Car)myVehicle).numberOfDoors
```

Polymorfizmus



- triedy `Rectangle`, `Oval` a `RoundRect` sú podtriedami triedy `Shape`

```
class Shape {
   Color color;
   void setColor(Color newColor) {
      color = newColor;
      redraw();
   }
   void redraw() {
      ???
   }
}
```

- problém: ako implementovať redraw() tak, aby sme vykreslili správny tvar?
- riešenie: *polymorfizmus*

- môžme požiadať objekt, aby sa prekreslil sám:

```
class Rectangle extends Shape {
   void redraw() {
      // commands for drawing a rectangle
   }
}
class Oval extends Shape {
   void redraw() {
      // commands for drawing an oval
   }
}
class RoundRect extends Shape {
   void redraw() {
      // commands for drawing a rounded rectangle
   }
}
```

- ak je premenná `oneShape` typu `Shape`, môže ukazovať na objekt typu `Rectangle` alebo `Oval` alebo `RoundRect`
- chceme dosiahnuť aby `oneShape.redraw()` zavolal metódu `redraw()` podľa typu objektu, na ktorý `oneShape` ukazuje
- ale `redraw()` musí byť nejak definovaná v `Shape`

abstraktná trieda

```
public abstract class Shape {
   Color color;
   void setColor(Color newColor) {
      color = newColor;
      redraw();
   }
   abstract void redraw();
}
```