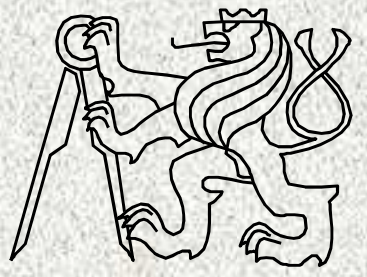
Třídy, polymorfismus



A0B36PR2-Programování 2

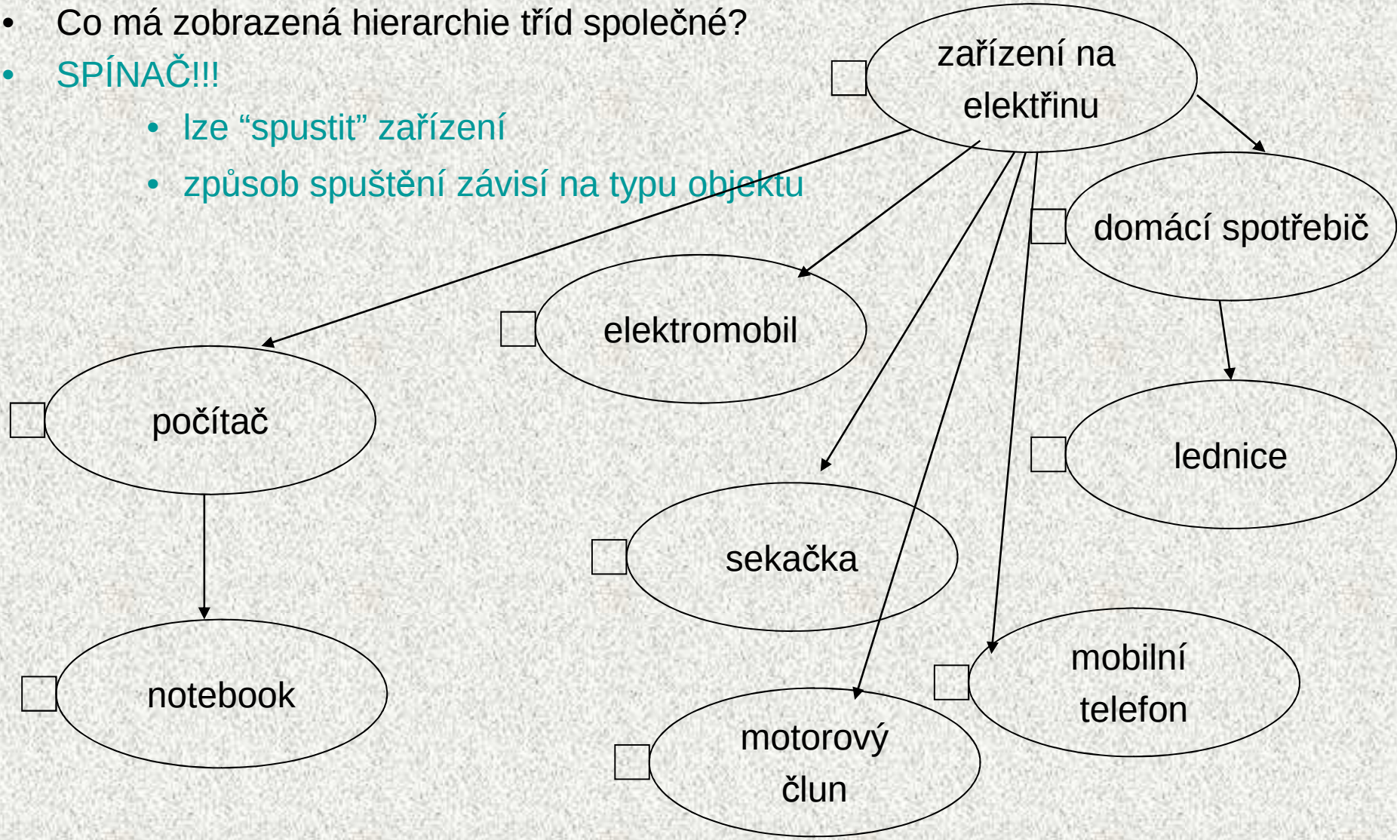
Fakulta elektrotechnická
České vysoké učení technické

Polymorfizmus

- Polymorfizmus ~ vícetvarost
  - základní vlastnost objektového přístupu
  - základní princip polymorfismu:
    - schopnost metody pracovat („přizpůsobit se“) podle typu objektu, na který působí
  - příklady:
    - *„točit volantem“*,
    - *„zaplatit“*
    - *„vypni“*,

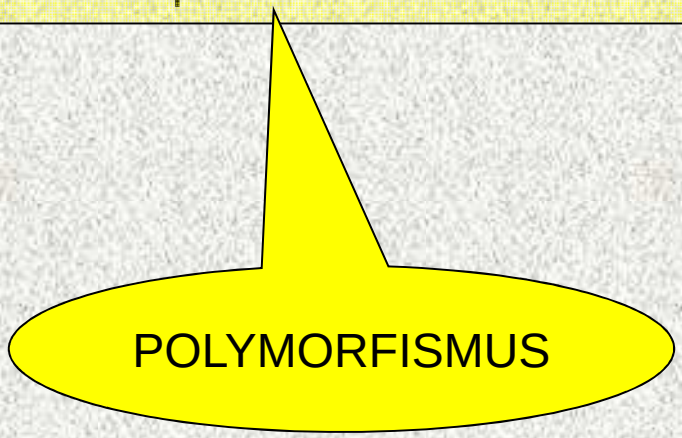
Polymorfismus - příklad

- Co má zobrazená hierarchie tříd společné?
- SPÍNAČ!!!
  - lze “spustit” zařízení
  - způsob spuštění závisí na typu objektu



Polymorfismus, řešení: abstraktní třída

- Jedno řešení polymorfismu je použití abstraktní třídy (AT)
  - AT slouží pro specifikaci společných vlastností
    - AT zajistí jednotné pojmenování společných metod
    - při zachování specifického chování dotčeného objektu



Často nepoužitelné – nutná hierachie tříd
přednost má interface

Abstraktní třída

- Abstrakce (abstraktní třídy) - **přínos objektového přístupu**
- Příklad: čítače - abstraktní metody

  `zvetsi()`
  `zmensi()`
  `reset()`
- Všechny třídy implementující čítače mají uvedené metody, resp. každá třída, která implementuje čítač musí tyto metody implementovat
- Lze tedy zavést jednu „abstraktní třídu“ charakterizující hierarchii všech tříd čítačů, tato abstraktní třída je kořenem této hierarchie
- Abstraktní třída je společný předek tříd, jejichž metody (abstraktní) vyžadujme naprogramovat podle potřeb příslušných podtříd

Abstraktní třída

```
public abstract class CitacAbstraktni {
    static int hodnota = 0;
    abstract int getHodnota();
    abstract void zvetsi();
    abstract void zmensi();
    abstract void nastav();
}
```

Abstraktní datový typ, pouze rozhraní

```
class MujCitac extends CitacAbstraktni {
 public int getHodnota(){return hodnota;}
 public void zvetsi(){hodnota++;}
 public void zmensi(){hodnota--;}
 public void reset(){hodnota = 0;}
}
```

Implementace datového typu

Pro uživatele je implementace skryta, používá jen veřejné metody objektu

Abstraktní třída

```
class JinyCitac extends MujCitac {
public void zvetsi(){
            if(hodnota >=4)hodnota=0; else hodnota++;}
public void zmensi(){
            if(hodnota < 0)hodnota=4;else hodnota--;}
}
```

Abstraktní třída

```
class CitacZnakovy extends CitacAbstraktni {
    static char hodnota = 'a';
   public int getHodnota(){return hodnota;}
   public void zvetsi(){hodnota=(char)(((int)hodnota)+1);}
   public void zmensi(){hodnota=(char)((int)hodnota-1);}
   public void nastav(){hodnota = 'a';}
}
```

Abstraktní třída, abstraktní metody

Abstrakce (abstraktní třídy) - **přínos objektového přístupu**

- Příklad: grafické objekty - abstraktní metody

```
abstract void otoc();
abstract void zmensi(); …
```

- Společný předek tříd, jejichž (některé) metody vyžadujme naprogramovat podle potřeb příslušných podtříd
- Implementace je **přenechána následníkům resp. implementace je vynucena**
- **Abstraktní třída může obsahovat datové složky a neabstraktní metody**
- **Třídy mají společnou vlastnost vyjádřenou jako abstraktní metoda**
- **Předpoklad systematického polymorfismu**
  - **stejný název metody, různá funkce pro různé objekty**
  - **lze vytvořit referenci na abstraktní třídu, nikoli její instanci!**
- Metody jsou označeny **`abstract`,** mají svůj typ, ale nemají parametry ani tělo
  - Třída s alespoň jedinou abstraktní metodou je abstraktní třídou, označená rovněž **`abstract class`**
  - Přeprogramovávat je povinné jen abstraktní metody
  - Detaily implementace jsou ponechány na příslušných potomcích, proto abstraktní třídy neimplementují konkrétní těla metod

Abstraktní třída - příklad

```
abstract class Rodic {
public int i;
  abstract int znasob();
  void setI(int noveI) { i = noveI; }
}
class Potomek1 extends Rodic {
  int znasob() { return i * 2; }
}
class Potomek2 extends Rodic {
  int znasob() { return i * 3; }

public static void main(String[] args) {
Rodic pot;
pot = new Potomek1();
pot.setI(3);
System.out.println("Hodnota je: " + pot.znasob());
pot = new Potomek2();
pot.setI(3);
System.out.println("Hodnota je: " + pot.znasob());
}}
```

Výstup:

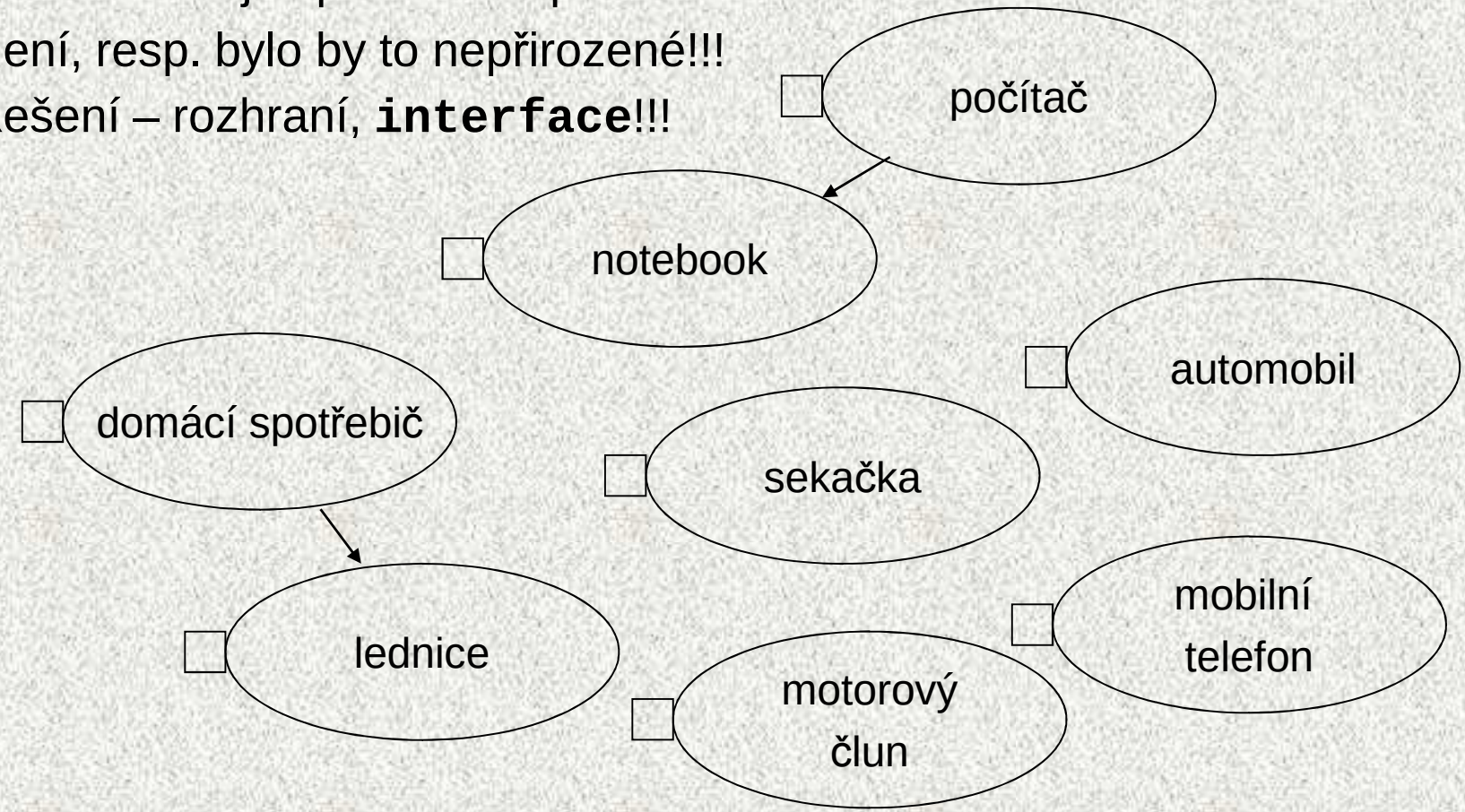
```
Hodnota je: 6
Hodnota je: 9
```

Abstraktní třídy a polymorfizmus

- Pomocí referenční proměnné předka lze využívat i metody potomka
  - Lze použít referenční proměnnou na abstraktní třídu
  - Často se využívají abstraktní metody k definici „universálního“ předka
- Jasně nadefinujeme, jakou signaturu musejí mít některé metody následníků pro jednotné ovládání, donutíme programátory to respektovat (nebo rozhraním – viz dále)
  - Možnost rozšiřování, není nutný žádný **`switch`**
  - Použití abstraktní třídy není nutné, kořenová třída nemusí být abstraktní
- Jiná možnost pro polymorfismus - interface

Rozhraní – `interface`

- Co mají zobrazené třídy společné?
- SPÍNAČ!!!
- Je možné najít společného předka?
- Není, resp. bylo by to nepřirozené!!!
- Řešení – rozhraní, **`interface`**!!!



Rozhraní - `interface`

- Java neumožňuje vícenásobnou dědičnost (problematická záležitost)
    - řešení rozhraní (**`interface`**)
- Koncepčně můžeme interface považovat za totálně abstraktní třídu, která může obsahovat pouze:
    - abstraktní metody
    - konstanty (implicitně **`public static final`**)
- Interface je konstrukce, která definuje **přidané a vyžadované** vlastnosti třídy výčtem jejích instančních metod

- Deklarace rozhraní = deklarace hlaviček metod bez implementace, podobně jako v abstraktních metodách
- Třída, která chce rozhraní použít, musí všechny metody rozhraní implementovat (překrýt)

Použití interface, implementace

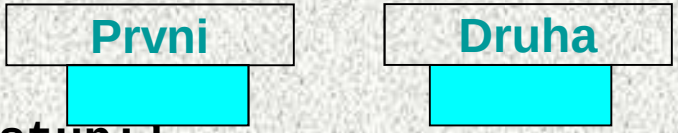
- **Rozhraní jsou implementována třídami**

- Objekty tříd, které implementují stejné rozhraní jsou "zaměnitelné" tímto rozhraním obdobně, jako jsou zaměnitelné hardwarové prvky se stejným rozhraním

- Rozhraní má tyto vlastnosti (na rozdíl od abstraktní třídy):
  - nedeklaruje žádné proměnné
  - třída může implementovat více rozhraní !!
  - rozhraní nesouvisí s dědičností tříd, s hierarchií tříd
    - různé třídy a implementace téhož rozhraní !
  - nevynucuje „příbuzenské“vztahy
  - vnucuje dovednosti těm, co by toho měly být schopni
    - příklad **`Serializable !! - viz PR1`**

Rozhraní - příklad

```
interface Inter {
 public void jednotnyVystup();
}
class Prvni implements Inter {
int data;
Prvni(int vstup) { this.data = 14 * vstup;}
public void jednotnyVystup (){
  System.out.println("Data jsou ciselna: " + this.data);}
}
class Druha implements Inter {
String data;
Druha(char vstup) { this.data = "string " + vstup;}
public void jednotnyVystup (){
  System.out.println("Data jsou retezec: " + this.data);}
}
public class Testrozhrani{
        public static void main(String[] args) {
          new Prvni(14).jednotnyVystup();
          new Druha('g').jednotnyVystup();
        }
      }
```




```
Data jsou ciselna: 196
Data jsou retezec: string g
```

Použití rozhraní jako referenční proměnné

- Proměnnou rozhrani je možné jako referenční proměnnou pro reference na instance tříd, které toto rozhraní implementují
- Je možné volat takto metody rozhraní, nikoli metody tříd!

```
public class Test_rozhrani{
public static void main(String[] args) {
 Inter i;
 Prvni p = new Prvni(14);
 i=p;
 i.jednotnyVystup();
 Druha d = new Druha('g');
 i=d;
 i.jednotnyVystup();        }
 }
```

Příklad 2: ADT - programově

```
public interface Citac {
   public int getHodnota();
   public void zvetsi();
   public void zmensi();
   public void reset();
}
public class MujCitac implements Citac {
   private int hodnota = 0;
   public int getHodnota(){return hodnota;}
   public void zvetsi(){hodnota++;}
   public void zmensi(){hodnota--;}
   public void reset(){hodnota = 0;}
}
```

Abstraktní datový typ, pouze rozhraní

Implementace datového typu

Pro uživatele je implementace skryta, používá jen veřejné metody objektu

Příklad2: ADT - programově

```
public class JinyCitac implements Citac {
   private static int hodnota;
   public int getHodnota(){return hodnota;}
   public void zvetsi(){if {(hodnota >=4 )
                            hodnota=0;else hodnota++;}
   public void zmensi(){if (hodnota < 0)
                            hodnota=4;else hodnota--;}
   public void reset(){hodnota = 0;}
}
```

Oblasti použití rozhraní

- Vnucení metod třídě bez nutnosti zařazení do hierarchie
- Vytváření "vícenásobného" dědění
- Nalezení podobných "dovedností" pro třídy různých hierarchií
    - mohly vzniknout děděním tříd z knihovny, jiných autorů ...
    - společný předek by byl „vykonstruovaný“

- Trend - použití rozhraní namísto abstraktních tříd
    - dodávka: neabstraktní třída + rozhraní (popisuje všechny metody)

Pozn: Abstraktní třída může „implementovat“ rozhraní bez skutečné implementace těla metod rozhraní

Polymorfizmus a rozhraní - příklad I

- Je-li nutné přistupovat k třídám různých hierarchií stejným způsobem a nelze-li vytvořit společného předka, pak použijeme rozhraní

```
interface Vazitelny {
  public void vypisHmotnost();
}
class Clovek implements Vazitelny {
  int vaha;
  String profese;
  Clovek(String povolani, int tiha) {
    profese = povolani;
    vaha = tiha;}
  public void vypisHmotnost() {
    System.out.println(profese + ": " + vaha);}
  public int getHmotnost() { return vaha; }
}
```

Polymorfizmus a rozhraní - příklad II

```
class Kufr implements Vazitelny {
  int vaha;
  Kufr(int tiha) { vaha = tiha; }
  public void vypisHmotnost() {
    System.out.println("kufr: " + vaha);}
}
```

Polymorfizmus a rozhraní - příklad III

```
public class PolymRozhra {
public static void main(String[] args) {
int vahaLidi = 0;
Vazitelny[] kusJakoKus = new Vazitelny[3];
kusJakoKus[0] = new Clovek("programator", 100);
kusJakoKus[1] = new Kufr(20);
kusJakoKus[2] = new Clovek("modelka", 51);
for (int i = 0;  i < kusJakoKus.length;  i++) {
kusJakoKus[i].vypisHmotnost();
if (kusJakoKus[i] instanceof Clovek == true)
vahaLidi += ((Clovek) kusJakoKus[i]).getHmotnost();
}                    //nutno přetypovat!!
System.out.println("Ziva vaha: " + vahaLidi);
  }}
```

programator: 100
kufr: 20
modelka: 51
Ziva vaha: 151

Srovnání rozhraní a abstraktní třídy

interface	abstraktní třída
obsahuje pouze abstraktní metody	může obsahovat i neabstraktní metody
obsahuje pouze konstanty	může obsahovat libovolné položky
nemusí mít předchůdce	vždy má nadtřídu (např. `object`)
třída může implementovat více rozhraní	třída může rozšiřovat právě jednu nadtřídu

Třída x abstraktní třída x rozhraní

Rozhraní = interface

„totálně abstraktní třída“

- nemůže obsahovat atributy
- může obsahovat pouze konstanty
- nemůže mít implementované metody
- všechny jeho metody jsou abstraktní metody
- nemůže mít konstruktor
- nelze vytvořit instanci

Abstraktní třída = abstract class

„nedodělaná třída“

- může obsahovat atributy
- může mít implementované metody
- může mít i neabstraktní metody
- může mít konstruktor
- nelze vytvořit instanci (konstruktor lze volat z potomka)

Třída = class

- nesmí obsahovat abstraktní metody

Konec

Ukázka vytváření UML diagramů tříd v Netbeans, R6.7

- Reverzní inženýrství
  - Pravým tlačítkem na projekt
  - Vybrat Reverse engineering, O.K.
  - rozkliknout v UML projektu “… - Model“ záložku „Model“
  - pravým tlačítkem kliknout na balíček z java projektu a zvolit "Create Diagram".
  - Potom vybrat třeba "Class diagram" a mělo by se otevřít okénko

Pro experimety

Abstraktní třída – příklad

- Příklad – návrh menu (a základních funkcí) pro grafický editor

- Třída **Grafický objekt** má metody společné pro všechny podtřídy
- Třída **Grafický objekt** nemá konkrétní instanci
- Od všech následníků požadujeme, aby se uměly vytvořit, zmenšit, posunout, otočit a překlopit

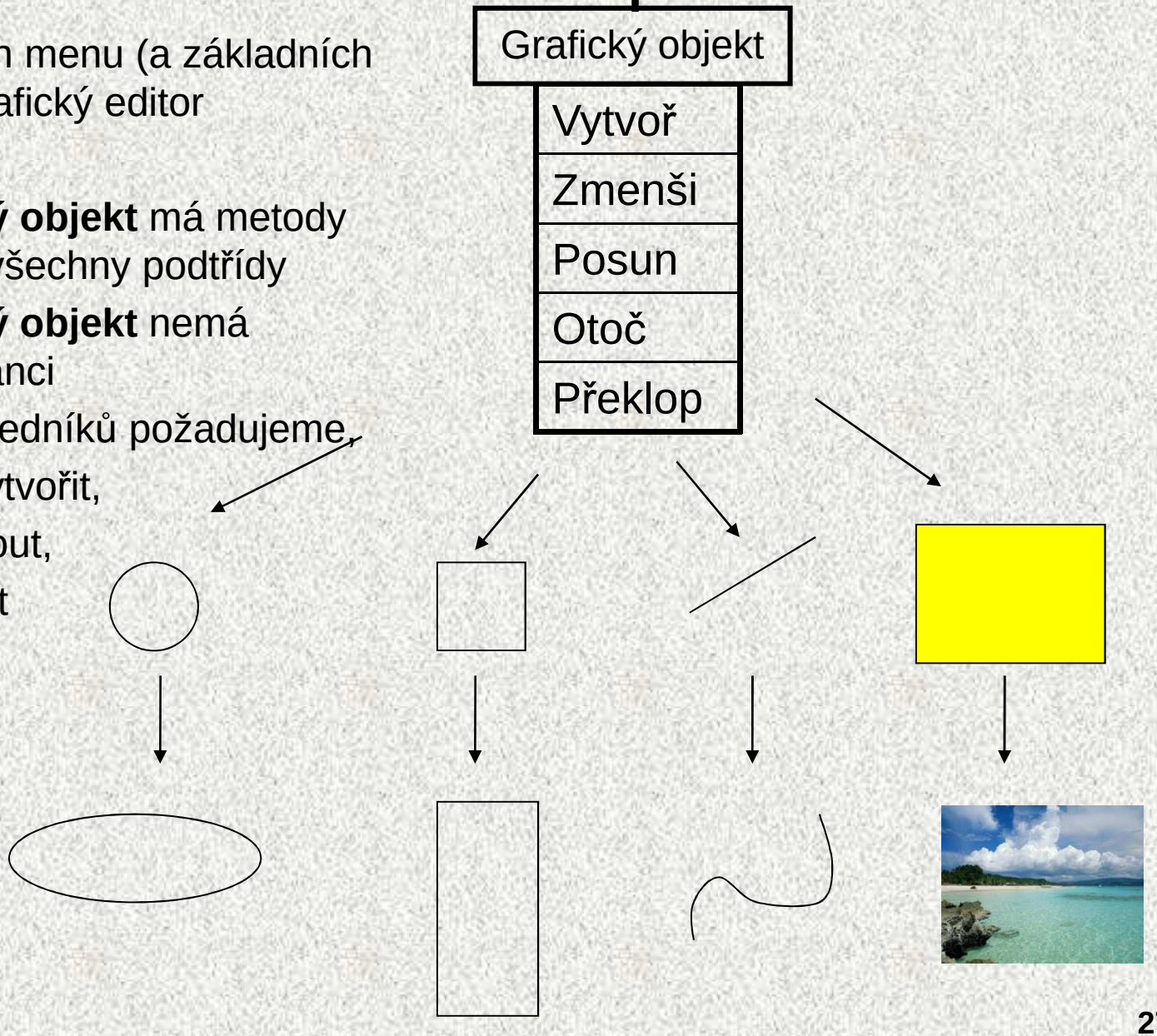
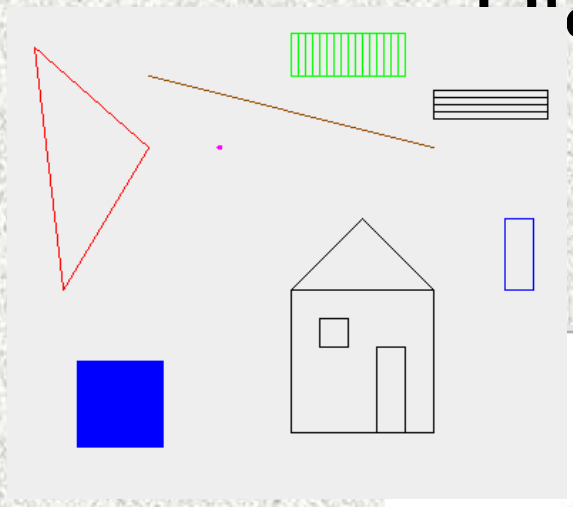
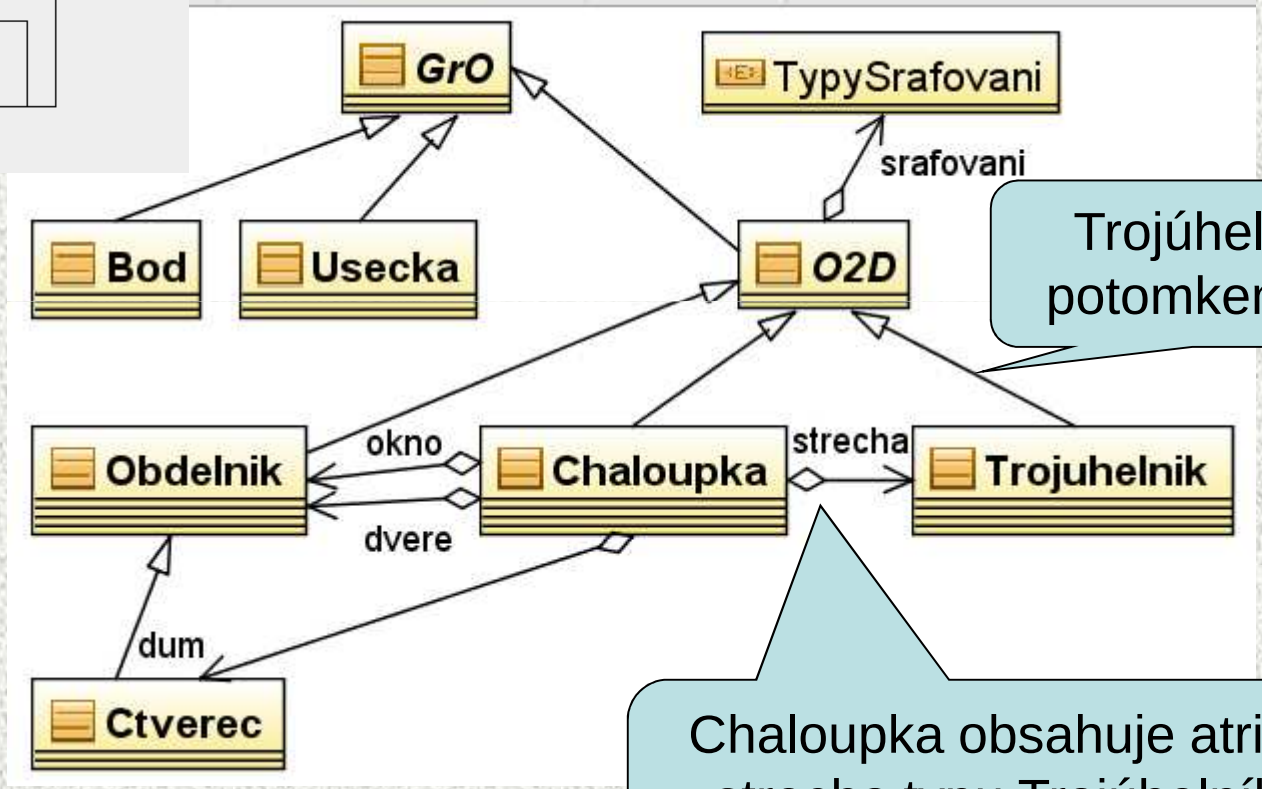


Diagram tříd - příklad

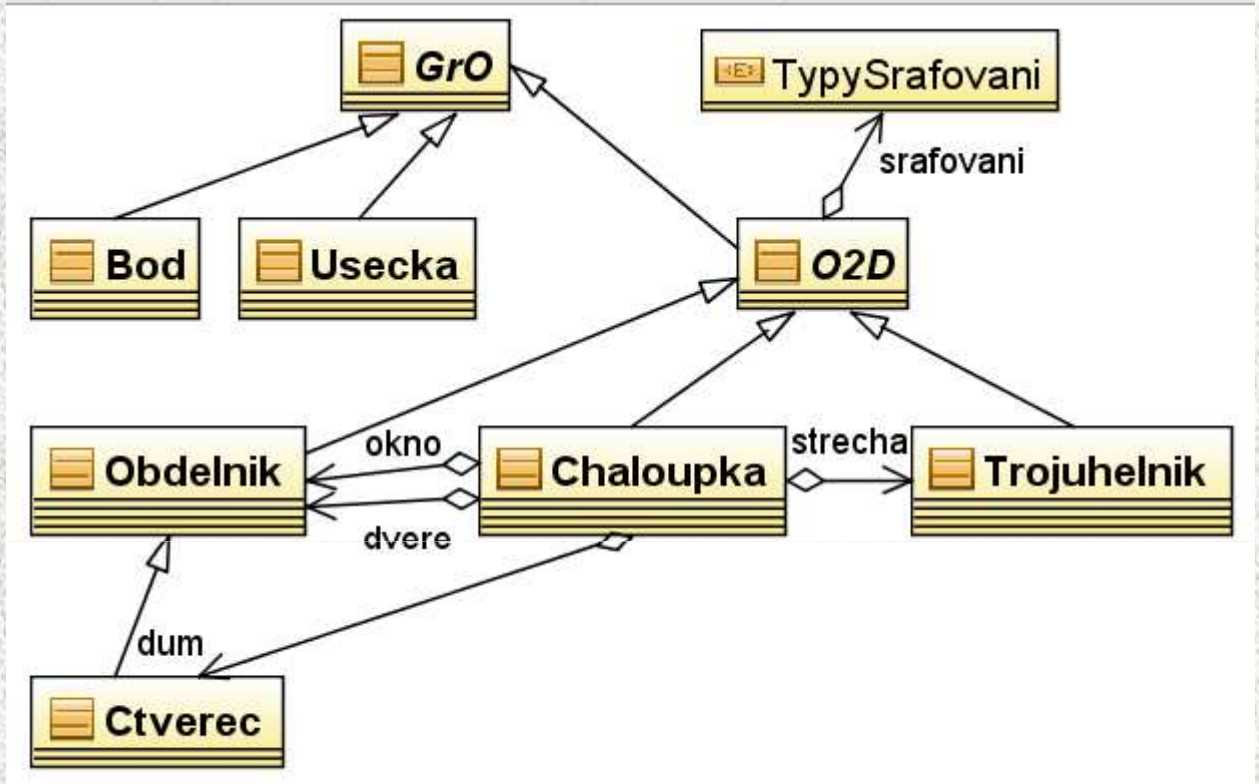




Trojúhelník je potomkem O2D

Chaloupka obsahuje atribut strecha typu Trojúhelník - kompozice

Diagram tříd - příklad



Kam bychom zařadili

- textový řetězec?
- mnohoúhelník?
- městečko?