Analýza a návrh informačných systémov II 4

objektovo orientovaný návrh

Peter Bednár

Zapúzdrenie v UML

- Pre označenie prístupu sa v UML používajú nasledujúce značky pred názvom metódy, alebo členskej premennej:

Značka	Prístup
+	`public`
#	`protected`
-	`private`
~	Triedy z balíka (bez ohraničenia v kóde)

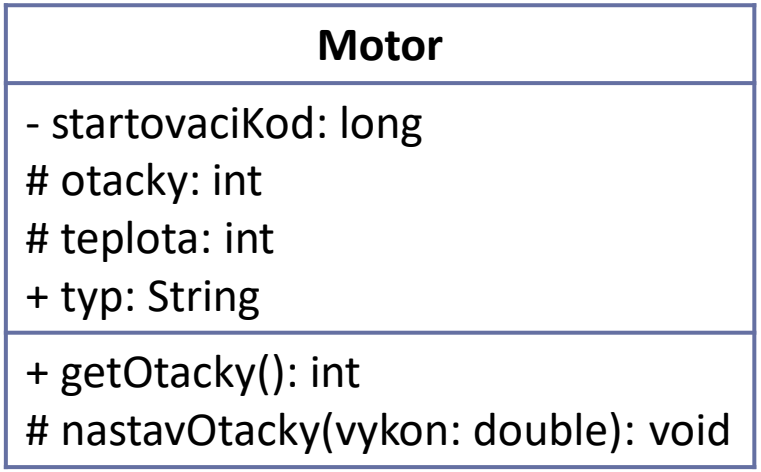
Zapúzdrenie v UML – príklad

```
public class Motor {

    private long startovaciKod;
    protected int otacky;
    protected int teplota;
    public String typ;

    public int getOtacky() {
        return otacky;
    }

    protected void nastavOtacky(double vykon) {
        ...
    }
}
```



Rozhrania

Rozhrania (1)

- Rozhrania sú zovšeobecnením abstraktných tried
- Všetky metódy, ktoré definuje rozhranie sú abstraktné a musia byť implementované v triede, ktorá rozhranie implementuje
- Z rozhrania nie je možné vytvoriť objekt (podobne ako z abstraktnej triedy)
- Trieda môže byť odvodená od jednej nadtriedy, ale môže implementovať viacero rozhraní

Rozhrania (2)

- Rozhrania sa definujú kľúčovým slovom **`interface`**

```
interface Rozhranie {
    definícia metód
}
```

- Všetky metódy, ktoré rozhranie definuje, sú abstraktné (bez tela) a voľne dostupné - kľúčové slová **`abstract`** a **`public`** netreba uvádzať
- Rozhranie nemôže definovať konštruktory

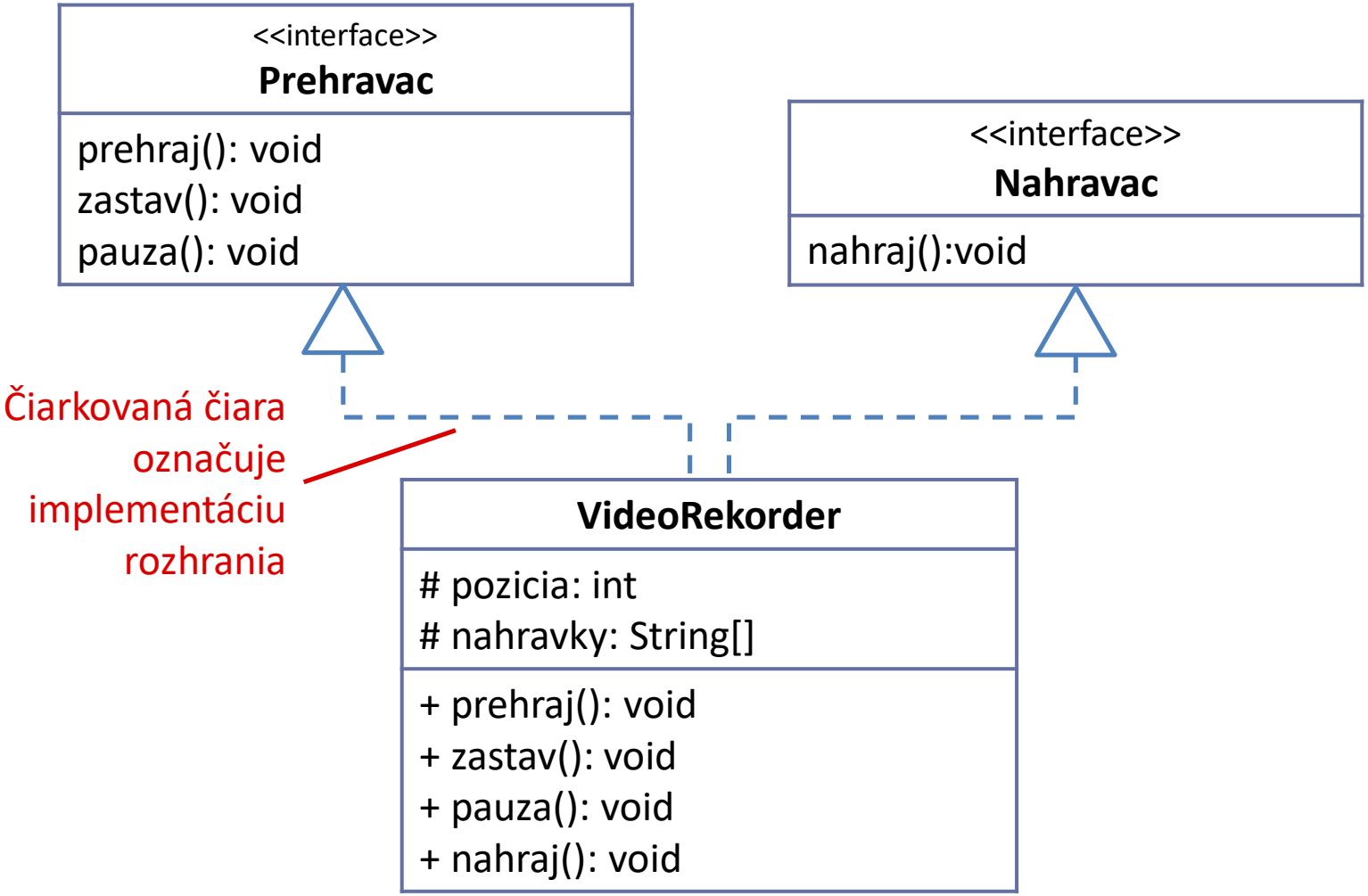
Rozhrania (3)

- Rozhranie môže byť odvodené od iného rozhrania

```
interface Rozhranie extends Nadrozhranie {
    definícia metód
}
```

- Ak trieda implementuje dané rozhranie, musí implementovať všetky metódy aj nadradených rozhraní

Rozhrania v UML - príklad



Rozhrania – príklad (1)

```
public interface Prehravac {
    void prehraj();
    void zastav();
    void pauza();
}
```

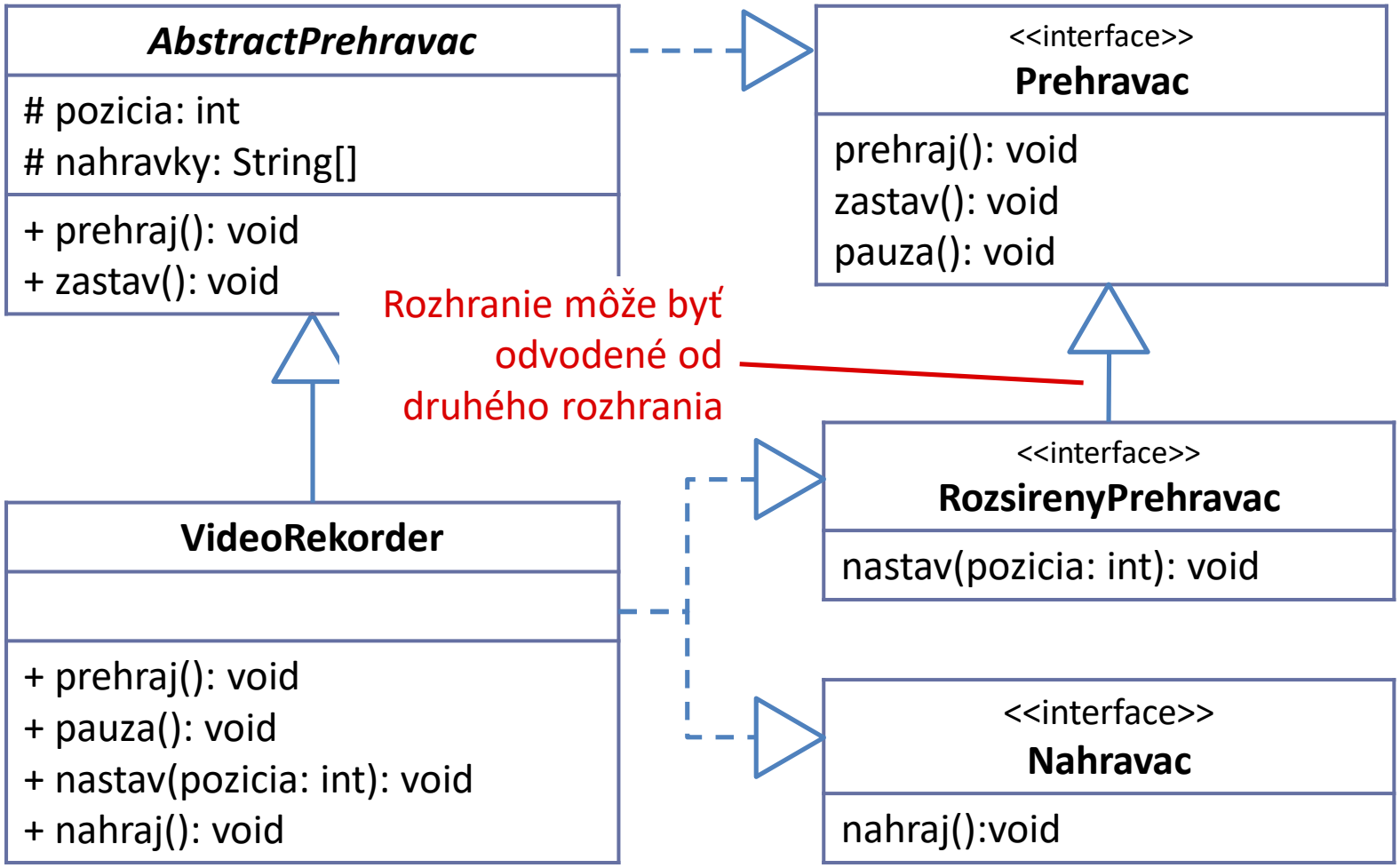
```
public interface Nahravac {
    void nahraj();
}
```

Zoznam rozhraní ktoré trieda implementuje

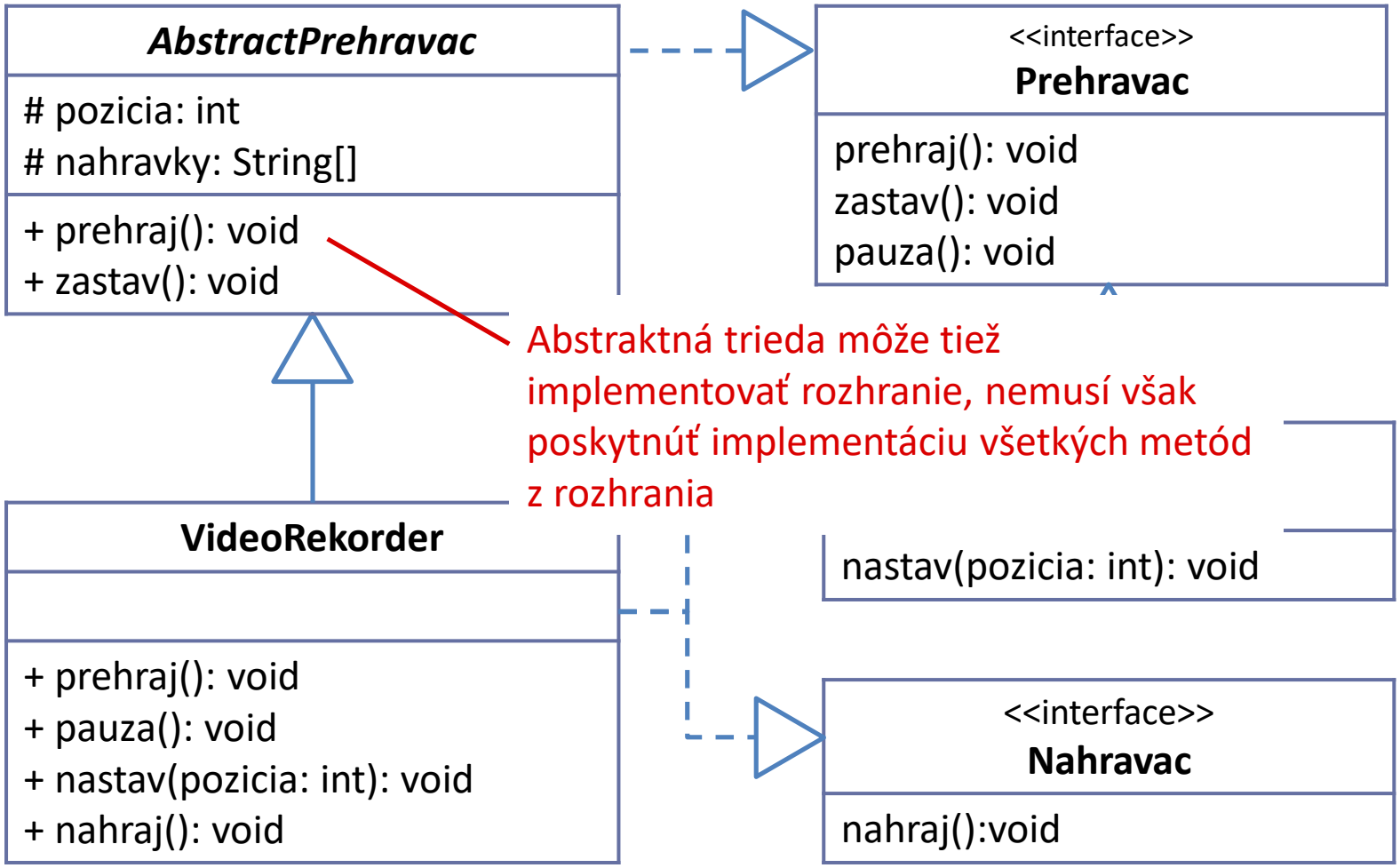
```
public class VideoRekorder implements Prehravac, Nahravac {
    protected int pozicia;
    ...
    @Override
    public void prehraj() {
        System.out.println("prehravam...");
    }
    ...
}
```

Trieda musí implementovať všetky funkcie zo všetkých rozhraní ktoré implementuje

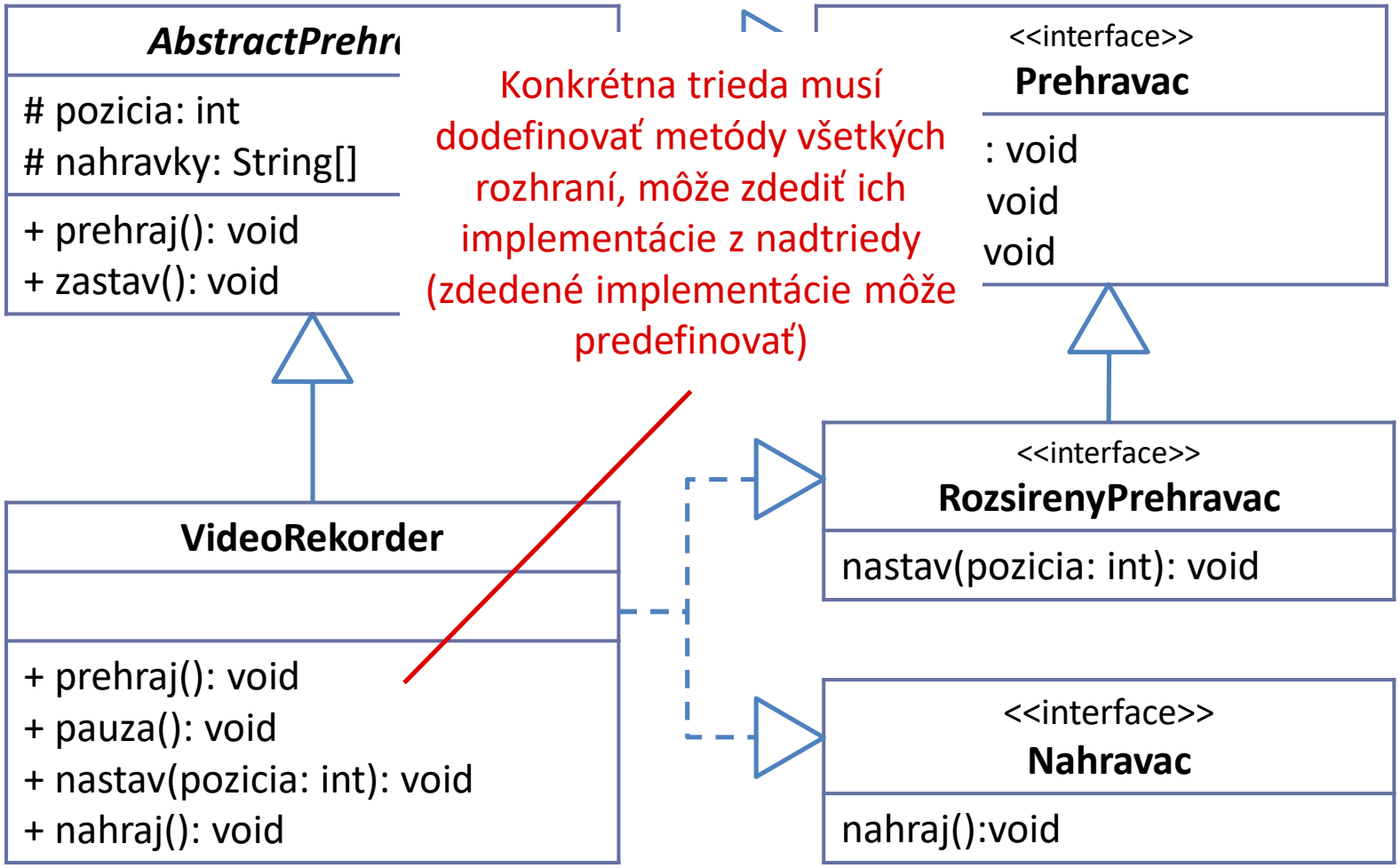
Rozhrania – príklad (2)



Rozhrania – príklad (3)



Rozhrania – príklad (4)



Rozhrania – príklad (5)

```
public interface Prehravac {
    void prehraj();
    void zastav();
    void pauza();
}
```

```
public interface Nahravac {
    void nahraj();
}
```

```
public interface RozsirenyPrehravac extends Prehravac {
    void nastav(int pozicia);
}
```

Rozhrania – príklad (6)

```
public abstract class AbstractPrehravac implements Prehravac {
    protected int pozicia;
    protected String[] nahravky;

    public AbstractPrehravac(String[] nahravky) {
        ...
    }
    @Override
    public void prehraj() {
        ...
    }
    @Override
    public void zastav() {
        ...
    }
}
```

Abstraktná trieda nemusí implementovať všetky metódy rozhrania

Rozhrania – príklad (7)

```
public class VideoRekorder extends AbstractPrehravac
        implements RozsirenyPrehravac, Nahravac {

    public VideoRekorder(String[] nahravky) {
        super(nahravky);
    }
    @Override // predefinovaná z AbstractPrehravac
    public void prehraj() { ... }
    @Override // implementácia Prehravac
    public void pauza() { ... }
    @Override // implementácia RozsirenyPrehravac
    public void nastav(int pozicia) { ... }
    @Override // implementácia Nahravac
    public void nahraj() { ... }
}
```

Rozhrania – príklad (8)

```
String[] nahravky = {"You Don't Own", "I Will Follow Him"};
// rozhranie môžete použiť pre typ premenných
RozsirenyPrehravac prehravac = new VideoRekorder(nahravky);

// k metódam ktoré sú definované v type premennej (rozhranie
// RozsirenyPrehravac môžete pristupovať priamo
prehravac.prehraj();
prehravac.nastav(1);

// to, či trieda implementuje dané rozhranie môžete otestovať
// operátorom instanceof
if (prehravac instanceof Nahravac) {
    // objekt môžeme pretypovať na rozhranie, ktoré implementuje je
    // jeho trieda
    ((Nahravac) prehravac).nahraj();
}
```

Rozhrania a Abstraktné triedy - zhrnutie

- Kedy použiť rozhranie:
  - Ak chcete oddeliť implementáciu od definície typu
  - Ak chcete pre jeden typ poskytnúť viacero implementácií
  - Ak chcete definovať viacero typov, ktorý musí jeden objekt implementovať
- Kedy použiť abstraktnú triedu:
  - Ak chcete poskytnúť spoločnú implementáciu metód

Finálne triedy, metódy a premenné

Finálne typy (1)

- Kľúčové slovo **`final`** je možné použiť pri definícií:
  - tried
  - metód
  - členských premenných
- Pre triedy:
  - Nie je možné od danej triedy odvodiť novú triedu, napr:

```
final class Automobil
```

- Pre metódu:
  - Nie je možné danú metódu predefinovať v podtriede

```
final void nastartuj(int otacky) {
    ...
}
```

Finálne typy (2)

- Pre členské premenné:
  - konštantné premenné, po priradení hodnoty ju nemožno zmeniť, napr.:

```
public class Motor  {

    protected final int MAX_VYKON;
    ...

    public Motor(int maxVykon) {
        MAX_VYKON = maxVykon;
        MAX_VYKON = 10;
    }
    ...
}
```

Konštantné hodnoty sú pomenované veľkými písmenami a jednotlivé slová sú oddelené _

Hodnotu je možné inicializovať v konštruktore (každá inštancia môže byť inicializovaná inak)

CHYBA: po inicializácii sa hodnota už nesmie meniť

Finálne typy (3)

- Ak majú všetky objekty rovnaké hodnoty, je možné ich označiť ako finálne a statické a priamo inicializovať

```
protected static final int MAX_TEPLOTA = 250;
```

Ošetrenie chýb - výnimky

Ošetrenie chýb pomocou výnimiek

- Pri ošetrení chýb pomocou výnimiek sa zjednodušuje prerušenie kódu pri výskyte chyby, jej zachytenie a vykonanie kódu na jej spracovanie
- Výnimky v Jave sa rozdeľujú na:
  - Nekontrolované – kompilátor nekontroluje ich vyvolanie, alebo spracovanie
  - Kontrolované – kompilátor vyžaduje, aby programátor buď chybu ošetril alebo deklaroval, že daná metóda môže chybu vyvolať

Výnimky v Jave (1)

- V Jave sú výnimky reprezentované ako objekty triedy `Exception`
- Java poskytuje podtriedy pre najčastejšie prípady, ako napr.:
  - `IndexOutOfBoundsException` – prístup k indexu mimo rozsahu
  - `NullPointerException` – prístup k objektu cez `null` hodnotu
  - `ClassCastException` – hodnota sa nedá pretypovať na požadovaný typ

Výnimky v Jave (2)

- `IllegalArgumentException` – parameter metódy má chybnú hodnotu (napr. chybný formát reťazca pri prevode na číslo)
- `ArithmeticException` – numerická chyba pri výpočte (napr. delenie 0)

Vyvolanie výnimky

- Triedy výnimiek majú zvyčajne bezparametrický konštruktor a konštruktor s jedným parametrom typu `String`, ktorý reprezentuje popis chyby
- Výnimky sa vyvolávajú príkazom **`throw`** objekt výnimky, napr.:

```
throw new NullPointerException();
throw new IllegalArgumentException("číslo musí byť kladné");
```

Ošetrenie výnimky

- Výnimky sa ošetrujú príkazmi **try**/**catch**:

```
try {
    testovaný kód v ktorom sa môže vyskytnúť chyba
} catch (typ výnimky) {
    kód pre ošetrenie chyby daného typu
}
```

- Za výrazom **try** môže nasledovať aj voliteľný blok **finally**, ktorý sa vykoná vždy, nezávisle či k chybe došlo, alebo nie

Vyvolanie a ošetrenie výnimky – príklad (1)

```
A1  public class Motor {
A2      protected int otacky;
A3      protected float plyn;
A4
A5      public void nastartuj(int plyn) {
A6          this.plyn = plyn;
A7          nastavOtacky(plyn);
A8          System.out.println("naštartované");
A9      }
A10     protected void nastavOtacky(float plyn) {
A11         if (plyn < 0 || plyn > 100) {
A12             throw new IllegalArgumentException("plyn musí byť 0..100");
A13         }
A14         otacky = (int)(50 * plyn + 2000);
A15     }
A16 }
```

Vyvolanie a ošetrenie výnimky – príklad (2)

```
B1  public static void main(String args[]) {
B2      Motor motor1 = new Motor();
B3      Motor motor2 = new Motor();
B4      try {
B5          motor1.nastartuj(10);
B6          motor2.nastartuj(-10);
B7          System.out.println("motory bežia");
B8      } catch (IllegalArgumentException chyba) {
B9          System.out.println(chyba.getMessage());
B10     } finally {
B11         System.out.println("koniec štartovania");
B12     }
B13     System.out.println("koniec programu");
B14 }
```

Vyvolanie a ošetrenie výnimky – príklad (3)

1. Volanie B5 prebehne bez chyby, a vypíše sa správa "naštartované" z A8
2. Pri volaní B6 sa najprv nastaví plyn na A6, potom sa zavolá metóda `nastavOtacky` na A7
3. Keďže parameter `plyn` je mimo rozsahu, vyvolá sa výnimka `IllegalArgumentException` na riadku A12
4. Po prerušení sa A14 nevykoná a ani A8 v nadradenom volaní, tak isto sa preskočí aj B7 a riadenie sa prenesie až do bloku **`catch`**, tzn. nezáleží, kde k výnimke dôjde a ako je volanie metód vnorené, vždy sa nájde najbližší ohraničujúci blok **`try`**/**`catch`**, ktorý ju môže zachytiť

Vyvolanie a ošetrenie výnimky – príklad (4)

5. Keďže sa typ chyby zhoduje s typov v bloku `catch`, chyba sa zachytí a vypíše sa na obrazovku správa "plyn musí byť 0..100" nastavená na A12
6. Blok `finally` sa vykoná vždy, aj keď došlo k chybe a vypíše sa "koniec štartovania" z B11
7. Po ošetrení chyby a vykonaní bloku `finally` sa ďalej pokračuje v hlavnom programe (tzn. nevráti sa vykonávanie na miesto kde došlo k chybe) a vypíše sa "koniec programu" z B13

Vyvolanie a ošetrenie výnimky – príklad (5)

- Ak by sme volanie na B6 opravili

1. Volanie B5 prebehne bez chyby, a vypíše sa správa "naštartované" z A8
2. Volanie B6 prebehne bez chyby, a vypíše sa správa "naštartované" z A8
3. Vypíše sa "motory bežia" na B7
4. Blok `finally` sa vykoná aj bez chyby, tzn. vypíše sa "koniec štartovania"
5. Pokračuje sa za blokom `try`/`catch` a vypíše sa "koniec programu" z B13

Zhrnutie

- Rozhrania
- Finálne triedy, metódy a premenné
- Ošetrenie chýb:
  - `throw`
  - `throws`
  - `try/catch/finally`