Rozhraní

```
<link rel="stylesheet"  href="http://cdnjs.cloudflare.com/ajax/libs/font-
awesome/3.1.0/css/font-awesome.min.css">
```

Motivace

- Jsou objektové jazyky, kde vůbec rozhraní nemáme.
- Jsou dokonce objektové jazyky, kde nemáme ani třídy (JavaScript).
- Většinou ale *třídy jsou*, u většiny OO jazyků: Java, C++, C#, Python, Ruby…
- Přímočaré řešení je pak použít na implementaci nějaké potřebné funkčnosti třídu.
- Problém je, že někdy máme takový požadavek na funkčnost, který se jednou třídou špatně implementuje, resp. evidentně potřebujeme více *zcela různorodých tříd*, které budou tento požadavek plnit.
- Příklad: požadavkem je, aby objekty uměly o sobě sdělit informace, tedy aby měly třeba metodu `retrieveInfo()`.
- Zcela jistě existuje mnoho typů objektů, které o sobě umějí informovat — od psa až po laserovou tiskárnu :-)

Příklad jednoduchého rozhraní

- Velmi jednoduché rozhraní s jedinou metodou:

```
// Informing = can describe information about itself
public interface Informing {
  // this method is used for describing
  String retrieveInfo();
}
```

Přesněji k rozhraní

- Rozhraní (`interface`) je *seznam metod* (budoucích) tříd objektů.
- *Neobsahují vlastní kód* těchto metod.
- Je to tedy *seznam hlaviček metod* s popisem, co přesně mají metody dělat, typicky dokumentačním komentářem.
- Rozhraní by nemělo příliš smyslu, kdybychom neměli třídy, které jej naplňují, realizují, přesněji *implementují*
- Říkáme, že třída *implementuje* rozhraní, pokud obsahuje *všechny metody*, které jsou daným rozhraním předepsány.

- třída *implementuje rozhraní* = objekt dané třídy *se chová jak* rozhraní předepisuje,
- např. osoba nebo pes se chová jako běhající

Deklarace rozhraní

- Stejně jako třída, jedině namísto slova `class` je `interface`.
- Všechny *metody v rozhraní přebírají viditelnost* z celého rozhraní:
  - viditelnost hlaviček metod není nutno psát;
  - rozhraní v našem kurzu budou pouze `public` (není to vůbec velká chyba tak dělat stále).
- Těla metod v deklaraci rozhraní se nepíší vůbec, ani složené závorky, jen středník za hlavičkou.

```
// Informing = can describe information about itself
public interface Informing {
  // this method is used for describing
  String retrieveInfo();
}
```

Implementace rozhraní

- Třídy `Person` a `Donkey` implementují rozhraní `Informing`:

```
public class Person implements Informing {
  public String retrieveInfo() {
    return "People are clever.";
  }
}
public class Donkey implements Informing {
  public String retrieveInfo() {
    return "Donkeys are simple.";
  }
}
```

Když třída implementuje rozhraní, musí implementovat *všechny* jeho metody!

Přetypování třídy na rozhraní

```
public void printInfo(Informing informing) {
  System.out.println("Now you learn the truth!");
  System.out.println(informing.retrieveInfo());
}
...
Person p = new Person();
printInfo(p);
...
Donkey d = new Donkey();
printInfo(d);
```

- Parametr metody je typu rozhraní, můžeme tam tedy použít *všechny třídy, které ho implementují*.
- To je velice časté a užitečné používat jako typ parametru rozhraní.

Přetypování obecně

- Píše se `(typ) hodnota`.
- Ve skutečnosti se nejedná o změnu obsahu objektu, nýbrž pouze o potvrzení (*běhovou typovou kontrolu*), že běhový typ objektu je ten požadovaný.

```
Person p = new Person();
Informing i = (Informing) p;
Object o = (Object) i;
```

U *primitivních typů* se jedná o skutečný převod, např. `long` přetypujeme na `int` a ořeže se tím rozsah.

Proměnná typu rozhraní

- Můžeme taky vytvořit proměnnou typu rozhraní:

```
public class Person implements Informing {
  public String retrieveInfo() {
    return "People are clever.";
  }
  public void emptyMethod() { }
}
...
Informing i = new Person();
i.retrieveInfo(); // ok
i.emptyMethod(); // cannot be done
```

- Proměnná `i` může používat pouze metody definované v rozhraní (ztrácí metody v třídě `Person`).

- To je opět velice časté a užitečné používat jako typ proměnné rozhraní.

Příklad z reálného světa

- máme různé tiskárny, různých značek
- nechceme psát kód, který ošetří všechny značky, všechny typy
- chceme použít i budoucí značky/typy
- vytvoříme pro všechny jedno uniformní rozhraní:

```
public interface Printer {
  void printDocument(File file);
  File[] getPendingDocuments();
}
```

- náš kód bude používat tohle rozhraní, každá tiskárna která ho implementuje, bude kompatibilní
- budoucí tiskárny, které rozhraní implementují ho budou moct používat také

Anotace `@Override`

- Pro jistotu, že *přepisujeme metodu předepsanou rozhraním* a ne jinou, použijeme znovu anotaci `@Override`:

```
public class Person implements Informing {
  @Override
  public String retrieveInfo() {
    return "People are clever.";
  }
}
```

Repl.it demo k rozhraním

- https://repl.it/@tpitner/PB162-Java-Lecture-04-interfaces-Shapes

Implementace více rozhraní I

- Jedna třída může implementovat *více rozhraní*.
- Jednoduše v případě, kdy objekty dané třídy toho "mají hodně umět".
- Příklad: Třída `Person` implementuje 2 rozhraní:

```
public interface Informing { String retrieveInfo(); }
public interface Screaming { String scream(); }

public class Person implements Informing, Screaming {
  public String retrieveInfo() { ... }
  public String scream() { ... }
}
```

Implementace více rozhraní II

- Co kdyby obě rozhraní měla stejnou metodu?

```
public interface Informing { String retrieveInfo(); }
public interface Screaming { String retrieveInfo(); }
```

- Mají-li úplně stejnou hlavičku, je to v pořádku:

```
public class Person implements Informing, Screaming {
  @Override
  public String retrieveInfo() { ... }
}
```

Implementace více rozhraní III

- Mají-li stejný název i parametry, ale různý návratový typ, je to PROBLÉM.

```
public interface Informing { String retrieveInfo(); }
public interface Screaming { void retrieveInfo(); }
public class Person implements Informing, Screaming { ... }
```

- To samozřejmě nelze — viz totéž u přetěžování metod:

```
Person p = new Person();
// do we call method returning void or
// string and we ignore the result?
p.retrieveInfo();
```

Nesnesou se tedy metody lišící se *pouze návratovým typem*.

Rozhraní — vtip

Metody i samotné rozhraní by mělo obsahovat kvalitní dokumentaci s detailním popisem.

> Rozhraní je jako vtip. Když ho třeba vysvělovat, není tak dobré.

Zajímavost — rozhraní bez metod

- Občas se kupodivu používají i prázdná rozhraní, *nepředepisující žádnou metodu*.
- Úplně bezúčelné to není — deklarace, že třída implementuje určité rozhraní, poskytuje typovou informaci o dané třídě.
- Např. `java.lang.Cloneable`, `java.io.Serializable`.