Kapitola 1. Lambda výrazy a spol.

Obsah

- lambda výrazy (*lambda expressions* nebo *lambdas*) jsou od Java 8
- podobné konstrukce se vyskytují i v jiných jazycích
  - Java se inspirovala a vzala si to dobré
- lambda výrazy jsou čím dál tím víc používány a jejich znalost je nezbytná
- výhody:
  - výrazně zjednodušují zápis **krátkých částí** kódu, např. oproti anonymním vnitřním třídám
    - hodí hlavně tam, kde je potřeba **předat jako parametr** akci (tj. část kódu), která se má s daty provést (obsluha tlačítka, výběr dat z kolekce)
  - mohou být deklarovány jako hodnoty proměnných typu funkčního rozhraní
  - mohou být předávány jako parametry metod
  - při použití v kolekcích umožňují využít paralelismus
- zdroje:
  - Java Tutorial
  - `www.lambdafaq.org`

1.1. Funkční rozhraní

- *functional interface*
  - též *Single Abstract Method (SAM)*

Poznámka

Opakování a rozšíření z dřívějška.

- funkční rozhraní jsou klíčová pro použití lambda funkcí – jsou to jejich typy
- je to rozhraní, které má jen jednu abstraktní metodu
  - může mít dále defaultní a/nebo statické metody, což není pro lambda výrazy podstatné
- existuje-li jen jedna metoda, pak ji jednoznačně určuje už samotný název rozhraní
  - to lze dále zobecnit – je-li objekt typu funkčního rozhraní, pak disponuje pouze jednou metodou

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- aby mohl překladač ohlídat unikátnost metody, lze použít anotaci `@FunctionalInterface`

```
@FunctionalInterface
public interface IFunkcniRozhrani {
  public int jedinaMetoda(int a, int b);
}
```

- co se týče počtu formálních parametrů a typu návratové hodnoty, neklade funkční rozhraní žádné omezení

```
@FunctionalInterface
public interface IFunkcniRozhraniVoid {
  public void jedinaMetoda();
}
```

- velmi často bývá funkční rozhraní generické, např.

```
@FunctionalInterface
public interface IFunkcniRozhraniGenericke<T> {
  public void jedinaMetoda(T typ);
}
```

- pokud nechceme využívat lambda výrazy, lze funkční rozhraní používat v běžném smyslu, jako jakákoliv jiná rozhraní

1.1.1. Funkční rozhraní z Java Core API

- protože se očekává stále častější využívání funkčních rozhraní, jsou některé typově nejčastější již připraveny v balíku `java.util.function`
- balík obsahuje asi 70 rozhraní, ale ty jsou často seskupovány následovně:
  - `Consumer` – generické rozhraní – `void accept(T typ)`
  - `IntConsumer` – rozhraní pro práci s typem `int` – `void accept(int value)`
  - `LongConsumer` – rozhraní pro práci s typem `long` – `void accept(long value)`
  - `DoubleConsumer` – rozhraní pro práci s typem `double` – `void accept(double value)`
- přehled skupin generických rozhraní (na jménu metody ve skutečnosti nezáleží – podstatný je počet formálních parametrů a návratový typ):
  - `Consumer` – `void accept(T typ)`
  - `BiConsumer` – `void accept(T typT, U typU)`
  - `Supplier` – `T get()`
  - `Predicate` – `boolean test(T typ)`
  - `Function` – `R apply(T typ)`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- `UnaryOperator` – `T apply(T typ)`

- takže pokud bychom chtěli vytvářet vlastní funkční rozhraní, je vhodné zjistit, zda již v `java.util.function` neexistuje
  - např. místo dříve uvedeného `IFunkcniRozhraniGenericke` by se použilo `Consumer`
- k těmto funkčním rozhraním je třeba přidat podobná rozhraní již známá z dřívějška
  - `java.lang.Comparable` – `int compareTo(T typ)`
  - `java.lang.Iterable` – `Iterator<T> iterator()`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/