Výchozí a statické metody rozhraní

Tomáš Pitner, upravil Marek Šabo

Výchozí a statické metody rozhraní

- Java 8 přidává ohledně metod v rozhraní nové možnosti.
- Neuvidíme je tedy ve starém kódu a mnozí vývojáři je neznají.
- S tím se nemusíme trápit, většinou se tyto postupy hodí pro *modifikace stávajícího kódu* a
- pro návrh nového kódu moc jen omezeně.
- Jde o možnosti, jak již do rozhraní přímo *implementovat funkčnost*, nenechávat to až na třídy.
- To trochu popírá základní princip, že *v rozhraní funkční kód metod není*.
- Proto to má omezené použití a nemělo by se zneužívat.
- Jsou dva nové základní typy metod:
  - statické (static)
  - výchozí (default)

Statické metody

- Rozhraní může počínaje Java 8 obsahovat *statické metody*.
- Ty se píší i chovají stejně jako ve třídě, tj. nesmějí pracovat s atributy a metodami objektu, nýbrž jen celé třídy = *dalšími statickými*.

```
interface A {
  void methodToImplement();
  static void someStaticMethod() {
    /* code inside */
  }
}
...
A.someStaticMethod();
```

Výchozí metody rozhraní

- výchozí = `default`
- Přidávají možnost *implementovat v rozhraní* i určitou funkcionalitu.
- Hlavním smyslem je, aby se při přidávání metod do STÁVAJÍCÍHO rozhraní *nenarušil stávající kód*.
- Jestli přidáme novou metodu do STÁVAJÍCÍHO rozhraní, musíme ji ve všech STÁVAJÍCÍCH implementacích rozhraní implementovat, což znamená ve stávajících třídách dopisovat nové metody atd.
- To vyžaduje většinou *brutální zásah* do stávajícího kódu s riziky vnesené chyb (regrese), přináší pracnost apod.

- Proto obvykle zpětně nepřidáváme do stávajícího rozhraní nic.
- *Výchozí metody* nám umožní udělat výjimku a něco tam přidat bez narušení stávajícího kódu,
- protože výchozí metody rovnou obsahují i svou (výchozí) implementaci.
- V definici rozhraní jsou výchozí metody uvozeny klíčovým slovem default a *obsahují implementaci* metody.

Příklad

```
public interface Addressable {
  String getStreet();
  String getCity();

  default String getFullAddress() {
    return getStreet() +", " + getCity();
  }
}
```

- Výchozí metodu můžeme samozřejmě ve třídách překrýt.
- TIP: Více informací: Java SE 8's New Language Features

Statické a výchozí metody

- Statické metody se mohou v rozhraní využít při psaní výchozích metod:

```
interface A {
  static void someStaticMethod() {
    /* some stuff */
  }
  default void someMethod() {
    // can call static method
    someStaticMethod();
  }
}
```

Významná použití výchozích metod

- Výchozí metody se mohou zdát zbytečností, ale je několik situací, kdy se velmi hodí:
  - *Vývoj existujících rozhraní*: Dříve nebylo možné přidat do existujícího rozhraní metodu, aniž by všechny třídy *implementující toto rozhraní* musely implementovat i novou metodu. Jinak by stávající kód přestal fungovat.
  - *Zvýšení pružnosti návrhu* Výchozí metody nás zbavují nutnosti použít abstraktní třídu pro implementaci obecných metod, které by se jinak v implementujících neabstraktních třídách

opakovaly. Abstraktní třída nutí nejen k *implementaci*, ale i k *dědění*, což narušuje javový koncept preferující implementaci rozhraní před dědičností tříd.

Rozšiřování rozhraní s výchozí metodou

- Mějme rozhraní A obsahující nějakou výchozí metodu, třeba dm().
- Definujeme-li nyní rozhraní B jako rozšíření (extends) rozhraní A, mohou nastat tři různé situace:
  1. Jestliže výchozí metodu dm() v rozhraní B nezmiňujeme, pak se podědí z A.
  2. V rozhraní B uvedeme metodu dm(), ale *jen její hlavičku* (ne tělo). Pak ji nepodědíme, stane se abstraktní jako u každé obyčejné metody v rozhraní a každá třída implementující rozhraní B ji *musí sama implementovat*.
  3. V rozhraní B implementujeme metodu znovu, čímž se původní výchozí metoda překryje — jako při dědění mezi třídami.

Více výchozích metod — chybně

- Následující kód Java 8 (a samozřejmě ani žádná starší) nezkompiluje.

```
interface A {
   default void someMethod() { /*bla bla*/ }
}
interface B {
   default void someMethod() { /*bla bla*/ }
}
class C implements A, B {
   // překladač by nevěděl, kterou someMethod() použít
}
```

Více výchozích metod — překryté, OK

- Následující kód Java 8 (ale samozřejmě žádná starší) bez potíží zkompiluje.

```
interface A {
   default void someMethod() { /*bla bla*/ }
}
interface B {
   default void someMethod() { /*bla bla*/ }
}
class D implements A, B {
// překryjeme-li (dvojitě) poděděnou metodu, není problém
// překladač nemusí "přemýšlet", kterou someMethod() použít
   public void someMethod() {
   // the right stuff, this will be used
   }
}
```

Jedna metoda výchozí, druhá abstraktní

- Následující kód Java 8 opět nezkompiluje.
- Jedno rozhraní default metodu má a druhé ne.

```
interface A { void someMethod(); }
interface B { default void someMethod() { /* whatever */ } }
class E implements A, B {
 // nepřeloží, protože zůstává otázka:
 // má či nemá překladač použít výchozí metodu?
}
```

Dokumentace

- Oracle The Java Tutorial: Default Methods