

Výchozí a statické metody rozhraní

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

Výchozí a statické metody rozhraní

- Jde o možnosti, jak **do rozhraní přímo implementovat funkčnost**, nenechávat to až na třídy.
- Popírá základní princip, že *v rozhraní funkční kód metod není*.
 - ⇒ má omezené použití a nemělo by se zneužívat
- Existují dva základní typy metod:
 - statické — `static`
 - výchozí — `default`

Statické metody

- Rozhraní může obsahovat *statické metody*.
- Statické metody smějí pracovat jen s dalšími statickými metodami a proměnnými.
- Nesmějí pracovat s metodami a atributy objektu.

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

Výchozí metody — motivace

- Nechť existuje rozhraní, které implementuje 10 tříd.
- Do rozhraní chceme přidat novou metodu.
- Metoda musí být (bohužel) implementována ve všech rozhraních!
- Co kdyby rozhraní poskytovalo i svou **výchozí implementaci**, kterou by třídy nemuseli implementovat?
- výchozí = `default`



Výchozí metody — příklad

Výchozí metodu můžeme samozřejmě ve třídách překrýt.

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



Zdroj: [Java SE 8's New Language Features](#)

Výchozí metody — použití

Výchozí metody používáme, když chceme:

- **přidat novou metodu do existujícího rozhraní**
 - všechny třídy implementující rozhraní pak nemusí implementovat novou metodu
- **nahradit abstraktní třídu za rozhraní**
 - abstraktní třída vynucuje dědičnost
 - preferujeme implementaci rozhraní před dědičností tříd

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();  
    }  
}
```

Rozšiřování rozhraní s výchozí metodou

- Mějme rozhraní **A** obsahující výchozí metodu `defaultMethod()`.
- Definujeme-li rozhraní **B** jako rozšíření rozhraní **A**, mohou nastat 3 různé situace:
 1. Jestliže výchozí metodu `defaultMethod()` v rozhraní **B** nezmiňujeme, pak se podědí z **A**.
 2. V rozhraní **B** uvedeme metodu `defaultMethod()`, 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 se **nezkompiluje**:

```
interface A {
    default void someMethod() { /*bla bla*/ }
}
interface B {
    default void someMethod() { /*bla bla*/ }
}
class C implements A, B {
    // compiler does not know which default method should be used
}
```

Více výchozích metod — překryté, OK

Následující kód je zkompiluje:

```
interface A {
    default void someMethod() { /*bla bla*/ }
}
interface B {
    default void someMethod() { /*bla bla*/ }
}
class D implements A, B {
    @Override
    public void someMethod() {
        // now we can define the behaviour
        A.super.someMethod();
    }
}
```

Jedna metoda výchozí, druhá abstraktní

- Následující kód se 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 {  
    // compiler should or should not use default method?  
}
```

Repl.it demo k výchozím a statickým metodám rozhraní

- <https://repl.it/@tpitner/PB162-Java-Lecture-13-intf-default-and-static>