6. Třída Object

Třída Object je v Javě předkem všech tříd (včetně speciálních typů – polí (`array`) a výčtového typu (`enum`)) a jako jediná žádného předka nemá. Pokud při deklaraci své třídy neuvedete slovo `extends` a jméno předka, bude jako předek vaší třídy automaticky dosazena třída `Object`.
Třída `Object` neobsahuje žádné zvnějšku viditelné datové atributy, je zde však definováno několik metod, které se dědí a je možno je používat či překrývat. Koncepcí Javy je též zaručeno, že všechny třídy mají k dispozici tyto metody – zděděné nebo překryté.

6.1. Metoda toString()

Tato metoda vrací textovou reprezentaci objektu, která je snadno čitelná. Metoda se používá automaticky v mnoha situacích, např. pokud je instance parametrem metody `System.out.println()`, metoda **println()** vypíše výsledek metody `toString()`**.** Pokud se při spojování řetězců (použití operátoru + pro spojování řetězců) jako parametr zadá instance, též se na její místo doplní výsledek metody `toString()`. Standardní formát výpisu metody `toString()` třídy `Object` však není příliš vhodný (vypisuje se jméno třídy a výsledek metody `hashcode()`), proto se doporučuje tuto metodu překrýt, pokud ji chceme využívat např. pro kontrolní výpisy.
V projektu `Skola` je ve třídě `Osoba` překryta metoda `toString()` následovně:

```
public String toString() {
     return titulPred+" "+jmeno+" "+titulZa;
}
```

6.2. Metoda equals()

Metoda **equals()** se používá v případě, že chceme zjistit, zda se dva objekty rovnají. Ve třídě `Object` je napsána nejpřísnějším možným způsobem, tj. porovnávané objekty jsou si rovny, pokud oba názvy odkazují na stejnou instanci (na stejné místo v paměti). Vrací tedy stejný výsledek jako použití operátoru ==. Tento způsob porovnávání u věcných tříd obvykle nevyhovuje, a proto je vhodné metodu `equals()` překrýt a porovnávat obsahy. Je potřeba si uvědomit, že se tato metoda používá automaticky v mnoha situacích – např. při ukládání instancí do množin (`Set`), při používání instance jako klíče v mapách (`Map`). Metoda `equals()` je překryta ve většině základních tříd, např. ve třídách `String`, `Integer`.
Při překrývání metody `equals()` musíme dodržet následující pravidla:

- musí být reflexivní, tj. `x.equals(x) == true`,
- musí být symetrická, tj. `x.equals(y)== true` právě tehdy, když `y.equals(x)==true`,
- musí být tranzitivní, tj. když `x.equals(y)==true` a `y.equals(z)==true,` tak `x.equals(z)` musí vrátit také `true`,
- musí být konzistentní, tj. pro nezměněné instance vrací vždy stejnou hodnotu,
- pro parametr `null` vracet hodnotu `false`, tj. `x.equals(null)` vrátí `false.`

Tato pravidla najdete v dokumentaci metody `equals()` ve třídě `Object`. Dále je vhodné, aby se pro porovnání instancí používaly ty datové atributy, které se v průběhu života instance nemění.
Metoda `equals()` ve třídě `Mistnost` v projektu Adventura (viz kapitola19) vypadá takto:

```
    public boolean equals (Object o) {
         if (o instanceof Mistnost) {
              Mistnost druha = (Mistnost)o;
              return nazev.equals(druha.nazev);
         }
         else {
              return false;
         }
    }
```

Z kódu vyplývá, že dvě instance třídy `Mistnost` si jsou rovny, pokud mají stejný název (stejnou hodnotu v datovém atributu `nazev` typu `String`).

6.3. Metoda hashCode()

Metoda **hashCode()** vrací pro každou instanci číslo typu `int`. Využívá se pro optimalizaci ukládání do dynamických datových struktur `HashSet` nebo `HashTable`. Pro metodu `hashCode()` jsou v dokumentaci Javy stanovena tato pravidla:

- je konzistentní, tj. pro nezměněné instance vrací vždy stejnou hodnotu,
- pro dvě instance, které jsou si rovny na základě použití metody `equals()` vrací stejnou hodnotu,
- pro dvě instance, u kterých metoda `hashCode()` vrací stejnou hodnotu, ještě nemusí platit, že jsou si rovny na základě použití metody `equals()`.

Z toho vyplývá, že když překryjete ve své třídě metodu `equals()`, měly byste překrýt i metodu `hashCode()`.
Pro třídu `Mistnost` v projektu `Adventura` je metoda `hashCode()` jednoduchá, neboť využívá metodu `hashCode()` třídy `String`:

```
    public int hashCode() {
         return nazev.hashCode();
    }
```

6.4. Metoda getClass()

Tato metoda nám za běhu programu může vrátit informace o objektu. Vrací je jako instanci třídy **Class**, která je potomkem třídy `Object`. Následujícím kódem za běhu zjistíme jméno třídy pro instanci, na kterou odkazuje proměnná moje:

```
String jmenoTridy = moje.getClass().getName();
```

Metoda `getClass()` má modifikátor `final`, tj. nemůžeme ji překrýt.

6.5. Metoda clone()

Metoda **clone()** se používá pro vytváření identické kopie jedné instance (je potřeba rozlišovat kopii instance od vytvoření dalšího odkazu na stejný objekt, který se vytváří přiřazovacím příkazem). Po provedení metody `clone()` jsou obě instance, původní i nová, na sobě nezávislé. Pokud chceme použít ve vlastní třídě klonování, je nutné metodu `clone()` překrýt a navíc musí naše třída implementovat rozhraní **Cloneable**, jinak bude vždy vyhozena výjimka **CloneNotSupportedException**. Rozhraní `Cloneable` neosahuje žádnou metodu, není tedy třeba implementovat jinou metodu než `clone()`.

6.6. Metoda finalize()

Tuto metodu spouští **Garbage Collector (GC)** při čištění paměti od neplatných objektů. Pokud GC zjistí, že na konkrétní instanci nevede žádný odkaz, spustí tuto metodu. Není ale garantováno, že tato

metoda proběhne – pokud končí aplikace, tak se okamžitě uvolní paměť a metody **finalize()** se nevolají. Z tohoto důvodu nelze metodu `finalize()` používat pro operace, které musí proběhnout, např. pro uzavření souborů.

6.7. Metody notify(), notifyAll(), wait()

Tyto metody se vztahují k použití více vláken **(thread)** v programu. Problematika vláken je mimo rozsah těchto skript.