Konstruktory a metody u `record`

Výhody záznamů

- viz https://medium.com/@reetesh043/record-java-new-feature-daf97797bf3a

Stručnost

Záznamy snižují množství překombinovaného kódu automatickým generováním standardních implementací pro běžné úlohy, jako jsou přístupy ke čtení atributů, metody `equals()`, `hashCode()` a `toString()`. Vývojáři se tak mohou soustředit na definici datových polí, případně i jejich chování.

Neměnnost

Záznamy jsou ve výchozím nastavení navrženy jako neměnné, což znamená, že jakmile je objekt vytvořen, jeho stav nelze změnit. Tato vlastnost je žádoucí v mnoha scénářích, kde je důležitá integrita a konzistence dat. Zejména vede k robustnějšímu kódu u vícevláknových aplikací.

Čitelnost

Při použití záznamů je záměr kódu přehlednější. Záznamy explicitně sdělují, že třída je především datový kontejner s určitou sadou polí.

Jako třída

Záznamy jsou přesto v mnoha ohledech jako běžné třídy - mohou mít konstruktory a další metody.

Z tříd záznamů nelze dědit

- Cestou je vytvořit vždy nový typ, pokud nemá metody, tak to stejně nevadí.
- Důvod je mj. v tom, že bychom neměli záruku, že Account je opravdu jedině Account a ne třeba CheckingAccount.

```
record Account(int id, String name) {}
// won't compile, cannot inherit
record CheckingAccount extends Account {}
```

Záznamy nemohou mít (další) proměnné

- Je to logické: ty proměnné by stejně musely být neměnné

```
record Account(int id, String name) {
    // won't compile, cannot have other attrs
    private int age;
```

```
}
```

Záznamy mohou mít (další) konstruktory

```
public record Account(int id, String name) {
    public Account(int id) {
        // call of the implicit record constructor
        this(id, null);
    }
}
```

Záznamy mohou nahrazovat implicitní konstruktor

```
public record Person(String name, String address) {
    // new syntax: no parentheses after classname
    public Person {
        Objects.requireNonNull(name);
        Objects.requireNonNull(address);
    }
}
```

Záznamy mohou implementovat rozhraní

- Obvykle se proto musí doplnit metody.

```
record Account(int id, String name) implements Runnable, Serializable
```

Záznamy mohou mít další nestatické metody

```
public record Account( int id, String name) {
    public String nameAsUpperCase() {
        return name().toUpperCase();
    }
}
```

Záznamy mohou mít další statické metody

```
public record Account( int id, String name) {
    public static String nameAsUpperCase(Account acc) {
        return acc.name().toUpperCase();
    }
}
```