

# Konstruktory a metody u **record**

## Table of Contents

Výhody záznamů .....	1
Výhody záznamů .....	1
Z tříd záznamů nelze dědit .....	1
Záznamy nemohou mít (další) proměnné .....	2
Záznamy mohou mít (další) konstruktory .....	2
Záznamy mohou nahrazovat implicitní konstruktor .....	2
Záznamy mohou implementovat rozhraní .....	2
Záznamy mohou mít další nestatické metody .....	3
Záznamy mohou mít další statické metody .....	3

## Výhody záznamů

### *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 dat, 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.

## Výhody záznamů

### *Čitelnost*

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

### *Jsou jako třída*

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

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

## Z tříd záznamů nelze dědit

- Cestou je vytvořit vždy nový typ, pokud nemá metody, tak to stejně nevádí.
- Důvod je mj. v tom, že bychom neměli záruku, že `Account` je opravdu jediné `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();  
    }  
}
```