

# Pole, třída *Arrays*

Tomáš Pitner, Radek Ošlejšek, Marek Šabo

# Opakování

- Vytvoření, naplnění a získání hodnot vypadá následovně:

```
int[] array = new int[2];
array[0] = 1;
array[1] = 4;
System.out.println("First element is: " + array[0]);
```

- Deklarace: `typ[] jméno = new typ [velikost];`
- `typ` může být i objektový: `Person[] p = new Person[3];`

## Velikost pole

- *velikost* pole je daná při jejím vytvoření a **nelze ji měnit**
- V budoucnu budeme probírat *kolekce* (seznam, slovník), což je mocnější složený datový typ než pole
  - jejich počty prvků se mohou dynamicky měnit

## Mělká kopie

- Přiřazení proměnné objektového typu (což je i pole) vede pouze k **duplikaci odkazu**, nikoli celého odkazovaného objektu.
- Modifikace jedné proměnné se pak projeví u i té druhé.

```
int[] array = new int[] {1, 4, 7};
int[] array2 = array;
array[1] = 100;
System.out.println(array[1]); // prints 100
System.out.println(array2[1]); // prints 100
```

- Takovému kopírování se říká **mělká kopie** (*shallow copy*).

## Hluboká kopie

Provedeme-li vytvoření nového pole, pak `array2` obsahuje kopii (duplikát) původního pole.

```
int[] array = new int[] {1, 4, 7};
int[] array2 = Arrays.copyOf(array, array.length);
array[1] = 100;
System.out.println(array[1]); // prints 100
System.out.println(array2[1]); // prints 4
```

- Takovému kopírování se říká **hluboká kopie** (*deep copy*).

Metoda `copyOf` bere dva parametry — původní pole a počet prvků, kolik se má nakopírovat.

## Kopie u objektů I

- Obdobně to funguje i u objektů.

```
Person[] people = new Person[] { new Person("Jan"), new Person("Adam")};
Person[] people2 = Arrays.copyOf(people, people.length);
people[1] = new Person("Pepa");
System.out.println(people[1].getName()); // prints Pepa
System.out.println(people2[1].getName()); // prints Adam
```

- Co kdybychom změnili jenom *jméno* (atribut objektu)?

## Kopie u objektů II

- Do cílového pole se zduplikují jenom *odkazy na objekty* `Person`, nevytvoří se kopie objektů `Person`!

```
Person[] people = new Person[] { new Person("Jan"), new Person("Adam")};
Person[] people2 = Arrays.copyOf(people, people.length);
people[1].setName("Pepa"); // changes Adam to PepaList<T> asList(T... a)
System.out.println(people[1].getName()); // prints Pepa
System.out.println(people2[1].getName()); // prints Pepa
```

- Jinými slovy, pole mají sice *různý odkaz* (šipku), ale na **stejný objekt**.
- V předešlém příkladu jsme změnili odkaz na jiný objekt.
- Teď jsme změnili obsah objektu, na který ukazují oba odkazy.

## Třída `Arrays`

- nabízí jen statické metody a proměnné, tzv. utility class
- nelze od ní vytvářet instance
- metody jsou implementovány pro všechny primitivní typy i objekty

- pro jednoduchost použijeme pole typu `long`



Javadoc třídy `Arrays`

## Metody třídy `Arrays` I

- `String toString(long[] a)`
  - vrátí textovou reprezentaci
- `long[] copyOf(long[] original, int newLength)`
  - nakopíruje pole `original`, vezme prvních `newLength` prvků
- `long[] copyOfRange(long[] original, int from, int to)`
  - nakopíruje prvky `from-to`
  - nové pole vrátí
- `void fill(long[] a, long val)`
  - naplní pole `a` hodnotami `val`

## Metody třídy `Arrays` II

- `boolean equals(long[] a, long[] a2)`
  - `true` jestli jsou pole stejná
- `int hashCode(long[] a)`
  - haš pole
- `void sort(long[] a)`
  - setřídí pole
- `... asList(...)`
  - z pole vytvoří kolekci (budou probírány později)

## Příklad

```
long[] a1 = new long[] { 1L, 5L, 2L };
a1.toString(); // [J@4c75cab9
Arrays.toString(a1); // [1, 5, 2]

long[] a2 = Arrays.copyOf(a1, a1.length);
Arrays.equals(a1, a2); // true

Arrays.fill(a2, 3L); // [3, 3, 3]
Arrays.sort(a1); // [1, 2, 5]
```