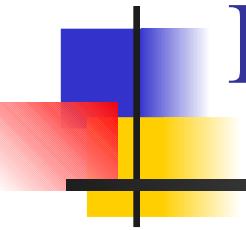


Kolekce

Kolekce (kontejnery, datové struktury)

- Jedná se o skupinu tříd, které slouží pro hromadnou práci s daty.
- Větší množství dat můžete uchovávat v operační paměti ve statickém poli, ale není to praktické.
- V řadě aplikací máte v různých situacích různé požadavky:
  - Uchovávat množství dat v poli, ale předem nevíte, kolik jich bude.
  - Mít strukturu, do které bude možné jednoduše přidávat / odebírat ze začátku a konce.
  - Mít strukturu, ve které budete moci jednoduše najít prvek podle nějakého klíče.
- Toto všechno umožňují kolekce.

Dynamické pole (ArrayList)

- Struktura, která je velice podobná statickému poli, ale nemá omezení maximálního počtu prvků.

Nikde neříkáme, jaká je maximální velikost pole!

```
ArrayList<String> col = new ArrayList<>();
col.add("Prvni");
col.add("Druhy");
col.add("Treti");
System.out.println("prvek na indexu nula: " +
                   col.get(0));
```

Tímto způsobem řeknete, že v poli budou pouze prvky typu String … tomuto se říká Generics a je to od Java SE 5.

Prvky přidáváme metodou add()

Metodou get() získáte prvek na nějakém indexu

- ArrayList je výborná náhrada statického pole, umí to samé a pracuje se s ním velice podobným způsobem.

Evoluce kolekcí

- Do Java SE 5 neexistovaly špičaté závorky, čili bylo možné vytvořit dynamické pole pouze tímto způsobem (a uvnitř mohly být jablka a hrušky):

```
ArrayList col = new ArrayList();
```

  - Tento způsob je v současné době nedoporučovaný (ale funguje a ve starším kódu se s ním setkáte).
- Od Java SE 5 je možné specifikovat typ prvků v poli tímto způsobem:

```
ArrayList<String> col = new ArrayList<String>();
```

- Od Java SE 7 je možné používat diamond operátor <>:

```
ArrayList<String> col = new ArrayList<>();
```

Obousměrně zřetězený seznam (LinkedList) I.

- LinkedList se hodí pro implementaci datových struktur **fronta** (**FIFO** – First In First Out) nebo **zásobník** (**LIFO** – Last In First Out)

```
LinkedList<String> list = new LinkedList<String>();
list.addLast("Druhy");
list.addLast("Treti");
list.addFirst("Prvni");
System.out.println("prvni prvek: " + list.getFirst());
System.out.println("posledni prvek: " +
                   list.getLast());
```

Přidá prvek na konec listu

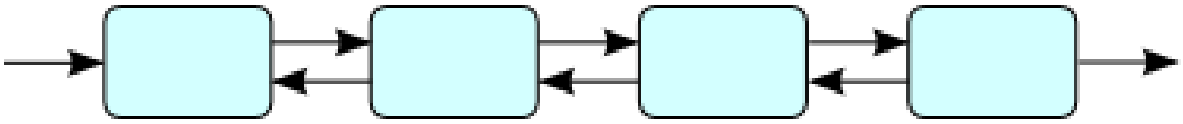
Přidá prvek na začátek listu

Získá prvek ze začátku listu

Získá prvek z konce listu

Obousměrně zřetězený seznam (LinkedList) II.

- **Fronta (FIFO):**
  - Na začátek listu budete vkládat prvky a z konce listu je budete odebírat (nebo obráceně).
  - Jako fronta u přepážky v bance / poště / obchodě. Tato struktura má velké využití.



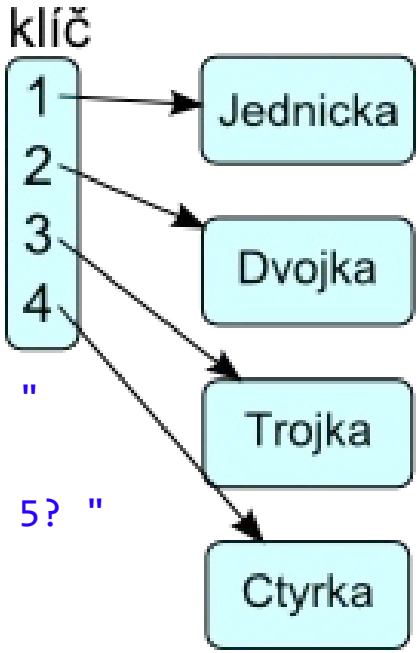
- **Zásobník (LIFO):**
  - Na začátek listu budete vkládat prvky a ze začátku listu je budete také odebírat (nebo obráceně).
  - Moc často se nepoužívá.

Hashovací mapa (HashMap)

- Jedná se o kolekci, která slouží pro efektivní vyhledávání záznamu podle klíče (klíčem i hodnotou, která je na něj namapovaná může být jakýkoli objekt):

```
HashMap<Integer, String> map =
  new HashMap<Integer, String>();
map.put(1, "Jednicka");
map.put(2, "Dvojka");
map.put(3, "Trojka");
map.put(4, "Ctyrka");
System.out.println("Prvek s klicem jedna je: "
  + map.get(1));
System.out.println("Je v mape prvek s klicem 5? "
  + map.containsKey(5));
// Odebere prvek s klíčem 2
map.remove(2);
```

Klíč bude typu Integer a bude namapovaný na hodnotu typu String



Procházení prvků posloupnosti I.

- **Pomocí foreach cyklu**
  - ArrayList a LinkedList:

```
for (String string : col) {
    System.out.println(string);
}
```

  - U map musíte vybrat, jestli chcete procházet hodnoty nebo klíče:

```
Collection<String> values = map.values();
for (String string : values) {
    System.out.println(string);
}
```

Procházení prvků posloupnosti II.

- **Pomocí Iterátoru** – objekt typu `Iterator`:

```
Iterator<String> iterator = col.iterator();
while (iterator.hasNext()) {
    String string = (String) iterator.next();
    System.out.println(string);
}
```

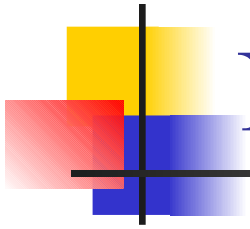
- Jedná se o objekt, pomocí něhož je možné procházet celou řadu posloupností jako je `ArrayList` a `LinkedList`.
- Poznámka: uvnitř foreach cyklu není možné záznam z kolekce odebrat. To je možné pouze při použití iterátoru, který obsahuje metodu `remove()`:
  - http://stackoverflow.com/questions/1196586/calling-remove-in-foreach-loop-in-java

Procházení prvků posloupnosti III.

- **Pomocí enumerátoru** – objekt typu Enumerator:

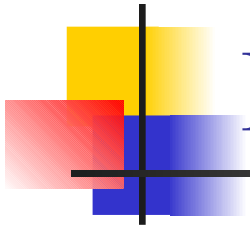
```
Enumeration<Object> enumeration =
  System.getProperties().keys();
while (enumeration.hasMoreElements()) {
    String string = (String)
      enumeration.nextElement();
    System.out.println(string);
}
```

- Starší způsob procházení prvků, který byl v řadě nových struktur nahrazen Iterátorem. Stále se s ním ale můžete setkat.



Některé kontejnery

- **Set:** Kontejner, který nemůže obsahovat duplikované elementy (HashSet, TreeSet)
- **List:** Uspořádaný kontejner. Elementy jsou přístupné přes index (ArrayList, LinkedList)
- **Maps:** Mapuje klíč na hodnotu. Mapování nemůže obsahovat duplikované klíče (HashMap, TreeMap, Hashtable)



Implementace kontejnerů

- **HashSet** – množina s hashovací tabulkou, pořadí prvků není zaručeno, je povoleno vkládání nullových referencí,
- **TreeSet** – uspořádaná (seřazená) množina se stromovou strukturou, většina operací probíhá v logaritmickém čase,
- **ArrayList** – pole proměnné velikosti, přístup přes index a přidávání na konec v konstantním čase, přidávání a odebírání na začátku nebo uvnitř kolekce v lineárním čase,
- **LinkedList** – obousměrně zřetězený dynamický seznam,
- **HashMap** – asociativní kontejner s hashovací tabulkou, klíče nejsou seřazené, je povoleno vkládání nullových referencí,
- **TreeMap** – asociativní kontejner se stromovou strukturou, prvky jsou uchovávány seřazeny podle klíčů.

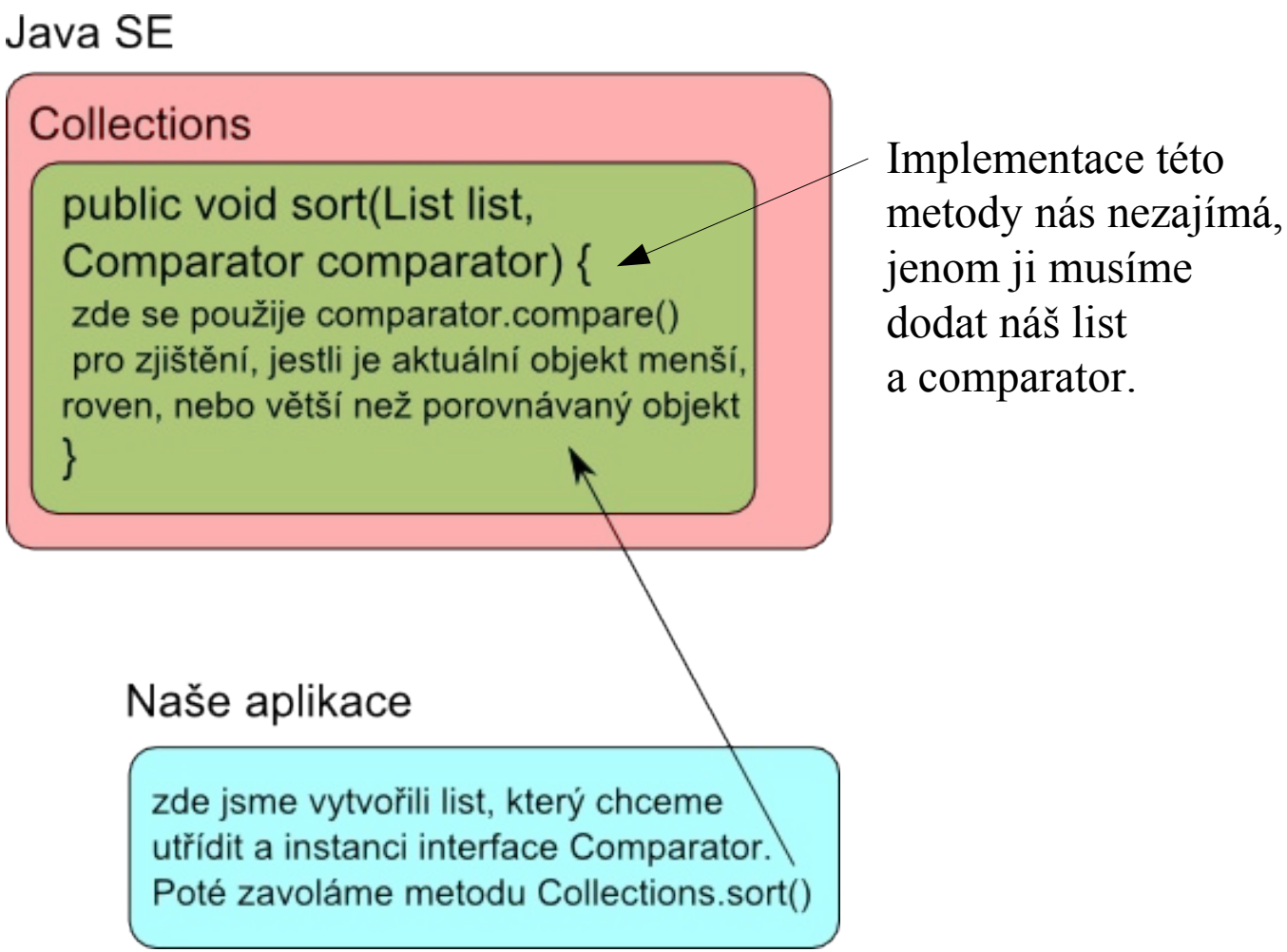
Interface I.

- Interface je podobný třídě, ale je oproti ní velice omezený v tom, může obsahovat pouze:
  - Konstanty (ale nikoli obyč. atributy)
  - Hlavičky metod (ale nikoli jejich těla)
  - Default metody (od Java SE 8)
  - Statické metody (od Java SE 8)
  - Vnitřní třídy
- Není možné vytvořit instanci interface, místo toho je možné:
  - Buď provést implementaci interface a přitom dodefinovat těla metod (pomocí klíč. slova `implements`)
  - Nebo udělat interface, který je potomkem jiného (pomocí klíč. slova `extends`).

Interface II.

- Vzhledem k tomu, že interface neobsahuje implementaci, umožňuje vystavit veřejné rozhraní pro práci s nějakou knihovnou (API – Application Programming Interface).
- Příklad:
  - V Java SE existuje třída `Collections`, která obsahuje metodu `sort()` pro utřídění libovolného listu. Podle jakého kritéria se ale má utřídění provést? Dejme tomu, že uvnitř listu máme instance třídy `Clovek`, která má atributy `id`, `jmeno` a `email`. Jednou chceme utřídit tento list pomocí atributu `id`, jindy přes složený klíč obsahující atributy `jmeno` a `email` atd. Je jasné, že toto kritérium musíme definovat my. Jak na to? Pomocí interface `Comparator`.

Interface III.



Interface IV.

- Vytvořme třídu `Clovek`:

```
public class Clovek {
   public int id;
   public String jmeno;
   public String email;

   public Clovek(int id, String jmeno, String email) {
      this.id = id;
      this.jmeno = jmeno;
      this.email = email;
   }

}
```

Atributy jsou public pro zkrácení kódu v tomto příkladu, jinak by byly private!

Interface V.

- Vytvořme třídu `JmenoComparator`:

```
public class JmenoComparator implements Comparator<Clovek> {

   @Override
   public int compare(Clovek o1, Clovek o2) {
      return o1.jmeno.compareTo(o2.jmeno);
   }

}
```

Tato metoda má vracet -1, 0, nebo 1, podle toho, jestli je první vstupní parametr menší, roven nebo větší druhému.

Tato metoda vrací -1, 0, nebo 1, podle toho, jestli je aktuální objekt (na kterém se tato metoda volá) menší, roven nebo větší porovnávanému objektu.

- http://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html

Interface VI.

- A nakonec metodu `main()`:

```
public static void main(String[] args) {
    ArrayList<Clovek> list = new ArrayList<Clovek>();
    list.add(new Clovek(1, "Jirka Pinkas", "jirka.pinkas@gmail.com"));
    list.add(new Clovek(2, "Aaron Eckhart", "aaron@eckhart.com"));

    Collections.sort(list, new JmenoComparator());

    for (Clovek clovek : list) {
        System.out.println(clovek.jmeno);
    }
}
```

Vytvoření listu

Utřídění listu

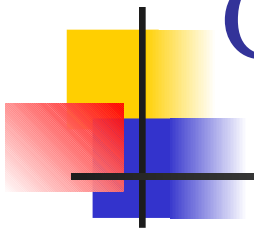
Výpis na konzoli

Interface VII.

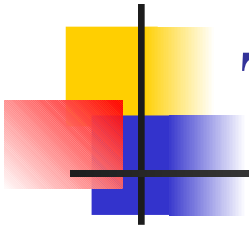
- Nemusíme vytvářet třídu `JmenoComparator`, alternativně místo ní můžeme použít vnitřní anonymní třídu:

```
public static void main(String[] args) {
   ArrayList<Clovek> list = new ArrayList<Clovek>();
   list.add(new Clovek(1, "Jirka Pinkas", "jirka.pinkas@gmail.com"));
   list.add(new Clovek(2, "Aaron Eckhart", "aaron@eckhart.com"));
   Collections.sort(list, new Comparator<Clovek>() {
      @Override public int compare(Clovek o1, Clovek o2) {
         return o1.jmeno.compareTo(o2.jmeno);
      }
   });
   for (Clovek clovek : list) {
      System.out.println(clovek.jmeno);
   }
}
```

Comparator, Comparable



- Interface `Comparator` je univerzální, můžete ho použít pro třídění podle libovolného Vámi definovaného kritéria.
- Na předcházejícím snímku jsme pro porovnání Stringů použili metodu `compareTo()`, která pochází z interface `Comparable`. Řada tříd – wrapper classes (`Integer`, `Double`, …), `String`, `Date` apod. tento interface implementují a můžete ho implementovat i Vy ve Vašich třídách. Má to ale tu nevýhodu, že je pomocí něj možné porovnávat instance nějaké třídy pouze podle jediného kritéria (toho, které je v metodě `compareTo()` implementované).

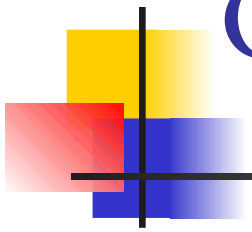


Třída Collections

- V třídě `Collections` jsou zajímavé následující metody pro práci s kontejnery:
  - `sort(List)` – třídění pomocí merge algoritmu
  - `shuffle(List)` – udělá se permutace prvků
  - `reverse(List)` – obrátí pořadí prvků listu
  - `copy(List dest, List src)` – zkopíruje prvky zdrojového listu do cílového listu
  - `binarySearch(List, Object)` – hledá prvek v utříděném seznamu pomocí binárního vyhledávání

Interface & Java SE 8

- Od Java SE 8 je možné v interface mít:
  - Statické metody
    - Užitečné v situaci, kdy byste měli interface a k němu helper metody (statické metody, které používají tento interface). Dřív bylo nutné definovat třídu, která takové metody obsahovala. Příklad takové třídy je třída `Collections`. Kdyby kolekce byly navrhovány v Java 8, pak by tyto metody byly v interface `Collection`.
      - http://docs.oracle.com/javase/8/docs/api/java/util/Collection.html
      - http://docs.oracle.com/javase/8/docs/api/java/util/Collections.html
  - Default metody
    - Instanční metody. V Java 8 se používají pro dodefinování nového chování interface bez rozbití zpětné kompatibility. Například nová metoda v `java.util.Map`:
      - `map.getOrDefault("klic", "default hodnota");`



Generics

- Generics můžete používat i ve vlastních třídách a metodách:
  - http://docs.oracle.com/javase/tutorial/java/generics/index.html