Kolekcie v Jave

Java Collections Framework jednoducho

List, Set, Map a Deque: ako ukladať údaje a vybrať vhodnú kolekciu

1. Čo je kolekcia?

Kolekcia nám umožňuje uložiť viacero objektov spolu a vykonávať nad nimi spoločné operácie.
Napríklad zoznam mien:

```
List<String> names = new ArrayList<>();

names.add("Anna");
names.add("Bohdan");
names.add("Denys");
```

Teraz sú všetky mená uložené v jednom objekte `names`.
S kolekciami môžeme:

- pridávať prvky;
- získavať prvky;
- vyhľadávať prvky;
- odstraňovať prvky;
- prechádzať všetky prvky.

Typ `String` v špicatých zátvorkách

```
List<String>
```

znamená, že tento zoznam uchováva reťazce.

Hlavná myšlienka

Kolekcia je užitočná vtedy, keď chceme pracovať so skupinou objektov namiesto jedného objektu.

2. Základné druhy kolekcií

Java má niekoľko dôležitých rozhraní pre prácu s kolekciami.

List

Uchováva prvky v určitom poradí. Duplicitné prvky sú povolené.
Príklad implementácie:

```
ArrayList
```

Set

Uchováva iba jedinečné prvky.
Príklad implementácie:

```
HashSet
```

Map

Uchováva dvojice:

kľúč ⟶ hodnota.

Príklad implementácie:

```
HashMap
```

Deque
Je vhodný pre rady a zásobníky.
Príklad implementácie:

```
ArrayDeque
```

Dôležité

`Map` stojí trochu mimo ostatných typov kolekcií.
Nededí z `Collection`, pretože neuchováva jednotlivé prvky, ale dvojice „kľúč — hodnota“.

3. List a ArrayList

List je usporiadaný zoznam prvkov.
Umožňuje:

- zachovať poradie prvkov;
- pristupovať k prvkom pomocou indexu;
- ukladať rovnaké prvky viackrát.

Príklad:

```
List<String> names = new ArrayList<>();

names.add("Anna");
names.add("Bohdan");
names.add("Anna");
```

V zozname budú tri prvky.
Získanie prvku podľa indexu:

```
String first = names.get(0);
```

Prvý prvok má index `0`.
Zmena prvku:

```
names.set(1, "Denys");
```

Odstránenie prvku:

```
names.remove(0);
```

Zistenie počtu prvkov:

```
int size = names.size();
```

Dôležité

Pre väčšinu bežných zoznamov je `ArrayList` dobrá prvá voľba.

4. Set a HashSet

Set používame vtedy, keď majú byť prvky jedinečné.
Príklad:

```
Set<String> languages = new HashSet<>();

languages.add("Java");
languages.add("Python");
languages.add("Java");
```

Hoci sme hodnotu `"Java"` pridali dvakrát, uložená bude iba raz.
Overenie, či prvok existuje:

```
boolean exists =
    languages.contains("Java");
```

Odstránenie prvku:

```
languages.remove("Python");
```

Je dôležité si zapamätať:
`HashSet` nezaručuje poradie prvkov.
Ak potrebujeme zachovať poradie pridávania, môžeme použiť:

```
LinkedHashSet
```

Ak majú byť prvky automaticky zoradené:

```
TreeSet
```

5. Map a HashMap

Map uchováva dvojice:

$$\text{kľúč} \longrightarrow \text{hodnota}.$$

Napríklad môžeme priradiť menu žiaka počet bodov:

```
Map<String, Integer> scores =
    new HashMap<>();

scores.put("Anna", 10);
scores.put("Bohdan", 8);
scores.put("Denys", 9);
```

Získanie hodnoty podľa kľúča:

```
int annaScore = scores.get("Anna");
```

Overenie, či kľúč existuje:

```
boolean exists =
    scores.containsKey("Bohdan");
```

Odstránenie dvojice:

```
scores.remove("Denys");
```

Kľúče v `Map` sú jedinečné.
Ak znovu uložíme hodnotu s rovnakým kľúčom:

```
scores.put("Anna", 12);
```

stará hodnota `10` sa nahradí hodnotou `12`.

Dôležité

`HashMap` je vhodná vtedy, keď potrebujeme rýchlo nájsť hodnotu podľa kľúča.

6. Queue, Deque a ArrayDeque

Rad funguje podľa pravidla:

prvý dnu ⟶ prvý von.

Tento princíp sa označuje ako **FIFO**:

First In, First Out.

V Jave je pre bežný rad vhodná trieda `ArrayDeque`.

```
Deque<String> tasks =
    new ArrayDeque<>();

tasks.offerLast("Task A");
tasks.offerLast("Task B");

String next = tasks.pollFirst();
```

Premenná `next` bude obsahovať:

```
"Task A"
```

Pozrieť sa na prvý prvok bez jeho odstránenia:

```
String first = tasks.peekFirst();
```

`Deque` môžeme použiť aj ako zásobník.
Zásobník funguje podľa pravidla:

posledný dnu ⟶ prvý von.

Tento princíp sa označuje ako **LIFO**:

Last In, First Out.

```
Deque<String> stack =
    new ArrayDeque<>();

stack.push("A");
stack.push("B");

String top = stack.pop();
```

Tu sa odstráni prvok `"B"`.

7. Ako rýchlo vybrať kolekciu

Pre väčšinu školských úloh stačia nasledujúce pravidlá.

- Potrebujeme bežný zoznam s indexmi:

```
ArrayList
```

- Potrebujeme iba jedinečné prvky:

```
HashSet
```

- Potrebujeme nájsť hodnotu podľa kľúča:

```
HashMap
```

- Potrebujeme jedinečné prvky a zachovať poradie pridávania:

```
LinkedHashSet
```

- Potrebujeme dvojice „kľúč — hodnota“ a zachovať poradie pridávania:

```
LinkedHashMap
```

- Prvky majú byť automaticky zoradené:

```
TreeSet
```

- Kľúče mapy majú byť zoradené:

```
TreeMap
```

- Potrebujeme rad alebo zásobník:

```
ArrayDeque
```

8. Prechádzanie prvkov

Jedným z najjednoduchších spôsobov, ako prejsť všetky prvky zoznamu, je cyklus `for-each`.

```
for (String name : names) {
    System.out.println(name);
}
```

Rovnakým spôsobom môžeme prechádzať aj `Set`:

```
for (String language : languages) {
    System.out.println(language);
}
```

Pri `Map` môžeme prejsť všetky dvojice „kľúč — hodnota“:

```
for (Map.Entry<String, Integer> entry
        : scores.entrySet()) {

    System.out.println(
        entry.getKey()
        + ": "
        + entry.getValue()
    );
}
```

Tu:

- `entry.getKey()` vráti kľúč;
- `entry.getValue()` vráti hodnotu.

9. Zoraďovanie

Bežný zoznam môžeme zoradiť.
Napríklad:

```
List<String> names =
    new ArrayList<>();

names.add("Denys");
names.add("Anna");
names.add("Bohdan");

Collections.sort(names);
```

Po zoradení budú prvky v abecednom poradí:

```
Anna
Bohdan
Denys
```

Čísla sa prirodzene zoraďujú od menšieho po väčšie.
Ak chceme, aby sa jedinečné prvky automaticky uchovávali v zoradenom poradí, môžeme použiť `TreeSet`:

```
Set<Integer> numbers =
    new TreeSet<>();

numbers.add(5);
numbers.add(2);
numbers.add(8);
```

Poradie prvkov bude:

```
2, 5, 8
```

10. Rýchlosť základných operácií

Pri výbere kolekcie je užitočné vedieť, ako rýchlo sa zvyčajne vykonávajú základné operácie.
Označenia:

- $O(1)$ — operácia zvyčajne trvá približne konštantný čas;

- $O(\log n)$ — čas rastie pomaly so zvyšujúcim sa počtom prvkov;
- $O(n)$ — v najhoršom prípade treba prejsť veľa alebo všetky prvky.

Kolekcia	Operácia	Zvyčajne
`ArrayList`	prístup podľa indexu	$O(1)$
`ArrayList`	vyhľadanie prvku	$O(n)$
`HashSet`	vyhľadanie / pridanie	$O(1)^*$
`HashMap`	vyhľadanie / pridanie podľa kľúča	$O(1)^*$
`TreeSet`	vyhľadanie / pridanie	$O(\log n)$
`TreeMap`	vyhľadanie / pridanie	$O(\log n)$
`ArrayDeque`	pridanie / odstránenie na krajoch	$O(1)$

* V priemere. V niektorých prípadoch môže operácia trvať dlhšie.

Dôležité

Pri prvých programoch nevyberajte kolekciu iba podľa časovej zložitosti.
Najprv si premyslite, **čo presne** potrebujete s údajmi robiť.

11. equals() a hashCode()

Pri `HashSet` a kľúčoch v `HashMap` je dôležité správne určiť, kedy sa dva objekty považujú za rovnaké.
Java na to používa metódy:

```
equals()
hashCode()
```

Zjednodušene:
equals()
odpovedá na otázku:

„Považujú sa tieto objekty za rovnaké?“

hashCode()
vracia špeciálne číslo, ktoré pomáha hašovacím kolekciám rýchlejšie nájsť objekt.
Hlavné pravidlo:

Dôležité

Ak sú dva objekty rovnaké podľa `equals()`, musia mať rovnaký `hashCode()`.

Pre štandardné typy v Jave, napríklad

```
String
Integer
Long
```

sú tieto metódy už správne implementované.
Pri vytváraní vlastných tried sa táto téma preberá podrobnejšie.

12. Nemodifikovateľné kolekcie

Niekedy potrebujeme vytvoriť malú kolekciu, ktorú po vytvorení nemožno meniť.
Napríklad:

```
List<String> colors =
    List.of("red", "green", "blue");
```

Z takéhoto zoznamu môžeme čítať:

```
String first = colors.get(0);
```

Nemôžeme však pridať nový prvok:

```
colors.add("yellow"); // Chyba
```

Pre mapu:

```
Map<String, Integer> scores =
    Map.of(
        "Anna", 10,
        "Bohdan", 8
    );
```

Takéto kolekcie sú užitočné vtedy, keď má množina údajov zostať nezmenená.

13. Časté chyby

Chyba 1.
Očakávať konkrétne poradie prvkov v `HashSet` alebo `HashMap`.
Ich poradie nie je zaručené.

Chyba 2.
Zabudnúť, že indexy zoznamu začínajú od nuly.
Pre zoznam s tromi prvkami sú platné indexy:

$$0, \quad 1, \quad 2.$$

Chyba 3.
Očakávať, že `Set` zachová duplikáty.

```
Set<String> set = new HashSet<>();

set.add("Java");
set.add("Java");
```

V množine zostane iba jeden prvok.

Chyba 4.
Zabudnúť, že vloženie novej hodnoty do `Map` pod rovnakým kľúčom nahradí pôvodnú hodnotu.

```
scores.put("Anna", 10);
scores.put("Anna", 12);
```

Hodnota pre `"Anna"` je teraz `12`.

Chyba 5.
Odstraňovať prvky priamo z kolekcie vnútri cyklu `for-each`.
Pri jednoduchom odstraňovaní podľa podmienky je vhodnejšie použiť:

```
names.removeIf(String::isBlank);
```

14. Stručný prehľad

ArrayList
Bežný usporiadaný zoznam. Má indexy a povoľuje duplicitné prvky.
HashSet
Množina jedinečných prvkov. Poradie nie je zaručené.
HashMap
Uchováva dvojice „kľúč — hodnota“. Každému kľúču zodpovedá jedna hodnota.
LinkedHashSet
Jedinečné prvky so zachovaním poradia pridávania.
LinkedHashMap
Dvojice „kľúč — hodnota“ so zachovaním poradia pridávania.
TreeSet
Jedinečné prvky uchovávané v zoradenom poradí.
TreeMap
Dvojice „kľúč — hodnota“ so zoradenými kľúčmi.
ArrayDeque
Vhodná implementácia radu alebo zásobníka.

Hlavné pravidlo výberu

Najprv si položte tri otázky.

1. Potrebujem zachovať určité poradie prvkov?
2. Môžu sa prvky opakovať?
3. Ako chcem údaje vyhľadávať: podľa indexu, hodnoty alebo kľúča?

Potom je výber zvyčajne jednoduchý:

- bežný zoznam — `ArrayList`;
- jedinečné prvky — `HashSet`;
- kľúč a hodnota — `HashMap`;
- rad alebo zásobník — `ArrayDeque`.

Nie je potrebné vyberať najzložitejšiu kolekciu.
Vyberte tú, ktorá najlepšie zodpovedá konkrétnej úlohe.

Táto pomôcka je určená na prvé zoznámenie s kolekciami v Jave. Príklady sú zámerne jednoduché, aby vysvetlili hlavné myšlienky a najčastejšie operácie.