

СОРТУВАННЯ ОБ'ЄКТІВ JAVA

У Java існують два інтерфейси, призначені для порівняння об'єктів: [Comparable](#) та [Comparator](#).

Comparable об'єкт може порівнювати себе із іншими об'єктами. Для порівняння своїх екземплярів клас повинен імплементувати інтерфейс Comparable.

Компаратор-об'єкт (Comparator) може порівнювати два різних об'єкти. Клас порівнює не власні екземпляри, а екземпляри інших класів. Клас компаратора повинен імплементувати інтерфейс Comparator.

Кожен із інтерфейсів має метод, котрий повинен бути імплементований.

java.lang.Comparable: int compareTo(Object o1) — метод порівнює поточний екземпляр класу із об'єктом o1 і повертає один із можливих результатів:

- додатній – даний об'єкт більший за o1;
- 0 – даний об'єкт та o1 – однакові;
- від'ємний – даний об'єкт менший за o1.

java.util.Comparator: int compare(Object o1, Object o2) – метод порівнює об'єкти o1 та o2 і повертає один із можливих результатів:

- додатній – об'єкт o1 більший за o2;
- 0 – об'єкти o1 та o2 – однакові;
- від'ємний – об'єкт o1 менший за o2.

Методи java.util.Collections.sort(List) та java.util.Arrays.sort(Object[]) можуть бути використані для природного сортування об'єктів.

Методи java.util.Collections.sort(List, Comparator) and java.util.Arrays.sort(Object[], Comparator) можуть бути використані для порівняння об'єктів за наявності відповідного об'єкта Comparator.

Нехай, скажімо, існує клас, котрий має два поля: int id та int x. Скажімо, природнім сортуванням вважатимемо сортування на основі порівнянь за полем id. Для такого сортування можна скористатися інтерфейсом Comparable. Уявімо, що виникла потреба сортування за полем x. Можна змінити реалізацію методу compareTo(). При цьому втратиться попередня реалізація порівняння за полем id. У такому випадкові слід створити Comparator.

Сортування Object[] o

```
import java.util.Arrays;
public class Main {
    public static void main(String[] args) {
        int[] arr = {32, 2};
        Arrays.sort(arr);
        arr.toString();
        for (int var : arr) {
            System.out.println(var);
        }
    }
}
```

Результат:

2
32

У даному прикладі масив arr містить елементи примітивного типу (int). Для int існує відповідний клас-обгортка Integer, котрий імплементує інтерфейс Comparable. Батьківським класом для Int є клас Object. Змінимо тип елементів масиву на String та ініціалізуємо його елементи значеннями відповідного типу:

```
import java.util.Arrays;
public class Main {
    public static void main(String[] args) {
        String[] arr = {"world", "hello"};
        Arrays.sort(arr);
        arr.toString();
        for (String var : arr) {
            System.out.println(var);
        }
    }
}
```

Результат:

```
hello
world
```

Arrays.sort(Object o) сортує елементи String за рахунок того, що клас String імплементує інтерфейс Comparable. Батьківським класом для String є клас Object.

Сортування об'єктів за допомогою Comparable

Створимо клас, який імплементує інтерфейс Comparable. При цьому клас повинен реалізувати метод `int compareTo(T o)`:

```
import java.util.Arrays;
class Data implements Comparable<Data> {
    private int x;
    public Data(int x) {
        this.x = x;
    }
    @Override
    public int compareTo(Data o) {
        int result = 0;
        if (x < o.x)
            result = -1;
        else if (x > o.x)
            result = 1;
        return result;
    }
    public int getX() {
        return x;
    }
}
```

```

    }
    public class Main {
        public static void main(String[] args) {
            Data var1 = new Data(123);
            Data var2 = new Data(2);
            Data var3 = new Data(321);
            Data[] arr = {var1, var2, var3};
            Arrays.sort(arr);
            arr.toString();
            for (Data var : arr) {
                System.out.println(var.getX());
            }
        }
    }
}

```

Результат:

```

2
123
321

```

Напрям сортування можна змінити, помінявши місцями -1 та 1 у методі `compareTo(Data o)`.

Оскільки тип поля `x` класу `Data` – `int`, то порівняння можна реалізувати наступним чином:

```

@Override
public int compareTo(Data o) {
    return Integer.compare(x, o.getX());
}

```

У такому випадкові елементи будуть посортовані за зростанням.

Сортування List

```

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
public class Main {
    public static void main(String[] args) {
        List<Integer> lst = new ArrayList<>();
        lst.add(11);
        lst.add(5);
        lst.add(77);
        System.out.println(lst.toString());
    }
}

```

```
        Collections.sort(lst);
        System.out.println(lst.toString());
    }
}
```

Результат:

```
[11, 5, 77]
[5, 11, 77]
```

При додаванні елементів до ArrayList у даному випадкові використовується [autoboxing](#), оскільки замість типу Integer параметри методу add() мають тип примітивний тип int. Елементи ArrayList у даному випадкові сортуються за допомогою методу sort() класу Collections за рахунок того, що клас Integer імплементує інтерфейс Comparable.

Сортування об'єктів за допомогою Comparator

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Comparator;
class Data {
    private int x;
    public Data(int x) {
        this.x = x;
    }
    public int getX() {
        return x;
    }
    public void setX(int x) {
        this.x = x;
    }
}
class CustomComparator implements Comparator<Data> {
    @Override
    public int compare(Data o1, Data o2) {
        return ((Integer) o1.getX()).compareTo(o2.getX());
    }
}
public class Main {
    public static void main(String[] args) {
        Data d1 = new Data(4);
        Data d2 = new Data(1);
        Data d3 = new Data(0);
        List<Data> arr = new ArrayList<>();
        arr.add(d1);
        arr.add(d2);
        arr.add(d3);
    }
}
```

```
        Collections.sort(arr, new CustomComparator());  
        for (Data element : arr) {  
            System.out.println(element.getX());  
        }  
    }  
}
```

Результат:

0
1
4