Java Developer

Java Core

Sumário

Objetivos deste Módulo

Ao final deste módulo, objetiva-se que o aluno adquira:

Conhecimentos:

- Ordenação de coleções com e sem uso de stream

1. Ordenação de coleções

Para ordenar uma lista, utiliza-se o **Collections.sort(l)**.

Funciona automaticamente para classes do Java como String, Long, etc.

Em classes que criamos, implementamos a interface **Comparable**:

```
public class Conta implements Comparable<Conta> {

    private String nome;
    private double saldoTotal;

    // construtor default - sem argumentos
    public Conta(){
    }

    // construtor recebendo nome
    public Conta(String nome) {
        this.nome = nome;
    }

    public double getSaldoTotal() {
        return saldoTotal;
    }

    public void setSaldoTotal(double saldoTotal) {
        this.saldoTotal = saldoTotal;
    }

    // somente leitura
    public String getNome() {
        return nome.substring(0,20) + "...";
    }

    // escrever
    public void setNome(String nome) {
        this.nome = nome;
    }

    @Override
    public int compareTo(Conta proximaConta) {
        if (this.saldoTotal < proximaConta.saldoTotal) {
            return -1; //Se Menor
        }
        if (this.saldoTotal > proximaConta.saldoTotal) {
            return 1; //Se Maior
        }

        // Se quiser na ordem inversa:
        /*if (proximaConta.saldoTotal < this.saldoTotal) {
            return -1; //Se Menor
        }
        if (proximaConta.saldoTotal > this.saldoTotal) {
            return 1; //Se Maior
```

```
        }*/
        return 0; //Se Igual
    }


    // https://www.devmedia.com.br/java-object-class-entendendo-a-classe-object/30513
    @Override
    public String toString() {
        //return "Nome " + nome + " R$ " + saldoTotal;

        return String.format("Nome %s R$ %.2f \n", nome, saldoTotal);
    }

}
```

====

Vamos testar:

```
import java.util.*;

public class ContaListagem {

    public static void main(String[] args) {

        List<Conta> contas = new ArrayList<>();

        Conta c1 = new Conta();
        c1.setNome("Rogerio");
        c1.setSaldoTotal(1_000);

        Conta c2 = new Conta("Lucas");
        c2.setNome("Lucas");
        c2.setSaldoTotal(50_000);

        Conta c3 = new Conta("Gabriela");
        c3.setNome("Gabriela");
        c3.setSaldoTotal(80);

        contas.add(c1);
        contas.add(c2);
        contas.add(c3);

        System.out.println("Ordenando por saldo da conta");
        Collections.sort(contas);
        System.out.println(contas);
    }
}
```

===

Console:

```
Ordenando por saldo da conta
[Nome Gabriela R$ 80,00
, Nome Rogerio R$ 1000,00
, Nome Lucas R$ 50000,00
]
```

Ou podemos criar uma **classe especializada** através da interface **Comparator**:

```
public class Cliente {
    public Integer cpf;
    public String nome;

    @Override
    public String toString() {
        return "cpf: " + cpf + " nome: " + nome;
    }
}
```

====

```
// Ordena pelo cpf do cliente
import java.util.Comparator;

public class ClienteByCpfComparator implements Comparator<Cliente> {


    @Override
    public int compare(Cliente clienteAtual, Cliente outroCliente) {
        return clienteAtual.cpf.compareTo(outroCliente.cpf);
    }
}
```

====

```
// Ordena pelo nome do cliente
import java.util.Comparator;

public class ClienteByNomeComparator implements Comparator<Cliente> {

    public int compare(Cliente c1, Cliente c2) {
        return c1.nome.compareTo(c2.nome);
    }
}
```

====

```
// classe para validar as ordenações
import java.util.*;

public class ClienteListagem {

    public static void main(String[] args) {

        Cliente cliente1 = new Cliente();
        cliente1.nome = "João";
        cliente1.cpf = 123456;

        Cliente cliente2 = new Cliente();
        cliente2.nome = "Adriano";
        cliente2.cpf = 456789;

        Cliente cliente3 = new Cliente();
        cliente3.nome = "Zalando";
        cliente3.cpf = 876543;

        List<Cliente> clientes = new ArrayList<>();
        clientes.add(cliente1);
        clientes.add(cliente2);
        clientes.add(cliente3);

        System.out.println("Ordenando pelo nome do cliente");
        ClienteByNomeComparator comparator = new ClienteByNomeComparator();
        Collections.sort(clientes, comparator);
        System.out.println(clientes);

        System.out.println("Ordenando pelo cpf do cliente");
        ClienteByCpfComparator comparatorByCpf = new ClienteByCpfComparator();
        Collections.sort(clientes, comparatorByCpf);
        System.out.println(clientes);


    }
}
```

E agora se utilizarmos o Stream para ordenar as listas?

Para ordenar uma lista com o uso de Stream em Java, você pode usar a função **sorted**() do Stream. Aqui está um exemplo de como fazer isso:

Suponha que você tenha uma lista de números inteiros e deseja ordená-los em ordem crescente:

```
public class StreamSortingExample {

    public static void main(String[] args) {

        List<Integer> numbers = List.of(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5);

        // Usando Stream para ordenar a lista
        List<Integer> sortedList = numbers.stream()
                .sorted()
                .collect(Collectors.toList());
```

```
        System.out.println("Lista ordenada: " + sortedList);

        List<Integer> reversedSorted = numbers.stream()
                .sorted(Comparator.reverseOrder())
                .collect(Collectors.toList());

        System.out.println("Lista ordenada inversa: " + reversedSorted);
    }
}
```

Neste exemplo, primeiro você cria um Stream a partir da lista de números e, em seguida, chama o método .**sorted**() no Stream para ordená-lo em ordem crescente. Por fim, você coleta os elementos ordenados de volta em uma lista usando .collect(Collectors.toList()).

Para ordenar na ordem decrescente, utilize o reverseOrder(), desta forma:

.**sorted**(Comparator.**reverseOrder**())

Você pode fazer o mesmo para ordenar listas de objetos personalizados, mas você precisa fornecer um **comparador personalizado** para o método **sorted**(). Por exemplo, se você tiver uma lista de objetos Person e desejar ordená-los pelo nome, você pode fazer o seguinte:

```
class Person {

    private String name;
    private Integer age;

    public Person(String name, Integer age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }


    public Integer getAge() {
        return age;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```

```
import java.util.Comparator;
import java.util.List;
import java.util.stream.Collectors;

public class StreamPessoaSortingExample {

    public static void main(String[] args) {

        List<Person> people = List.of(
                new Person("Alice", 20),
                new Person("Bob", 50),
                new Person("Charlie", 30),
                new Person("David", 5)
        );

        List<Person> sortedPeople = people.stream()
                .sorted((p1, p2) -> p1.getName().compareTo(p2.getName()))
                .collect(Collectors.toList());

        List<Person> sortedPeople2 = people.stream()
                .sorted(Comparator.comparing(Person::getName))
                .collect(Collectors.toList());

        System.out.println("Pessoas ordenadas por nome: " + sortedPeople);
        System.out.println("Pessoas ordenadas por nome: " + sortedPeople2);

        List<Person> sortedPeopleByAge = people.stream()
                .sorted((p1, p2) -> p1.getAge().compareTo(p2.getAge()))
                .collect(Collectors.toList());
        System.out.println("Pessoas ordenadas por idade: " + sortedPeopleByAge);

        List<Person> sortedPeopleByAge2 = people.stream()
                .sorted((p1, p2) -> Integer.compare(p1.getAge(), p2.getAge()))
                .collect(Collectors.toList());

        System.out.println("Pessoas ordenadas por idade: " + sortedPeopleByAge);
        System.out.println("Pessoas ordenadas por idade com compare: " + sortedPeopleByAge2);

    }
}
```

2. Exercícios

- Crie uma classe `**Produto**` com os atributos: `nome`, `custoAquisicao`, `valorVenda`.

- Crie uma classe `**ExercicioOrdenacao**` com o método `main`.

- Crie os produtos na lista.

- Imprima a lista:

+ Ordenada por `**nome**`.
+ Ordenada por `**custoAquisicao**`.

Produtos:

Nome, CustoAquisicao, ValorVenda

Creme Dental 90g, R$ 2,49, R$ 2,99
Sabonete em Barra Corporal 90g, R$ 2,99, R$ 3,30
Protetor Solar 30 FPS 200ml, R$ 37,39, R$ 39,12
Fralda P Confort - 50 Unidades, R$ 44,90, R$ 44,90
Condicionador 200ml, R$ 18,90, R$ 19,50

```
Produto cremeDental = new Produto();
cremeDental.setNome("Creme Dental 90g");
cremeDental.setCustoAquisicao(2.49);
cremeDental.setValorVenda(2.99);

Produto sabonete = new Produto();
sabonete.setNome("Sabonete em Barra Corporal 90g");
sabonete.setCustoAquisicao(2.99);
sabonete.setValorVenda(3.30);

Produto protetorSolar = new Produto();
protetorSolar.setNome("Protetor Solar 30 FPS 200ml");
protetorSolar.setCustoAquisicao(37.39);
protetorSolar.setValorVenda(39.12);

Produto fralda = new Produto();
fralda.setNome("Fralda P Confort - 50 Unidades");
fralda.setCustoAquisicao(44.90);
fralda.setValorVenda(44.90);

Produto condicionador = new Produto();
condicionador.setNome("Condicionador 200ml");
condicionador.setCustoAquisicao(18.90);
condicionador.setValorVenda(19.50);
```

Realizar esse exercício sem uso de stream e com seu uso.