LINGUAGEM DE PROGRAMAÇÃO JAVA

Cefet – Maracanã -RJ

BCC/TSI

Prof. Gustavo Guedes

E-mail: gustavo.guedes@cefet-rj.br

Collections

- Já vimos que as listas são percorridas de maneira pré-determinada de acordo com a inclusão dos itens.
- Muitas vezes queremos uma lista ordenada.
- A classe Collections traz um método estático sort que recebe um List como argumento e ordena por ordem crescente.

Collections.sort

- List lista = new ArrayList();
- lista.add("Manoel");
- lista.add("José");
- lista.add("Marcia");
- System.out .println(lista);
- Collections.sort (lista);
- System.out .println(lista);

COLLECTIONS.SORT

- Como ordenar os objetos de uma classe desenvolvida por nós?

```
public class Conta {
    private double saldo;
    private int cpf;
    public Conta (int cpf, double saldo) {
        setSaldo(saldo);
        setCpf(cpf);
    }
    public double getSaldo() {
        return saldo;
    }
    public void setSaldo(double saldo) {
        this.saldo = saldo;
    }
    public int getCpf() {
        return cpf;
    }
    public void setCpf(int cpf) {
        this.cpf = cpf;
    }
    public String toString() {
        return "cpf: " + cpf + " saldo: " + saldo;
    }
}
```

COLLECTIONS.SORT

- A interface Comparable

```
public class Conta implements Comparable<Conta> {
    private double saldo;
    private String cpf;
    public Conta (String cpf, double saldo) {
        setSaldo(saldo);
        setCpf(cpf);
    }
    public double getSaldo() {
        return saldo;
    }
    public void setSaldo(double saldo) {
        this.saldo = saldo;
    }
    public String getCpf() {
        return cpf;
    }
    public void setCpf(String cpf) {
        this.cpf = cpf;
    }
    public String toString() {
        return "cpf: " + cpf + " saldo: " + saldo;
    }
    @Override
    public int compareTo(Conta o) {
        return cpf.compareTo(o.cpf);
    }

}
```

Collections.sort

- Desta forma nossa classe se tornou um comparável. Repare que o critério de ordenação é totalmente aberto, definido pelo programador.
- Se quisermos ordenar por outro/outros atributo/atributos, basta modificar a implementação do método compareTo na classe.
- Agora sim, o método sort de Collections saberá ordenar nossas contas.

COLLECTIONS.FREQUENCY

- Para contar quantos objetos “o” existentes em “c”, basta utilizar o método frequency.
- Collections.frequency(Collection c, Object o) -> Qual método que precisa ser implementado no objeto para isso funcionar?

- List lista = new ArrayList();
- lista.add("Manoel");
- lista.add("José");
- lista.add("Marcia");
- System.out .println(lista);
- Collections.sort (lista);
- System.out .println(lista);
- Collections.frequency(lista, “Marcia”)

COLLECTIONS.FREQUENCY

- Para contar quantos objetos “o” existentes em “c”, basta utilizar o método frequency.
- Collections.frequency(Collection c, Object o) -> Qual método que precisa ser implementado no objeto para isso funcionar? equals

- List lista = new ArrayList();
- lista.add("Manoel");
- lista.add("José");
- lista.add("Marcia");
- System.out .println(lista);
- Collections.sort (lista);
- System.out .println(lista);
- Collections.frequency(lista, “Marcia”)

Collections.Shuffle

- Para embaralhar os elementos de uma coleção, basta utilizar o método Collections.shuffle(Colletion c)

- List lista = new ArrayList();
- Collections.shuffle(lista);

Collections.copy

- Para copiar os elementos de uma Lista para outra
- Collections.copy(List dest, List font)