Exercícios com Collections.

1) Utilize o projeto fornecido pelo professor.
2) No pacote de testes, crie a classe TestaCollection1 conforme abaixo.

```
 8 public class TestaCollection {
 9     public static void main(String[] args) {
10         ArrayList lista1 = new ArrayList();
11         lista1.add("Rafael");
12         lista1.add("Maria");
13         lista1.add("Fulano");
14         lista1.add("Rafael");
15         //Percorre a lista
16         for (int i = 0; i < lista1.size(); i++)
17             System.out.println(lista1.get(i));
18         //Verifica e remove a 1ª ocorrencia do elemento
19         if(lista1.contains("Rafael"))
20             lista1.remove("Rafael");
21 
22         //Ordena a lista e percorre novamente
23         Collections.sort(lista1);
24         for (int i = 0; i < lista1.size(); i++)
25             System.out.println(lista1.get(i));
26 
27         lista1.remove("Maria");
28         //Adiciona um elemento diferente
29         lista1.add(false);
30         //Vai dar erro (ClassCastException) porque boolean não é ordenável
31         //Collections.sort(lista1);
32         for (Object elemento : lista1)
33             System.out.println(elemento);
34     }
35 }
```

3) Na linha 9 teste fazer uso do polimorfismo. Declare lista1 como List (pacote java.util.list) e mude a implementação para Vector(). Perceba que não vai haver problemas. Agora vc pode escolher utilizar ArrayList, Vector ou LinkedList uma vez que todas são listas (

```
10         List lista1 = new Vector();//new ArrayList();//new LinkedList();
```

4) No pacote de testes, crie a classe TestaCollection2 e conheça um pouco sobre o funcionamento dos conjuntos.

```java
public class TestaCollection2 {

    public static void main(String[] args) {
        Set conjunto = new HashSet();//new LinkedHashSet
        conjunto.add("Rafael");
        conjunto.add("Maria");
        conjunto.add("Fulano");
        conjunto.add("Rafael");//Vai retornar false e não vai adicionar

        System.out.println(conjunto);

        for (Object elemento : conjunto)
            System.out.println(elemento);

        if(conjunto.contains("Maria"))
            System.out.println("Maria esta no conjunto");
        else
            System.out.println("Maria não esta no conjunto");

    }
}
```

5) No pacote de testes, crie a classe TestaCollection3. Veja como funciona uma lista heterogênea e quais são os problemas de se lidar com isso.

```java
    public static void main(String[] args) {
        List lista1 = new ArrayList();
        lista1.add("Rafael");
        lista1.add("Maria");
        lista1.add("Fulano");
        ContaCorrente cc = new ContaCorrente(2);
        cc.deposita(5000);
        lista1.add(cc);
        ContaPoupanca cp = new ContaPoupanca(1);
        cp.deposita(2500);
        lista1.add(cp);

        // O método sort não sabe ordenar contas
        // Collections.sort(lista1);

        // Percorre a lista
        for (int i = 0; i < lista1.size(); i++) {
            if (lista1.get(i) instanceof Conta) {
                Conta conta = (Conta) lista1.get(i);
                System.out.println("Conta nº "+conta.getNumero()+" com saldo de R$"+conta.getSaldo());
            } else
                System.out.println(lista1.get(i));
        }
    }
}
```

6) Para resolver o problema, utilizamos Generics para definir com que tipo de dado nossa estrutura vai trabalhar. Assim não precisamos nos preocupar em fazer casting. Escreva a classe TestaCollection4 conforme abaixo e teste você mesmo.

```java
public class TestaCollection4 {
    public static void main(String[] args) {
        List<Conta> lista1 = new ArrayList();
        ContaCorrente cc = new ContaCorrente(2);
        cc.deposita(5000);
        ContaPoupanca cp = new ContaPoupanca(1);
        cp.deposita(2500);
        ContaCorrente cc2 = new ContaCorrente(5);
        cc2.deposita(3000);
        lista1.add(cc);
        lista1.add(cp);
        lista1.add(cc2);

        for (int i = 0; i < lista1.size(); i++) {
            Conta conta = lista1.get(i);
            System.out.println("Conta nº "+conta.getNumero()
            +" com saldo de R$"+lista1.get(i).getSaldo());
        }

        System.out.println();
        //Como ordenar Contas????
        //Collections.sort(lista1);

        for (Conta conta : lista1) {
            System.out.println("Conta nº "+conta.getNumero()
            +" com saldo de R$"+conta.getSaldo());
        }
    }
}
```

7) Não sabemos como ordenar contas. Vamos definir que contas são ordenáveis pelo seu saldo. Faça com que conta implemente a interface Comparable e forneça uma implementação para o método compare. Utilize genérics para facilitar sua vida. Veja como ficaria.

```java
public class Conta implements Comparable<Conta>{
    // Atributos
    protected Cliente titular; //Apenas getTitular (IMUTÁVEL)
    protected double saldo; //getSaldo, saca, deposita, transferePara
    protected int numero; //getNumero, setNumero

    @Override
    public int compareTo(Conta outraConta) {
        if(this.saldo>outraConta.saldo)
            return 1;
        else if(this.saldo<outraConta.saldo)
            return -1;
        else
            return 0;
    }

    //Construtor que exige um número
```

8) Agora descomente a linha 33 de TestaCollection4 e rode a aplicação novamente. Você vai perceber que no foreach, depois da ordenação, as contas estarão ordenadas pelo saldo.
9) Na verdade, o método estático sort de Collections espera uma lista de elementos Comparable. Para que um Comparable saiba se ordenar não é necessário retornar 1, -1 ou 0. Na verdade basta retornar um número positivo, um número negativo ou 0. Veja uma outra forma muito mais simples de se fazer a mesma coisa com uma única linha.

```
3  public class Conta implements Comparable<Conta>{
4      // Atributos
5      protected Cliente titular; //Apenas getTitular (IMUTÁVEL)
6      protected double saldo; //getSaldo, saca, deposita, transferePara
7      protected int numero; //getNumero, setNumero
8
9      @Override
10     public int compareTo(Conta outraConta) {
11         /*if(this.saldo>outraConta.saldo)
12             return 1;
13         else if(this.saldo<outraConta.saldo)
14             return -1;
15         else
16             return 0;*/
17         return (int) (this.saldo - outraConta.saldo);
18     }
19
20     //Construtor que exige um número
```

10) Agora vamos definir que contas são ordenáveis pelo seu número. Faça com que conta implemente a interface Comparable e forneça uma implementação para o método compare. Utilize genérics para facilitar sua vida. Veja como ficaria.

```
3  public class Conta implements Comparable<Conta>{
4      // Atributos
5      protected Cliente titular; //Apenas getTitular (IMUTÁVEL)
6      protected double saldo; //getSaldo, saca, deposita, transferePara
7      protected int numero; //getNumero, setNumero
8
9      @Override
10     public int compareTo(Conta outraConta) {
11         /*if(this.saldo>outraConta.saldo)
12             return 1;
13         else if(this.saldo<outraConta.saldo)
14             return -1;
15         else
16             return 0;*/
17         //return (int) (this.saldo - outraConta.saldo);
18         return (this.numero - outraConta.numero);
19     }
20
21     //Construtor que exige um número
```

11) Rode TestaCollection4 novamente. Você vai perceber que no foreach, depois da ordenação, as contas estarão ordenadas pelo número.