UNIVERSIDADE VILA VELHA-ES

Programação Java III

Prof. Vinicius Rosalen

Java Collection Framework

- Com o objetivo de permitir a manipulação mais eficiente de estruturas de dados,
  - A linguagem Java traz implementada, a partir da versão 1.2, a *Java Collection Framework*

- É uma coleção de interfaces e classes que implementam alguns dos algoritmos e estruturas de dados mais comuns...
  - Por exemplo, não é necessário reinventar a roda e criar uma lista ligada mas sim utilizar aquela que a API disponibilizou.

UNIVERSIDADE VILA VELHA-ES

Java Collection Framework

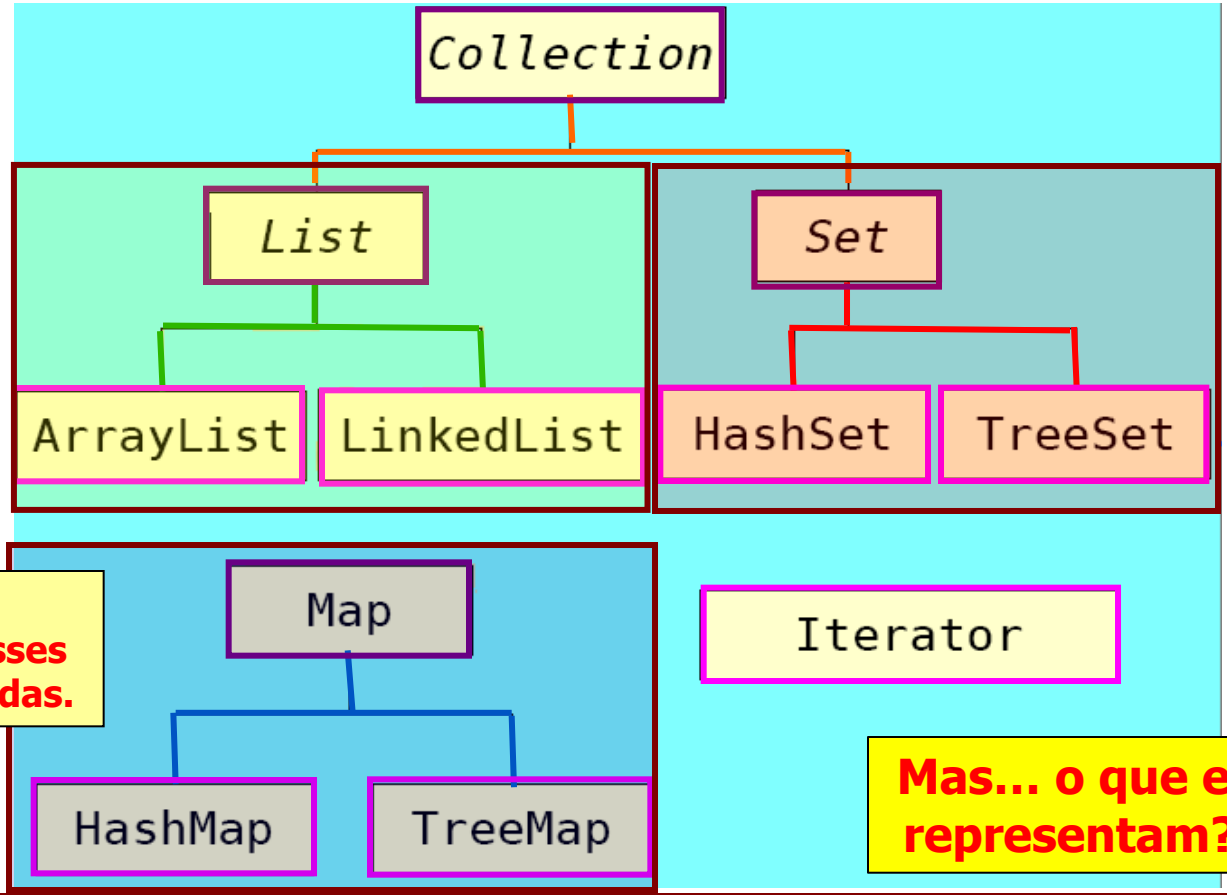
Quais as vantagens da Java Collection Framework ?

- Reduzem o esforço de programação
- Estruturas de dados e algoritmos variados e prontos para usar
- Aumentam a velocidade e a qualidade do programa
- Permitem a interoperabilidade entre APIs não relacionadas
  - Várias APIs utilizam Collections para trocar dados
- Reduzem o esforço para aprender e utilizar novas APIs:
- Reduzem o esforço para se projetar novas APIs
- Estimulam o reuso de software

Java Collection Framework

Hierarquia (vamos falar mais sobre isso logo logo..)

- Os elementos que compreendem a estrutura de coleções estão no pacote java.util.



List - Coleções indexadas

- O primeiro recurso que a API de Collections traz são Listas ou Coleções indexadas

- Uma lista é uma coleção que permite:
  - Elementos duplicados,
  - Mantém uma ordenação específica entre os elemento,
  - E os elementos podem ser acessados pelos seus respectivos índices dentro da lista
  - Ela resolve todos os problemas que levantamos em relação a array (busca, remoção, tamanho “infinito”,...).
  - Esse código já está pronto! ☺

UNIVERSIDADE VILA VELHA-ES

List - Coleções indexadas

Possui as seguintes implementações

ArrayList: usa vetores

- Trabalha com uma array interna para gerar uma lista
  - lista de objetos armazenados em um vetor interno
- Ela é mais rápida na pesquisa (desempenho melhor);
- Melhor se você precisa de acesso com índice

LinkedList: usa lista encadeada

- Lista de objetos armazenados em uma lista encadeada
  - Normalmente ordenada pela ordem de inserção.
- É mais rápida na inserção e remoção de itens nas pontas.

List - Coleções indexadas

- Métodos principais:
  - add(Object), add(int, Object), addAll(Collection);
  - clear(), remove(int), removeAll(Collection);
  - contains(Object), containsAll(Collection);
  - get(int), indexOf(Object), set(int, Object);
  - isEmpty(), toArray(), subList(int, int), size().

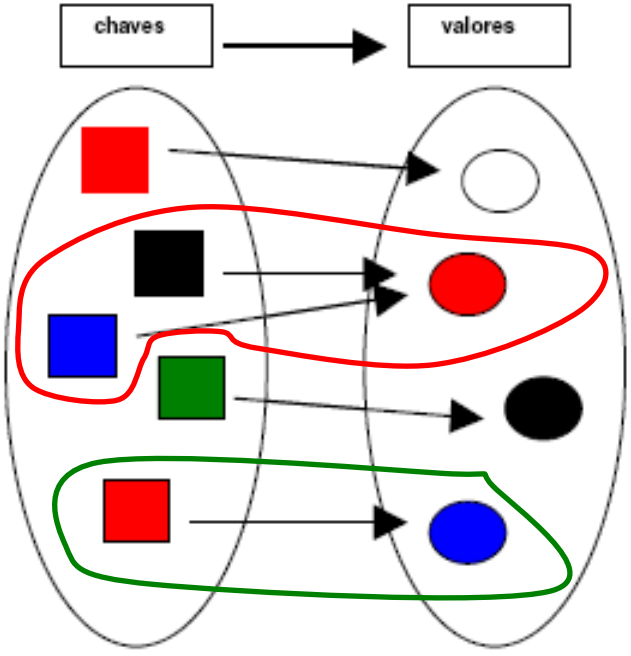
Map - Coleções pares chave x valor

- Outro recurso que a API de Collections traz são os mapeamentos (Map) ou Coleções de pares chave x valor
  - É basicamente o conceito de hash mas para o Collection Framework

- Um mapa é composto de uma associação de um objeto chave a um objeto valor.
  - É equivalente ao conceito de dicionário usado em várias linguagens
  - Exemplo: Mapeia “Vinicius” à chave “CPF.

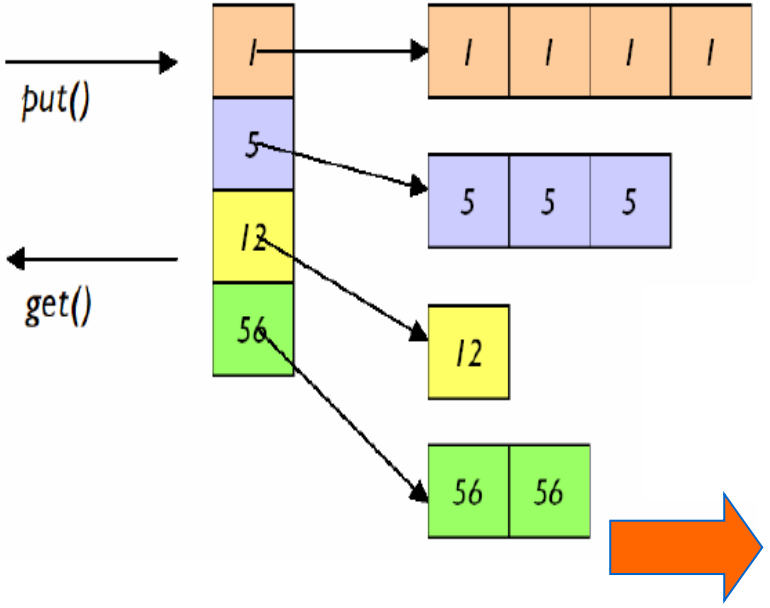
Map - Coleções pares chave x valor

- Objetos Map não podem conter chaves duplicadas
- Cada chave só pode mapear um valor apenas
- Valores podem ser repetidos para chaves diferentes



Possíveis ações em um mapa:

Mapeie uma chave a um valor
O que está mapeado na chave X?
Remapeie uma certa chave
Quero o conjunto de chaves.
Quero o conjunto de valores.
Desmapeie a chave X.



Map - Coleções pares chave x valor

Possui as seguintes implementações

HashMap: Usa tabela *hash*;

- Correspondências chave-valor, onde as chaves não estão ordenadas.
- Permite elementos e chaves nulos
- Usa hashCode() para otimizar a busca por uma chave

TreeMap: Usa árvore e chaves é ordenada

- Usado quando preciso que os elementos sejam ordenados (definido pela chave)
- Objetos devem implementar Comparable ou Comparator
  - Os objetos precisam implementar a interface Comparable ou Comparator (vamos ver mais pra frente...)

Map - Coleções pares chave x valor

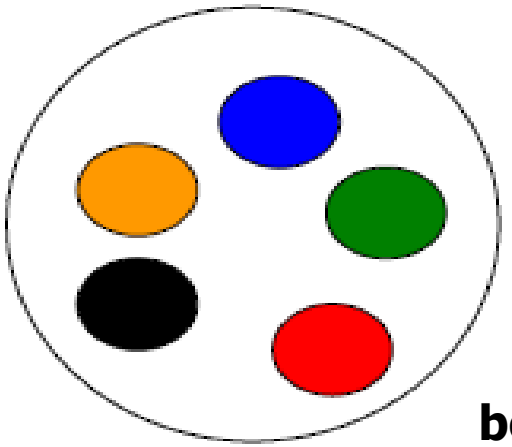
- Métodos principais:
  - remove(Object), clear();
  - containsKey(Object), containsValue(Object);
  - isEmpty(), size();
  - put(Object key, Object), get(Object key), putAll(Map);
  - entrySet(), keySet(), values().

Set - Coleções não indexadas

- Outro recurso que a API de Collections traz são os conjuntos (Set) ou Coleções não indexadas

- Representa uma coleção "desordenada" de dados e não permite elementos duplicados.
  - "Desordenada" pois a priori a ordem não importa...

Não pode haver dois objetos iguais



bolsa....

UNIVERSIDADE VILA VELHA-ES

Set - Coleções não indexadas

Possui as seguintes implementações

HashSet: usa tabela hash;

- Conjunto de objetos armazenados em uma tabela hash

LinkedHashSet:

- Armazenamento do conjunto de objetos em uma lista encadeada

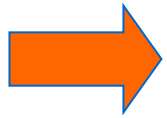
TreeSet: usa árvore e é ordenado.

- Conjunto de objetos armazenados em árvore binária
- O armazenamento dos objetos pode ser ordenados
  - Os objetos precisam implementar a interface Comparable ou Comparator (vamos ver mais pra frente...)
- Mais rápidas que os outros conjuntos O(log(n))

UNIVERSIDADE VILA VELHA-ES

Set - Coleções não indexadas

- Por que o Set é importante e usado?
  - Para perceber se um item já existe em uma lista é muito mais rápido usar um Set do que um List,
  - E os TreeSets já vem ordenados de acordo com as características que desejarmos!
- IMPORTANTE:
  - Sets funcionam apenas se os objetos inseridos implementam equals() e hashCode()



Set - Coleções não indexadas

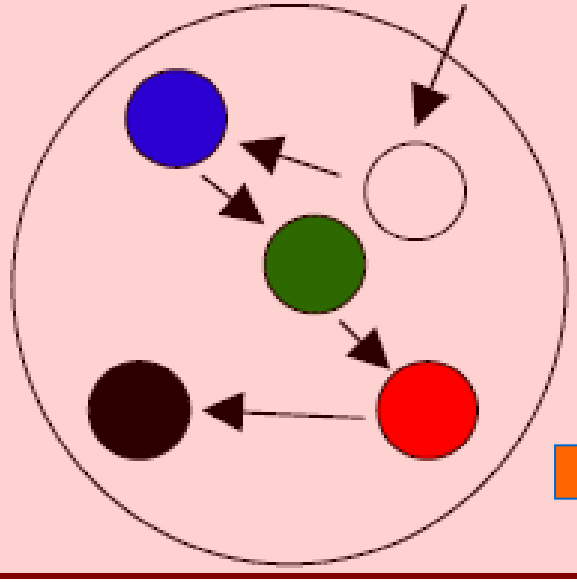
- Métodos principais:
  - add(Object), addAll(Collection);
  - clear(), remove(Object), removeAll(Collection),
  - retainAll(Collection);
  - contains(Object), containsAll(Collection);
  - isEmpty(), toArray(), size().

- Cadê o get????

Iteradores - Coleções

- Em conjuntos, não há um método para obter o objeto pelo índice, pois não há índice;
  - Set não possui uma função para pegar o primeiro, o segundo ou o quinto elemento do conjunto

- Para acessar os elementos de conjuntos, usamos *ITERADORES*



Possíveis ações em um iterador:

- Existe um próximo elemento na coleção?.
- Pegue o próximo elemento.
- Remova o elemento atual da coleção.

```
ContaCorrente c =
  (ContaCorrente) iterator.next();
```

Iteradores - Coleções

Os interadores são obtidos via o método iterator()

Outos métodos utilizados são:

- hasNext(), next() e remove().

java.util.Iterator

Retorna um objeto Iterator

```
Iterator it = myCollection.iterator();
while(it.hasNext()) {
    chame it.next() para obter o próximo objeto
    faça algo com esse objeto
}
```

Ex:.

```
public void listNotes()
{
    Iterator it = notes.iterator();
    while(it.hasNext()) {
        System.out.println(it.next());
    }
}
```

coleção

- Funciona também para listas e outras coleções.

Iteradores - Coleções

- Podemos também usar o iterator ou o enhanced for

coleção

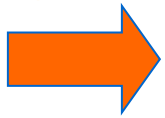
```
Iterator i = numeros.iterator();
// O while só termina quando todos os elementos do conjunto forem percorridos,
// isto é, quando o método hasNext mencionar que não existem mais itens.
while (i.hasNext())
    System.out.println(i.next());
```

```
// A partir do Java 5.0, surgiu uma nova sintaxe para laços que usam iteradores;
System.out.println("\n");
for (Object o : numeros)
    System.out.println(o);
```

coleção

Tipos Genéricos

- Blz,
  - Entendido esses conceitos, podemos conversar sobre uma questão que é bastante utilizada em desenvolvimento de código…
  - Toda a API de coleções foi adaptada para permitir o uso de Tipos Genéricos
  - Mas que raios é isso…
    - Vamos entender pois vamos escrever nossos códigos assim….



UNIVERSIDADE VILA VELHA-ES

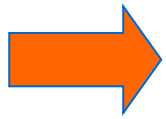
Tipos Genéricos e Coleções

- A idéia por trás dos tipos genéricos é melhorar a redigibilidade, legibilidade e confiabilidade

Uma das principais características desse recurso é permite a abstração do tipo, eliminando o casting e provê compile-time type safety

Ou seja, podemos usar o recurso de **Generics** para restringir as listas a um determinado tipo de objetos

- E não qualquer Object



Tipos Genéricos e Coleções

- Repare no uso de um parâmetro ao lado de List e ArrayList:
  - Ele indica que nossa lista foi criada para trabalhar exclusivamente com objetos do tipo ContaCorrente.

```
List<ContaCorrente> contas = new ArrayList<ContaCorrente>();
contas.add(c1);
contas.add(c3);
contas.add(c2);
```

```
contas.get(i); // sem casting!
```

- Isso também nos traz uma segurança em tempo de compilação:

```
contas.add("uma string"); // isso não compila mais!!
```

Tipos Genéricos e Coleções

Outro exemplo de Genérico e List

```
List listaPac = new ArrayList(10);
listaPac.add(p1);
listaPac.add(0, p2);
listaPac.add(1,p3);
listaPac.add(p4);
```

Antes

```
for(int i = 0; i<listaPac.size(); i++) {
    Paciente umPac = (Paciente) listaPac.get(i);
    System.out.println(umPac);
}
```

Depois

```
List <Paciente> listaPac = new ArrayList<Paciente>(10);
. . .
for(int i = 0; i<listaPac.size(); i++) {
    Paciente umPac = listaPac.get(i);
    System.out.println(umPac);
}
```

Tipos Genéricos e Coleções

Genérico e Set

Antes

```
Set conjunto = new HashSet();
conjunto.add("item 1");
conjunto.add("item 2");
conjunto.add("item 3");

// retorna o iterator
for(Object elemento : conjunto) {
    String palavra = (String) elemento;
    System.out.println(palavra);
}
```

Depois

```
Set<String> conjunto = new HashSet<String>();
conjunto.add("item 1");
conjunto.add("item 2");
conjunto.add("item 3");

// retorna o iterator
for(String palavra : conjunto) {
    System.out.println(palavra);
}
```

Tipos Genéricos e Coleções

Genérico e Map

- Assim como as coleções, um mapa é parametrizado.
  - O interessante é que ele recebe dois parâmetros:
    - A chave e o valor:

```
ContaCorrente c1 = new ContaCorrente();
c1.deposita(10000);

ContaCorrente c2 = new ContaCorrente();
c2.deposita(3000);
```

```
// cria o mapa
Map<String,ContaCorrente> mapaDeContas = new
HashMap<String,ContaCorrente>();

// adiciona duas chaves e seus valores
mapaDeContas.put("diretor", c1);
mapaDeContas.put("gerente", c2);
// qual a conta do diretor? (sem casting!)
ContaCorrente contaDoDiretor = mapaDeContas.get("diretor");
```

- Se você tentar colocar algo diferente de String na chave e ContaCorrente no valor… Vai ter um erro de compilação.

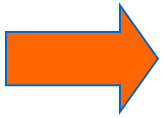
Tipos Genéricos e Coleções

A) Qual a saída do código abaixo? B) O que acontece quando a linhas 11 é descomentada?

```
1  import java.util.*;
2
3  public class testCollection2 {
4
5    private void testCollection() {
6      List<String> list = new ArrayList<String>();
7
8      list.add(new String("Hello world!"));
9      list.add(new String("Good bye!"));
10     list.add("Blz");
11 //    list.add(new Integer(95));
12
13     printCollection(list);
14   }
15
16 private void printCollection(List<String> c) {
17
18   for (String item : c) {
19     System.out.println("Item FOR: "+item);
20   }
21   }
22
23
24   public static void main(String argv[]) {
25
26     testCollection2 e = new testCollection2();
27     e.testCollection();
28   }
29 }
```

Implementação de relações – 1-N

- Blz,
  - Pra fechar o nosso assunto e começar a praticar a gente pode conversar sobre as relações 1 pra vários…
  - Relembrando…..



Implementação de relações – 1-N

Relações de 1 para 2 bidirecional

B "é usado por" A
e
A "é usado por" B

B "tem um" A
e
A "tem dois" B

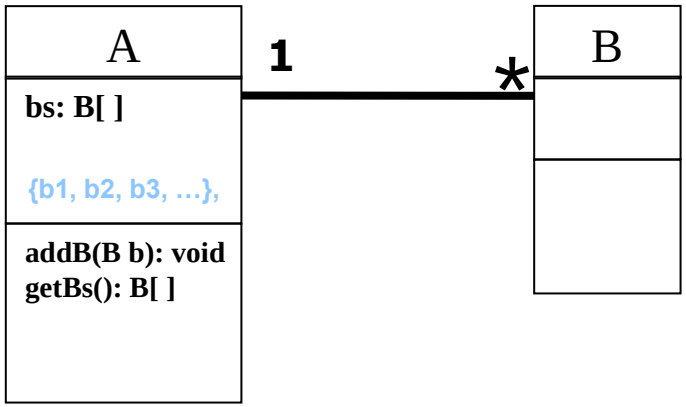
```
class A{

    private B b1;
    private B b2;
    ...
    public void setB1(B aB){
        b1=aB;
    }
    public B getB1(){
        return b1;
    }

    public void setB2(B aB){
        b2=aB;
    }
```

```
class B{

    private A a;
    ...
    public void setA(A aA){
        a=aA;
    }

    public A getA(){
        return a;
    }
    ...

}
```

E se tivermos uma relação de 1 para n??

Implementação de relações – 1-N e Array

Relações de 1 para n bidirecional



B "tem um" A
e
A "tem N's" B
A "tem vários" B

```
class A{

    private B[] bs;
    private int pos;
    ...
    public A(){
        bs=new B[10];
        pos=0;
    }
    ...

    public void addB(B aB){
        if(pos<bs.length) {
            bs[pos]=aB;
            pos=pos+1;
        }
    }

    public B[] getBs(){
        return bs;
    }
....
}
```

```
class B{

    private A a;
    ...
    public void setA(A aA){
        a=aA;
    }

    public A getA(){
        return a;
    }
    ...

}
```

Blz, reserva memória para 10 objetos do tipo B, trata posição...
E se forem mais de 10?...

Implementação de relações – 1-N e Coleções

É aí que entra a Coleções...Poderíamos fazer assim..

```
class A{

    private ArrayList bs;
    ...
    public A(){
        bs=new ArrayList( );
    }
    ...

    public void addB(B aB){
        bs.add(aB);
    }

    public ArrayList getBs(){
        return bs;
    }
....
}
```

```
class B{

    private A a;
    ...
    public void setA(A aA){
        a=aA;
    }

    public A getA(){
        return a;
    }
    ...
}
```

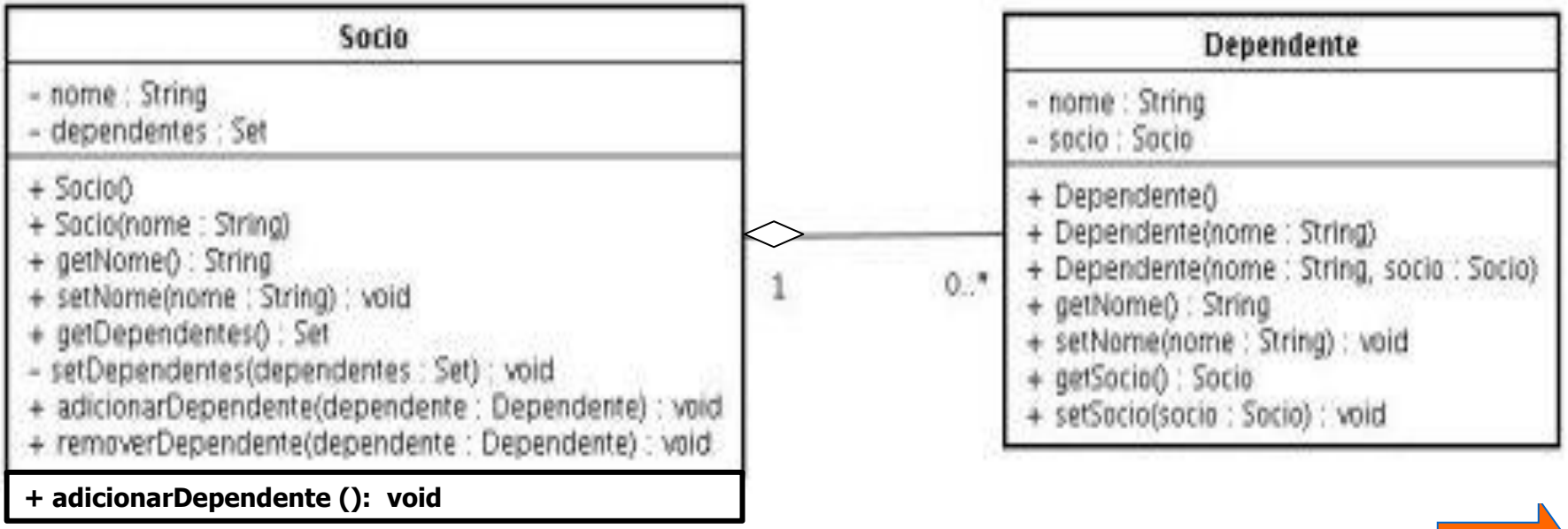
B "tem um" A
e
A "tem N's" B
A "tem vários" B

Relações de 1 para n bidirecional

Vamos analisar um exemplo mais completo....

Implementação de relações – 1-N e Coleções

- O relacionamento entre as classes Sócio e Dependente.
  - Esse relacionamento é um relacionamento do tipo um-para-muitos (um sócio pode ter nenhum (zero) ou muitos (*) dependentes.)



Relações de 1 para n bidirecional

Implementação de relações – 1-N e Coleções

- Para implementar relacionamentos desse tipo, baseado na nossa modelagem bidirecional, podemos:

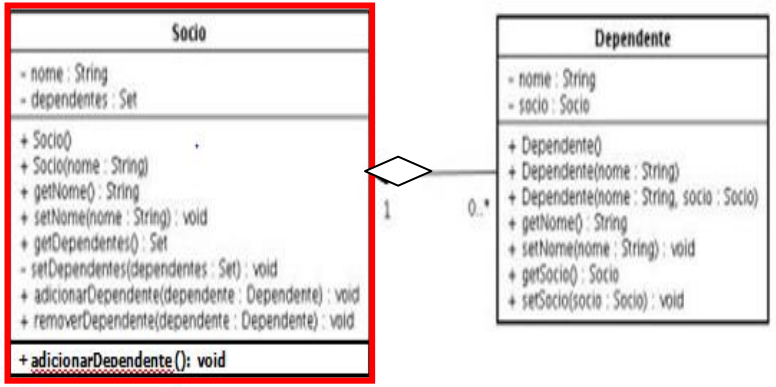
Inserir na classe que recebe a classe com multiplicidade N um atributo do tipo coleção (Set, por exemplo);

Inserir os métodos get e set correspondentes ao atributo coleção;

Inserir métodos para adicionar e remover objetos na coleção

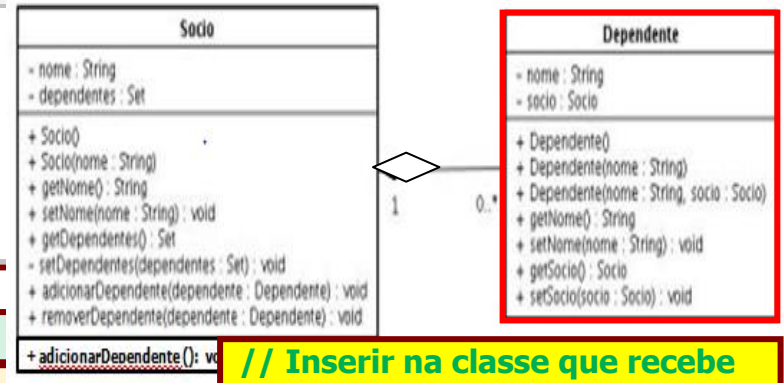
Inserir na classe que recebe a classe com multiplicidade 1 um atributo do tipo correspondente;

Implementação de relações – 1-N e Coleções




```
1  import java.util.HashSet;
2  import java.util.Set;
3
4  public class Socio {
5      private String nome;
6
7      private Set<Dependente> dependentes;
8
9      public Socio(){
10         super();
11         this.dependentes = new HashSet<Dependente>();
12     }
13     public Socio(String nome)..
18
19     public void setNome(String nome)..
22     public String getNome()..
25
26     private void setDependentes(Set<Dependente> dependentes)..
29     public Set<Dependente> getDependentes()..
32
33     public void adicionarDependente(Dependente dependente){
34         this.getDependentes().add(dependente);
35     }
36     public void removerDependente(Dependente dependente){
37         this.getDependentes().remove(dependente);
38     }
39 }
```

Implementação de relações – 1-N e Coleções



// Inserir na classe que recebe a classe com multiplicidade 1 um atributo do tipo correspondente;

```
1  public class Dependente {
2      private String nome;
3
4      private Socio socio;
5
6      public Dependente()
9      public Dependente(String nome)
13
14     public Dependente(String nome, Socio socio){
15         super();
16         this.nome = nome;
17         this.setSocio(socio);
18     }
19     public void setNome(String nome)
22     public String getNome()
25
26     public Socio getSocio()
29
30
31     public void setSocio(Socio socio){
32         this.socio = socio;
33         if(this.socio!=null){
34             this.socio.adicionarDependente(this);
35         }
36     }
37
38     @Override
39     public boolean equals(Object obj)
55
56     @Override
57     public int hashCode()
63 }
64
```

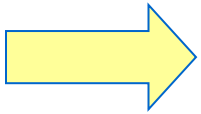
Implementação de relações – 1-N e Coleções

Qual a saída impressa?
E se descomentar a linha 16??

```
1  public class AppUmParaMuitosSocioDependente {
2
3      public static void main(String[] args) {
4
5  //=========================================================================
6  // Fluxo de execuçao tomando como exemplo a relação de AGREGACAO / ASSOCIACAO
7  //=========================================================================
8  System.out.println("*************************************************************");
9  System.out.println("***************** AGREGACAO / ASSOCIACAO*********************");
10 System.out.println("*************************************************************");
11
12         Socio socioAgregacaoAssociacao = new Socio("Fulano");
13
14         Dependente dependente00 = new Dependente("Beltrano", socioAgregacaoAssociacao);
15         Dependente dependente01 = new Dependente("Cicrano", socioAgregacaoAssociacao);
16
17         Dependente dependente02 = new Dependente("Outro");
18         dependente02.setSocio(socioAgregacaoAssociacao);
19
20         System.out.println("Lista de Dependentes do socio " + socioAgregacaoAssociacao.getNome() + ":");
21         for(Dependente dependente : socioAgregacaoAssociacao.getDependentes()){
22             System.out.println(dependente.getNome());
23         }
24
25         for(Dependente dependente : socioAgregacaoAssociacao.getDependentes()){
26             System.out.println("==========================");
27             System.out.println("Socio ao qual o dependente estah associado. "+
28                                "Nome do Dependente: " +
29                               dependente.getNome());
30             Socio socioDoDependente = dependente.getSocio();
31             System.out.println(socioDoDependente.getNome());
32         }
33     }
34 }
```

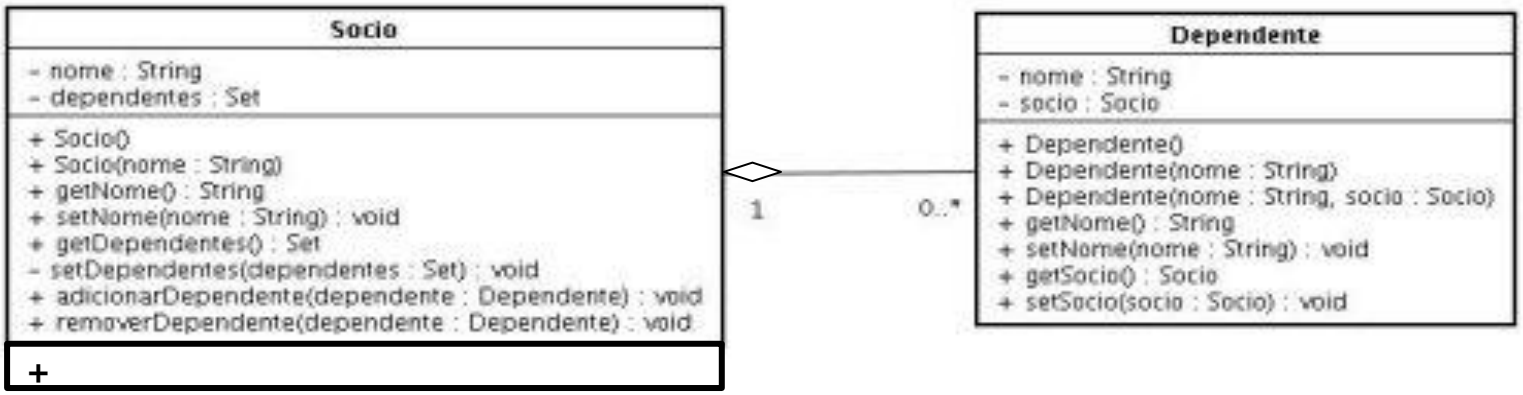
Exercício Junto....

- Blz, entendemos muita coisa hoje.....
  - ⇨ Vamos agora construir juntos um exemplo que tenha:
    - Coleção Tipada
    - Relacionamento 1 - N
  - – Baseado no seguinte diagrama....



Exercício Junto....

- Fazer um exemplo de 1-N junto… Agregação



  - Criar as classses Socio e Dependente com os atributos básicos.
  - Gerar os gets e set para todos atributos e construtores padrão.
    - Criar o equals e hashCode na classe Dependente
  - Criar na classe Socio (a que tem a coleção) os métodos para adicionar e remover objetos na coleção.
  - Alter o método SetSocio(..) na classe Dependente para que o Dependente possa ser vinculada a coleção de dependente do Socio
  - Agora podemos criar classe Principal, criando um sócio, chamando o método para adicionar seus dependentes e depois 1: Listar quais são os dependentes do Sócio; 2: Listar qual é o sócio associados aos dependentes

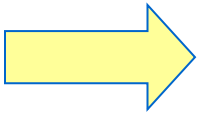
UNIVERSIDADE VILA VELHA-ES

Exercícios

- Blz... Agora é hora de exercitar.....

- Tente resolver os seguintes problemas...
  - Em dupla
  - Apresentar ao professor no final da aula
  - Pontuação em Atividades em sala de aula...

  - Faça o JAVADOC de todos os exercícios!!!



UNIVERSIDADE VILA VELHA-ES

Exercício

- Adicione a sequência de números 2, 5, 3, 9, 2, 4, 3, 8, 5
  - a um conjunto (Set)
  - e a uma lista (List), escolhendo a implementação que desejar.

- Em seguida imprima o conteúdo de ambas as coleções usando um enhanced for
  - E analise as diferenças.

UNIVERSIDADE VILA VELHA-ES

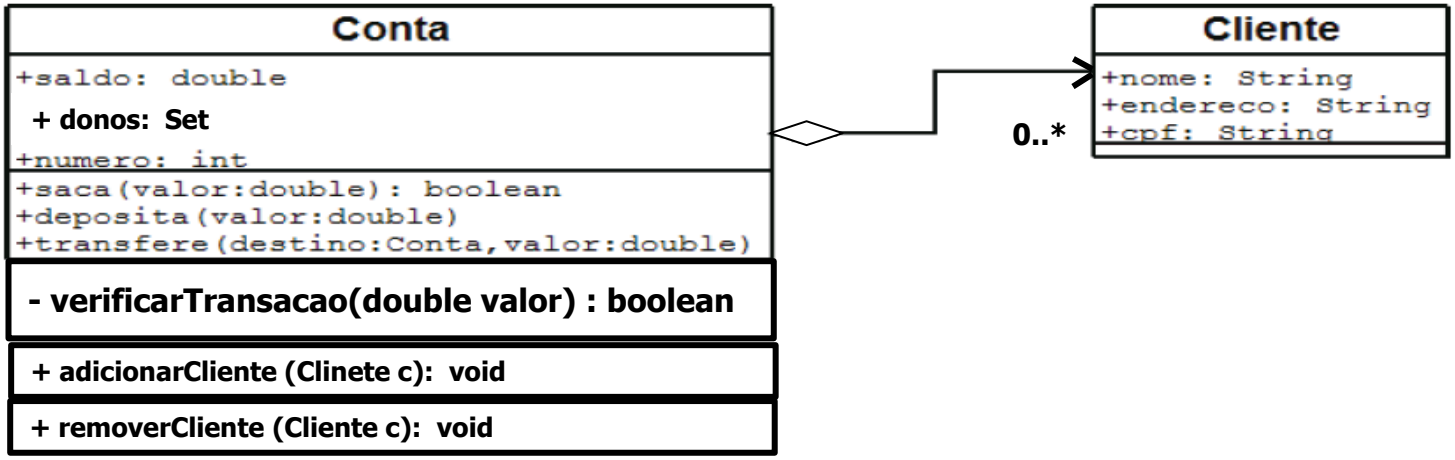
Exercício

- Escreva um programa Java que:
  - Crie 5 objetos Circulo de tamanho diferentes,
  - Insira-os em uma lista
  - E depois percorra a lista imprimindo a área de cada círculo armazenado.

UNIVERSIDADE VILA VELHA-ES

Exercício

- Identifique alguns atributos e comportamentos simples para as classes abaixo e implemente.



  - Criar as classses Conta e Cliente com os atributos básicos.
  - Gerar os gets e set para todos atributos e construtores padrão.
    - Criar o equals e hashCode na classe Cliente
    - this.donos = new HashSet<Cliente>() no construtor da Conta
  - Criar na classe Conta (a que tem a coleção) os métodos para adicionar e remover objetos na coleção.
  - Agora podemos criar classe Principal, criando uma Conta, adicionamos vários clientes e esta conta depois 1: Listar quais são as clientes/donos da Conta;

UNIVERSIDADE VILA VELHA-ES

Exercício

- Crie um programa em Java para testar a classe AgendaTelefonica abaixo
  - Teste a classe com pelo menos 5 contatos diferentes na agenda de telefones.

UNIVERSIDADE VILA VELHA-ES

Exercício

- Escreva uma aplicação de dicionário com três funções:
  - Adicionar um termo ao dicionário,
  - Procurar um termo no dicionário e
  - Listar todos os termos existentes em ordem alfabética.

- Qual classe você usou para implementar a coleção de palavras? Por quê?

UNIVERSIDADE VILA VELHA-ES

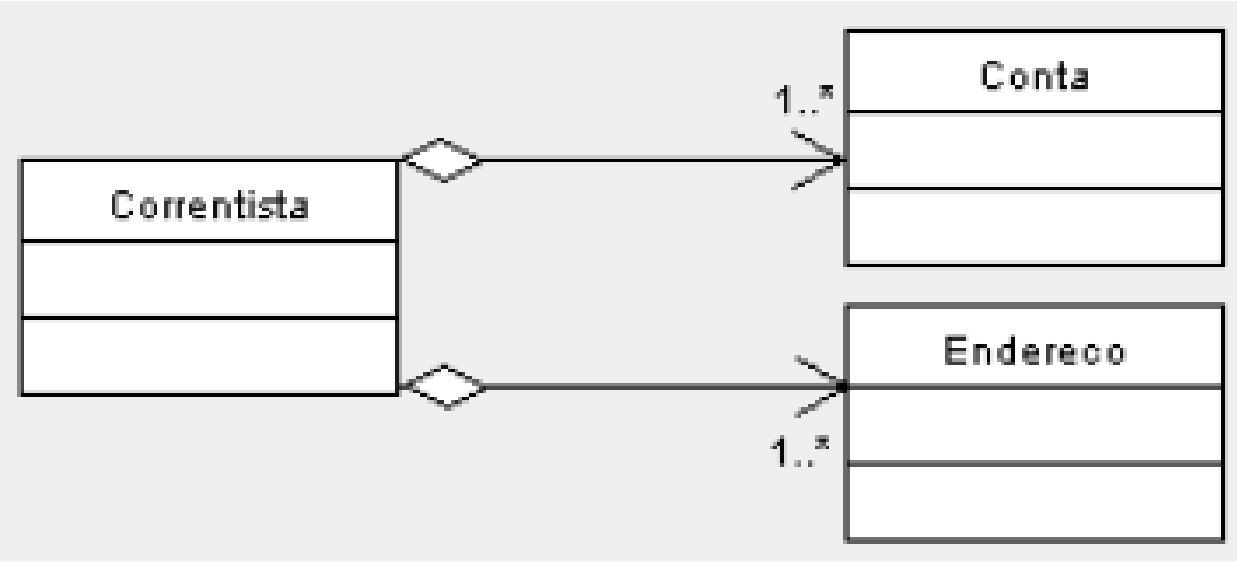
Exercício

- Implementar um coleção do tipo mapa que associa nomes de pessoas a sua cor favorita.
  - Use Tipos Genéricos

UNIVERSIDADE VILA VELHA-ES

Exercício

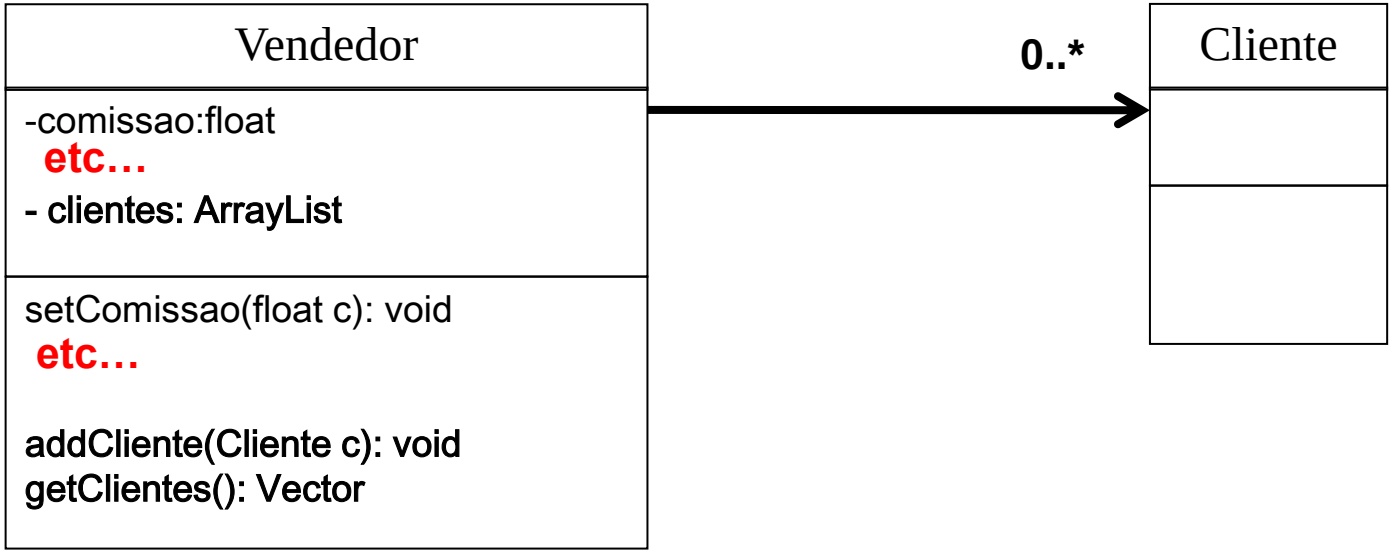
- Identifique alguns atributos e comportamentos simples para as classes abaixo e implemente.



UNIVERSIDADE VILA VELHA-ES

Exercício

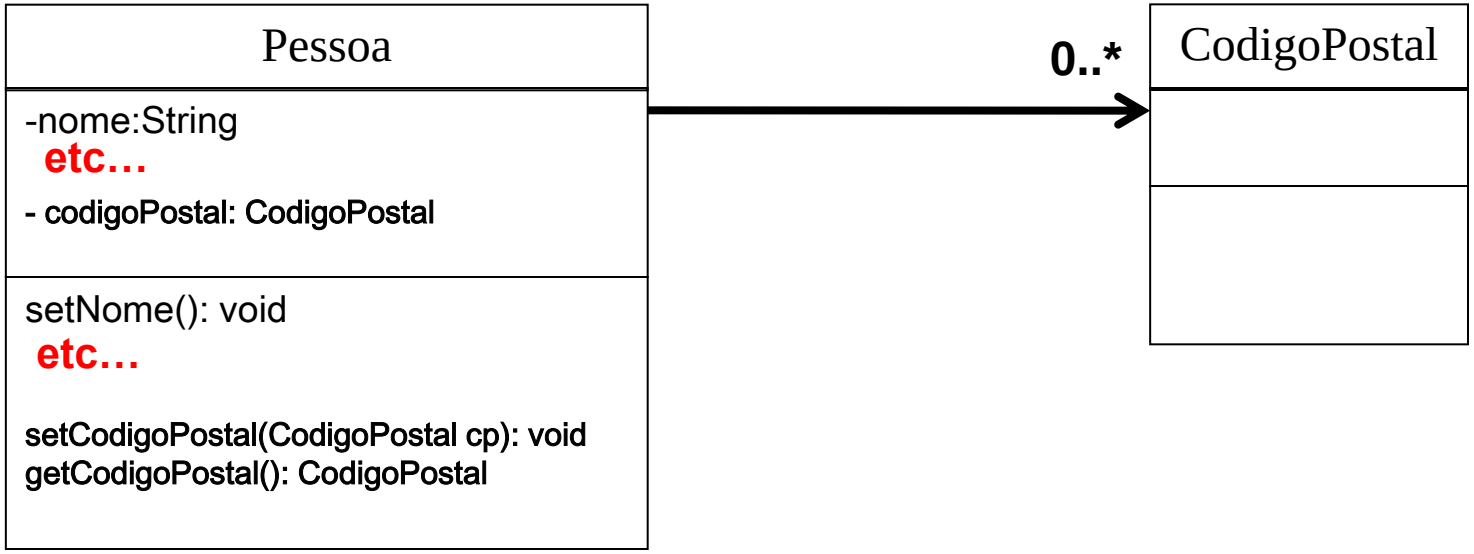
- Relação de 1 para n, unidirecional
  - Implemente a associação entre as classes Vendedor e Cliente. Suponha que um determinado vendedor possuí vários clientes.
  - Escreva um programa de teste capaz de verificar a implementação da relação.



UNIVERSIDADE VILA VELHA-ES

Exercício

- Relação unidirecional
  - Considere as classes Pessoa e CodigoPostal abaixo e implemente a relação existente entre ambas
    - Cada instância de Pessoa possui um CodigoPostal.
  - Escreva um programa de teste capaz de verificar a implementação da relação.

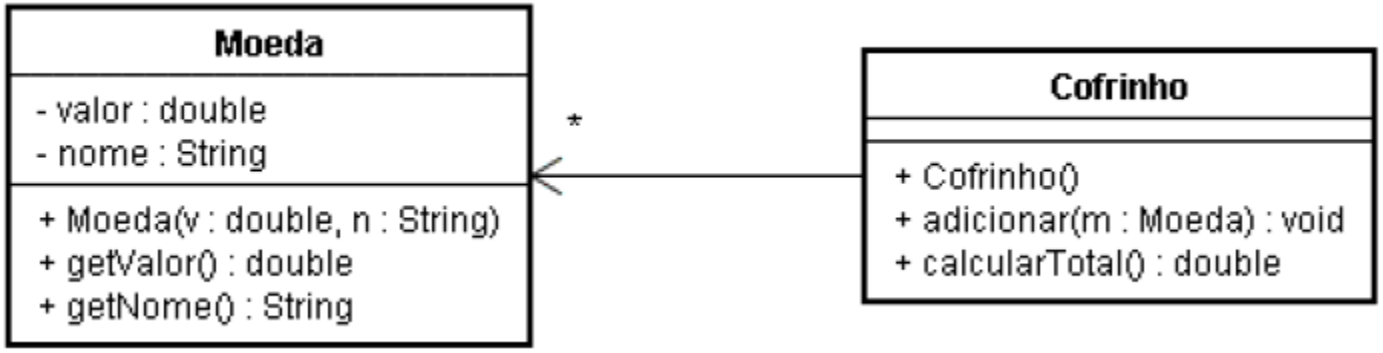


UNIVERSIDADE VILA VELHA-ES

Exercício

- Implementar um cofrinho de moedas com a capacidade de receber moedas e calcular o total depositado no cofrinho.
  - Implementa uma coleção de Moeda como uma lista.
    - Use o ArrayList
  - Faça um classe de teste

```
public class Cofrinho {
    private List<Moeda> moedas;
    public Cofrinho() {}
    public ....  get/setMoeda(...) {...}
}
```



UNIVERSIDADE VILA VELHA-ES

Exercício

- Altere a classe Cofrinho de modo que ela implemente métodos para:
  - Contar o número de moedas armazenadas
  - Contar o número de moedas de um determinado valor
  - Informar qual a moeda de maior valor

UNIVERSIDADE VILA VELHA-ES

Exercício

- Sistema Empresa...
  - Criar uma *classe Empregado* que contenha:
    - Os dados de uma pessoa (*classe Pessoa*)
    - Mais dados de matrícula e salário
  - Criar uma *classe programa* que utilize a classe Empresa.
    - Os dados devem ser obtidos via Scanner
  - Os métodos construtores devem ser utilizados para a propagação dos valores

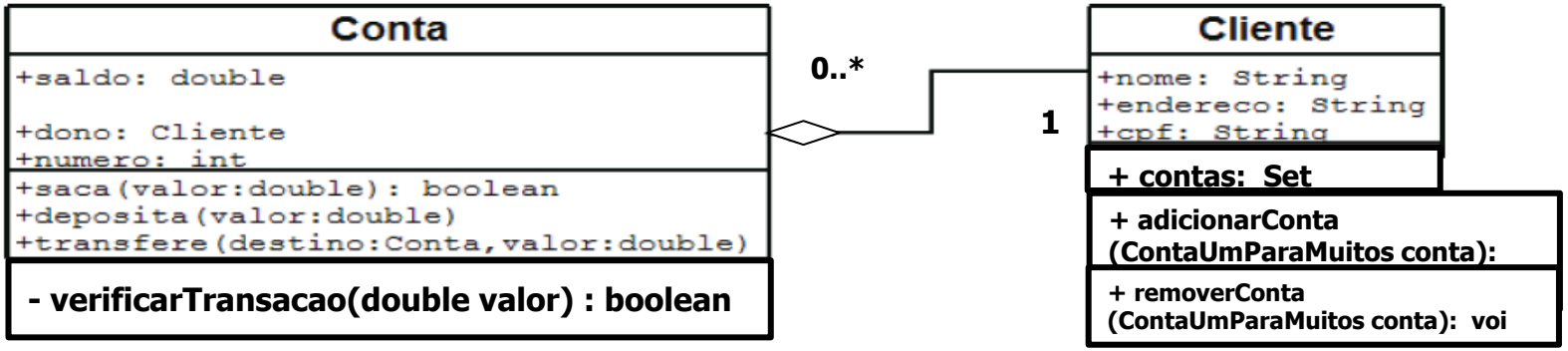
Exercício

- Sistema Empresa...
  - Criar uma *classe Empregado* que contenha:
    - Os dados de uma pessoa (*classe Pessoa*)
    - Mais dados de matrícula e salário
  - Criar uma *classe programa* que utilize a classe Empresa.
    - Os dados devem ser obtidos via Scanner
  - Os métodos construtores devem ser utilizados para a propagação dos valores

UNIVERSIDADE VILA VELHA-ES

Exercício

- Identifique alguns atributos e comportamentos simples para as classes abaixo e implemente.



  - Criar as classses Conta e Cliente com os atributos básicos.
  - Gerar os gets e set para todos atributos e construtores padrão.
    - Criar o equals e hashCode na classe Conta
    - this.contas = new HashSet<Conta>() no construtor do Cliente
  - Criar na classe Cliente (a que tem a coleção) os métodos para adicionar e remover objetos na coleção.
  - Alter o método SetDono(..) na classe Conta para que a Conta possa ser vinculada a coleção de contas do Cliente
  - Agora podemos criar classe Principal, criando um cliente, chamando o método para adicionar suas contas e depois 1: Listar quais são as contas do Cliente; 2: Listar qual é o Cliente associado as Contas

UNIVERSIDADE VILA VELHA-ES

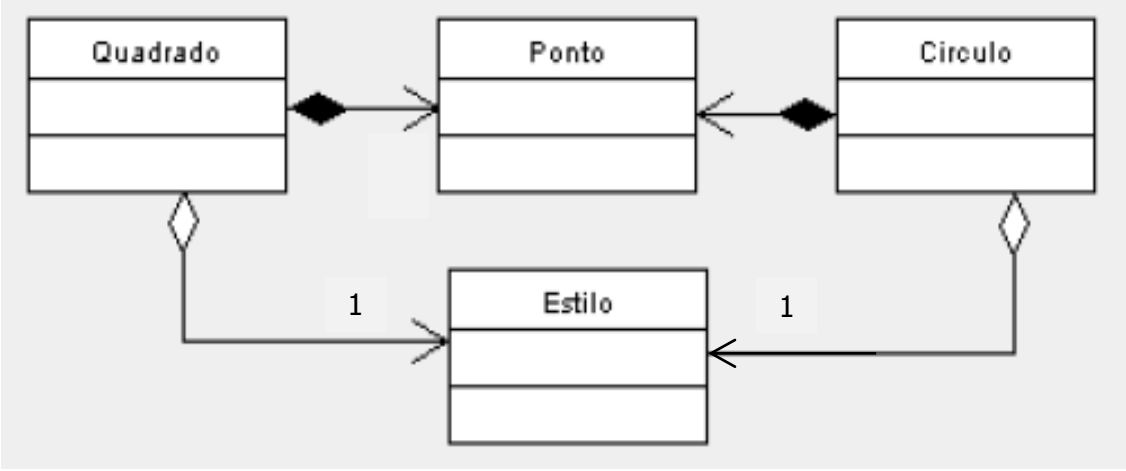
Exercício

- Identifique alguns atributos e comportamentos simples para as classes abaixo e implemente.

UNIVERSIDADE VILA VELHA-ES

Exercício

- Identifique alguns atributos e comportamentos simples para as classes abaixo e implemente.
  - A relação de Ponto com Circulo e Quadrado é uma composição, pois os mesmos não podem ser compartilhados
  - Enquanto que o mesmo objeto de Estilo pode ser compartilhado por Circulo e Ponto (agregação)



UNIVERSIDADE VILA VELHA-ES

Exercício

- Escrever um programa que:
  - Recebe um vetor de inteiros de tamanho 4
  - E um valor inteiro x
  - E retorna a quantidade de vezes que x aparece no vetor.
  - Os elementos do vetor e o nº x devem ser informados pelo usuário.

UNIVERSIDADE VILA VELHA-ES

Exercício

- Escreva um programa Java que:
  - Permita determinar qual o valor máximo existente num vetor do tipo float com 10 elementos.
  - Utilize como estrutura de controle um ciclo for.

- Complemente o programa anterior de modo que:
  - Indique também qual o valor mínimo existente no vetor
  - E qual a soma dos seus 10 valores.

UNIVERSIDADE VILA VELHA-ES

Exercício

- Crie um algoritmos que ordene em ordem crescente de descrente um conjunto de 10 objetos Integer
  - Dica:
    - List<Integer> lista = new ArrayList<Integer>();
    - Collections.sort(lista)
    - for (Integer o : lista)

UNIVERSIDADE VILA VELHA-ES

Exercício

Sort1.java

- No código a seguir, compile, execute no computador e analise os resultados.. Estude na prática!!!

```
1  // Using algorithm sort.
2  import java.util.List;
3  import java.util.Arrays;
4  import java.util.Collections;
5
6  public class Sort1
7  {
8     private static final String suits[] =
9        { "Hearts", "Diamonds", "Clubs", "Spades" };
10
11    // display array elements
12    public void printElements()
13    {
14       List< String > list = Arrays.asList( suits ); // create List
15
16       // output list
17       System.out.printf( "Unsorted array elements:\n%s\n", list );
18
19       Collections.sort( list ); // sort ArrayList
20
21       // output list
22       System.out.printf( "Sorted array elements:\n%s\n", list );
23    } // end method printElements
24
25    public static void main( String args[] )
26    {
27       Sort1 sort1 = new Sort1();
28       sort1.printElements();
29    } // end main
30 } // end class Sort1
```

UNIVERSIDADE VILA VELHA-ES

Exercício

TotalNumbersErrors.java

- Dado o código em TotalNumbersErrors.java pede-se
  - 1. Encontre o erro.
  - 2. Encontre uma forma de corrigir o erro
  - 3. Após corrigido, qual a saída impressa?

```
1  // Summing the elements of an ArrayList.
2  import java.util.ArrayList;
3
4  public class TotalNumbersErrors
5  {
6     public static void main( String[] args )
7     {
8        // create, initialize and output ArrayList of Integers
9        // then display total of the elements
10       Integer[] integers = { 1, 2, 3, 4 };
11       ArrayList< Integer > integerList = new ArrayList< Integer >();
12
13       for ( Integer element : integers )
14          integerList.add( element ); // place each number in integerList
15
16       System.out.printf( "integerList contains: %s\n", integerList );
17       System.out.printf( "Total of the elements in integerList: %.1f\n",
18          sum( integerList ) );
19    } // end main
20
21    // calculate total of ArrayList elements
22    public static double sum( ArrayList< Number > list )
23    {
24       double total = 0; // initialize total
25
26       // calculate sum
27       for ( Number element : list )
28          total += element.doubleValue();
29
30       return total;
31    } // end method sum
32 }
```

UNIVERSIDADE VILA VELHA-ES

Exercício

- No código a seguir, compile, execute no computador e analise os resultados.. Estude na prática!!!

BinarySearchTest.java

```
1  // Using algorithm binarySearch.
2  import java.util.List;
3  import java.util.Arrays;
4  import java.util.Collections;
5  import java.util.ArrayList;
6
7  public class BinarySearchTest
8  {
9     private static final String colors[] = { "red", "white", "blue",
10                                              "black", "yellow", "purple", "tan", "pink" };
11    private List< String > list; // ArrayList reference
12
13    // create, sort and output list
14    public BinarySearchTest()
15    {
16       list = new ArrayList< String >( Arrays.asList( colors ) );
17       Collections.sort( list ); // sort the ArrayList
18       System.out.printf( "Sorted ArrayList: %s\n", list );
19    } // end BinarySearchTest constructor
20
21    // search list for various values
22    private void search()
23    {
24       printSearchResults( colors[ 3 ] ); // first item
25       printSearchResults( colors[ 0 ] ); // middle item
26       printSearchResults( colors[ 7 ] ); // last item
27       printSearchResults( "aqua" ); // below lowest
28       printSearchResults( "gray" ); // does not exist
29       printSearchResults( "teal" ); // does not exist
30    } // end method search
31
32    // helper method to perform searches
33    private void printSearchResults( String key )
34    {
35       int result = 0;
36       System.out.printf( "\nSearching for: %s\n", key );
37       result = Collections.binarySearch( list, key );
38       if ( result >= 0 )
39          System.out.printf( "Found at index %d\n", result );
40       else
41          System.out.printf( "Not Found (%d)\n",result );
42    } // end method printSearchResults
43
44    public static void main( String args[] )
45    {
46       BinarySearchTest binarySearchTest = new BinarySearchTest();
47       binarySearchTest.search();
48    } // end main
49 } // end class BinarySearchTest
```

UNIVERSIDADE VILA VELHA-ES

Exercício

WordTypeCount.java

- Analise o código abaixo e responda:
  - O que ele está fazendo? Qual a saída?

```java
// Program counts the number of occurrences of each word in a string
import java.util.StringTokenizer;
import java.util.Map;
import java.util.HashMap;
import java.util.Set;
import java.util.TreeSet;
import java.util.Scanner;

public class WordTypeCount
{
   private Map< String, Integer > map;
   private Scanner scanner;

   public WordTypeCount()
   {
      map = new HashMap< String, Integer >(); // create HashMap
      scanner = new Scanner( System.in ); // create scanner
      createMap(); // create map based on user input
      displayMap(); // display map content
   } // end WordTypeCount constructor

   // create map from user input
   private void createMap()
   {
      System.out.println( "Enter a string:" ); // prompt for user input
      String input = scanner.nextLine();

      // create StringTokenizer for input
      StringTokenizer tokenizer = new StringTokenizer( input );

      // processing input text
      while ( tokenizer.hasMoreTokens() ) // while more input
      {
         String word = tokenizer.nextToken().toLowerCase(); // get word

         // if the map contains the word
         if ( map.containsKey( word ) ) // is word in map
         {
            int count = map.get( word ); // get current count
            map.put( word, count + 1 );  // increment count
         } // end if
         else
            map.put( word, 1 ); // add new word with a count of 1 to map
       } // end while
   } // end method createMap

   // display map content
   private void displayMap()
   {
      Set< String > keys = map.keySet(); // get keys

      // sort keys
      TreeSet< String > sortedKeys = new TreeSet< String >( keys );

      System.out.println( "Map contains:\nKey\t\tValue" );

      // generate output for each key in map
      for ( String key : sortedKeys )
         System.out.printf( "%-10s%10s\n", key, map.get( key ) );

      System.out.printf(
         "\nsize:%d\nisEmpty:%b\n", map.size(), map.isEmpty() );
   } // end method displayMap

   public static void main( String args[] )
   {
      new WordTypeCount();
   } // end main
} // end class WordTypeCount
```

UNIVERSIDADE VILA VELHA-ES

Exercício

Analise2.java

- Diga se as linhas de códigos abaixo são válidas ou não e explique o motivo.

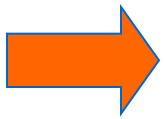
```
1  import java.util.List;
2  import java.util.ArrayList;
3
4  class Analise2{
5
6      public static void main(String args[]){
7          List lista = new ArrayList<Integer>();
8          lista.add(8);
9          metod(lista);
10     }
11
12     static void metod(ArrayList a){
13         System.out.println(a);
14     }
15 }
```

UNIVERSIDADE VILA VELHA-ES

Exercício

- Desafio – Performance
  - 1) Crie um código que insira 100 mil números numa ArrayList e pesquise-os.
    - Vamos usar um método de System para cronometrar o tempo gasto:

```
1.    public class TestaPerformance {
2.
3.          public static void main(String[] args) {
4.                System.out.println("Iniciando...");
5.                long inicio = System.currentTimeMillis();
6.                Collection<Integer> teste = new ArrayList<Integer>();
7.
8.                for (int i = 0; i < 30000; i++) {
9.                      teste.add(i);
10.               }
11.
12.               for (int i = 0; i < 30000; i++) {
13.                     teste.contains(i);
14.               }
15.
16.               long fim = System.currentTimeMillis();
17.               double tempo = (fim - inicio) / 1000.0;
18.               System.out.println("Tempo gasto: " + tempo);
19.         }
20.   }
```



UNIVERSIDADE VILA VELHA-ES

Exercício

- Desafio – Performance
  - 2) Troque a ArrayList por um HashSet e verifique o tempo que vai demorar

UNIVERSIDADE VILA VELHA-ES

Exercício

- Desafio – Performance
  - 1) Crie uma comparação entre ArrayList e LinkedList, para ver qual é a mais rápida
    - Para se adicionar elementos na primeira posição (list.add(0, elemento))
    - Para se percorrer usando o get(indice)
  - 2) Depois faça utilizando o enhanced for ou o iterator.

UNIVERSIDADE VILA VELHA-ES

Exercício

- Desafio – Performance
  - Refaça os “Desafio – Performance” só que agora coletando dados com várias quantidades de elementos diferentes, construa uma tabela e um gráfico e analise a curva de performance mostrando a partir de que ponto cada implementação se torna mais interessante.

UNIVERSIDADE VILA VELHA-ES

Exercício

- Desafios
  - 1) Gere todos os números entre 1 e 1000 e ordene em ordem decrescente utilizando um TreeSet.
  - 2) Gere todos os números entre 1 e 1000 e ordene em ordem decrescente utilizando um ArrayList.