Programação para Web I
Aula 1

Profa. Silvia Bertagnolli

Revisão O.O.

Orientação a Objetos

A linguagem Java utiliza-se do paradigma da Orientação a Objetos

Principais conceitos:

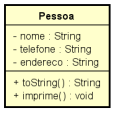
- abstração (classe, objeto, etc.)
- herança e composição de objetos
- polimorfismo
- encapsulamento

Classe

Orientação a Objetos: Classe

```
public class Pessoa{
    private String nome;

    public Pessoa(){}
    public Pessoa(String nome){ this.nome = nome;}

    public String toString(){return "Pessoa [nome=" + nome + "]";}
    public boolean equals(Object e){ … }
    public void setNome(String nome){ this.nome = nome; }
    public String getNome(){return nome;}
    //outros métodos
}
```

Como fica o código da classe Pessoa?




```
public class Pessoa{

        private String nome;




}
```

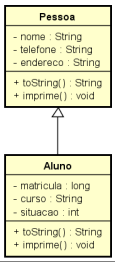
?

Herança

Orientação a Objetos: Herança

- Superclasse e subclasse
- Relacionamento **é-um?**
- Na linguagem de programação Java **Object** é a superclasse das classes que não definem uma superclasse comum
- Para referenciar construtores e métodos da superclasse usamos a palavra reservada **super**

Como fica o código da classe Aluno?




```
public class Aluno _____________________ {

}
```

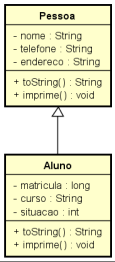
?

Orientação a Objetos: anotação @Override

- Esta anotação diz ao compilador que está sendo reescrito um método na classe
- Isso impede que alterações sejam realizadas na assinatura do método toString(), por exemplo
- Evita problemas com o uso da sobrecarga, exemplo:

```
public double calculaSalario(double taxa){...}

public int calculaSalario(int taxa){...}
```

Como fica o código da classe Aluno?




```
public class Aluno _____________________ {

    @Override
    public String toString(){
        return ...;
    }

}
```

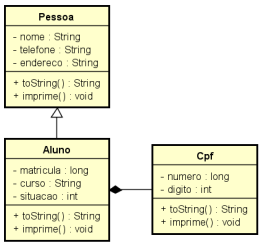
?

Composição

Orientação a Objetos: composição

- Relacionamento **tem-um?**
- Composição e Agregação (todo-parte)
- Associação (origem-destino)
- Multiplicidade deve ser analisada
  - 1..1
  - 1..*
  - 2..3

Como fica o código da classe Aluno?

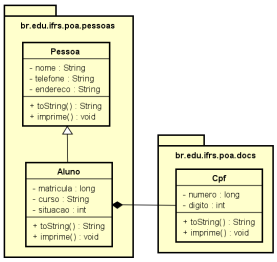


Como fica o código da classe Aluno agora?

```
public class Aluno __________________ {




}
```

?

Como fica o código da classe usando Pacotes?



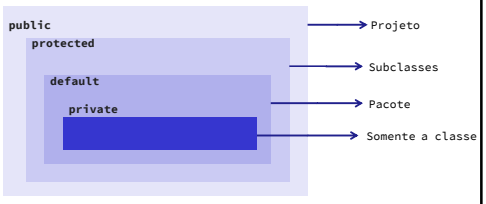
Como fica o código da classe Aluno agora?

```
___________________________________
___________________________________
public class Aluno __________________ {




}
```

?

Encapsulamento

Modificadores e sua Visibilidade



Recomendações

Modificador/Elemento	Classe	Método	Atributo
public	✓	✓	✓
private	x	✓	✓
protected	x	✓	✓

Recomendações

Modificador/Elemento	Classe	Método	Atributo
public	✓	✓	✓
private	x	✓	✓
protected	x	✓	✓

Resumo

Visibilidade	public	protected	private	default
Da mesma classe	✓	✓	✓	✓
De qualquer classe no mesmo pacote	✓	✓		✓
De qualquer classe que não seja uma subclasse externa ao pacote	✓			
De uma subclasse do mesmo pacote	✓	✓		✓
De uma subclasse externa ao pacote	✓	✓		

Instanceof x Cast

instanceof é um operador e determina qual a classe de um dado objeto

```
Pessoa p[] = new Pessoa[3];
p[0] = new Pessoa();
p[1] = new Aluno();
if(p[0] instanceof Pessoa)
    System.out.println("Objeto é um aluno");
```

Instanceof x Cast

cast "força" que um objeto seja convertido para outro

```
Pessoa p[] = new Pessoa[3];
p[0] = new Pessoa();
p[1] = new Aluno();
if(p[1] instanceof Aluno){
   Aluno a = (Aluno)p[1];
   System.out.println(a.getCpf());

//ou:
if(p[1] instanceof Aluno){
System.out.println(((Aluno)p[1]).getCpf());
```

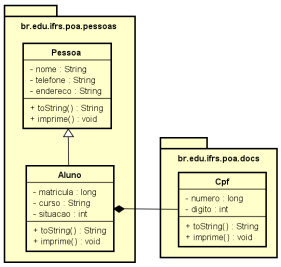
Exercícios

Exercícios

1. Crie as classes da figura ao lado
2. Agora, monte o menu abaixo usando vetores

1 - Cadastrar Aluno
2 - Pesquisar Aluno pelo nome
3 - Pesquisar Aluno pelo CPF
4 - Listar todos os alunos
5 - Sair

Obs.: Para resolver o item 3 do menu você deverá utilizar **instanceof** e/ou **cast** de objetos



Coleções

Coleção: o que é?

Estrutura de dados que permite armazenar vários objetos (Listas, Filas, Pilhas, Árvores binárias, etc.)

Coleções são usadas para:

- armazenar
- recuperar
- manipular
- pesquisar

} objetos

Coleções Primitivas

A versão 1.0 do Java tinha suporte para:

- Vector, Stack, Hashtable, Properties, BitSet e Enumeration

Após vários outros tipos de coleções foram introduzidos:

- ArrayList, LinkedList, TreeSet, LinkedHashSet, HashSet, TreeMap, HashMap, LinkedHashMap

Coleções Java

Collections podem ser:

- **organizadas** coleções serão percorridas na mesma ordem em que os elementos foram inseridos - LinkedHashSet, ArrayList, Vector, LinkedList, LinkedHashMap

- **ordenadas** possui métodos/regras para ordenação dos elementos - TreeSet, PriorityQueue, TreeMap

Coleções Java

Listas: ordenadas, podem conter elementos duplicados, mantendo a ordem em que foram adicionados e usam índices – implementam List

Conjuntos: itens exclusivos, mantém sua própria ideia de ordem, sem pesquisa através de índices – implementam Set

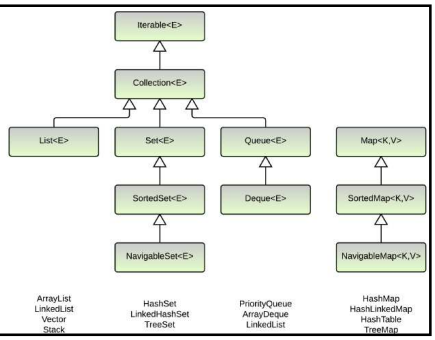
Queue: filas que são usadas para conter elementos antes do processamento (usado com threads)

Coleções Java

Deque: Do inglês “Double Ended QUEue”, onde uma queue permite apenas inserções no fim da fila e remoções do seu início, um deque permite que inserções e remoções sejam feitas no início e no fim da fila, ou seja, um objeto que implemente Deque pode ser usado tanto como uma fila FIFO (firstin, firstout), quanto uma fila LIFO (lastin, firstout).

Mapas: usados para armazenar pares de objetos. Possuem itens com uma identificação (chave) exclusiva – implementam Map

Hierarquia de Interfaces



Interface Iterable

Seu objetivo é definir que qualquer coleção “filha” possa ser percorrida pelo “for melhorado”

Métodos que permitem percorrer qualquer tipo de coleção

- boolean hasNext(): retorna true se existem mais elementos a serem acessados na coleção vinculada a esse Iterator
- E next(): retorna o próximo elemento disponível na coleção vinculada a esse iterator
- void remove(): remove da coleção o último elemento acessado através desse Iterator

Interface Collection

Define os métodos mais gerais, independentes da estrutura e da forma de acesso da coleção:

Define um padrão de operações básicas para as coleções:

- size() – determina o número de elementos armazenados
- remove() – remove o elemento informado
- add() – adiciona o elemento informado
- isEmpty() – verifica se está vazio
- iterator() – percorre a coleção
- contains() – verifica se um elemento está armazenado

Interface Collection

Operações em massa:

- boolean addAll(Collection<? extends E> c): adiciona à coleção todos os elementos da coleção passada como parâmetro
- void clear(): Esvazia essa collection, mas não elimina da memória os objetos que ela referenciava, a não ser que não haja mais nenhuma outra referência para os mesmos
- boolean containsAll(Collection<?> c): retorna true se a coleção contém todos os elementos da coleção informada como parâmetro

Interface Collection

Operações em massa:

- boolean removeAll(Object o): remove da coleção todos os elementos da coleção informada como parâmetro
- boolean retainAll(Collection<?> c): mantém na coleção somente os elementos da coleção informada como parâmetro
- Object[] toArray(): converte essa coleção para um array de Object.

Listas

Interface List

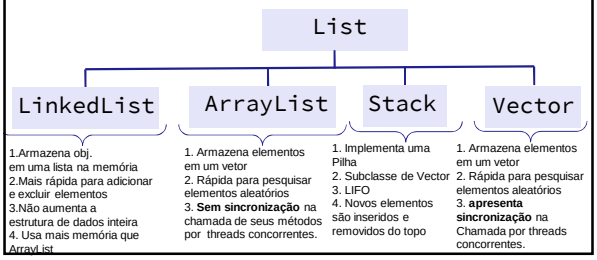
Define coleções que se organizam como arrays de tamanho dinâmico, de forma que cada elemento seja acessível por um índice

Permite acesso posicional – índices (0 até size-1)

Novos elementos podem ser criados ou removidos em qualquer posição e pode haver elementos duplicados.

Nem todas as listas garantem acesso indexado com tempo constante

Hierarquia da Classes de Listas



ArrayList

- é uma lista em array, logo tem acesso direto aos seus elementos através do índice e permite adicionar e remover elementos de forma eficiente apenas no fim

- caso seja necessário aumentar o seu tamanho, o custo será alto, já que todo o array será copiado para um novo com maiores dimensões

ArrayList: exemplo

```
public class Lista1{
     public static void main(String args[]){
            ArrayList<Integer> lista = new ArrayList<>();
            lista.add(new Integer(10));
            lista.add(new Integer(20));
            for(Integer obj: lista) {
                  System.out.println(obj);
            }
            System.out.println(lista.indexOf(20));
     }
}
```

LinkedList

- Usa uma lista duplamente encadeada de elementos, onde cada elemento sabe quem é o próximo e quem é o anterior
- Primeiro e último elementos podem ser acessados de maneira direta, mas os restantes terão um custo de acesso (elemento na posição N da lista, temos que passar por todos os elementos de 0 até N1)
- O maior benefício é que ela pode crescer indefinidamente
- Caso a lista cresça constantemente a melhor opção é a LinkedList

LinkedList: Métodos (1/2)

Inclusão:

```
public void add(in index, Object element)
public void addFirst(Object element)
public void addLast(Object element)
```

Recuperação:

```
public Object getFirst()
public Object getLast()
```

LinkedList: Métodos (2/2)

Exclusão:

```
public boolean remove(Object element)
public Object removeFirst()
public Object removeLast()
```

Os métodos addFirst() e removeFirst() podem simular uma pilha (LIFO - Last In First Out)

Os métodos addLast() e removeFirst() podem simular uma fila (FIFO - First In First Out)

LinkedList: exemplo

```
public class Lista3{
    public static void main(String args[]){
            LinkedList<Number> lista = new LinkedList<>();
            lista.add(new Integer(10));
            lista.add(new Integer(20));
            lista.add(new Integer(30));
            lista.add(new Integer(20));
            lista.removeFirst();
            lista.removeLast();
    }
}
```

LinkedList:uso do get

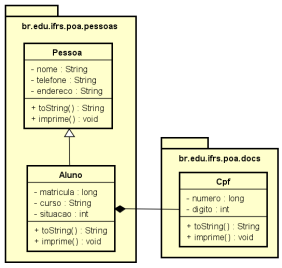
Não usar o método get(), porque o acesso aos elementos é aleatório o que ocasionar perda de desempenho

```
for(int i=0; i< list.size(); i++)
        System.out.println(list.get(i));
```

Exercícios

Exercícios

1. Como declarar um ArrayList para objetos do tipo Cpf?
2. Como declarar uma LinkedList para armazenar objetos do tipo Pessoa e Aluno?



Conjuntos

Interface Set

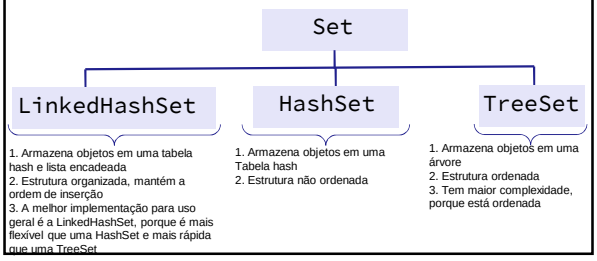
Dá relevância à exclusividade – não admite elementos duplicados, Caso dois objetos sejam iguais, considerando o método equals, apenas um será incluído

Representação para a abstração matemática de "conjuntos"

Suporta as operações de união, intersecção e diferença entre conjuntos

Podem ser vazios, mas não podem ser infinitos

Hierarquia da Classes de Conjuntos



1. Armazena objetos em uma tabela hash e lista encadeada
2. Estrutura organizada, mantém a ordem de inserção
3. A melhor implementação para uso geral é a LinkedHashSet, porque é mais flexível que uma HashSet e mais rápida que uma TreeSet

1. Armazena objetos em uma Tabela hash
2. Estrutura não ordenada

1. Armazena objetos em uma árvore
2. Estrutura ordenada
3. Tem maior complexidade, porque está ordenada

Interface Set Principais Métodos

`public boolean addAll(Collection c)`

adiciona ao conjunto que o invocar todos os elementos da coleção passada como parâmetro – equivale a operação de **UNIÃO** de conjuntos

`public boolean retainAll(Collection c)`

mantém no conjunto que executar o método somente os elementos encontrados no conjunto passado como parâmetro – equivale a operação de **INTERSECÇÃO** de conjuntos

`public boolean removeAll(Collection c)`

Remove do conjunto que executar o método todos os objetos iguais encontrados no conjunto passado como parâmetro – equivale a operação de **DIFERENÇA** de conjuntos

HashSet

Como utiliza hash é mais rápido para operações de modificação

Conjunto não organizado/classificado e não ordenado

Usar: quando for necessário um conjunto sem duplicatas e sem ordem para iteração

HashSet: exemplo

```
HashSet<String> conjunto = new HashSet<>();
conjunto.add("Dois");
conjunto.add("Tres");
conjunto.add("Um");
conjunto.add("Um");
for(String num : conjunto) {
        System.out.println(num);
}
```

LinkedHashSet

Permite a iteração na ordem em que os elementos foram acessados

Ordenado pela sequência de inserção

LinkedHashSet: exemplo

```
LinkedHashSet<String> cidades = new LinkedHashSet<>();
cidades.add("Porto Alegre");
cidades.add("Canoas");
cidades.add("Alvorada");
cidades.add("Viamão");
for(String cidade: cidades) {
        System.out.println(cidade);
}
```

Sáida:
Porto Alegre
Canoas
Alvorada
Viamão

TreeSet

Permite que os elementos fiquem em seqüência ascendente – ordem natural

Mais lenta que HashSet ou LinkedList

Árvore com n elementos = $\log_2 n$ comparações para localizar a posição correta do novo elemento

Permite customizar a ordem dos elementos – definir as regras de ordenação

TreeSet: exemplo

```
TreeSet<String> cidades = new TreeSet<>();
cidades.add("Porto Alegre");
cidades.add("Canoas");
cidades.add("Alvorada");
cidades.add("Viamão");
for(String cidade: cidades) {
        System.out.println(cidade);
}
```

Saída:
Alvorada
Canoas
Porto Alegre
Viamão

Exercícios

Exercícios

1. Como declarar um TreeSet para objetos do tipo Integer?
2. Crie dois conjuntos e faça a união, diferença ou intersecção deles usando métodos de Set
3. Como declarar um LinkedHashSet para armazenar objetos do tipo Aluno