UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

Faculdade de Ciências e Tecnologia

Bacharelado em Ciência da Computação

Programação Orientada a Objetos II

Classes e Objetos

Material preparado pelo prof. Me. Bruno Santos de Lima

1

Programação Orientada a Objetos

2

Programação Orientada a Objetos

unesp

- O Paradigma **Orientado a Objetos** utiliza uma **abordagem** em que unidades de **dados** são **vistas** como **"objetos"** ativos.
  - Encontramos **muitos objetos ativos no mundo real**
    - ✓Veículos, Portas, Elevadores, Pessoas, etc.
- As **ações** a serem realizadas sobre esses objetos estão **incluídas nos próprios objetos**.
  - Por exemplo:
    - ✓Um Veículo, Carro, pode ser dirigido, ligar, dar seta, subir vidro, etc...
    - ✓Uma Pessoa, João, pode andar, correr, pular, falar, cantar, etc...
- Essas ações são chamadas de **métodos** e podem ser executadas quando **recebem um estimulo** para isso.

3

Programação Orientada a Objetos

unesp

- O Paradigma **Orientado a Objetos** faz uma **abstração** do que acontece no mundo real.
- Os **objetos são entidade abstratas** relacionadas com entidades do mundo real.
- **A abordagem Orientada a Objetos foca mais no problema.**
  - Sendo uma importante abordagem para resolver problemas através de simulação
- Em Orientação a Objetos chamamos de **CLASSE** a entidade que especifica um objeto determinando seu **estrutura e comportamento**.

4

Classes e Objetos

5

Classes

unesp

- As **Classes** representam **conceitualmente toda e qualquer definição da abstração do mundo real modelada computacionalmente**.

- Todo conhecimento sobre o **domínio do problema** é utilizado para definir um conjunto de Classes.
  - ✓Cada Classe funciona como uma espécie de **molde** para a criação de um dado objeto.
  - ✓Os **objetos** são vistos como representações físicas **concretas** da abstração definida por uma Classe.

6

Classes

unesp

- Uma **Classe** é formada pelo conjunto: **Atributos** e **Métodos. [estrutura e comportamento]**
  - **Atributos:**
    - ✓Características gerais e próprias que os objetos pertencentes a Classe possuem.
    - ✓Corresponde aos dados que um objeto contém.
      - ✓Um atributo é uma variável que pertence a um objeto.
      - ✓Os dados de um objeto são armazenados em seus atributos.
  - **Métodos:**
    - ✓Representam os comportamentos que um objeto possui.
    - ✓São expressos na forma de procedimentos ou funções que realizam as ações próprias do objeto.
    - ✓Por meio dos métodos que os objetos interagem com outros objetos.

7

Classes

unesp

- As **Classes** representam **conceitualmente toda e qualquer definição da abstração do mundo real modelada computacionalmente**.
  - **Definem as características e comportamentos que os Objetos dessa Classe devem ter.**
    - ✓ **Exemplo:** Classe Carro

Características/Atributos

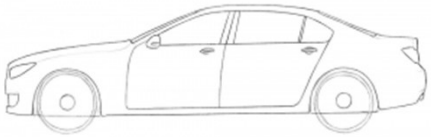
Modelo: ?
Cor: ?
Placa: ?
Número de portas: ?
Etc..

8

Classes

unesp

- As **Classes** representam **conceitualmente toda e qualquer definição da abstração do mundo real modelada computacionalmente**.
  - **Definem as características e comportamentos que os Objetos dessa Classe devem ter.**
    - ✓ **Exemplo:** Classe Carro



Comportamentos/Métodos

Ligar()
Acelerar()
Frear()
AcionarSeta()
Etc..

"Ao considerar o domínio do problema como um simulador automotivo"

9

Objetos

unesp

- Os **Objetos** são entidades com **existência física** que pertence a uma determinada **Classe** ou **conjunto de Classes**.
  - **Um Objeto é dito como uma instância de uma Classe.**

- Entidade capaz de **reter um estado** (informações) e que oferece uma série de informações (comportamentos) para **modificação desse estado** se necessário.
  - ✓Por meio dos objetos, das interações entre objetos, **troca de mensagens** entre objetos, que um sistema ou aplicação desenvolvida seguindo o Paradigma Orientada a Objetos.

10

Classes e Objetos

- Os **Objetos** são entidades com **existência física** que pertence a uma determinada **Classe** ou **conjunto de Classes**.
  - **Um Objeto é dito como uma INSTÂNCIA de uma Classe.**
    - ✓ Exemplo: Classe Carro, Objeto Ferrari e Objeto Onix

Modelo: ?
Cor: ?
Placa: ?
Número de portas: ?

CLASSE

Modelo: Onix
Cor: Prata
Placa: HBA-2210
Número de portas: 4

OBJETO 1

Modelo: Ferrari
Cor: Vermelho
Placa: BKS-4547
Número de portas: 2

OBJETO 2

Classes e Objetos

- Os **Objetos** são entidades com **existência física** que pertence a uma determinada **Classe** ou **conjunto de Classes**.
  - **Um Objeto é dito como uma INSTÂNCIA de uma Classe.**
    - ✓ Exemplo: Classe Carro, Objeto Onix e Objeto Ferrari

CLASSE CARRO	OBJETO CARRO 1	OBJETO CARRO 2
ATRIBUTOS	**Modelo:** Onix	**Modelo:** Ferrari
	Cor: Prata	**Cor:** Vermelho
	Placa: HBA-2210	**Placa:** BKS-4547
	Número de portas: 4	**Número de portas:** 2
MÉTODOS	Ligar	
	Acelerar	
	Frear	
	Acionar Seta	

Classes e Objetos

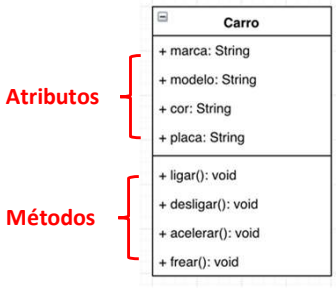
CLASSE: **É o modelo ou molde utilizado para construção de objetos. O modelo define as características e comportamentos que os objetos irão possuir. Consiste na abstração.**

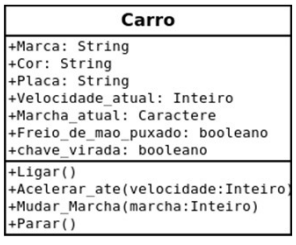
OBJETO: **É a entidade com existência física, uma instância da Classe**

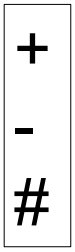
13

Modelagem de Classes: UML

- **UML – *Unified Modeling Language***
  - Linguagem de Modelagem Unificada
- Linguagem padrão para a elaboração da estrutura de projetos de software







14

Exemplo Prático:
Aplicação Banco

15

Exemplo Prático: Aplicação Banco

unesp

- Com objetivo de aplicar na prática e exercitar os principais conceitos de Classes e Objetos, vamos utilizar um exemplo de Aplicação Bancária.

16

Exemplo Prático: Aplicação Banco

unesp

- Nosso exemplo começa com a implementação da classe Conta.
  - ➢Classe **CONTA**:
    - **Atributos:** O que toda conta possui?
      - ✓ Número da conta
      - ✓ Nome do dono da conta
      - ✓ Saldo da conta
      - ✓ Limite da conta
    - **Métodos:** O que toda conta faz? O que gostaríamos de "pedir à conta"?
      - ✓ Sacar uma quantidade x
      - ✓ Depositar uma quantidade x
      - ✓ Extrato do saldo atual
      - ✓ Transfere uma quantidade x para outra conta
      - ✓ Imprime todos os dados da conta

Exemplo Prático: Aplicação Banco

unesp

- **Atributos e Instância de Objeto**

Atributos

```
public class Conta {

    int numero;
    String dono;
    double saldo;
    double limite;
}
```

```
public class Principal {

    public static void main(String[] args) {

        Conta minhaConta;
        minhaConta = new Conta();

    }

}
```

Instância do objeto chamado *minhaConta* pertencente a classe Conta.

Exemplo Prático: Aplicação Banco

unesp

```
public class Principal {

    public static void main(String[] args) {

        Conta minhaConta;
        minhaConta = new Conta();

        minhaConta.dono = "Luke";
        minhaConta.saldo = 1000.0;

        System.out.println("Saldo atual: " + minhaConta.saldo);
    }

}
```

Acesso ao objeto *minhaConta* alterando diretamente os atributos dono e saldo, pois são public

19

Exemplo Prático: Aplicação Banco

unesp

- **Métodos:**
  - Inicialmente, dois métodos são definidos
    - **saca()**
      - ✓ Recebe como parâmetro um valor do tipo *double*.
      - ✓ Retira esse valor do saldo da conta.
    - **deposita()**
      - ✓ Recebe como parâmetro um valor do tipo *double*.
      - ✓ Adiciona esse valor do saldo da conta.

```
public class Conta {

    int numero;                                   // Atributos
    String dono;
    double saldo;
    double limite;

    void saca(double valor){                      // Método
        this.saldo = this.saldo - valor;
    }

    void deposita(double valor){                  // Método
        this.saldo = this.saldo + valor;
    }
}
```


20

Exemplo Prático: Aplicação Banco

unesp

- **Métodos:**
  - Uso dos métodos **saca** e **deposita** pelo objeto ***minhaConta.***

```
public class Principal {

    public static void main(String[] args) {

        Conta minhaConta;
        minhaConta = new Conta();

        minhaConta.dono = "Luke";
        minhaConta.saldo = 1000.0;

        minhaConta.saca(200);
        minhaConta.deposita(500);

        System.out.println("Saldo atual: " + minhaConta.saldo);
    }
}
```


Exemplo Prático: Aplicação Banco

unesp

- **Método:**
  - Os métodos são procedimentos que permitem:
    - ✓ Passagem de parâmetros.
    - ✓ Retorno de tipos de dados, inclusive retorno de outros objetos.
  - O método saca() foi modificado
    - ✓ Retorna um booleano indicando se o saque foi ou não realizado.
    - ✓ Possui verificação se existe saldo suficiente na conta para o saque.

```
void saca(double valor){
    this.saldo = this.saldo - valor;
}
```

```
public class Conta {

    int numero;
    String dono;
    double saldo;
    double limite;

    boolean saca(double valor){
        if(this.saldo > valor){
            this.saldo = this.saldo - valor;
            return true;
        }
        else return false;
    }

    void deposita(double valor){
        this.saldo = this.saldo + valor;
    }
}
```

Estado válido, consistente. Mas, ainda é possível alterar o saldo diretamente.

Exemplo Prático: Aplicação Banco

unesp

- **Método:**

```
public class Principal {

    public static void main(String[] args) {

        Conta minhaConta;
        minhaConta = new Conta();

        minhaConta.dono = "Luke";
        minhaConta.saldo = 1000.0;

        boolean result = minhaConta.saca(2000);

        if(result) System.out.println("Saque realizado!");
        else System.out.println("Saque não realizado, falta de saldo!");

    }
}
```


Exemplo Prático: Aplicação Banco

unesp

- **Instanciação de múltiplos objetos de uma mesma Classe:**

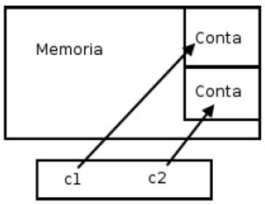
```
public class Principal {

    public static void main(String[] args) {

        Conta c1 = new Conta();
        c1.dono = "Luke";
        c1.saldo = 227;

        Conta c2 = new Conta();
        c2.dono = "Luke";
        c2.saldo = 227;
    }

}
```



Apesar dos objetos c1 e c2 serem da mesma classe e possuírem os mesmos valores para os atributos. Eles são objetos diferentes, ocupam espaços diferentes em memória, endereços diferentes.

Exemplo Prático: Aplicação Banco

unesp

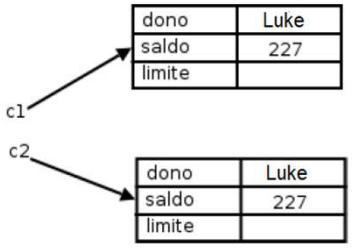
```
public class Principal {

    public static void main(String[] args) {

        Conta c1 = new Conta();
        c1.dono = "Luke";
        c1.saldo = 227;

        Conta c2 = new Conta();
        c2.dono = "Luke";
        c2.saldo = 227;
    }

}
```

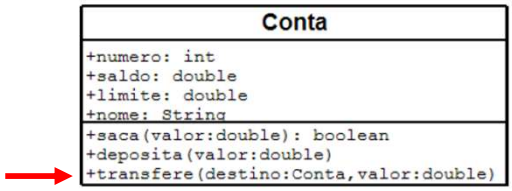


25

Exemplo Prático: Aplicação Banco

unesp

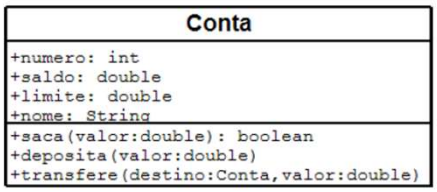
- Classe: CONTA
  - Novo método: **transfere()**
    - ✓Possibilita transferir um determinado valor para uma outra conta especificada.



26

Exemplo Prático: Aplicação Banco

- Classe: CONTA
  - Novo método: **transfere()**
    - ✓Possibilita transferir um determinado valor para uma outra conta especificada.

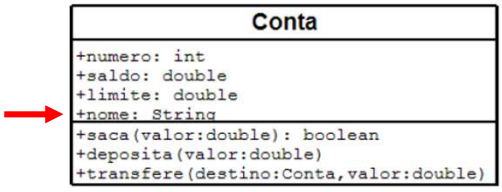



```
boolean transfere(Conta destino, double valor){
    boolean retirou = this.saca(valor);
    if(retirou == true){
        destino.deposita(valor);
        return true;
    }
    else return false;
}
```


27

Exemplo Prático: Aplicação Banco

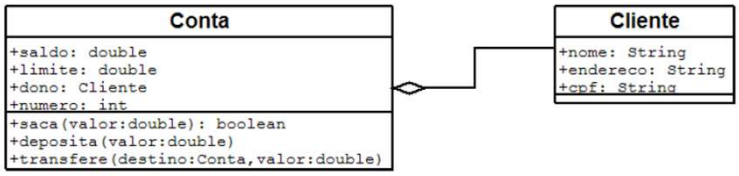
- Classe: CONTA
  - Inicialmente a classe Conta foi modelada possuindo como atributo o nome do dono da conta...
    - Entretanto, uma classe descreve apenas características dela mesma.
      - **Uma conta não possui nome!** Quem possui nome é o dono da conta, ou seja, o Cliente do Banco.
        - Logo, o correto é criar uma Classe **CLIENTE** para representar todos os clientes do Banco como objetos dentro da aplicação.



28

Exemplo Prático: Aplicação Banco

- **Definição dos objetos Conta e Cliente**
  - Uma conta possui um Titular (Cliente), logo foi realizado uma **composição**, em que um dos atributos da conta é um objeto da classe CLIENTE.
    - ✓ A Classe Cliente possui todas as definições próprias de um cliente.
    - ✓ A Classe Conta possui todas as definições próprias de uma conta.




```
public class Conta {

    int numero;
    String dono;
    double saldo;
    double limite;
}
```

```
public class Conta {

    int numero;
    double saldo;
    double limite;
    Cliente titular;

}
```

```
public class Cliente {

    String nome;
    String sobrenome;
    String cpf;
}
```

29

29

Exemplo Prático: Aplicação Banco

- **Instância de objeto Conta e objeto Cliente**

```
public class Conta {

    int numero;
    double saldo;
    double limite;
    Cliente titular;

}

public class Cliente {

    String nome;
    String sobrenome;
    String cpf;
}
```

```
public class Principal {

    public static void main(String[] args) {

        Conta minhaConta = new Conta();
        Cliente c = new Cliente();
        minhaConta.titular = c;

    }

}
```


30

Construtores

31

Exemplo Prático: Aplicação Banco

unesp

```
public class Conta {

    int numero;
    double saldo;
    double limite;
    Cliente titular;

}
public class Cliente {

    String nome;
    String sobrenome;
    String cpf;
}
```

```
public class Principal {

    public static void main(String[] args) {

        Conta minhaConta = new Conta();
        Cliente c = new Cliente();
        minhaConta.titular = c;

    }

}
```


32

Construtores

- Os **construtores** são utilizados para inicializar objetos de um determinada classe.
  - O uso de construtores possibilitam prover um estado inicial (inicializar) os atributos de um objeto de uma determinada classe.
  - **Construtor:**
    - ✓Possui o mesmo nome da Classe
    - ✓Pode possuir um ou mais parâmetros
      - ✓Por padrão toda classe possui um construtor sem nenhum parâmetro
    - ✓Sempre é chamado pelo comando new
    - ✓Não possui valor de retorno
      - ✓Logo, não é um método!
    - ✓Não pode ser chamado como um método da classe

Exemplo Prático: Aplicação Banco

- **Construtor: Cliente**

```
public class Cliente {

    String nome;
    String sobrenome;
    String cpf;

    Cliente(){
    }

    Cliente(String nome, String sobrenome, String cpf){
        this.nome = nome;
        this.sobrenome = sobrenome;
        this.cpf = cpf;
    }
}
```

```
public class Principal {

    public static void main(String[] args) {

        Cliente cliente1 = new Cliente("Luke", "Skywalker", "001");
        cliente1.imprimir();

    }

}
```

Exemplo Prático: Aplicação Banco

unesp

- **Construtor: Conta**

```
public class Conta {

    int numero;
    double saldo;
    double limite;
    Cliente titular;

    Conta(){
    }

    Conta(int numero, double saldo, double limite, Cliente titular){
        this.numero = numero;
        this.saldo = saldo;
        this.limite = limite;
        this.titular = titular;
    }
```

```
public class Principal {

    public static void main(String[] args) {

        Cliente cliente1 = new Cliente("Luke", "Skywalker", "001");
        cliente1.imprimir();

        Conta conta1 = new Conta(123, 2000.0, 5000.0, cliente1);
        conta1.imprimir();

    }
}
```

35

35

Exemplo Prático: Aplicação Banco

unesp

- **Novos métodos:**
  - Imprimir() : Imprime os atributos especificados pela classe de um determinado objeto.

✓ Método imprimir() da classe Cliente:

```
void imprimir(){
    System.out.println("\nCliente: " + this.nome + " " + this.sobrenome +
                       "\nCPF: " + this.cpf);
}
```

✓ Método imprimir() da classe Conta:

```
void imprimir(){
    System.out.println("\nConta\nN° conta: " + this.numero +
                       "\nSaldo: " + this.saldo +
                       "\nLimite: " + this.limite);
    this.titular.imprimir();
}
```

36

36

Modificadores de Acesso

37

Modificadores de Acesso

unesp

- Os **modificadores de acesso** são palavras-chave usadas para especificar a acessibilidade da declaração de um membro ou tipo de dado.
  - Os modificadores de acesso são aplicáveis a atributos, classes, métodos, interfaces e enumeradores.
  - **Níveis de modificadores de acesso:**
    - ***Public***
      - ✓ O uso desse modificador possibilita acesso livre em qualquer lugar do programa.
    - ***Private***
      - ✓ O uso desse modificador possibilita acesso restrito e visível somente dentro da classe onde ocorreu a declaração.
    - ***Protected***
      - ✓ O uso desse modificador possibilita acesso somente para classe que possui a declaração e derivados dessa classe.

38

Exemplo Prático: Aplicação Banco

unesp

- Modificadores de Acesso:

```
public class Cliente {

    private String nome;
    private String sobrenome;
    private String cpf;

    Cliente(){
    }

    Cliente(String nome, String sobrenome, String cpf){
        this.nome = nome;
        this.sobrenome = sobrenome;
        this.cpf = cpf;
    }

    public void imprimir(){
        System.out.println("\nCliente: " + this.nome + " " + this.sobrenome +
                           "\nCPF: " + this.cpf);
    }
}
```

39

Encapsulamento

Encapsulamento

- Ideia de encapsular:
  - **Caixa preta**
    - ✓ **Esconder todos os membros, atributos de uma classe, além de esconder o funcionamento dos métodos dessa classe**

Encapsulamento: "Capacidade de ocultar, esconder os dados de um modelo, além disso, impende a obtenção de informações sobre como foi implementado ou manipulado determinado dado"

  - **Exemplo:**
    - **Carteira/Pagamento com smartphone**

Exemplo Prático: Aplicação Banco

- Em nosso exemplo prático, para **possibilitar o encapsulamento**:
  - Todos os atributos da classe **Cliente** e da Classe **Conta** devem utilizar como **modificador de acesso**: ***private*** ou ***protected*****.**
  - Vamos utilizar o ***private***, com ele não é possível modificar ou ler o dado de um atributo.
    - Para isso utilizamos os métodos *Getters()* e *Setters()*:
      - ***Getters()***
        - Utilizado especialmente para acessar (ler) dados de atributos privados.
      - ***Setters()***
        - Utilizado especialmente para modificar (escrever) dados de atributos privados.
    - Os *Getters()* e *Setters()* são implementações de responsabilidade da própria classe, por meio dele que a classe pode ou não prover acesso de leitura e escrita a um de seus dados.
      - Todos os dados? Não, somente o que a classe desejar prover acesso, e como será o acesso ou manipulação é omitido para outras partes do programa.

Exemplo Prático: Aplicação Banco

unesp

- Alguns *Getters()* e *Setters()* da classe CONTA:

```
public class Conta {

    private int numero;
    private double saldo;
    private double limite;
    private Cliente titular;
```

```
public double getSaldo() {
    return saldo;
}

public void setSaldo(double saldo) {
    this.saldo = saldo;
}

public Cliente getTitular() {
    return titular;
}

public void setTitular(Cliente titular) {
    this.titular = titular;
}
```

43

43

Exemplo Prático: Aplicação Banco

unesp

- Alguns *Getters()* e *Setters()* da classe CLIENTE:

```
public class Cliente {

    private String nome;
    private String sobrenome;
    private String cpf;
```

```
public String getNome() {
    return nome;
}

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

public String getSobrenome() {
    return sobrenome;
}

public void setSobrenome(String sobrenome) {
    this.sobrenome = sobrenome;
}

public String getCpf() {
    return cpf;
}
```

44

44

Lista de Objetos

45

Lista de Objetos

unesp

- Em muitos casos, é necessário implementar uma **Lista de Objetos**.
  - Exemplo: Uma classe **BANCO** poderia possuir como atributo uma Lista de objetos da classe **CLIENTE**.
  - Existe diferentes modos para implementar listas de objetos.
    - Array
    - ArrayList

46

Exemplo Prático: Lista de clientes

unesp

```
public class Principal {
    public static void main(String[] args) {

        Cliente c1 = new Cliente("Luke", "Skywalker", "001");
        Cliente c2 = new Cliente("Mestre", "Yoda", "002");
        Cliente c3 = new Cliente("Obi-Wan", "Kenobi", "003");
        Cliente c4 = new Cliente("Qui-Gon", "Jinn", "004");

        ArrayList<Cliente> clientes = new ArrayList();
        clientes.add(c1);
        clientes.add(c2);
        clientes.add(c3);
        clientes.add(c4);

        for(int i = 0; i < clientes.size(); i++){
            clientes.get(i).imprimir();
        }
    }
}
```


```
public class Principal {
    public static void main(String[] args) {

        Cliente c1 = new Cliente("Luke", "Skywalker", "001");
        Cliente c2 = new Cliente("Mestre", "Yoda", "002");
        Cliente c3 = new Cliente("Obi-Wan", "Kenobi", "003");
        Cliente c4 = new Cliente("Qui-Gon", "Jinn", "004");

        ArrayList<Cliente> clientes = new ArrayList();
        clientes.add(c1);
        clientes.add(c2);
        clientes.add(c3);
        clientes.add(c4);

        for(Cliente c : clientes){
            c.imprimir();
        }
    }
}
```


47

Exemplo Prático: Lista de clientes

unesp

```
for(int i = 0; i < clientes.size(); i++){
    clientes.get(i).imprimir();
```


```
public class Principal {
    public static void main(String[] args) {

        Cliente c1 = new Cliente("Luke", "Skywalker", "001");
        Cliente c2 = new Cliente("Mestre", "Yoda", "002");
        Cliente c3 = new Cliente("Obi-Wan", "Kenobi", "003");
        Cliente c4 = new Cliente("Qui-Gon", "Jinn", "004");

        ArrayList<Cliente> clientes = new ArrayList();
        clientes.add(c1);
        clientes.add(c2);
        clientes.add(c3);
        clientes.add(c4);

        for(Cliente c : clientes){
            c.imprimir();
        }
    }
}
```


48

Exercício

- Implementar e testar a aplicação banco
  - Classes: Cliente, Conta, Principal
  - Com todos os construtores
  - Com os setters e getters (acessibilidade private para os atributos)
- Implementar e testar a aplicação lista de clientes
- NetBeans (Java Application)

49

Obrigado!

50