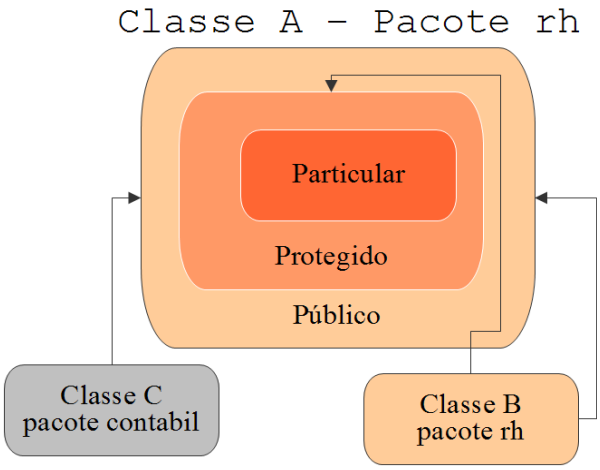
Programação OO em Java

Série Java – Cartão de Referência

1. **Classe** – ideia, não existe realmente

2. **Objeto** – derivado de uma classe, existe

Lembre-se que:
1ª – POO não é linguagem, é uma metodologia
2ª – Não tenha medo da reutilização
3ª – Pensar que POO é sempre uma solução "sucesso"
4ª – Nunca faça programação egoísta



Modificadores

public – acesso aberto a qualquer classe
protected – acesso aberto a classe do mesmo pacote
private – acesso limitado a classe que o criou

chamamos modificador default, a ausência de um dos modificadores de acesso, o acesso torna-se ilimitado para a família (por herança), mesmo de pacotes diferentes

final – classes, métodos e atributos não poderão ser modificados
static – métodos e atributos existem para a classe

synchronized – os métodos terão acesso sincronizado
volatile – os atributos serão muilti-acessados
transiente – os atributos só serão criados quando usados
native – utilizado para criar o programa em outra linguagem

Regra de Criação dos Tipos

Criar um objeto: modificador Classe nome = new Construtor();
Criar um atributo: modificador tipo nome = valorInicial;
Criar um método: modificador retorno nome([parâmetro]) {...}

Possíveis usos para os modificadores --------->

Modificador	Class	Atr/Obj	Método	Construtor
public	sim	sim	sim	sim
protected	não	sim	sim	sim
default	sim	sim	sim	sim
private	não	sim	sim	sim
final	sim	sim	sim	sim
static	não	sim	sim	não

Método Construtor

É um método normalmente com o acesso público, sem o valor de retorno e com o mesmo nome da classe, é utilizado no momento da instância da classe por um objeto (nascimento do objeto):

```
class Teste { pulic Teste() { } }
```

Na utilização por herança, a palavra "super()" se refere a utilização de um "construtor pai"

```
class A { pulic A(int a) { } }
class B extends A { pulic B(int b) { super(b); } }
```

Polimorfismo – Métodos Overload

São métodos com o mesmo nome que ocorrem dentro da mesma classe, entretanto devem receber parâmetros diferente de modo que sua chamada possa ser diferenciada.

```
public int calcCPF(long num) { }
public boolean calcCPF(long num, byte num) { }
```

Polimorfismo – Métodos Override

Ocorrem por herança para modificar ou complementar um determinado método, neste caso toda a assinatura do método deverá ser idêntica, ou respeitando-se a ordem de abertura dos modificadores de acesso.

Programação OO em Java

Série Java – Cartão de Referência

PRINCÍPIOS DA ORIENTAÇÃO A OBJETOS

1º Princípio da Orientação a Objetos – Abstração

Deve-se isolar os aspectos que sejam importantes para algum propósito no projeto e suprimir (eliminar) os que não forem.

"is-a" é feito por herança, enquanto que, "has a" é um novo objeto para classe. Por exemplo:

O usuário diz: O lar é uma (is-a) casa que tem uma (has-a) família.

O programador deve entender:

```
public class Lar extends Casa { private Familia familia; }
```

2º Princípio da Orientação a Objetos – Encapsulamento

Não é preciso conhecer o todo para saber o funcionamento. Para evitar acessos indevidos, todos os atributos e objetos de classe deverão ser definidos particulares para a classe (com o modificador "private"). Os atributos invariáveis (definidos com o modificador "final") escapam a essa regra.

```
private double salario;
public final int MATRICULA;
```

Para acessar os atributos particulares por outras classes, utilizamos os seguintes métodos modificadores:

Método **get** – para realizar o acesso ao atributo: `void setSalario(double s) { salario = s; }`
Método **set** – para obter o valor do atributo: double `getSalario() { return salario; }`

3º Princípio da Orientação a Objetos – Modularização

Quanto menos instruções existir em um método para executar, mais rápido este será executado. Devemos dividir para conquistar. Os nomes de classe e métodos devem representar exatamente o que devem fazer:

```
public class Teste {
  public void metodo() {
    // faz tudo
  }
}
```

```
public class Funcionario {
  public void adicionarSalario() { }
  public void promoveAumento(int p) { }
  public List retornarCurriculo() { }
}
```

4º Princípio da Orientação a Objetos – Herança ou Generalização

Uma classe pode gerar novas classes que sejam suas cópias perfeitas e a partir destas é possível readaptá-las ao meio em que vivem. Uma classe pode relacionar-se de três maneiras diferentes com outra:

- **Por Agrupamento** – quando uma classe domina completamente a outra, geralmente este tipo de relacionamento acarreta um atributo que representa um array de objetos da segunda classe na primeira. Ex. Pedido e ItemPedido, Funcionario e Dependente, Aquario e Peixe.
- **Por Associação** – quando uma classe utiliza simplesmente outra classe, geralmente este tipo de relacionamento acarreta um objeto da segunda classe na primeira. Ex. Pedido e Conta, Funcionario e Departamento, Aquario e Quarto.
- **Por Herança** – quando uma classe herda as características de uma classe já existente para realizar uma especialização (uma melhora). Ex. Pedido e PedidoUrgente, Funcionario e Gerente, Aquario e AquarioBerçario.

Classe Abstrata

Classe que possui métodos abstratos e pode possuir métodos reais. Não pode ser instanciada, deve obrigatoriamente ser extendida. Pode possuir atributos.

```
abstract class Mamifero {
  public abstract int mamar();
}
```

Interface

Só pode possuir métodos abstratos e públicos. Não pode ser instanciada e deve obrigatoriamente ser implementada. Atributos devem ser finais.

```
abstract interface SerVivo {
  public abstract void modoDeSuar();
}
```