LINGUAGEM DE PROGRAMAÇÃO JAVA

Cefet – Maracanã -RJ

BCC/TSI

Prof. Gustavo Guedes

E-mail: gustavo.guedes@cefet-rj.br

Herança

Herança

Enquanto programamos em Java, há a necessidade de trabalharmos com várias classes. Muitas vezes, classes diferentes tem características comuns, então, ao invés de criarmos uma nova classe com todas essas características usamos as características de um objeto ou classe já existente.

Ou seja, herança é, na verdade, uma classe derivada de outra classe.

Para simplificar de uma forma mais direta, vejamos:

Vamos imaginar que exista uma classe chamada Eletrodomestico, e nela estão definidos os seguintes atributos: ligado (boolean), voltagem (int) e consumo (int).

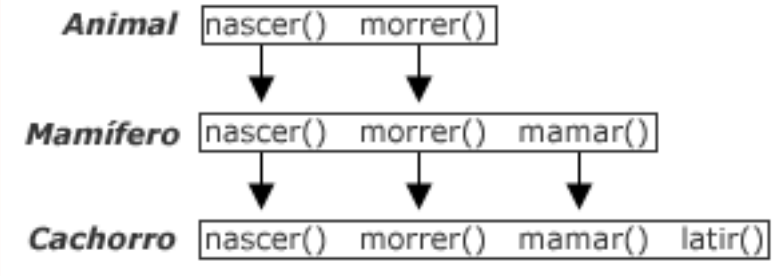
Se levarmos em conta a classe TV que estamos usando de exemplo até agora, podemos dizer que TV deriva de Eletrodomestico. Ou seja, a classe TV possui todas as características da classe Eletrodomestico, além de ter suas próprias características.

Extends e Super

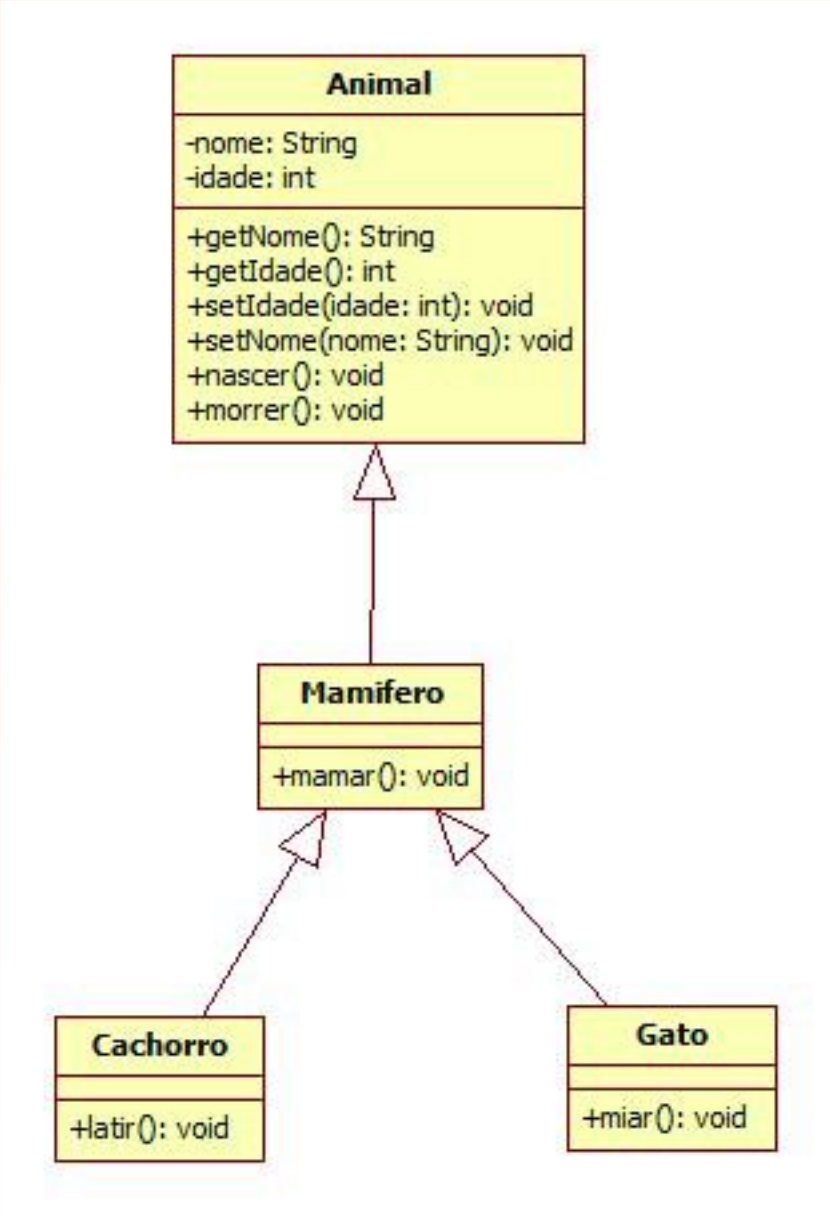
Para fazermos uma classe herdar as características de uma outra, usamos a palavra reservada `extends` logo após a definicação do nome da classe. Dessa forma:

class NomeDaClasseASerCriada **extends** NomeDaClasseASerHerdada

Importante: Java permite que uma classe herde apenas as características de uma única classe, ou seja, não pode haver heranças múltiplas. Porém, é permitido heranças em cadeias, por exemplo: se a classe Mamifero herda a classe Animal, quando fizermos a classe Cachorro herdar a classe Mamifero, a classe Cachorro também herdará as características da classe Animal.



exercício


Animal
-nome: String
-idade: int
+getNome(): String
+getIdade(): int
+setIdade(idade: int): void
+setNome(nome: String): void
+nascer(): void
+morrer(): void
Mamifero
+mamar(): void
Cachorro
+latir(): void
Gato
+miar(): void

Sobrescrita de método

- Além de produzir uma nova classe com base em uma antiga incluindo recursos adicionais, você pode modificar o comportamento atual da classe filha. Se um método é definido em uma subclasse, de modo que o nome, tipo de retorno e lista de argumentos de um método da classe filha seja igual ao nome, tipo de retorno e lista de argumentos de um método da classe pai, diz-se que o novo método sobrepôs ou sobrescreveu o antigo.

Sobrescrita de método

- Além de produzir uma nova classe com base em uma antiga incluindo recursos adicionais, você pode modificar o comportamento atual da classe filha. Se um método é definido em uma subclasse, de modo que o nome, tipo de retorno e lista de argumentos de um método da classe filha seja igual ao nome, tipo de retorno e lista de argumentos de um método da classe pai, diz-se que o novo método sobrepôs ou sobrescreveu o antigo.

Polimorfismo

- Polimorfismo é a capacidade de se referenciar a um objeto através de formas diferentes.
  - Um objeto tem somente uma forma.
  - Uma variável de referência pode se referir a objetos de diferentes formas.

- Vamos observar o exemplo abaixo:
- Animal a = new Cachorro();
- Quando fazemos isso, estamos olhando para o objeto Cachorro como um Animal. Dessa forma, poderemos chamar todos os métodos de Animal, mas nenhum de Cachorro. Pode parecer pouco realista criar um Cachorro
- e designar sua referência para uma variável do tipo Animal. Mas há motivos para se querer fazer isso.
- Um objeto tem apenas uma forma, aquela que lhe é dada quando construído. Entretanto, você pode olhar para um objeto de formas diferentes.

Polimorfismo

- O Java, assim como a maioria das linguagens OO, na verdade, permite que você se refira a um objeto com uma variável que seja um dos tipos da classe pai. Assim sendo, você pode dizer o seguinte:

- Mamifero m = new Gato();

- Usando a variável Mamifero como referência ao objeto Gato, você só pode acessar as partes do objeto que fazem parte de Mamifero ou de seus pais.
- As partes específicas de Gato estão ocultas. No que diz respeito ao compilador, Gato é um Mamifero, não um Gato.

Chamada de métodos

- Observe o cenário abaixo:

- Cachorro a = new Cachorro();
- Gato g = new Gato();

- Vamos supor que você tenha sobrescrito o métoto mamar() em Gato e em Cachorro. Ao carregar a.mamar() e g.mamar(), você chama comportamentos diferentes. O objeto Cachorro executa a versão mamar() associada à classe Cachorro, e o objeto Gato executa a versão de mamar() associada a classe Gato. Menos óbvio é o seguinte:

- Mamifero e = new Cachorro();
- e.mamar();

Chamada de métodos

- Na verdade, você tem o comportamento associado ao objeto ao qual a variável se refere no runtime. Esse momento não é determinado pelo tipo da variável no momento da compilação. Esse é o aspecto do polimorfismo, e uma característica importante das linguagens OO. Esse comportamento é muitas vezes denominado de chamada de método virtual. No exemplo anterior, o método e.mamar() executado é do tipo real do objeto, um Cachorro.

Argumentos polimórficos

- Você pode escrever métodos que aceitam um objeto "genérico" (neste caso, a classe Mamifero), e pode trabalhar adequadamente com os objetos de qualquer subclasse desse objeto. Você pode produzir um método em uma classe que alimenta um Mamifero. Usando recursos polimórficos, você pode fazer o seguinte:

```
public static void darMamadeira(Mamifero e){
    e.mamar();
}
...main... {
  Gato a = new Gato();
  Mamifero x = new Cachorro();
  Animal k = new Gato();
  darMamadeira (a);
  darMamadeira (x);
  darMamadeira (k);
}
```

O Operador instanceof

- Se você receber um objeto que usa uma referência do tipo Animal, ele pode
- "ser visto" como um Urso ou um Leao. Você pode testar usando o operador
- instanceof da seguinte forma:

```
public void facaAlgo (Animal e) {
    if (e instanceof Urso) {
        System.out.println("sou um urso");
    } else {
        System.out.println("sou um leao");
    }
}
```

Sobrecarga de métodos

- Métodos com o mesmo nome podem ser declarados na mesma classe, contanto que tenham diferentes conjuntos de parâmetros, determinados pelo número, tipo e ordem dos parâmetros.

```
public class Sobrecarga {
    public int getSoma(int x, int y) {
        return x+y;
    }
    public int getSoma(int x, int y, int z) {
        return x+y+z;
    }
}
public class SobrecargaTeste {
    public static void main (String [] args) {
        Sobrecarga s = new Sobrecarga();
        System.out.println(s.getSoma(1,2));
    }
}
```

Sobrecarga de métodos

- Métodos com o mesmo nome podem ser declarados na mesma classe, contanto que tenham diferentes conjuntos de parâmetros, determinados pelo número, tipo e ordem dos parâmetros.

```
public class Sobrecarga {
    public int getSoma(int x, int y) {
        return x+y;
    }
    public int getSoma(int x, int y, int z) {
        return x+y+z;
    }
}
public class SobrecargaTeste {
    public static void main (String [] args) {
        Sobrecarga s = new Sobrecarga();
        System.out.println(s.getSoma(1,2));
    }
}
```

Sobrecarga de construtores

- Ao criar uma instância de objeto, o programa pode querer fornecer vários construtores com base nos dados para o objeto que está sendo criado. Por exemplo, um sistema de folha de pagamento pode querer criar um objeto Empregado quando conhece todos os dados básicos (nome, salário inicial e data de nascimento). Eventualmente, o sistema pode não saber o salário inicial ou data de nascimento.

Sobrecarga de construtores

Chamando outro construtor

Um construtor só pode rodar durante a construção do objeto, isto é, você nunca conseguirá chamar o construtor em um objeto já construído. Porém, durante a construção de um objeto, você pode fazer com que um construtor chame outro, para não ter de ficar copiando e colando:

```
class Conta {

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

    // construtor
    Conta (Cliente titular) {
        //     faz mais uma série de inicializações e configurações
        this.titular = titular;
    }

    Conta (int numero, Cliente titular) {
        this(titular); // chama o construtor que foi declarado acima
        this.numero = numero;
    }
```

Sobrecarga de construtores

- Podemos notar que o segundo construtor chama o primeiro construtor passando o cliente.

- OBS: A palavra this (na chamada a outro construtor) dentro de um construtor precisa estar na primeira linha de código do construtor. Pode haver mais código de inicialização após a chamada this, mas nunca antes.

- Embora uma subclasse possa herdar os métodos e variáveis de uma classe pai (pública), **construtores não são herdados**. Assim como os métodos, os construtores podem chamar os construtores não privados de sua superclasse imediata. Para isso, basta usar a palavra reservada super a partir da primeira linha do construtor filho. Quando não há uma chamada para super com argumentos, o construtor pai com zero argumentos é chamado implicitamente. Nesse caso, se não houver nenhum construtor pai com zero argumentos, ocorrerá um erro do compilador.

Casting de objetos

- Em circunstâncias nas quais você recebeu uma referência para uma classe pai, e usando o operador instanceof, determinou que o objeto na verdade é uma subclasse particular, você pode restaurar a plena funcionalidade do objeto fazendo o casting da referência.

```
public void facaAlgo (Animal e) {
   if (e instanceof Elefante) {
      Elefante e2 = (Elefante)e;
      e2.metodoSoDeElefante();
    }
}
```

- Se você não fizer o casting, uma tentativa de executar e.metodoSoDeElefante(); não terá sucesso porque o compilador não pode localizar esse método na classe Animal. Se você não fizer o teste usando o instanceof, o casting pode falhar.

Casting de objetos

- Os casts feitos para cima na hierarquia de classe são sempr permitidos, de fato, não exigem o operador de casting. Elas podem ser feitas por meio de atribuição simples.

- Nos casts "para baixo", o compilador precisa aceitar que o cast é pelo menos possível. Por exemplo, qualquer tentativa de cast de uma referência Urso em uma referência Leao é definitivamente proibida, porque Urso não é um Leao. A classe para a qual o cast é feito deve ser alguma subclasse do tipo de referência corrente.

- Se o compilador permitir o cast, o tipo de objeto é verificado no runtime.

Casting de objetos

- Os castings feitos para cima na hierarquia de classe são sempre permitidos, de fato, não exigem o operador de casting. Eles podem ser feitas por meio de atribuição simples.

- Nos castings "para baixo", o compilador precisa aceitar que o casting é pelo menos possível. Por exemplo, qualquer tentativa de casting de uma referência "Urso" em uma variável de referência "Leao" é definitivamente proibida, porque Urso não é um Leao. A classe para a qual o casting é feito deve ser alguma subclasse do tipo de referência corrente.

- Se o compilador permitir o casting, o tipo de objeto é verificado no runtime.

A declaração package

- Quando um programador utiliza classes feitas por outro programador, surge um problema clássico: Como escrever duas classes com o mesmo nome?

A declaração package

- A linguagem de programação da tecnologia Java fornece a declaração package como uma forma de agrupar classes relacionadas. A declaração package tem o seguinte formato:
  - package <top_pkg_name>[.<sub_pkg_name>]*;
- Você pode indicar que as classes em um arquivo fonte pertencem a determinado pacote usando a declaração package como por exemplo:
  - package br.com.submarino.informatica;
  - public class Teclado {...

A declaração package

- A declaração package, quando existir, deve aparecer no início do arquivo fonte. Antes dela você só pode incluir espaços em branco e comentários, nada mais. Só é permitida uma declaração de pacote e ela governa todo o arquivo fonte. Se um arquivo fonte da tecnologia Java não tiver uma declaração de pacote, a(s) classe(s) declarada(s) naquele arquivo pertence(m) ao pacote não nomeado (default).

A declaração package

- Os nomes dos pacotes são hierárquicos e separados por pontos. É comum que os elementos do nome do pacote sejam informados em letras minúsculas. Entretanto, o nome da classe geralmente começa com letra maiúscula e você pode colocar a primeira letra de cada palavra adicional em maiúscula, para diferenciar as palavras dentro do nome da classe.

- Se não estiver incluída no arquivo nenhuma declaração de pacote, todas as classes declaradas em tal arquivo "pertencem" ao pacote default (ou seja, um pacote sem nome).

A declaração import

- A declaração import diz ao compilador onde encontrar as classes a serem utilizadas.
- A declaração import tem o seguinte formato:
  - import <pkg_name>[.<sub_pkg_name>].<class_name>;
  - ou
  - import <pkg_name>[.<sub_pkg_name>].*;

- Na verdade, o nome do pacote(por exemplo, br.com.submarino) é parte do nome das classes dentro do pacote. Você poderia ser referir a classe Telefone como br.com.submarino.Teclado, ou então poderia usar a declaração import e apenas o nome da classe Teclado.

- OBS: As declarações import devem preceder todas as declarações de classe.

A declaração import

- O fragmento de arquivo a seguir usa a declaração import:
- package shipping.reports;
- import shipping.domain.*;
- import java.util.List;
- public class VehicleCapacityReport {…

- Lembre-se que a declaração import é usada para colocar classes de outros pacotes à disposição da classe usada no momento.
- A declaração import especifica a classe que você deseja acessar. Por exemplo, se você quiser incluir somente a classe Writer (do pacote java.io), use:

- import java.io.Writer;

A declaração import

- Se você quiser ter acesso a todas as classes de um pacote, use ".*". Por exemplo, para acessar todas as classes no pacote java.io use:

- import java.io.*;

- OBS: Uma declaração import não faz o compilador carregar nada a mais na memória de trabalho.

- *Os pacotes são simplesmente diretórios.

A declaração import

- Uma variável ou método marcados com o modificador protected é, na verdade, mais acessível do que uma variável que tenha acesso default.

- Um método ou variável protected é acessível a partir de métodos em classes que são membros do mesmo pacote, e a partir de qualquer método em qualquer subclasse.

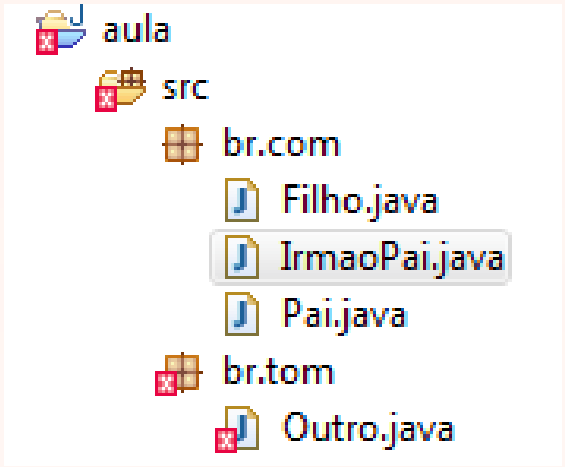
- Uma variável ou método marcados com o modificador public, podem ser acessados universalmente.

MODIFICADORES

Critérios de acessibilidade

Modificador	Mesma classe	Mesmo Pacote	Subclasse	Universo
private	sim	não	não	não
default	sim	sim	não	não
protected	sim	sim	sim	não
public	sim	sim	sim	sim

MODIFICADORES




```java
package br.com;

public class Pai {
    void metodoDefault() {

    }
    protected void metodoProtected() {

    }
    public void metodoPublico() {

    }
}
```

```java
package br.com;

public class IrmaoPai {
    public static void main(String[] args) {
        Pai p = new Pai();
        p.metodoDefault();
        p.metodoProtected();
    }
}
```

```java
package br.com;

public class Filho extends Pai{
    public static void main(String[] args) {
        Filho f = new Filho();
        f.metodoDefault();
        f.metodoProtected();
    }
}
```

```java
package br.tom;


import br.com.Pai;

public class Outro extends Pai{
    public static void main(String[] args) {
        Outro o = new Outro();
        o.metodoProtected();
        o.metodoDefault();
    }
}
```

A declaração import

- As variáveis e os métodos podem estar em um dos quatro níveis de acesso: public, protected, default ou private. As classes podem estar no nível public ou default.

- Uma variável ou método marcados como private, somente podem ser acessados por métodos que sejam membros da mesma classe.

- Uma variável, um método ou uma classe têm acessibilidade default quando não possuem um modificador de proteção explícito como parte de sua declaração. Essa acessibilidade significa que o acesso é permitido a partir de qualquer método em classes que são membros do mesmo pacote que o destino. Isso muitas vezes é chamado de "package-friendly".