Introdução

Orientação a Objetos

- Paradigma de análise, projeto e programação de sistemas
- Origem no campo de estudo da cognição
- Eliminar o "gap semântico"
  - Componentes de software que sejam o mais fiel na sua representação
- Linguagens
  - C++, C#, Java, Object Pascal, Objective-C, Python, Ruby e Smalltalk

JAVA

- Quais são os seus maiores problemas quando está programando?
  - ponteiros?
  - gerenciamento de memória?
  - organização?
  - falta de bibliotecas?
  - ter de reescrever parte do código ao mudar de sistema operacional?
  - custo de usar a tecnologia?

JAVA

- Java tenta amenizar esses “problemas”
- Idéia inicial: linguagem fosse usada em pequenos dispositivos
  - tvs, video-cassetes, aspiradores, liquidificadores...
- Apesar disso a linguagem teve seu lançamento focado no uso em clientes web (browsers) para rodar pequenas aplicações (applets)

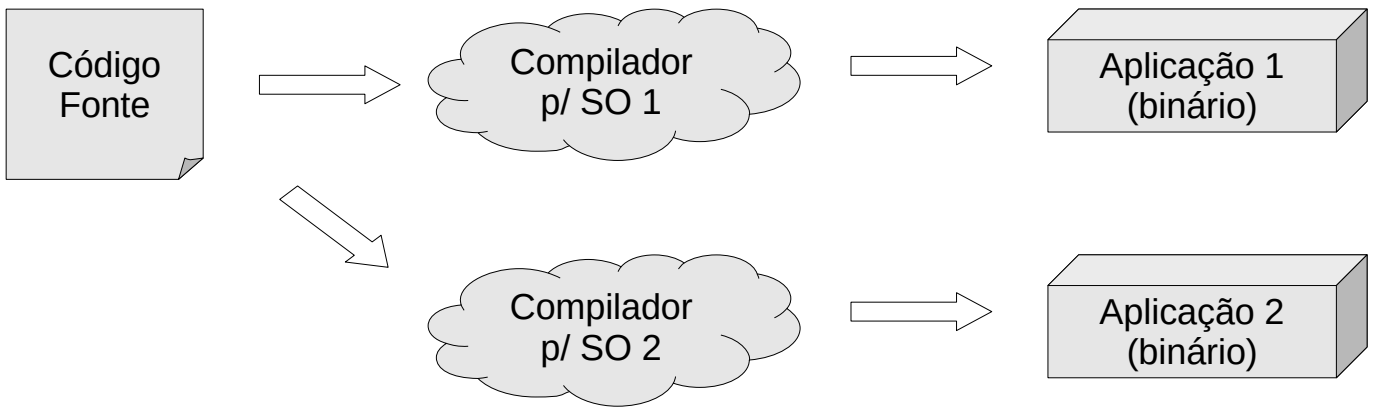
JAVA

- Linguagens convencionais são compiladas para código nativo

  - código fonte é compilado para uma plataforma e sistema operacional específicos

JAVA

- um código executável para cada sistema operacional. É necessário compilar uma vez para Windows, outra para o Linux, etc

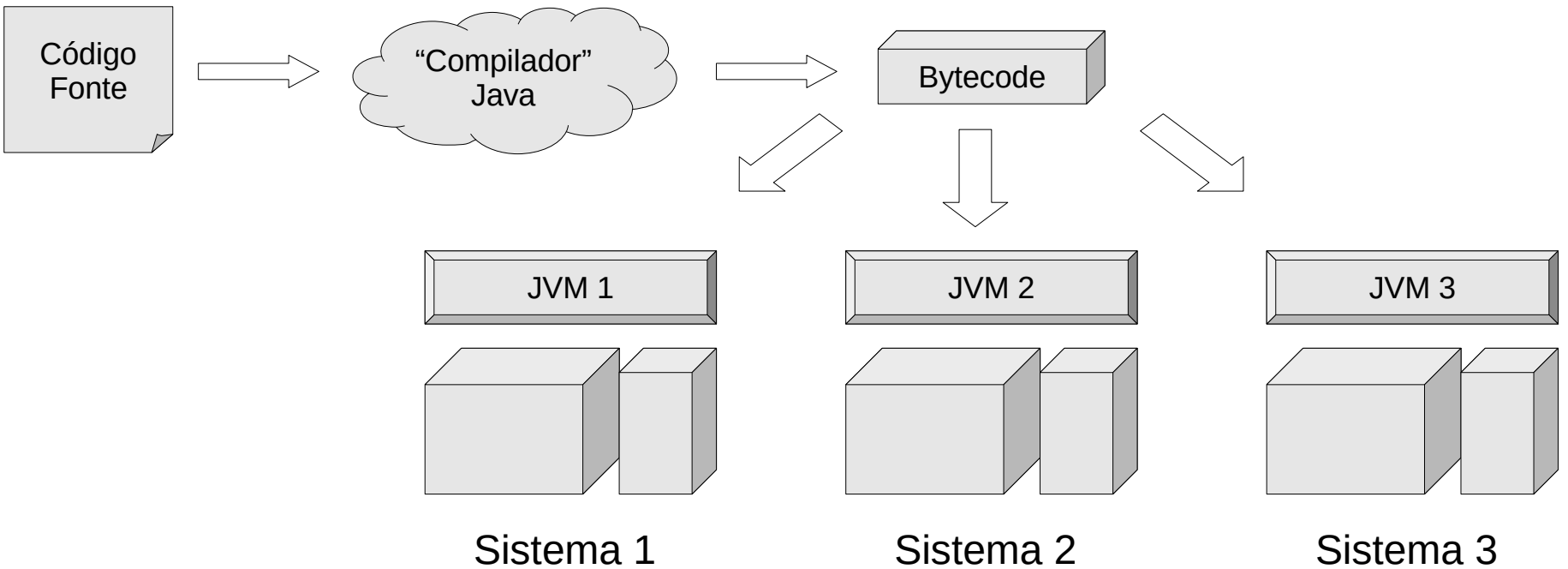


JAVA

- Java utiliza o conceito de **máquina virtual**
  - camada que “traduz” o que sua aplicação deseja fazer para as chamadas do sistema operacional onde ela está rodando
- O compilador Java gera **bytecode**
  - vai servir para diferentes sistemas operacionais
  - vai ser “traduzido” pela máquina virtual

JAVA

- “Write once, run anywhere” - slogan



JAVA

- Uma JVM isola totalmente a aplicação do sistema operacional
  - pode tirar métricas: decidir onde é melhor alocar a memória
  - Controla: memória, threads, pilha de execução
- Camada de isolamento
  - interessante quando um servidor não pode se sujeitar a rodar código que possa interferir na boa execução de outras aplicações

JAVA

- Mais lento?
  - Hotspot
    - detectar pontos quentes da aplicação
      - código que é executado muito, dentro de um ou mais loops
    - compilar para instruções nativas da plataforma
      - provavelmente melhora a performance da aplicação
  - JIT
    - Just in Time Compiler

JAVA

- Por que a JVM não compila tudo antes de executar a aplicação?
  - compilar dinamicamente, a medida do necessário, pode gerar uma performance melhor.
  - Compilador estático: otimização baseada em heurísticas
    - o compilador pode ter tomado uma decisão não tão boa
  - JVMs mais recentes, em alguns casos, chegam a ganhar, de códigos C compilados com o GCC 3.x, se rodados durante um certo tempo

JAVA

- Foco de Java não é de criar sistemas pequenos e sim aplicações de médio a grande porte
- Curva de aprendizado maior que outras linguagens, outros paradigmas
- Quantidade enorme de bibliotecas gratuitas para realizar os mais diversos trabalhos
  - pode criar uma aplicação sofisticada, usando diversos recursos

JAVA

- Nomenclaturas:
  - JVM = apenas a virtual machine
  - JRE = Java Runtime Environment, ambiente de execução Java, formado pela JVM e bibliotecas
    - tudo que se precisa para executar uma aplicação Java
  - JDK = Java Development Kit, ambiente de desenvolvimento JAVA.
    - tudo que se precisa para escrever códigos, gerar e executar aplicações Java.

Classe, Atributos e Métodos

Objeto

- Tudo pode ser um objeto
  - Usando uma sintaxe consistente
- Manipulação
  - Através da **referência** ao objeto

Referência

- Controle remoto pode existir sem uma TV
  - Referência pode existir sem um objeto
- Para ter uma frase, ou texto, deve-se criar uma referência de **String**

```
String s;
```

- Mas somente a referência é criada

Criando Objetos

- Ao criar uma referência, deve-se associá-la a um objeto
- Operador **new**
  - Significa: “Me faça um novo objeto deste tipo”

```
String s = new String(“abcd”);
```

- Não só cria um novo objeto
- Especifica **como** fazer o novo objeto
  - Ex: String inicializada com “abcd”

Orientação a Objetos

Criando um Tipo

- O que o tipo **tem** de importante?
  - Atributos
- O que o tipo **faz** de importante?
  - Métodos

Exemplo

- Programa para um banco → entidade Conta
- O que a Conta **tem** de importante?
  - Número da conta
  - Nome do dono
  - Saldo
  - Limite

Exemplo

- O que a conta **faz** de importante?
  - Saca uma quantidade x
  - Deposita uma quantidade x
  - Imprime o dono da conta
  - Devolve o saldo atual
  - Transfere um valor pra outra conta

Classe

- Projeto do tipo → Classe
  - Da biologia:
  - Indivíduos da mesma classe possuem atributos e comportamentos semelhantes, mas não são iguais
- Forma de descrever genericamente um objeto

Classe

- Atributos
  - Variáveis que descrevem características do objeto
- Exemplo: Conta

```
class Conta {
    // atributos
    int numero;
    String dono;
    double saldo;
    double limite;
}
```

Classe

- Métodos
  - Funções que descrevem comportamentos do objeto
- Sintaxe:

```
tipoRetorno Nome(tipo arg,...) {
   // corpo do método
}
```

Métodos

- Método de saque em conta
  - Quantidade informada como parâmetro

```
class Conta {
   // atributos
   double saldo;

   // métodos
   void saque (double quantidade) {
      saldo = saldo - quantidade;
   }
}
```

Métodos

- Método de depósito
  - Quantidade informada como parâmetro

```
void deposito(double quantidade) {
   saldo = saldo + quantidade;
}
```

Métodos

- Métodos com retorno
  - Exemplo: Saldo (devolve o valor atual do saldo da conta)

```
double ConsultaSaldo() {
  return saldo;
}
```

Métodos

- Método de transferência
  - Transfere um valor da conta específica para outra
    - Outra Conta é passada por parâmetro

```
void transfere(Conta rem, double val){
   this.saque(val);
   rem.deposita(val);
}
```

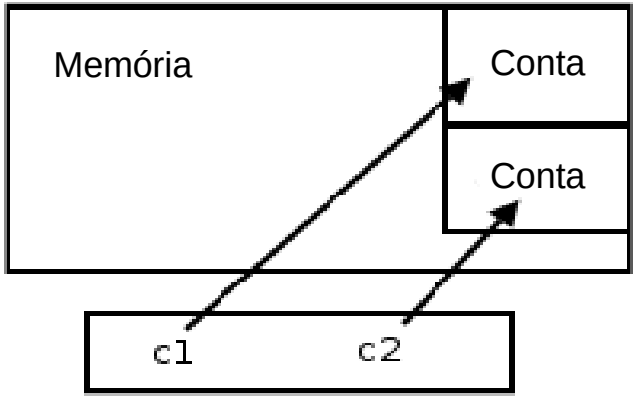
Criando Objetos

- Temos uma classe que molda um objeto
- É preciso criar Instâncias dessa classe
- Exemplo:
  - Criar um programa para controlar as Contas

```
class Programa {
    public static void main(String[] args) {
        Conta c1 = new Conta();
        Conta c2 = new Conta();
    }
}
```

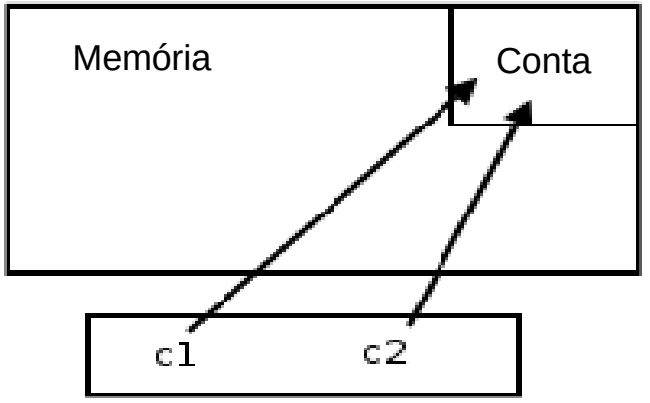
Criando Objetos

```
Conta c1 = new Conta();
Conta c2 = new Conta();
```



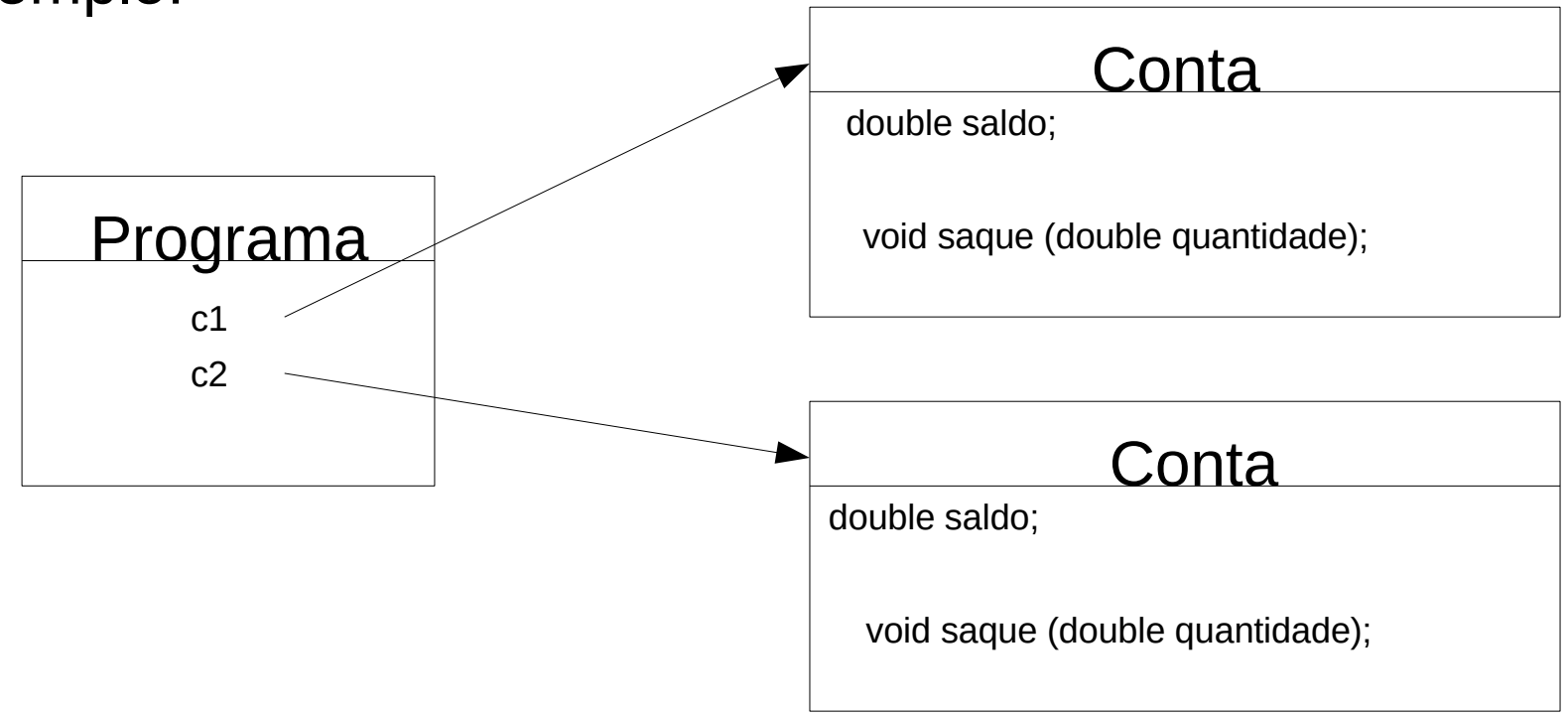
Criando objetos

```
Conta c1 = new Conta();
Conta c2 = c1;
```



Objetos

- Exemplo:



Visibilidade

- Modificador de acesso
  - public ou private
  - private
    - É visível somente dentro do escopo da Classe
  - public
    - Visível externamente
  - Via de regra: atributos → private

Static

- O objeto não é criado até o comando **new**
- Como acessar Atributos ou Métodos não instanciados?
- **Static**
  - Não é associado a um objeto
  - Pode ser chamado sem a criação do objeto

Main

- Método principal de uma classe (programa)
- Pelo menos uma classe do sistema deve conter o método **main**
- Onde o SO inicia a execução do sistema
- Onde os outros objetos são instanciados

Herança

Herança

- A ideia de herança é facilitar a programação
- Uma classe A deve herdar de uma classe B quando podemos dizer que A **é** um B
- Herança acontece quando duas classes são próximas
  - têm características mútuas
  - mas não são iguais
  - existe uma especificação de uma delas

Herança

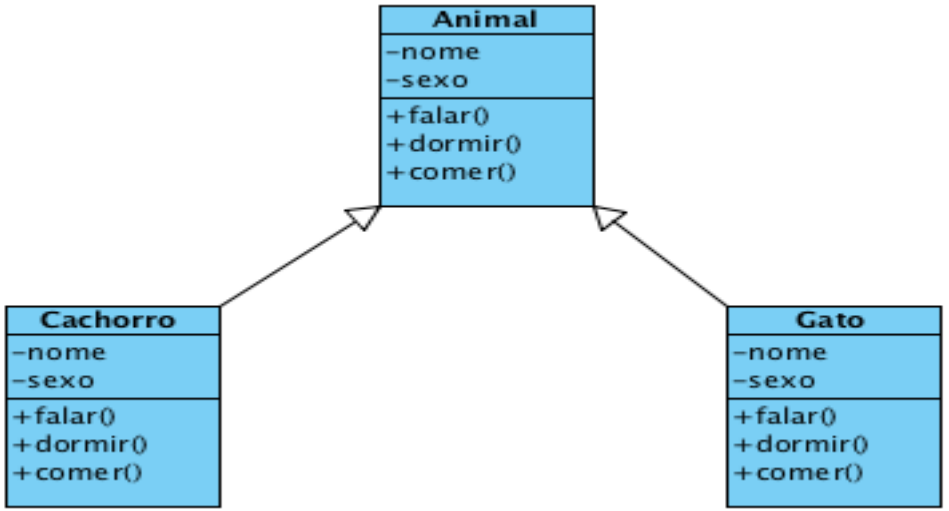
- É o relacionamento entre uma classe e um ou mais versões refinadas (especializadas) desta classe
- Permite a reutilização de código existente
- Facilita o projeto

Herança

- Classe base ou Superclasse
- A partir dela outras classes podem ser especificadas
  - cada classe derivada (subclasse) apresenta as características (estrutura e métodos) da classe base e acrescenta a elas o que for definido de particularidade para ela

Herança

- Exemplo

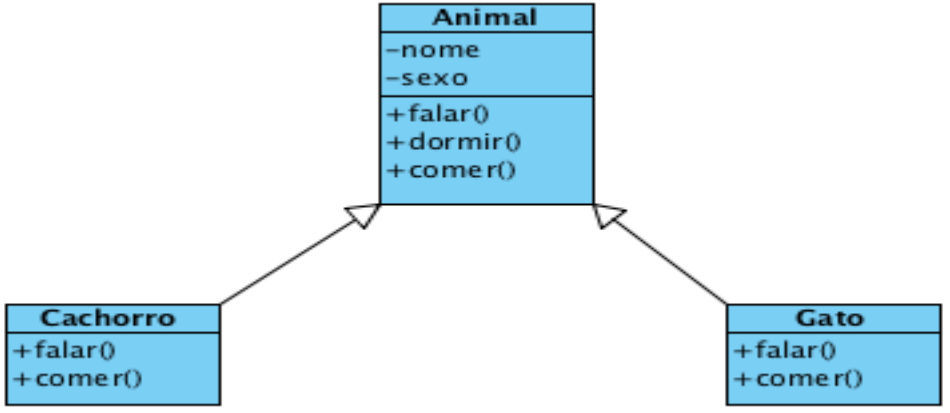


Superclasse: Animal

Subclasses: Cachorro, Gato

Herança

- No entanto, existem características comuns e outras diferentes entre as entidades envolvidas

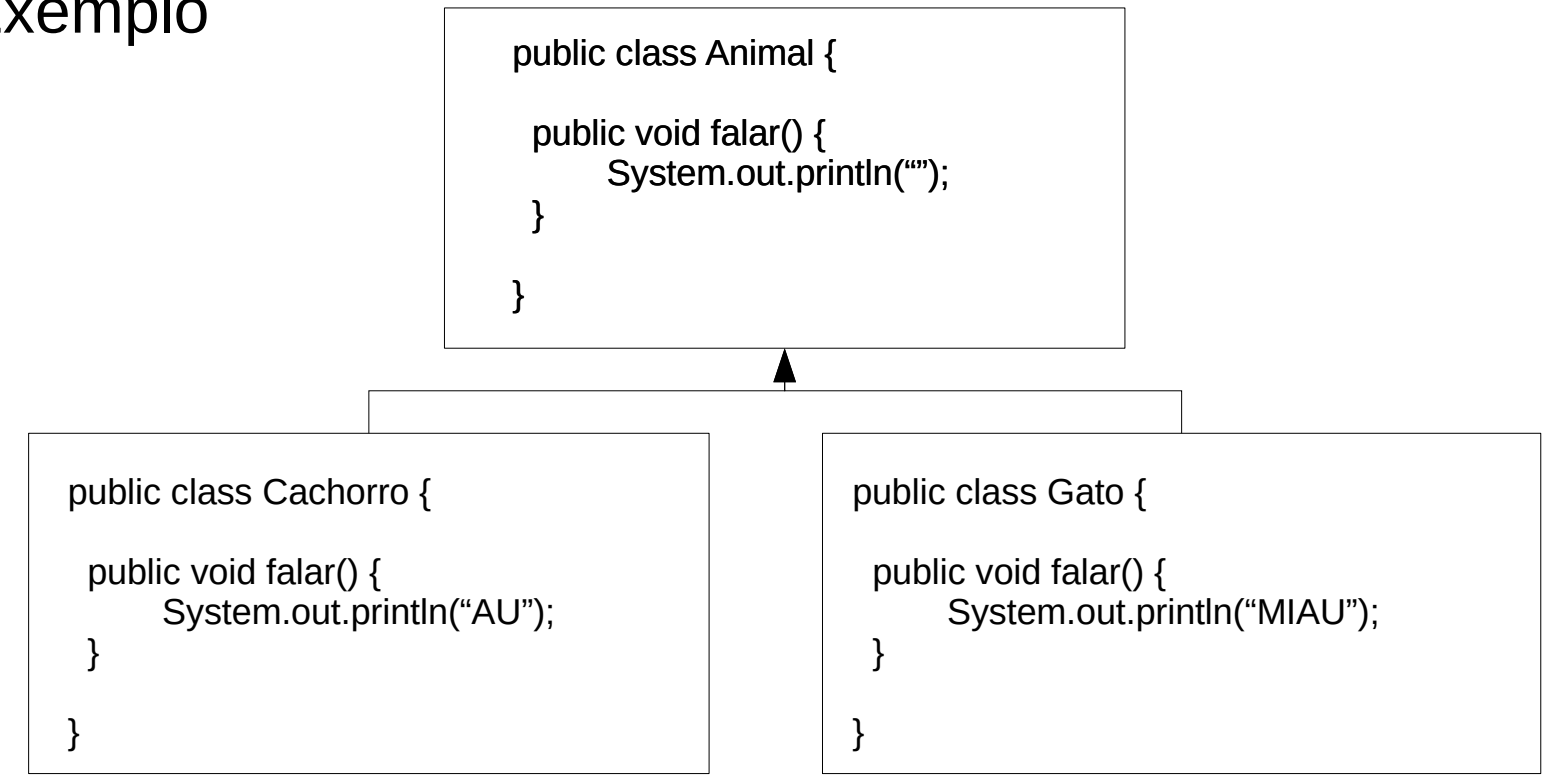


Sobrecarga de Métodos

- Em OO é possível reescrever os métodos da superclasse em uma subclasse
- Exemplo:
  - métodos “falar” e “comer” são redefinidos nas subclasses da classe Animal
  - define um método na classe com o mesmo nome
  - porém com propósitos diferentes

Sobrecarga de Métodos

- Exemplo



Sobrecarga de Métodos

- Permite a existência de vários métodos de mesmo nome
- Porém com assinaturas levemente diferentes
  - variando no número de argumentos
  - o tipo de argumentos
  - o valor de retorno

Sobrecarga de Métodos

- Exemplo

```
public class Soma {

  public int Soma(int x, int y) {
    return x+y;
  }

  public String Soma(String x, String y) {
    return x+y;
  }

  public double Soma(double x, double y) {
    return x+y;
  }

}
```

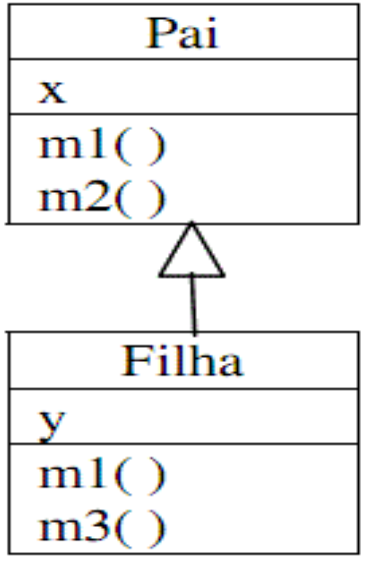
Polimorfismo

Polimorfismo

- Realização de uma tarefa de forma diferentes
  - Poli = muitas
  - Morphos = formas
- Conversão de objetos numa hierarquia de herança

Polimorfismo

- Herança



Polimorfismo

- Regra 1:
  - Em Java, podemos atribuir um objeto da subclasse a uma referência de sua superclasse.
- Esta operação é chamada upcasting
  - de up type casting
- Todo objeto da subclasse É UM objeto da sua superclasse

```
1.      Pai p = new Filha();
```

Polimorfismo

- Agora p, uma referência de superclasse, está apontando um objeto de subclasse

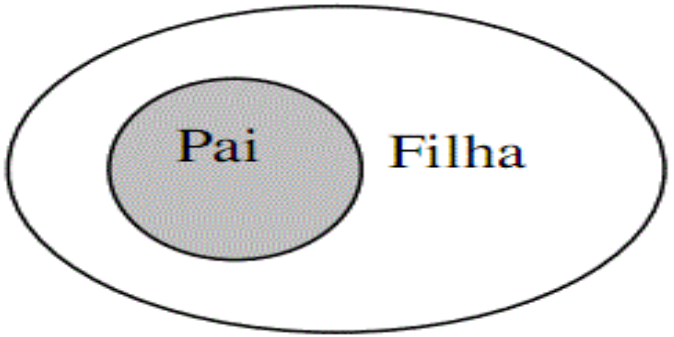
```
1. p.m1(); //chama m1( ) de Filha, porque ela sobrescreveu o m1 de Pai
2.
3. p.m2(); //chama m2( ) de Pai, herdado por Filha
```

- Porém, cuidado com a chamada seguinte:

```
1.  p.m3(); //ERRO de compilação
```

Polimorfismo

- Regra 2:
  - Uma referência de superclasse só reconhece membros disponíveis na superclasse, mesmo que esteja apontando para um objeto de subclasse.



Polimorfismo

- A Filha herda do Pai todo o seu conhecimento (atributos) e comportamentos (métodos)
  - pode acrescentar conhecimento e comportamentos novos exclusivamente seus
  - aos quais o Pai não tem acesso

Polimorfismo

- Regra 3:
  - Em Java, a atribuição de um objeto de superclasse a uma referência de subclasse, sem uma coerção explícita, não é permitida

```
1.Filha f = p; // ERRO de compilação
```

  - "forçar a barra" através de coerção

```
1.  Filha f = (Filha) p; //nome da classe destino entre parênteses
2.
3.  f.m3();
```

Polimorfismo

- Coerção:
  - Válido se soubermos que o objeto atualmente com referência de superclasse é, na realidade, um objeto da subclasse para a qual estamos convertendo
  - é algo verificado por Java apenas em tempo de execução
  - Se o objeto for do tipo da subclasse, a coerção será válida, mas se não for, ocorrerá uma ClassCastException

Polimorfismo

- Coerção anterior ficaria mais segura se codificada assim:

```
1. if (p instanceof Filha){
2.
3.   Filha f = (Filha) p;
4.
5.   f.m3();
6.
7.   }
```

Exercício

- Modelagem orientada a objetos (OO) para um campeonato de futebol. Objetivo, estatísticas gerais, jogos, gols, arbitragem, tabela, etc.

  1. Quantidade de classes
  2. Bons nomes de atributos e métodos
  3. Bom uso de herança → semântica e reutilização de código
  4. Encapsulamento e visibilidade
  5. Sobrecarga e Polimorfismo → Ponto extra