

Práticas de Desenvolvimento de Software

#

Aula 04

Modelando um problema com classes e objetos

Paradigma?

"Um paradigma de programação é uma forma de conceituar o que significa realizar computação e como tarefas a serem realizadas no computador devem ser estruturadas e organizadas."

Budd, 2001 (tradução livre)

Paradigma? (Aula 05)

Imperativo	Passos computacionais executados sequencialmente, levando o programa de um estado para outro.	C, Pascal
Funcional	Algoritmos matemáticos. Evita estados, favorecendo funções que dependem apenas de suas entradas.	LISP, Scheme, Scala
Lógico	Sentenças lógicas que expressam fatos e regras sobre o domínio do problema.	Prolog
Orientação a objetos	?	?

Orientação a objetos

- Forma de representar um problema e sua solução com elementos do domínio
- O problema é quebrado em partes menores e específicas
- Elementos do mundo real são representados como **abstrações** em objetos computacionais
- Cada objeto tem a sua **responsabilidade**
- O programa funciona como uma **coleção** de objetos interagindo
- Linguagens: Java, C#, C++, Ruby, Smalltalk, ...

Modelando um problema com classes e objetos



Objeto

- Elemento único do sistema que possui uma função específica
- Características: **estado** (dados) e **comportamento** (código)



Objeto
Conta Corrente

Estado

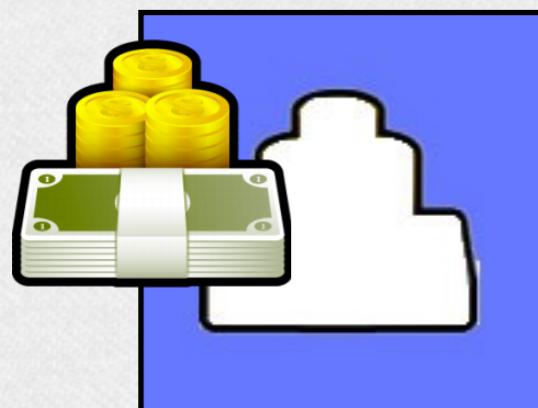
- Banco, agência e conta (111, 123, 681-2)
- Saldo (R\$ 200)
- Limite de crédito (R\$ 500)

Comportamento

- Depositar
- Sacar
- Transferir

Classe

- Molde ou definição de um novo tipo de dado
- Representa **características comuns** a um conjunto de objetos
- Um objeto é uma **instância** de uma classe



Classe
Conta Corrente

Atributos

- Representam o estado do objeto
- Cada objeto possui **valores diferentes** para os atributos

Comportamento

- Operações dos objetos da classe
- Normalmente mudam o estado do objeto
- Método: implementação de uma operação

Modelando um problema com classes e objetos

Exemplo

dog.rb

```
class Dog

  def initialize(name, age)
    @name = name
    @age = age
  end

  def bark
    puts "Woof! Woof!"
  end

  def run
    puts "I love to run!"
    bark()
  end

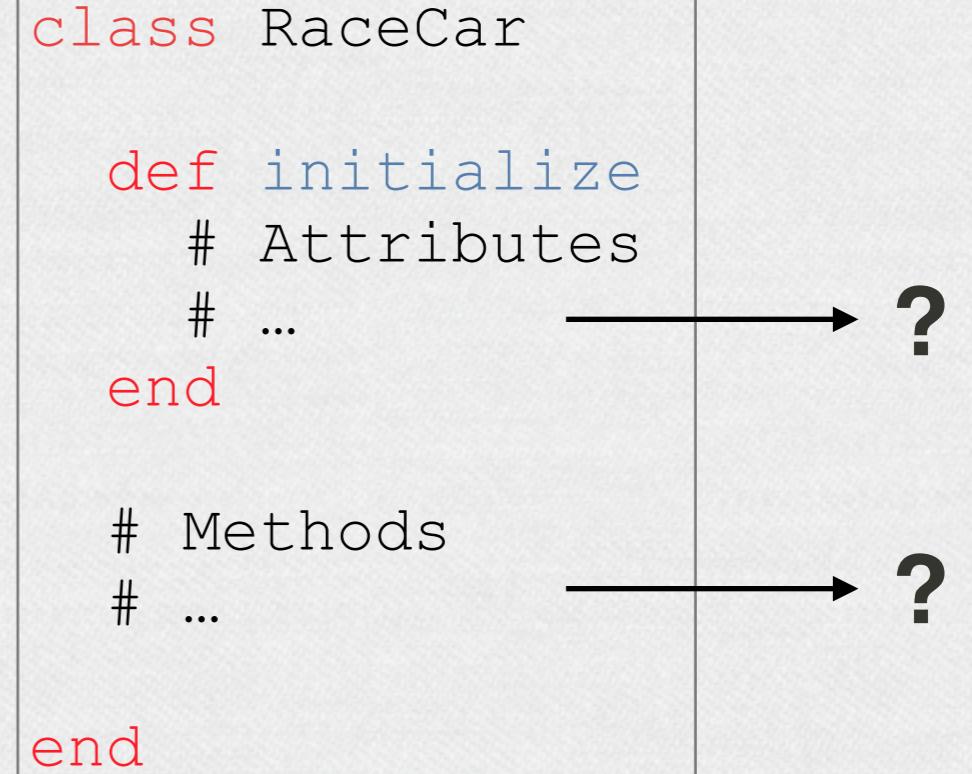
end
```

race_car.rb

```
class RaceCar

  def initialize
    # Attributes
    # ...
  end

  # Methods
  # ...
end
```



Modelando um problema com classes e objetos

Classe em Ruby

bank_account.rb

```
class BankAccount

    CREDIT_LINE = 500
    @@accounts = []

    def initialize(holder)
        @holder = holder
        @balance = 0
        @@accounts << self
    end

    def deposit(amount)
        @balance += amount
    end

    def withdraw(amount)
        @balance -= amount
    end
end
```

Nome do arquivo: **snake_case**

Nome da classe: **CamelCase**

Constante: **ALL_CAPS**
Todos os objetos da classe
podem acessar, mas nenhum
pode modificar

Variável de classe: **@@var**
Qualquer instância pode
acessar e modificar, mas a
modificação afeta todas as
demais instâncias da classe

Variável de instância: **@var**
Cada instância possui o seu
valor para cada atributo

Classe em Ruby

```
class BankAccount

    CREDIT_LINE = 500
    @@accounts = []

    def initialize(holder)
        @holder = holder
        @balance = 0
        @@accounts << self
    end

    def deposit(amount)
        @balance += amount
    end

    def withdraw(amount)
        @balance -= amount
    end
end
```

initialize: método construtor

Referência para o próprio objeto
Equivalente ao **this** do Java

Características de uma classe

```
class BankAccount
  def deposit(amount)
    @balance += amount
  end
end
```

Atributo do objeto

Cada instância tem o seus valores para cada atributo da classe.
Ex: o saldo da conta corrente.

```
class BankAccount
  @@accounts = []
end
```

Atributo da classe

É uma variável compartilhada por todas os objetos de uma classe.
Ex: uma lista de contas bancárias existentes.

Características de uma classe

```
class BankAccount
  def withdraw(amount)
    #
  end
end
```

Método do objeto

É executado no escopo da instância, acessando os dados específicos dela.

Ex: muda o saldo de uma conta bancária específica.

```
class BankAccount
  def self.daily_limit
    #
  end
end
```

Método da classe

É independente de uma instância específica, executado no escopo da classe.

Ex: retornar o limite diário, que é o mesmo para todas as contas.

Características de uma classe

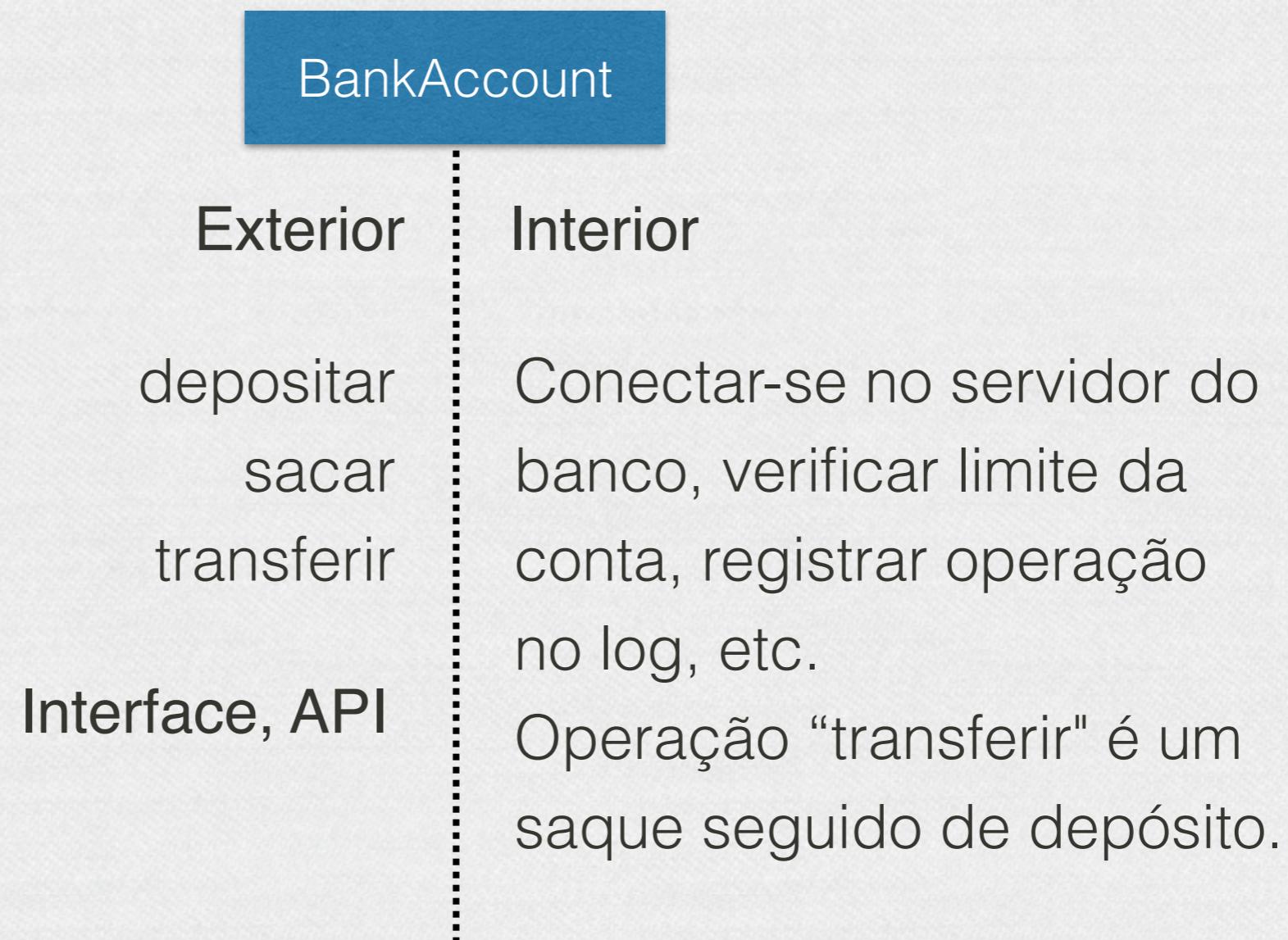
```
class BankAccount  
  CREDIT_LINE = 500  
end
```

Constante

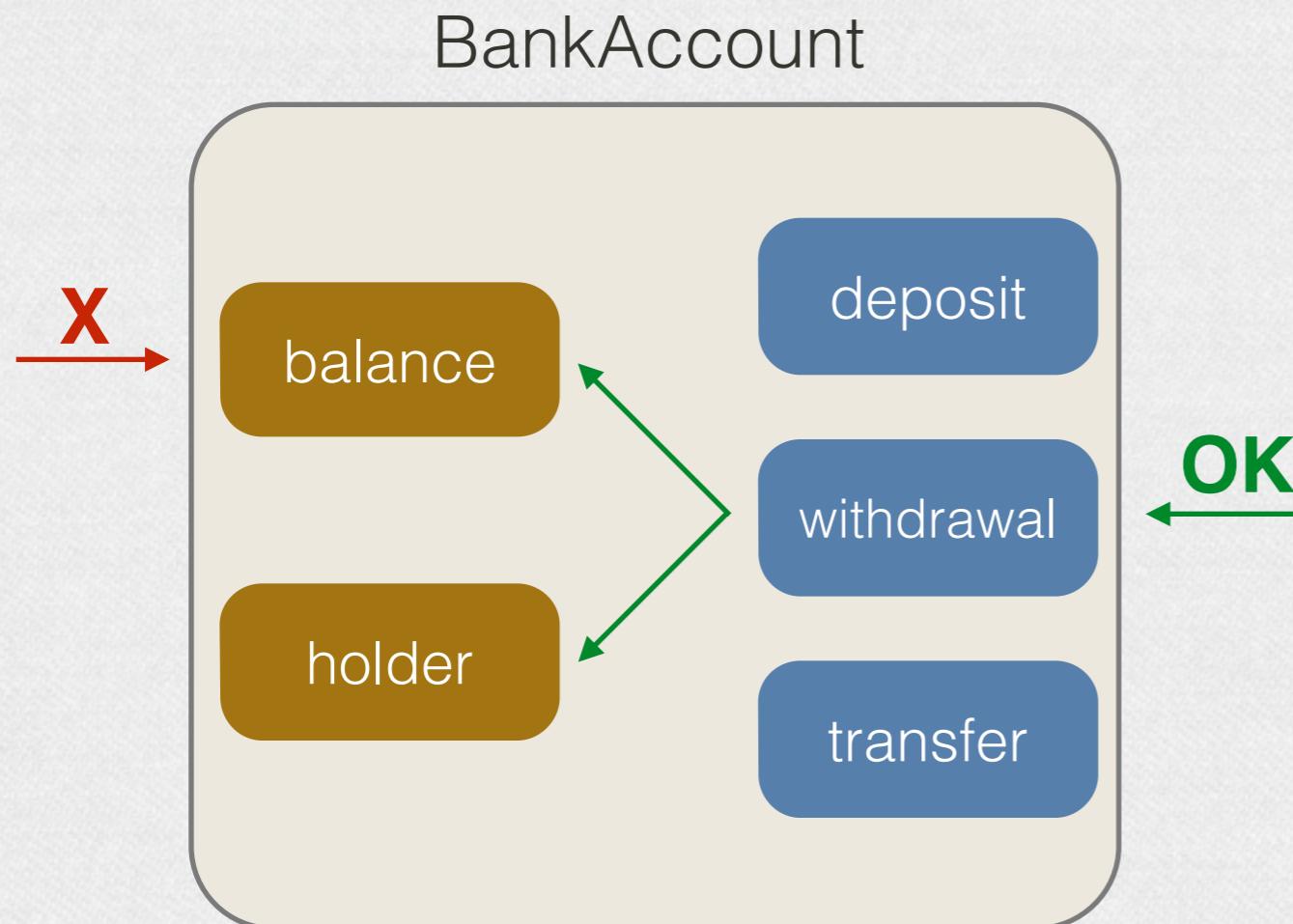
Todas as instâncias têm acesso, mas nenhuma pode modificar.
Ex: o limite de crédito é o mesmo para todas as contas bancárias.

Encapsulamento

Separação entre a visão externa e os aspectos internos da classe



Encapsulamento



```
class BankAccount  
  attr_accessor :holder  
end
```



```
class BankAccount  
  def holder=(holder)  
    @holder = holder  
  end
```

```
  def holder  
    @holder  
  end  
end
```

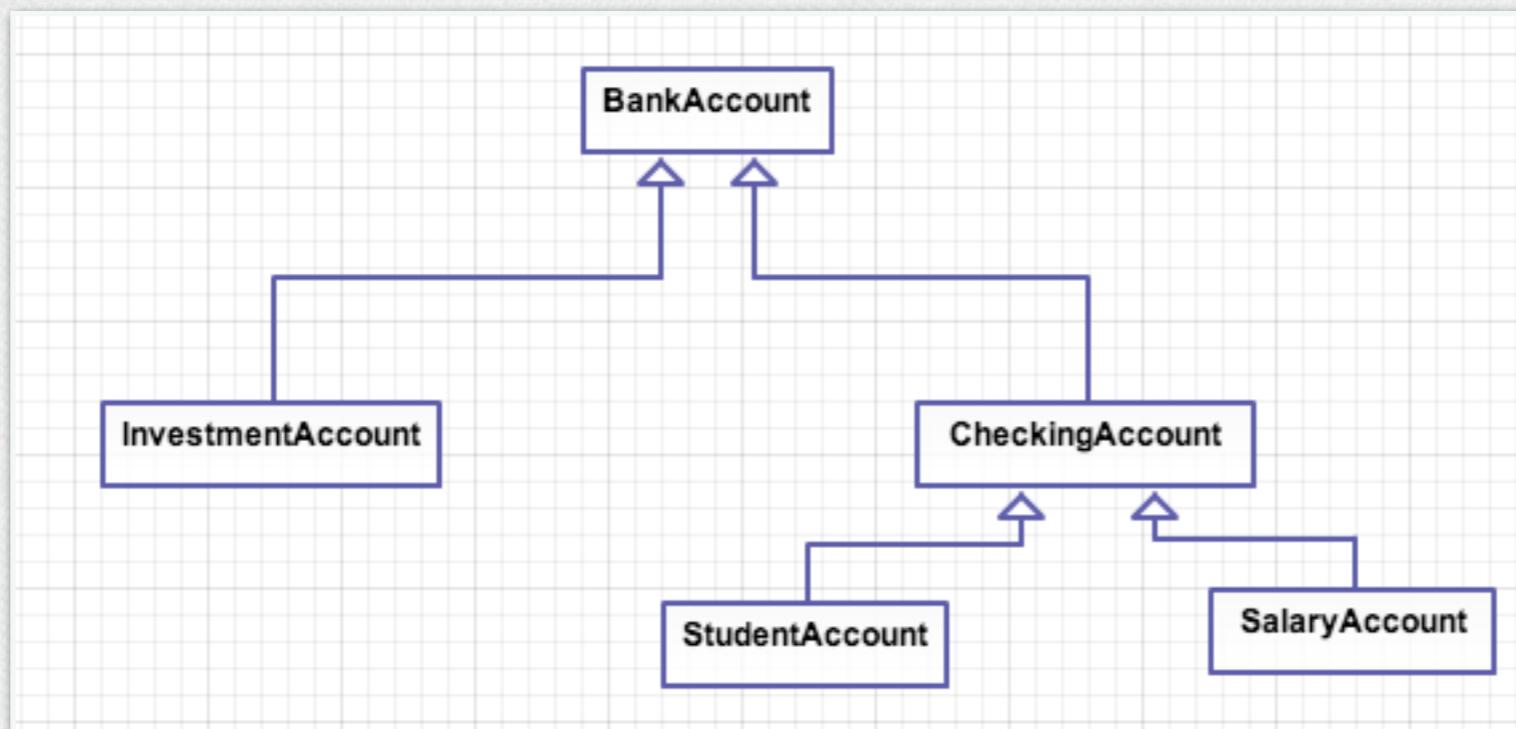
```
attr_writer :holder # (?)  
attr_reader :holder # (?)
```

Encapsulamento

- Baixo acoplamento
 - As classes devem ter o mínimo possível de dependência entre si, ou seja, ser fracamente acopladas
 - Facilita a compreensão e manutenção do código
- Alta coesão
 - Uma classe deve ter uma responsabilidade clara e limitar-se a resolver um problema específico
 - Melhora o entendimento e possibilita o reuso de código

Herança

- Representa hierarquia entre classes
- Permite que uma classe herde atributos e operações de alguma outra



SalaryAccount é uma CheckingAccount e, portanto, é também uma BankAccount

Herança

Generalização e especialização de comportamentos

Generalização

Todas as subclasses herdarão.

```
class BankAccount
  def initialize
    @balance = 0
    @monthly_fee = 10
  end
end

class CheckingAccount < BankAccount
  def deposit(amount)
    @balance += amount
  end

  def withdraw(amount)
    @balance -= amount
  end
end
```

Especialização

Adição de operação.

Herança

Especialização

`super`: invoca o método de mesmo nome na classe pai.

Especialização

Substituição de operação.
Só pode sacar se tiver saldo.

```
class CheckingAccount < BankAccount
  def deposit(amount)
    @balance += amount
  end

  def withdraw(amount)
    @balance -= amount
  end
end

class StudentAccount < CheckingAccount
  def initialize
    super
    @monthly_fee = 0
  end

  def withdraw(amount)
    if @balance >= amount
      @balance -= amount
    end
  end
end
```

Herança

Especialização

`super`: invoca o método de mesmo nome na classe pai.

Especialização

Substituição de operação.
Só pode sacar se tiver saldo.

```
class CheckingAccount < BankAccount
  def deposit(amount)
    @balance += amount
  end

  def withdraw(amount)
    @balance -= amount
  end
end

class StudentAccount < CheckingAccount
  def initialize
    super
    @monthly_fee = 0
  end

  def withdraw(amount)
    if @balance >= amount
      super(amount) # Equivalente :)
    end
  end
end
```

Visibilidade

- Forma de restrição de acesso
- Normalmente: public, protected, private

Modo	Classe	Subclasse	Mundo
public	OK	OK	OK
protected	OK	OK	X
private	OK	X	X

Modelando um problema com classes e objetos

Visibilidade

```
class MyClass
  def visibility_test
    public_method
    protected_method
    private_method
  end

  def public_method; end

  protected

  def protected_method; end

  private

  def private_method; end
end
```

```
m = MyClass.new

m.visibility_test
# => nil

m.public_method
# => nil

m.protected_method
# => NoMethodError: protected
method `protected_method' called
for #<MyClass:0x007fdac324ff18>

m.private_method
# => NoMethodError: private
method `private_method' called
for #<MyClass:0x007fdac324ff18>
```

Polimorfismo

- Enviar a mesma mensagem para objetos de diferentes classes
- Um nome pode ter diferentes significados
 - Sobrecarga de operação (*overload*)
 - Redefinição de operação (*override*)

Modelando um problema com classes e objetos

Polimorfismo

```
class Vehicle
  def start
    raise NotImplementedError
  end

  def self.park(vehicle)
    vehicle.start
    # ...
  end
end

class Car < Vehicle
  def start
    start_engine()
    puts 'The car is running!'
  end

  # ...
end
```

```
class Truck < Vehicle
  def start
    disable_alarm()
    inject_gas()
    start_engine()
    puts 'Ready to work!'
  end

  # ...
end
```

```
Vehicle.park(Car.new)
Vehicle.park(Truck.new)
```

Não é necessário saber a priori qual a classe do objeto, desde que todas as classes possíveis respeitem a mesma interface.

Polimorfismo

Sobrecarga de operação (*overload*)

ShoppingCart.java

```
public class ShoppingCart {  
  
    public void add(Product p, int quantity) {  
        // ...  
    }  
  
    public void add(Product p) {  
        self.add(p, 1);  
    }  
  
    public void add(Coupon c) {  
        // ...  
    }  
}
```

A quantidade diferente
de parâmetros define
qual é o método correto
a ser chamado.

Polimorfismo

Sobrecarga de operação (*overload*)

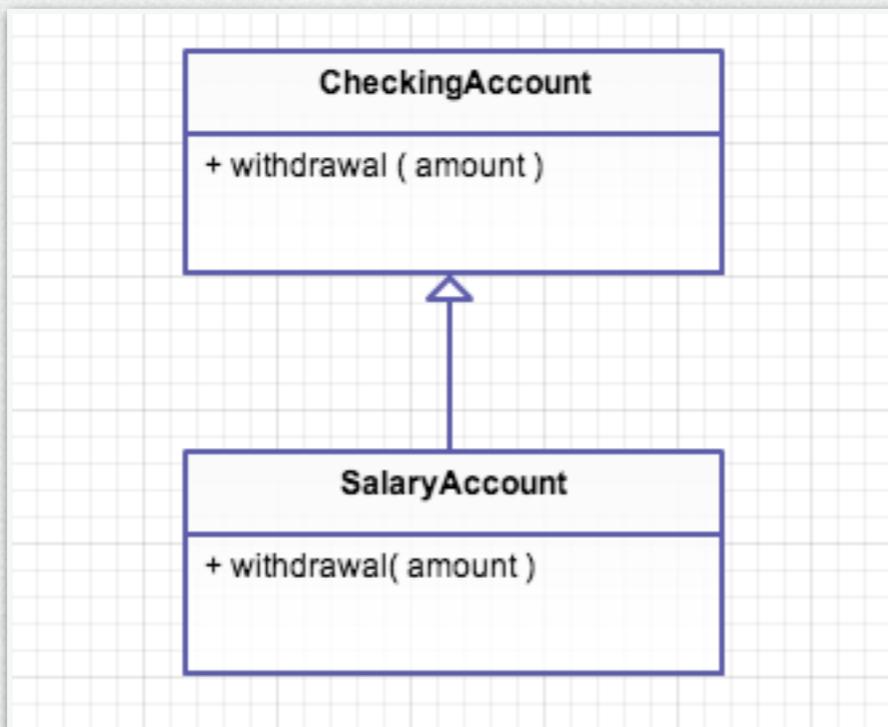
ShoppingCart.java

```
public class ShoppingCart {  
  
    public void add(Product p, int quantity) {  
        // ...  
    }  
  
    public void add(Product p) {  
        self.add(p, 1);  
    }  
  
    public void add(Coupon c) {  
        // ...  
    }  
}
```

Os tipos diferentes de parâmetros definem o método correto a ser chamado.

Polimorfismo

Redefinição de operação (*override*)



- A classe mais especializada sobrescreve o método com o mesmo nome da superclasse.
- A execução tenta sempre usar o método mais especializado.

Exercícios

Os exercícios devem ser entregues pelo GitHub seguindo o [tutorial da aula 1.](#)

Prazo: 08/04/2014 até às 18:30.

Modelando um problema com classes e objetos

Exercícios

A NeverEnding Software Inc. é uma empresa de desenvolvimento de software muito reconhecida no mercado pela fama de realizar projetos com ótima qualidade e dentro de um prazo razoável de tempo.

Você é o(a) líder de uma equipe de desenvolvimento e tem grande conhecimento em orientação a objetos.

Exercícios

A NeverEnding Software Inc. é uma empresa de desenvolvimento de software muito reconhecida no mercado pela fama de realizar projetos com ótima qualidade e dentro de um prazo razoável de tempo.

Você é o(a) líder de uma equipe de desenvolvimento e tem grande conhecimento em orientação a objetos.

4.1. A NeverEnding precisa finalizar uma proposta de projeto para uma grande transportadora. O sistema servirá para manter o registro das cargas transportadas, permitindo rastreamento dos itens, identificação do veículo responsável por um determinado frete e a qual cliente pertence cada carga. Para avaliar a dificuldade, o gerente de projeto requisitou a sua ajuda para levantar as 5 principais classes envolvidas, apontando quais seus atributos e métodos. Ele pediu para que você fosse sucinto(a) e finalizasse isso em 10 minutos, produzindo apenas um arquivo txt com os nomes das classes, os nomes dos seus atributos e dos métodos. (ver slide seguinte)

Exercícios

Formato de solução para o exercício 4.1.

ex1.txt

João da Silva Sauro

Classe: Cat

Atributos: name, age

Métodos: sleep, drink_milk, jump

Classe: Wheel

Atributos: size, material

Métodos: roll

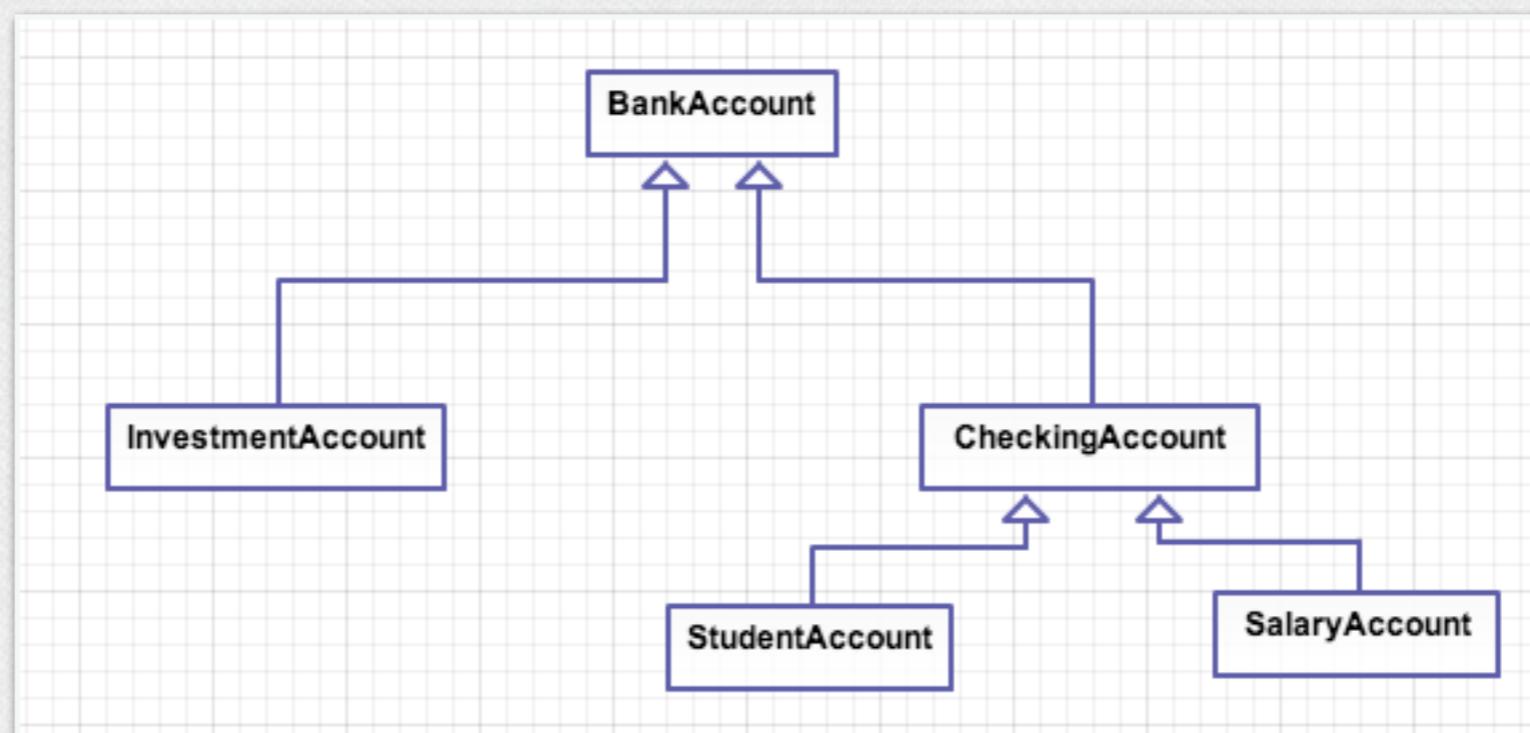
...

Modelando um problema com classes e objetos

Exercícios

O banco Coin é um cliente importante da NeverEnding. O desenvolvedor responsável pelo projeto está de férias e o banco requisitou algumas mudanças urgentes.

Pela sua experiência, você foi designado para o trabalho. O gerente de projeto pediu que você baixasse o código do repositório e começasse a trabalhar. Para ajudar, ele te passou o diagrama de classes do projeto.



Exercícios

4.2. Antes de sair de férias o desenvolvedor responsável deixou pendente a implementação da operação de transferência. O banco precisa desta funcionalidade o mais rápido possível e o gerente de projeto passou os requisitos para você:

- “*Uma transferência é um saque de uma conta seguido de um depósito em outra.*”
- “*O banco cobra o valor de R\$ 8,00 por transferência. Vamos implementar isso como um segundo saque na mesma operação de transferência. Não se esqueça de verificar se a conta tem saldo suficiente para cobrir a transferência e a tarifa. Qualquer erro será descontado diretamente na sua folha de pagamento!*”

4.3. O banco fechou uma parceria com uma grande empresa e precisa disponibilizar uma conta salário. Esta conta está prevista no diagrama de classes, mas ainda não existe no repositório. O gente de projeto passou algumas orientações:

- “*Este tipo de conta não deve possibilitar transferência.*” (*não use raise*)
- “*A tarifa mensal deve ter um desconto de 50% em relação à tarifa padrão.*”

Exercícios

4.4. O banco resolveu mudar a política do cheque especial. Até o momento, as contas bancárias podem ficar com saldo negativo de qualquer valor, mas o banco deseja que o saldo negativo possa ser no máximo igual à linha de crédito pré-aprovada. Ainda, as contas universitárias não terão cheque especial e os estudantes só poderão movimentar o dinheiro do saldo. O gente de projeto resumiu os requisitos para você:

- “*O limite de saldo negativo é a linha de crédito. O desenvolvedor deixou esse valor em uma constante da classe BankAccount.*”
- “*As contas universitárias não terão limite. Só podem usar o saldo.*”
- “*Para compensar, o banco deseja tornar as contas universitárias isentas de tarifa mensal.*”