
targettrust
treinamento e tecnologia

Riguel Figueiró

Consultor de software na ThoughtWorks
Apaixonado por metodologias ágeis, integração contínua, qualidade e disseminação e compartilhamento de conhecimento!

Como falar comigo:

- riguel@dkosoftware.com.br
- www.linkedin.com/in/riguel-figueiro/
- github.com/riguelbf

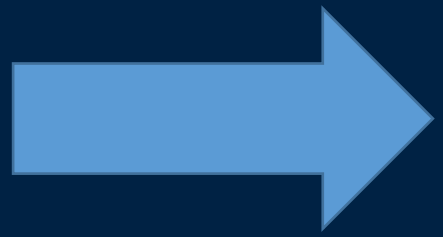
Orientação a objetos básica

O que toda conta tem e é importante para nós?

- Número da conta.
- Nome do titular da conta.
- Saldo.

O que toda conta faz e é importante para nós?

- Saca uma quantidade x.
- Deposita uma quantidade x.
- Imprime o nome do titular.
- Devolve o saldo anual.
- Transfere uma quantidade x para uma conta y.
- Devolve o tipo de conta.



A partir disso, se faz a especificação da conta

Classe = projeto de conta.

Objeto = o que se pode construir a partir do projeto.

Criando e usando um programa

- Programa.java
- Criar: *new*.
- Métodos: funções.
- Invocação de método: mandar mensagem ao objeto (ponto).

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

            minhaConta.titular = "Duke";
            minhaConta.saldo = 1000.0;

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

class Main {
    public static void main(String[] args) {
        Conta minhaConta;
        minhaConta = new Conta();

        minhaConta.titular = "Duke";
        minhaConta.saldo = 1000.0;

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

class Conta {
    int numero;
    String titular;
    double saldo;

    // ..
}
```

Métodos

- Invocação de método: mandar uma mensagem ao objeto (ponto).

```
class TestaAlgunsMetodos {
    public static void main(String[] args) {
        // criando a conta
        Conta minhaConta;
        minhaConta = new Conta();

        // alterando os valores de minhaConta
        minhaConta.titular = "Duke";
        minhaConta.saldo = 1000;

        // saca 200 reais
        minhaConta.saca(200);

        // deposita 500 reais
        minhaConta.deposita(500);
        System.out.println(minhaConta.saldo);
    }
}

class Main {
    public static void main(String[] args) {
        // criando a conta
        Conta minhaConta;
        minhaConta = new Conta();

        // alterando os valores de minhaConta
        minhaConta.titular = "Duke";
        minhaConta.saldo = 1000;

        // saca 200 reais
        minhaConta.saca(200);
```

Métodos

```
        // deposita 500 reais
        minhaConta.deposita(500);
        System.out.println(minhaConta.saldo);
    }
}

class Conta {
    int numero;
    String titular;
    double saldo;

    void saca(double quantidade) {
        double novoSaldo = this.saldo - quantidade;
        this.saldo = novoSaldo;
    }

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

Métodos com retorno

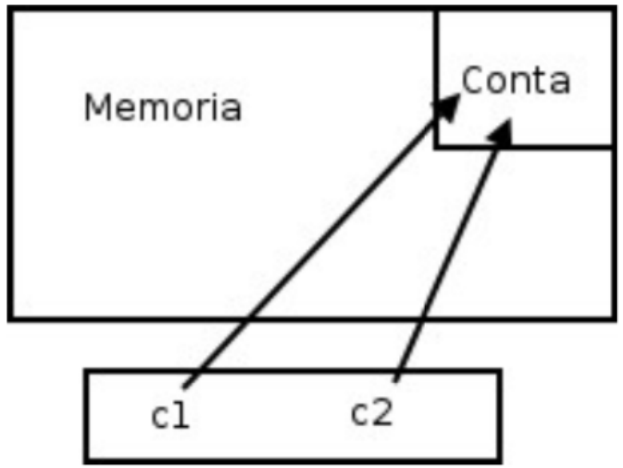
- Todo método deve definir o seu retorno.
- Sem retorno: *void*.
- É possível eliminar a variável temporária.

```
    minhaConta.saldo = 1000;
    boolean consegui = minhaConta.saca(2000);
    if (consegui) {
        System.out.println("Consegui sacar");
    } else {
        System.out.println("Não consegui sacar");
    }

class Main {
    public static void main(String[] args) {
        // criando a conta
        Conta minhaConta;
        minhaConta = new Conta();

        minhaConta.saldo = 1000;
        boolean consegui = minhaConta.saca(2000);
        if (consegui) {
            System.out.println("Consegui sacar");
        } else {
            System.out.println("Não consegui sacar");
        }
    }
}

class Conta {
    int numero;
    String titular;
    double saldo;

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

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

Objetos são acessados por referências

- A variável declarada guarda um caminho de acessar o objeto, a referência.
- *c1* = variável referência.

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

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



New: executa uma série de tarefas e devolve um valor de referência

O método transfere ()

- O método recebe um parâmetro ao invés de dois.

```
class Conta {

    // atributos e métodos...

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

Variáveis do tipo atributo

- Recebem um valor padrão.
- Exemplo:

  Classe: conta

  Atributos: nome, sobrenome, cpf.

  Os atributos são do cliente, e nào da

  classe. Logo, criamos uma nova classe

  com composição.

```
class Cliente {
    String nome;
    String sobrenome;
    String cpf;
}

class Conta {
    int numero;
    double saldo;
    Cliente titular;
    // ..
}
```

E dentro do `main` da classe de teste:

```
class Teste {
    public static void main(String[] args) {
        Conta minhaConta = new Conta();
        Cliente c = new Cliente();
        minhaConta.titular = c;
        // ...
    }
}
```

Pergunta

Façam o máximo de perguntas que desejarem!!

Exercícios

https://docs.google.com/document/d/1m1l9drAf-QgzMjNrxv9Pvt2yxt06-vFYhB65h1oSA94/edit?usp=sharing

Modificadores de acesso e atributos de classe

Controlando o acesso

- *If* = incluído dentro do método *saca* para evitar o saque com saldo insuficiente.
- Testar se o valor sacado não é maior que o saldo.

```
class TestaContaEstouro3 {

    public static void main(String[] args) {
        // a Conta
        Conta minhaConta = new Conta();
        minhaConta.saldo = 100;

        // quero mudar o saldo para -200
        double novoSaldo = -200;

        // testa se o novoSaldo é válido
        if (novoSaldo < 0) { //
            System.out.println("Não posso mudar para esse saldo");
        } else {
            minhaConta.saldo = novoSaldo;
        }
    }
}

class Main {

    public static void main(String[] args) {
        // a Conta
        Conta minhaConta = new Conta();
        minhaConta.saldo = 100;

        // quero mudar o saldo para -200
        double novoSaldo = -200;

        // testa se o novoSaldo é válido
        if (novoSaldo < 0) { //
            System.out.println("Não posso mudar para esse saldo");
        } else {
            minhaConta.saldo = novoSaldo;
        }
    }
}

class Conta {
    String titular;
    int numero;
    double saldo;


        // ..

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

Encapsulamento

- Encapsular: esconder os membros de uma classe.

Getters e setters

- Precisamos arrumar uma maneira de acessar o saldo da conta depois de usar o modificador *private*.
- *Get* ou *set*.

```
class Conta {

    private String titular;
    private double saldo;

    public double getSaldo() {
        return this.saldo;
    }

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

    public String getTitular() {
        return this.titular;
    }

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

```java
class Conta {

    private String titular;
    private double saldo;

    public double getSaldo() {
        return this.saldo;
    }

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

    public String getTitular() {
        return this.titular;
    }

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

```java
class Conta {

    private String titular;
    private double saldo;
    private double limite; // adicionando um limite a conta

    public double getSaldo() {
        return this.saldo + this.limite;
    }

    // deposita() saca() e transfere() omitidos

    public String getTitular() {
        return this.titular;
    }

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

Construtores

- Pode inicializar algum tipo de informação

  (exemplo: titular da conta).

```
String carlos = "Carlos";
Conta c = new Conta(carlos);
System.out.println(c.titular);

class Main {
    public static void main(String[] args) {
        String carlos = "Carlos";
        Conta c = new Conta(carlos);
        System.out.println(c.titular);
    }
}

class Conta {
    String titular;
    int numero;
    double saldo;

    // construtor
    Conta(String titular) {
        this.titular = titular;
    }

    // ..
}
```

Necessidade de um construtor

- Possibilitar o usuário de uma classe a passar argumentos (agregar informações) para o objeto durante o processo de criação.

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 {
    String titular;
    int numero;
    double saldo;

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

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

    //..
}
```

Atributos de classe

```
class Conta {
    private static int totalDeContas;
    //...

    Conta() {
        Conta.totalDeContas = Conta.totalDeContas + 1;
    }
}
```

```
class Conta {
    private static int totalDeContas;
    //...

    Conta() {
        Conta.totalDeContas = Conta.totalDeContas + 1;
    }

    public int getTotalDeContas() {
        return Conta.totalDeContas;
    }
}
```

Atributos de classe

```
Conta c = new Conta();
int total = c.getTotalDeContas();

public static int getTotalDeContas() {
    return Conta.totalDeContas;
}
int total = Conta.getTotalDeContas();
```

Pergunta

Façam o máximo de perguntas que desejarem!!

Exercícios

https://docs.google.com/document/d/1IS34Rjo2fHxEULlkotShjaVj9blij5A29Dw8vYz5Sss/edit?usp=sharing

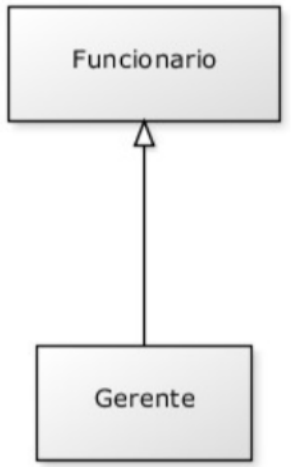
Herança, reescrita e polimorfismo

Herança

- Relacionar uma classe para que herde tudo que a outra tem.

  Exemplo: *gerente* deve ter tudo que o *funcionário* possui, sendo uma extensão (*extends*).

```
public class Gerente extends Funcionario {
    private int senha;
    private int numeroDeFuncionariosGerenciados;

    public boolean autentica(int senha) {
        if (this.senha == senha) {
            System.out.println("Acesso Permitido!");
            return true;
        } else {
            System.out.println("Acesso Negado!");
            return false;
        }
    }

    // setter da senha omitido
}
```



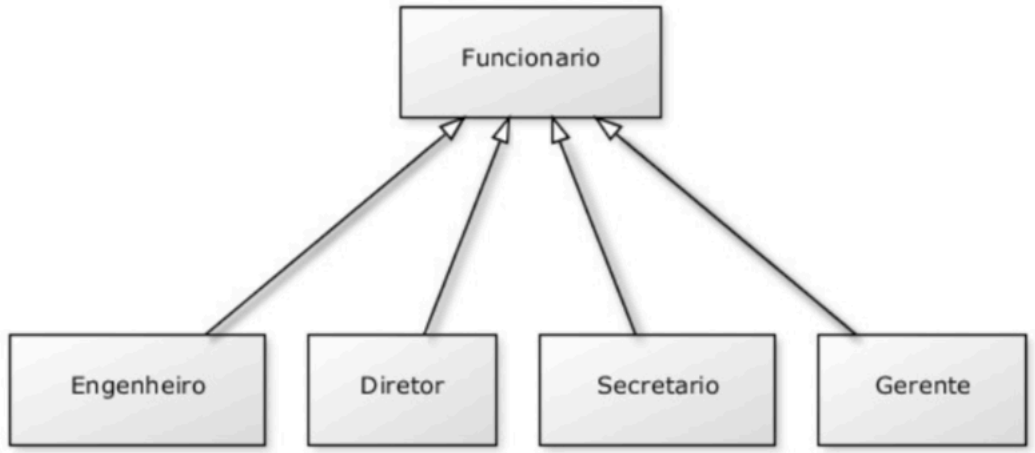
Herança

```
    public class TestaGerente {
        public static void main(String[] args) {
            Gerente gerente = new Gerente();

            // podemos chamar métodos do Funcionario:
            gerente.setNome("João da Silva");

            // e também métodos do Gerente!
            gerente.setSenha(4231);
        }
    }

public class Main {
    public static void main(String[] args) {
        Gerente gerente = new Gerente();

        // podemos chamar métodos do Funcionario:
        gerente.setNome("João da Silva");

        // e também métodos do Gerente!
        gerente.setSenha(4231);

        gerente.autentica(4231);
    }
}

class Gerente extends Funcionario {
    private int senha;
    private int numeroDeFuncionariosGerenciados;

    public boolean autentica(int senha) {
        if (this.senha == senha) {
            System.out.println("Acesso Permitido!");
            return true;
        } else {
            System.out.println("Acesso Negado!");
            return false;
        }
    }
```

```
    public void setSenha(int senha) {
        this.senha = senha;
    }
}

class Funcionario {
    private String nome;
    private String cpf;
    private double salario;


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

Herança

- Uma classe pode ter várias filhas mas apenas uma mãe.



Reescrita de método

- É possível alterar o comportamento do método herdado o reescrevendo:

```
public class Gerente extends Funcionario {
    int senha;
    int numeroDeFuncionariosGerenciados;

    public double getBonificacao() {
        return this.salario * 0.15;
    }
    // ...
}
```

```
    Gerente gerente = new Gerente();
    gerente.setSalario(5000.0);
    System.out.println(gerente.getBonificacao());

public class Main {
    public static void main(String[] args) {
        Gerente gerente = new Gerente();
        gerente.setSalario(5000.0);
        System.out.println(gerente.getBonificacao());
    }
}

class Gerente extends Funcionario {
    int senha;
    int numeroDeFuncionariosGerenciados;

    public double getBonificacao() {
        return this.salario * 0.15;
    }
}

class Funcionario {
    protected String nome;
    protected String cpf;
    protected double salario;




    public double getBonificacao() {
        return this.salario * 0.10;
    }

    public void setSalario(double salario) {
        this.salario = salario;
    }
}
```

Invocação do método reescrito

- Exemplo: bonificação de um gerente.

```
public class Gerente extends Funcionario {
    int senha;
    int numeroDeFuncionariosGerenciados;

    public double getBonificacao() {
        return this.salario * 0.10 + 1000;
    }
     // ...
}
```

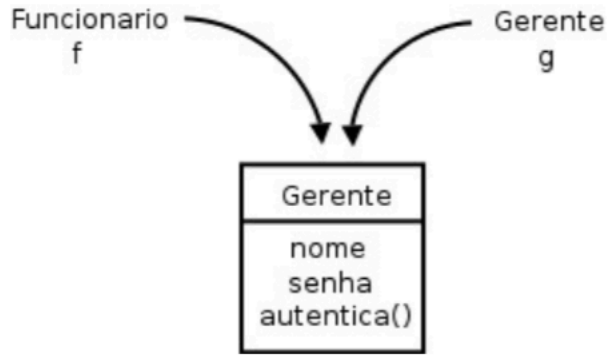
- Problema: o dia que o *getBonificacao do Funcionario* mudar, precisaríamos mudar o método. Resolução:

```
public class Gerente extends Funcionario {
    int senha;
    int numeroDeFuncionariosGerenciados;

    public double getBonificacao() {
        return super.getBonificacao() + 1000;
    }
     // ...
}
```

Polimorfismo

- Capacidade de um objeto de ser referenciado de varias formas.

```
Gerente gerente = new Gerente();
Funcionario funcionario = gerente;
funcionario.setSalario(5000.0);
```



Polimorfismo

```
        ControleDeBonificacoes controle = new ControleDeBonificacoes();

        Gerente funcionario1 = new Gerente();
        funcionario1.setSalario(5000.0);
        controle.registra(funcionario1);

        Funcionario funcionario2 = new Funcionario();
        funcionario2.setSalario(1000.0);
        controle.registra(funcionario2);

        System.out.println(controle.getTotalDeBonificacoes());

    public class Main {
        public static void main(String[] args) {
            ControleDeBonificacoes controle = new ControleDeBonificacoes()

            Gerente funcionario1 = new Gerente();
            funcionario1.setSalario(5000.0);
            controle.registra(funcionario1);

            Funcionario funcionario2 = new Funcionario();
            funcionario2.setSalario(1000.0);
            controle.registra(funcionario2);

            System.out.println(controle.getTotalDeBonificacoes());
        }
    }

    class ControleDeBonificacoes {
        private double totalDeBonificacoes = 0;

        public void registra(Funcionario funcionario) {
            this.totalDeBonificacoes += funcionario.getBonificacao();
        }

        public double getTotalDeBonificacoes() {
            return this.totalDeBonificacoes;
        }
    }
```

```
class Gerente extends Funcionario {
    int senha;
    int numeroDeFuncionariosGerenciados;

    public double getBonificacao() {
        return this.salario * 0.15;
    }
}

class Funcionario {
    protected String nome;
    protected String cpf;
    protected double salario;

    public double getBonificacao() {
        return this.salario * 0.10;
    }

    public void setSalario(double salario) {
        this.salario = salario;
    }
}
```

Pergunta

Façam o máximo de perguntas que desejarem!!

Exercícios

https://docs.google.com/document/d/1vq7zjllHhqFW371BXbgPbRy__-JuvK2ZtFal1qH8ojo/edit?usp=sharing