Construtores

- No domínio de um banco, todo cartão de crédito deve possuir um número. Toda agência deve possuir um número. Toda conta deve estar associada a uma agência. Após criar um objeto para representar um cartão de crédito, poderíamos definir um valor para o atributo numero. De maneira semelhante, podemos definir um número para um objeto da classe Agencia e uma agência para um objeto da classe Conta.

```
CartaoDeCredito cdc = new CartaoDeCredito ();
cdc . numero = 12345;

Agencia a = new Agencia ();
a. numero = 11111;

Conta c = new Conta ();
c. agencia = a;
```

Construtores

- Definir os valores dos atributos obrigatórios de um objeto logo após a criação dele resolveria as restrições do sistema do banco. Porém, nada garante que todos os desenvolvedores sempre lembrem de inicializar esses valores.
- Para não correr esse risco, podemos utilizar construtores. Um construtor permite que um determinado trecho de código seja executado toda vez que um objeto é criado, ou seja, toda vez que o operador new é chamado. Assim como os métodos, os construtores podem receber parâmetros.
- Contudo, diferentemente dos métodos, os construtores não devolvem resposta. EmJava, um construtor deve ter o mesmo nome da classe na qual ele foi definido.

Construtores

```
class CartaoDeCredito {
        int numero ;
        CartaoDeCredito ( int numero ) {
                this.numero = numero ;
        }
}

class Agencia {
        int numero ;
        Agencia ( int numero ) {
                this.numero = numero ;
        }
}

class Conta {
        Agencia agencia ;
        Conta ( Agencia agencia ) {
                this.agencia = agencia ;
        }
}
```

Construtores

- Na criação de um objeto com o comando new, os argumentos passados devem ser compatíveis com a lista de parâmetros de algum construtor definido na classe que está sendo instanciada. Caso contrário, um erro de compilação ocorrerá para avisar o desenvolvedor dos valores obrigatórios que devem ser passados para criar um objeto.

```
// Passando corretamente os parâmetros para os construtores

CartaoDeCredito cdc = new CartaoDeCredito (1111) ;

Agencia a = new Agencia (1234) ;

Conta c = new Conta (a);
```

Construtores

```
// ERRO DE COMPILAÇÃO
CartaoDeCredito cdc = new CartaoDeCredito ();

// ERRO DE COMPILAÇÃO
Agencia a = new Agencia ();

// ERRO DE COMPILAÇÃO
Conta c = new Conta ();
```

Construtor Padrão

- Toda vez que um objeto é criado, um construtor da classe correspondente deve ser chamado. Mesmo quando nenhum construtor for definido explicitamente, há um construtor padrão que será inserido pelo próprio compilador.
- O construtor padrão não recebe parâmetros e será inserido sempre que o desenvolvedor não definir pelo menos um construtor explicitamente.
- Portanto, para instanciar uma classe que não possui construtores definidos no código fonte, devemos utilizar o construtor padrão, já que este é inserido automaticamente pelo compilador.

```
class Conta {

}
```

Sobrecarga de Construtores

- O conceito de sobrecarga de métodos pode ser aplicado para construtores. Dessa forma, podemos definir diversos construtores na mesma classe.

```
class Pessoa {
        String rg;
        int cpf ;
        Pessoa ( String rg){
                this.rg = rg;
        }
        Pessoa ( int cpf ){
                this.cpf = cpf ;
        }
}
```

Sobrecarga de Construtores

- Quando dois construtores são definidos, há duas opções no momento de utilizar o comando new.

```
// Chamando o primeiro construtor
Pessoa p1 = new Pessoa (" 123456 X ");

// Chamando o segundo construtor
Pessoa p2 = new Pessoa (123456789) ;
```

Construtores chamando construtores

- Assim como podemos encadear métodos, também podemos encadear construtores.

```
class Conta {
        int numero ;
        double limite ;
        Conta ( int numero ) {
                this.numero = numero ;
        }
        Conta ( int numero , double limite ) {
        this ( numero );
                this.limite = limite ;
        }
}
```

Referências como Parâmetro

- Da mesma forma que podemos passar valores primitivos como parâmetro para um método ou construtor, também podemos passar valores não primitivos (referências).
- Considere um método na classe Conta que implemente a lógica de transferência de valores entre contas. Esse método deve receber como argumento, além do valor a ser transferido, a referência da conta que receberá o dinheiro.

```
public void transfere ( Conta destino , double valor ) {
        this.saldo -= valor ;
        destino.saldo += valor ;
}
```

Referências como Parâmetro

- Na chamada do método transfere(), devemos ter duas referências de contas: uma para chamar o método e outra para passar como parâmetro.

```
Conta origem = new Conta ();
origem.saldo = 1000;
Conta destino = new Conta ();
origem.transfere ( destino , 500) ;
```

* Quando a variável destino é passada como parâmetro, somente a referência armazenada nessa variável é enviada para o método transfere() e não o objeto em si. Em outras palavras, somente o “endereço” para a conta que receberá o valor da transferência é enviado para o método transfere().

Atributos Estáticos

- Num sistema bancário, provavelmente, criaríamos uma classe para especificar os objetos que representariam os funcionários do banco.
- Por exemplo:

```
class Funcionario {
        String nome ;
        double salario ;
        public void aumentaSalario ( double aumento ) {
                salario += aumento ;
        }
}
```

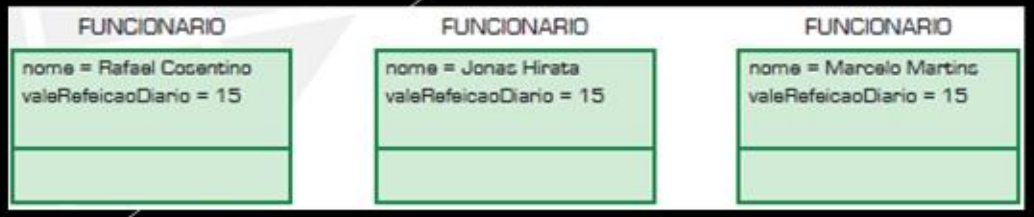
Atributos Estáticos

- Suponha que o banco paga aos seus funcionários um valor padrão de vale refeição por dia trabalhado. O sistema do banco precisa guardar esse valor. Poderíamos definir um atributo na classe Funcionario para tal propósito.
- Por exemplo:

```
class Funcionario {
        String nome ;
        double salario ;
        double valeRefeicaoDiario ;
        public void aumentaSalario ( double aumento ) {
                salario += aumento ;
        }
}
```

Atributos Estáticos

- O atributo valeRefeicaoDiario é de instância, ou seja, cada objeto criado a partir da classe Funcionario teria o seu próprio atributo valeRefeicaoDiario. Porém, não faz sentido ter esse valor repetido em todos os objetos, já que ele é único para todos os funcionários.



Atributos Estáticos

- Para que o atributo valeRefeicaoDiario não se repita em cada objeto da classe Funcionario, devemos torná-lo um atributo de classe ao invés de um atributo de instância. Para isso, devemos aplicar o modificador **static na declaração do atributo.**

```
class Funcionario {
        String nome ;
        double salario ;
        static double valeRefeicaoDiario ;
        public void aumentaSalario ( double aumento ) {
                salario += aumento ;
        }
}
```

Atributos Estáticos

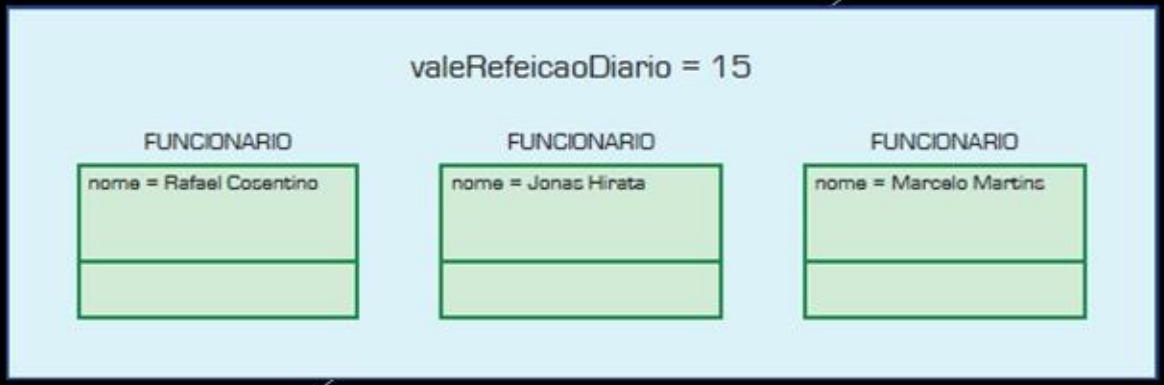
- Um atributo de classe deve ser acessado através do nome da classe na qual ele foi definido.

```
Funcionario.valeRefeicaoDiario = 15;
```

- Podemos acessar um atributo de classe através de uma referência de um objeto da classe na qual o atributo foi definido. Contudo, não seria a maneira conceitualmente correta já que o atributo pertence a classe e não ao objeto.

```
Funcionario f = new Funcionario ();
// Válido , mas conceitualmente incorreto
f. valeRefeicaoDiario = 15;
```

Atributos Estáticos



Métodos Estáticos

- Definimos métodos para implementar as lógicas que manipulam os valores dos atributos de instância. Podemos fazer o mesmo para os atributos de classe.
- Suponha que o banco tenha um procedimento para reajustar o valor do vale refeição baseado em uma taxa. Poderíamos definir um método na classe Funcionario para implementar esse reajuste.

```
public void reajustaValeRefeicaoDiario ( double taxa ) {
        Funcionario.valeRefeicaoDiario += Funcionario.valeRefeicaoDiario * taxa
        ;
}
```

- O método reajustaValeRefeicaoDiario() é de instância. Consequentemente, ele deve ser chamado a partir da referência de um objeto da classe Funcionario.

Métodos Estáticos

- Contudo, como o reajuste do valor do vale refeição não depende dos dados de um funcionário em particular, não faz sentido precisar de uma referência de um objeto da classe Funcionario para poder fazer esse reajuste.
- Neste caso, poderíamos definir o reajustaValeRefeicaoDiario() como método de classe ao invés de método de instância. Aplicando o modificador static nesse método, ele se tornará um método de classe. Dessa forma, o reajuste poderia ser executado independentemente da existência de objetos da classe Funcionario.

```
public static void reajustaValeRefeicaoDiario ( double taxa ) {
        Funcionario.valeRefeicaoDiario += Funcionario.valeRefeicaoDiario * taxa
        ;
}
```

Métodos Estáticos

- Um método de classe deve ser chamado através do nome da classe na qual ele foi definido.

```
Funcionario.reajustaValeRefeicaoDiario (0.1);
```

- Podemos chamar um método de classe através de uma referência de um objeto da classe na qual o método foi definido. Contudo, não seria a maneira conceitualmente correta já que o método pertence a classe e não ao objeto.

```
Funcionario f = new Funcionario ();
// Válido , mas conceitualmente incorreto
f. reajustaValeRefeicaoDiario (0.1) ;
```

Exemplo

1. Crie uma classe chamada **Conta. Defina um atributo de classe para contabilizar** o número de objetos instanciados a partir da classe Conta. Esse atributo deve ser incrementado toda vez que um objeto é criado. Você pode utilizar construtores para fazer o incremento.

```
public class Conta {
        // ATRIBUTO DE CLASSE
        static int contador ;
        // CONSTRUTOR
        Conta () {
                Conta.contador ++;
        }
}
```

Exemplo

2. Faça um teste criando dois objetos da classe Conta. Imprima o valor do contador de contas antes e depois da criação de cada objeto.

```
public class Testa {
        public static void main ( String [] args ) {
                System.out.println (" Contador : " + Conta.contador );
                Conta cliente1 = new Conta ();
                System.out.println (" Contador : " + Conta.contador );
                Conta cliente2 = new Conta ();
                System.out.println (" Contador : " + Conta.contador );
        }
}
```

Exemplo

3. O contador de contas pode ser utilizado para gerar um número único para cada conta. Acrescente na classe Conta um atributo de instância para guardar o número das contas. Implemente no construtor a lógica para gerar esses números de forma única através do contador de contas.

```
public class Conta {
        // ATRIBUTO DE CLASSE
        static int contador ;
        // ATRIBUTO DE INSTANCIA
        int numero ;
        // CONSTRUTOR
        Conta () {
                Conta.contador ++;
                numero = Conta.contador ;
        }
}
```

Exemplo

4. Altere o teste para imprimir o número de cada conta criada.

```
public class Testa {
        public static void main ( String [] args ) {
                System.out.println (" Contador : " + Conta.contador );
                Conta cliente1 = new Conta ();
                 System.out.println (" Contador : " + cliente1.numero );
                System.out.println (" Contador : " + Conta.contador );
                Conta cliente2 = new Conta ();
                 System.out.println (" Contador : " + cliente2.numero );
                System.out.println (" Contador : " + Conta.contador );
        }
}
```

Exemplo

5. Adicione um método de classe na classe Conta para zerar o contador e imprimir o total de contas anterior.

```
static void zeraContador () {
        System.out.println (" Contador : " + Conta.contador );
        System.out.println (" Zerando o contador de contas ... ");
        Conta.contador = 0;
}
```

Exemplo

6. Altere o teste para utilizar o método zeraContador() de contas

```
public class Testa {
        public static void main ( String [] args ) {
                System.out.println (" Contador : " + Conta.contador );
                Conta cliente1 = new Conta ();
                 System.out.println (" Contador : " + cliente1.numero );
                System.out.println (" Contador : " + Conta.contador );
                Conta cliente2 = new Conta ();
                 System.out.println (" Contador : " + cliente2.numero );
                System.out.println (" Contador : " + Conta.contador );
                Conta.zeraContador ();
        }
}
```