Dia 10 - Construtores

Definição de Construtor

Um construtor é um método de grande importância dentro de uma classe Java. Ele possui algumas características proprias:

- É um método que o mesmo nome que a classe onde se encontra
- Não possui um valor de retorno, nem mesmo void.
- É passível de possuir parâmetros (argumentos)
- Toda classe deve possuir ao menos 1 construtor
- Se o desenvolvedor não escrever um construtor em uma classe, a JVM irá prover um construtor sem argumentos, também chamado de default.

Ao ser criada, uma classe inicializa seus atributos da seguinte maneira:

- Tipos primitivos númericos (int, short, byte, long, float, double) são inicializados com valor 0 (Zero)
- Tipos primitivos booleanos (boolean) são inicializados com valor false
- Tipos primitivos characters (char) são inicializar com valor” (vazio)
- Objetos são inicializados como null

Execução de um Construtor

Vamos supor a seguinte hierarquia de classes Java, conforme Figura 01:

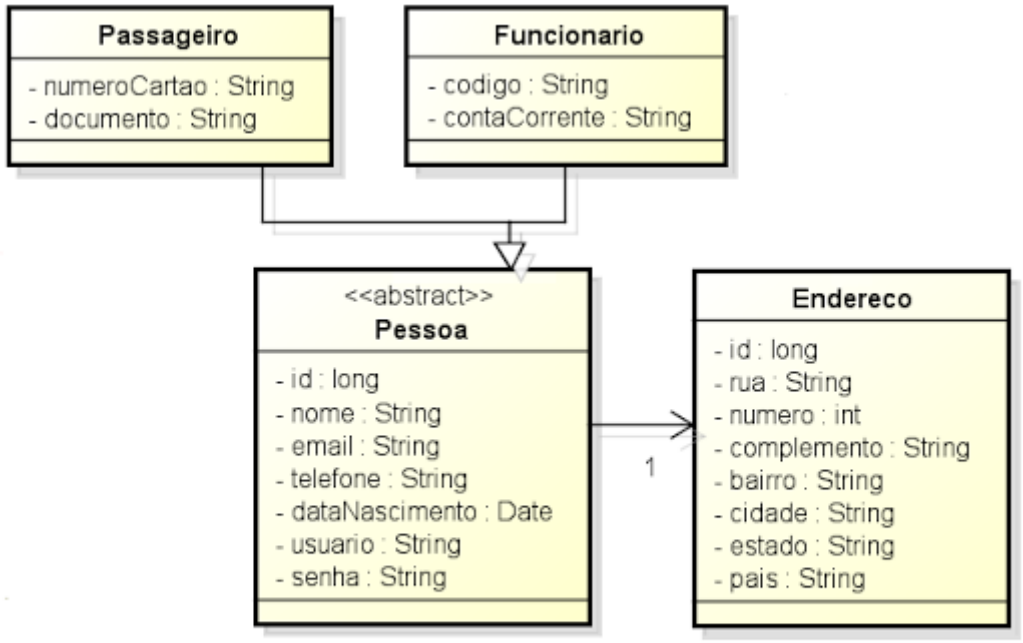

Figura 01 – Diagrama de *classes* de exemplo com a Hierarquia da *classe* Pessoa

Dia 10 - Construtores

Vamos supor o seguinte código para as *classes* Passageiro e Pessoa:

```java
public abstract class Pessoa {

    private long id;
    private String nome;
    private String email;
    private String telefone;
    private Date dataNascimento;
    private String usuario;
    private String senha;

    //Devido a associacao entre Pessoa e Endereco
    private Endereco endereco;

    //Metodos getters e setters
}

public class Passageiro extends Pessoa {

    private String documento;
    private String numeroCartao;

    //Metodos getters e setters
}
```

Como pode ser observado as classes não possuem um construtor explicito, logo a JVM cria um construtor default, sem argumentos. Como ocorre a execução da chamada ao construtor da classe Passageiro? Qual das opções abaixo:

- Apenas o construtor de Passageiro será chamado
- O construtor de Passageiro e o construtor de Pessoa será chamado
- O construtor de Passageiro, o construtor de Pessoa e o construtor de Object será chamado

Se você acha que é a opção 3 então você está certo. Lembrando que a classe Object é a Mãe de todas as classes em Java, logo ocorreria justamente o seguinte diagrama de sequência para a chamada ao construtor da classe Passageiro, como mostra a Figura 02:

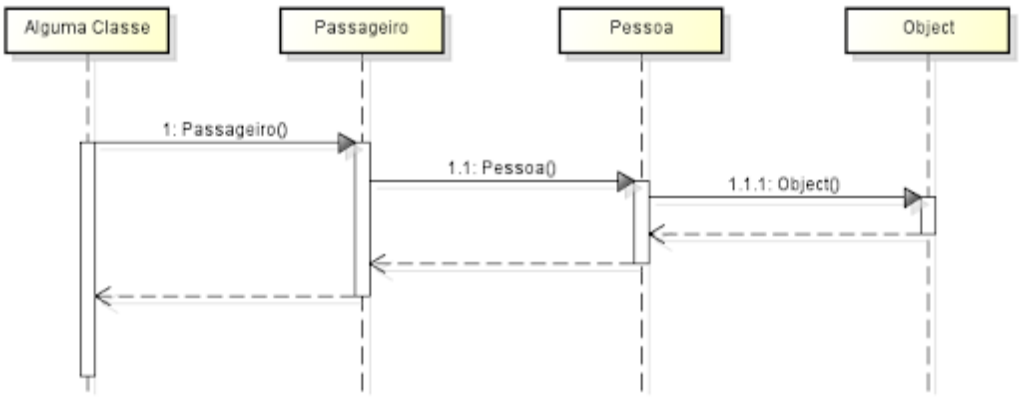

Figura 02 – Diagrama de Sequencia representando a chamada aos construtores da hierarquia de Pessoa-Passageiro

Mas como ocorre essa correlação entre a chamada de um construtor e a chamada do construtor da *classe* pai?

Essa chamada a um construtor pai se dá através do método super(). Assim a primeira linha de execução de um método construtor é uma chamada para um método super() da *classe* pai. Essa linha pode ser inserida automaticamente pela JVM ou pelo desenvolvedor. Um detalhe importante é que tem q ser a primeira linha do método do construtor. Caso seja após a primeira linha ocorrerá um erro de compilação.

```
1  public class Passageiro extends Pessoa {
2
3      private String documento;
4      private String numeroCartao;
5
6      public Passageiro(){
7          documento = "abc";
8          super(); //ERRO DE COMPILAÇÃO
9      }
10
11     //Metodos getters e setters
12 }
```

Um ponto importante a ser mencionado é que construtores da *classe* pai não são herdados por seus filhos.

Associações Obrigatórias

Antes de continuarmos é interessante explicar um pouco sobre associações obrigatórias. Quando vamos construir uma casa na realidade, quais são os “objetos” extremamente necessários para realizar essa construção?

- Água?
- Luz?
- Material de Construção?
- Terreno?

O primeiro objeto básico, realmente básico, é o terreno. Senão tivermos um terreno, como iremos fazer alguma construção? Assim antes de iniciar a construção de uma casa, antes de pensarmos no projeto, é necessário existir um “objeto” terreno. Bom partindo desse princípio, mas voltando para a realidade de software, como podemos obrigar a um desenvolvedor a enviar uma informação? A passar um objeto para uma instância?

Dia 10 - Construtores

Uma forma que poderia ser feita é no momento de gerar uma nova instância. Dessa forma ao utilizarmos o construtor da *classe*, podemos obrigar que o desenvolvedor envie objetos necessários para que aquela nova instância funcione corretamente.

Pegando o exemplo da casa, o seguinte código do construtor da *classe* Casa está correto:

```
1  public class Casa {
2      private Terreno terreno;
3
4      public Casa(Terreno terreno){
5          this.terreno = terreno;
6      }
7
8      public Terreno getTerreno(){
9          return terreno;
10     }
11
12     public void setTerreno(Terreno terreno){
13         this.terreno = terreno;
14     }
15 }
```

Mas pensando no contrário. É interessante que o objeto terreno conheça a casa? Como o terreno pode ser utilizado para vários tipos de objetos, como casa, aeroportos, prédios, talvez não seja interessante que o terreno possua o conhecimento da *classe* casa. Dessa forma, a Figura 03 mostra a representação UML dessa associação entre Casa e Terreno:

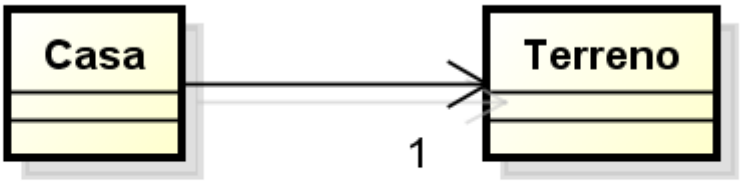

Figura 03 – Representação UML da associação entre Casa e Terreno

Perceba no código Java que foi criada um atributo do tipo Terreno dentro da *classe* Casa, representando a associação entre as *classes*. Caso a associação fosse bidirecional, cada uma das *classes* deveria ter a instância da outra.

Dia 10 - Construtores

Vamos observar novamente o diagrama de *classes* da hierarquia de Passageiro, Figura 04. Repare que entre Pessoa e Endereço existe uma associação obrigatória de 1 objeto.

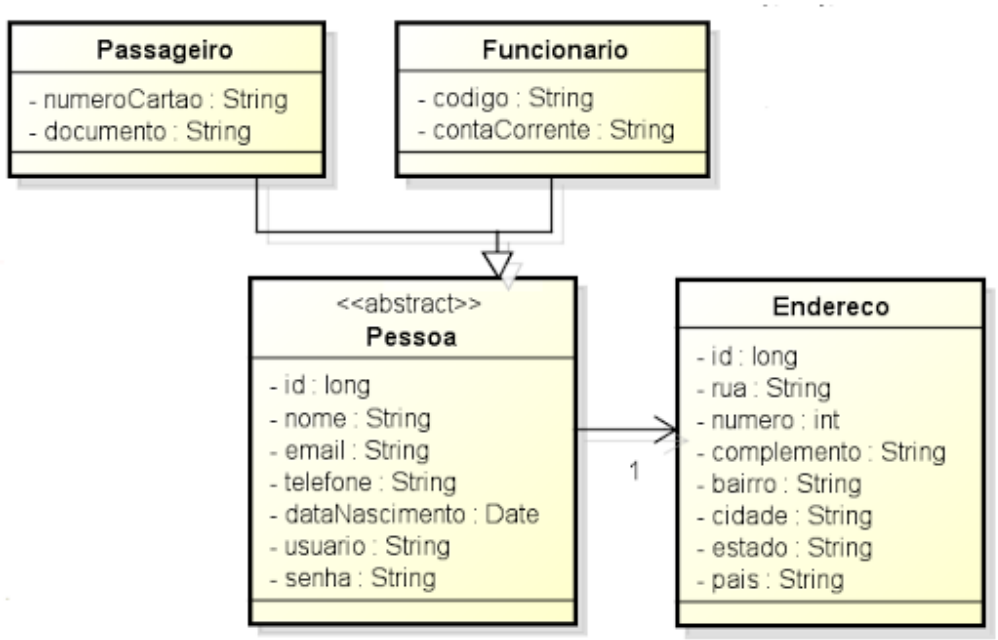

Figura 04 – Diagrama de *classes* de exemplo com a Hierarquia da *classe* Pessoa

Seguindo o que foi discutido na associação obrigatória, essa deve ser representada no construtor da *classe* Pessoa, assim será adicionado um parâmetro Endereço ao construtor da *classe* Pessoa:

```
1  public abstract class Pessoa {
2
3      private long id;
4      private String nome;
5      private String email;
6      private String telefone;
7      private Date dataNascimento;
8      private String usuario;
9      private String senha;
10
11     //Devido a associacao entre Pessoa e Endereco
12     private Endereco endereco;
13
14     public Pessoa(Endereco endereco){
15         this.endereco = endereco;
16     }
17
18     //Metodos getters e setters
19 }
```

Dia 10 - Construtores

Ao adicionar esse parâmetro ao construtor de pessoa, ocorrerá um erro de compilação na classe Passageiro, dizendo que não existe um construtor correspondente na classe Pessoa, sem argumentos. Por que isso ocorreu?

No primeiro momento desta aula, as classes da hierarquia Pessoa não possuíam construtores, dessa forma a JVM criou automaticamente os construtores sem argumentos, default, para todas as classes. Como percebido pelo diagrama de sequência, os construtores eram chamados entre si através do método super (), também inserido automaticamente pela JVM. Quando foi modificado o construtor de Pessoa para receber uma instância de Endereço, a chamada automática de super () da classe Passageiro torna-se errada, pois não havia mais um construtor default e sim, um construtor com argumentos.