Inatel

C206/C06 – Programação Orientada a Objetos com Java

Construtores e o Modificador Static

Prof. Christopher Lima

christopher@inatel.br

Objetivos

- Entender os Construtores
- Utilizar membros e métodos estáticos (*static*)

Construtores

- Já sabemos que a palavra chave "new" invoca o construtor de uma classe e cria a sua instância!

```
public class Pessoa {

    private String nome;
    private int idade;

    //Construtor
    public Pessoa() {
        System.out.println("Criando Instância de Pessoa");
    }
```

Construtores

- Quando não declaramos o construtor explicitamente, o Java invoca o construtor implícito (Padrão). Porém, se passamos a deixar um construtor visível, o padrão não é mais fornecido.

```
public class Pessoa {

    private String nome;
    private int idade;

    //Construtor
    //Não existe mais o construtor implícito agora
    public Pessoa() {
        System.out.println("Criando Instância de Pessoa");
    }
```

Construtores

- Os construtores podem receber parâmetros, facilitando a inicialização de instâncias.

```
public class Pessoa {

    private String nome;
    private int idade;

    //Construtor recebendo parametros
    //E inicializando os membros da classe
    public Pessoa(String nome, int idade) {
        this.nome = nome;
        this.idade = idade;
    }
```

```
public class Main {

    public static void main(String args[]) {
        Pessoa p = new Pessoa("Capiroto", 54);
    }

}
```

Construtores

- Perceba que no exemplo anterior, o construtor força o cliente a passar dois parâmetros durante a instanciação. Não é mais possível invocar o construtor vazio!

```
public class Main {

    public static void main(String args[]) {
        //Não compila
        Pessoa p = new Pessoa();
    }
```

Construtores

- Mas e se ainda desejamos oferecer uma opção de construtor vazio?
- Podemos fazer a sobrecarga de construtores. Basta escrever parâmetros diferentes!
- O Compilador sabe qual construtor invocar, desde que os parâmetros sejam diferentes

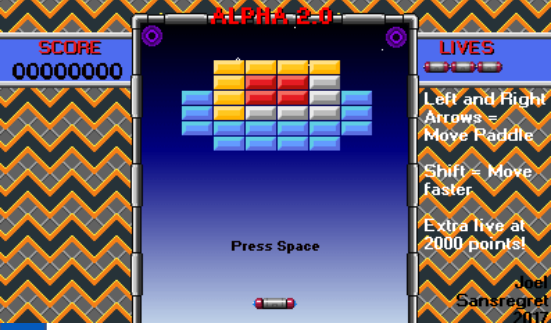
Construtores

```java
//Construtor recebendo parametros
public Pessoa(String nome, int idade) {
    this.nome = nome;
    this.idade = idade;
}
//Construtor Vazio
public Pessoa() {

}
//Recebendo apenas o nome
public Pessoa(String nome) {
    this.nome = nome;
}
```

```java
public static void main(String args[]) {
    //Invoca Construtor Vazio (sem parametros)
    Pessoa p = new Pessoa();
    //Invoca Construtor com um parâmetro String
    Pessoa p1 = new Pessoa("Capiroto");
    //Invoca Construtor com dois parâmetros
    Pessoa p2 = new Pessoa("Tinhoso", 56);
}
```

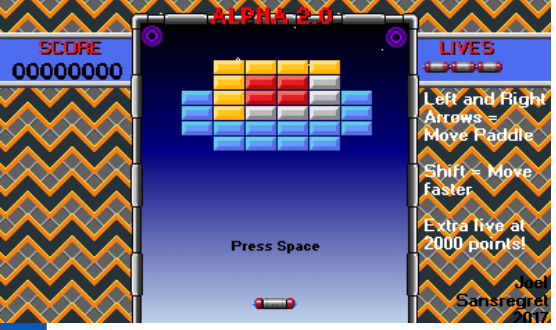
Modificador *Static* – Modificador de Acesso Global

- Imagine que estamos modelando um jogo estilo Arkanoid, e queremos saber quantos blocos existem no jogo



Modificador *Static*

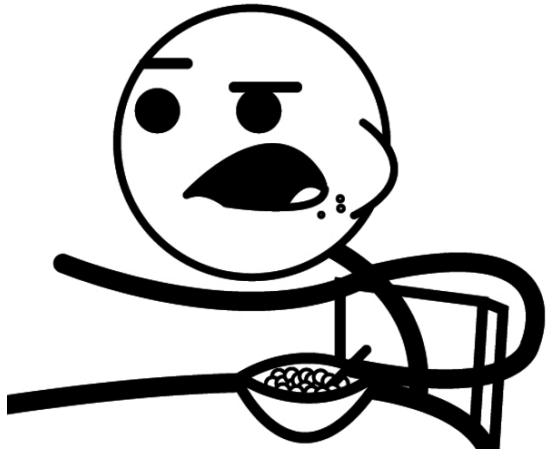
- Como podemos fazer isso? Precisamos saber quantos foram criados. Vamos colocar um contador dentro do construtor de uma classe Bloco.
- A cada nova instância faremos um incremento
- Será que funciona?



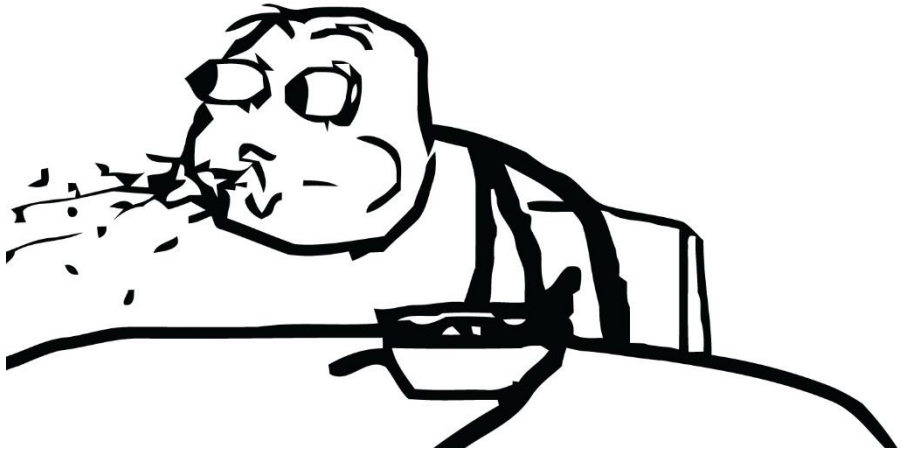
Modificador *Static*

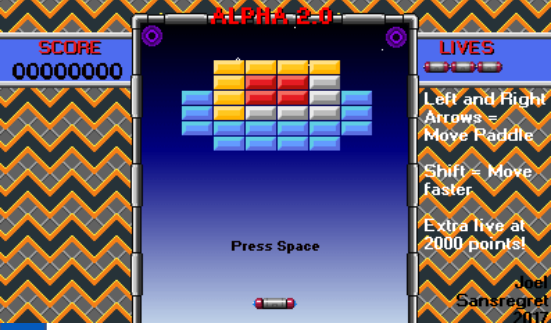
```
public class Bloco {

    private int numBlocos = 0;

    public Bloco() {
        numBlocos++;
    }
```

```
    public static void main(String[] args) {
        Bloco bloco1 = new Bloco();
        Bloco bloco2 = new Bloco();

        System.out.println(bloco2.getNumBlocos());

    }
```



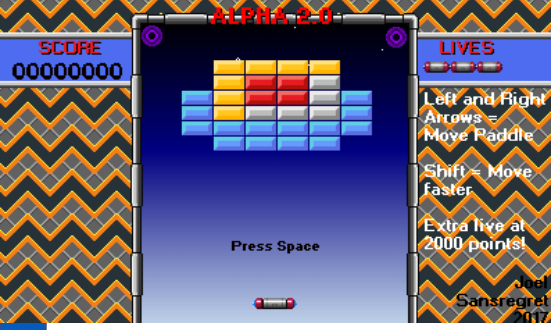
```
1
```





Modificador *Static*

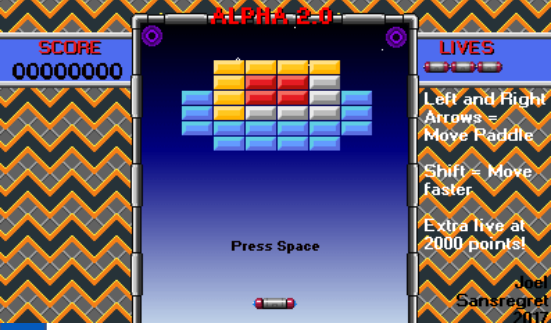
- Cada instância tem o seu próprio numBlocos, assim esse número sempre será incrementando uma única vez.
- Por isso eles são chamados membros da instância.
- Precisamos de algo compartilhado entre as instâncias, ou seja, algo que pertença a classe e não a instância.
- Vamos utilizar o modificador ***static***



Modificador *Static*

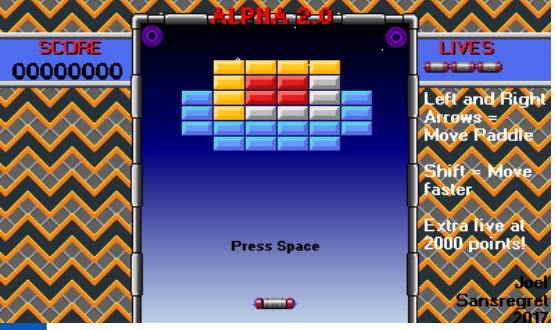
```
public class Bloco {

    private static int numBlocos = 0;

    public Bloco() {
        numBlocos++;
    }

    public static int getNumBlocos() {
        return numBlocos;
    }
```

- Qualquer instância da classe Bloco compartilha a mesma variável numBlocos. Essa variável pertence a classe.
- O acesso é dado via classe.



Modificador *Static*

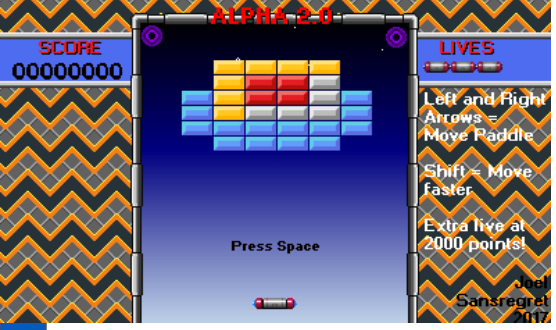
```
public class Main {

    public static void main(String[] args) {
        Bloco bloco1 = new Bloco();
        Bloco bloco2 = new Bloco();

        System.out.println(Bloco.getNumBlocos());

    }
```



Modificador *Static*

```
public class Main {

    public static void main(String[] args) {
        Bloco bloco1 = new Bloco();
        Bloco bloco2 = new Bloco();

        System.out.println(Bloco.getNumBlocos());

    }
```

2



Modificador ***Static***

- E se esse membro não fosse ***private***?

```
public class Bloco {

    static int numBlocos = 0;

    public Bloco() {
        numBlocos++;
    }
```

```
public class Main {

    public static void main(String[] args) {
        Bloco bloco1 = new Bloco();
        Bloco bloco2 = new Bloco();

        System.out.println(Bloco.numBlocos);

    }
```

2

Modificador *Static*

- E os métodos com modificador ***static***?
- São métodos que não dependem de nenhuma variável de instância.
- Quando invocados, executam uma função sem a dependência do conteúdo de um objeto.
- É criado como um membro de um objeto, mas não precisa de uma instância para ser acessado.

Modificador *Static*

- O método main é o principal exemplo:
- O Java chama o método Main, sem criar uma instância dessa classe já que o Main é ***Static.***

```
public class Main {

    public static void main(String[] args) {

    }
}
```

Modificador ***Static***

- Qual o propósito de métodos ***Static***?
- Métodos utilitários ou ***helpers*** que não requerem modificação de estado. Elimina-se a necessidade de criação de objetos só para acesso algum método.
- Utilizar um estado que não muda.

Exercício 1 – Space Shooter

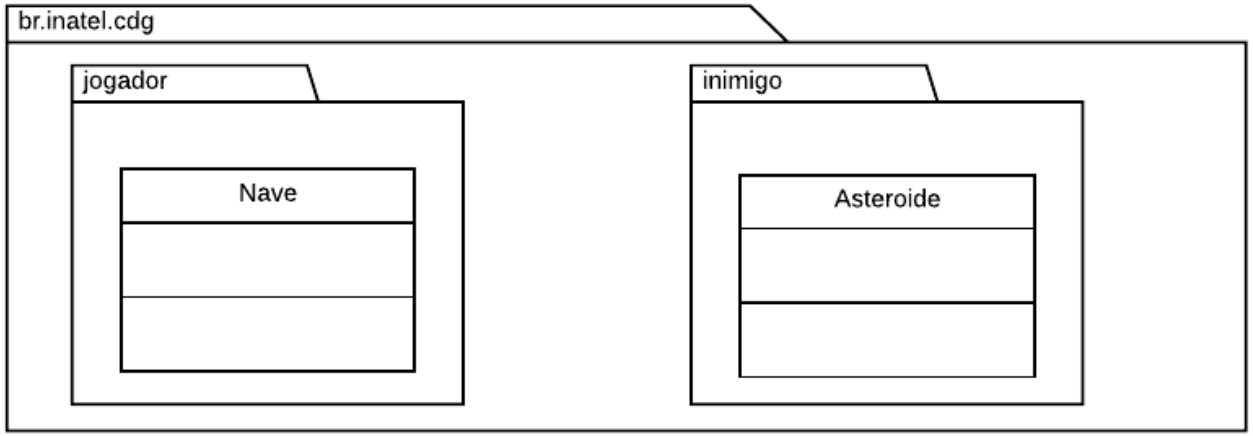
Exercício 1 – Space Shooter

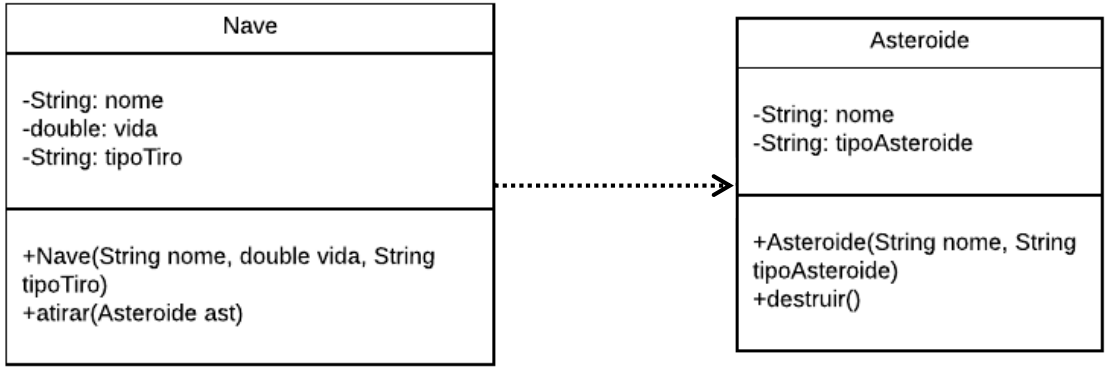
- Crie classes Java que sigam o UML apresentado a seguir.
- Temos uma estrutura de Classes e uma de Pacotes.
- Crie getters e setters que julgar necessário!
- Faça os seguintes testes
  - A Nave pode invocar o método atirar(), recebendo como parâmetro um Asteroide.
  - Existem dois tipos de asteroides: Pequeno e Grande
  - A nave possui dois tipos de tiro: Normal e Explosivo
  - Asteroides do tipo “Grande” são destruídos apenas com Naves que possuem um tipo de tiro “Explosivo”
  - Se o Asteroide for destruído, ele chama o método destruir(), que apenas imprime uma mensagem

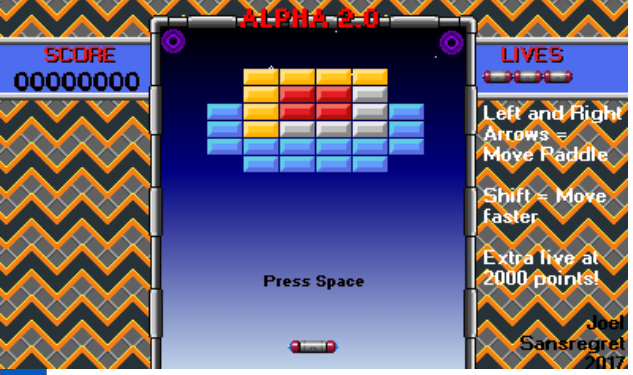
Exercício 1 – Space Shooter

- OBS: No diagrama de classes UML, quando temos uma seta, é uma relação de dependência. Assim, Nave depende de Asteroide. Pois um método da classe Nave, recebe uma variável do tipo Asteroide. Mas não temos um membro na classe Nave do tipo Asteroide, por isso não temos uma agregação e/ou composição!

Exercício 1 – Space Shooter







Exercício - Arkanoid

- Simule um jogo de Arkanoid. Construa duas classes, Jogador e Bloco. Lembre-se de utilizar pacotes para organizar.
- A cada novo bloco, incremente o número de blocos em jogo.
- A cada bloco destruído, decremente o número de blocos em jogo e aumente a pontuação do Jogador.
- Ao final, imprima o número de blocos criados, destruídos e a pontuação do jogador.
- Crie um método estático em uma classe Conversor que converte a pontuação do jogador em moedas. 1 ponto = 100 moedas. Mostre o número de moedas na classe Main.

Resolução dos Exercícios

https://github.com/chrislima-inatel/C206_C125

Material Complementar

- Capítulo 5 da apostila FJ-11
  - Modificadores de Acesso e Atributos(Membros) de Classe
  - A partir do item 5.4