C206 - POO

Arthur Openheimer

Inatel

Informações Gerais

- Atendimento → Terça-feira 17:30-19:10, prédio 1, sala 19
- Email → arthur.openheimer@ges.inatel.br
- Github →https://github.com/ArthurOpenheimer/C206-Monitoria

Inatel

Encapsulamento

Encapsulamento é um dos 4 pilares de POO, com ele podemos utilizar modificadores de acesso e limitar o acesso aos dados e aumentar o nível de segurança e organização

Modificadores de acesso

Em Java, existem quatro opções de modificadores de acesso, porém aqui no laboratório iremos focar em dois: Public e Private

Modificador	Classe	Mesmo Pacote	Pacote diferente (SubClasse	Pacote Diferente (Global)
Public	✅	✅	✅	✅
Protected	✅	✅	✅	❌
Default	✅	✅	❌	❌
Private	✅	❌	❌	❌

Inatel

Private

- Quando usamos o **Private**, apenas a própria classe pode acessar o atributo ou então chamar o método
- Apesar de parecer limitante e um trabalho extra, quando limitamos o acesso aos atributos e métodos, garantimos uma segurança maior no desenvolvimento. Podemos isolar dados onde eles não são necessários ou até mesmo onde não poderiam ser acessados

Getters e Setters

- Para alterar e acessar valores de atributos utilizamos métodos auxiliares chamados de getters e setters
- Não devemos criar obrigatoriamente getters e setters para todo atributo privado, apenas quando é necessário

```java
public class Personagem {

    private int vida;

}
```

```java
public class Main {
    Run | Debug
    public static void main(String[] args) {

        Personagem p1 = new Personagem();

        p1.vida = 100;

        System.out.println(p1.vida);
    }
}
```

Main.java 2 of 2 problems

The field Personagem.vida is not visible Java(33554503)

Inatel

Getters e Setters

```java
public class Personagem {

    private int vida;

    public int getVida() {
        return vida;
    }

    public void setVida(int vida) {
        this.vida = vida;
    }
}
```

```java
public class Main {
    Run | Debug
    public static void main(String[] args) {

        Personagem p1 = new Personagem();

        p1.setVida(vida:100);

        System.out.println(p1.getVida());
        // Output: 100
    }
}
```

Atributos estáticos

- O modificador **static** é utilizado quando queremos fazer com que um método ou atributo se torne da Classe e não do Objeto
- Métodos estáticos podem ser chamados sem a necessidade de criar um objeto da classe e um atributo estático é compartilhado entre todas as instâncias da classe

```java
public class Personagem {
    public static int id = 0;

    Personagem(){
        id++;
    }

    public static void mostraId(){
        System.out.println(id);
    }
```

```java
public static void main(String[] args) {

    Personagem p1 = new Personagem();
    Personagem p2 = new Personagem();
    Personagem p3 = new Personagem();

    System.out.println(Personagem.id);
    // Output: 3
    Personagem.mostraId();
    // Output: 3
```

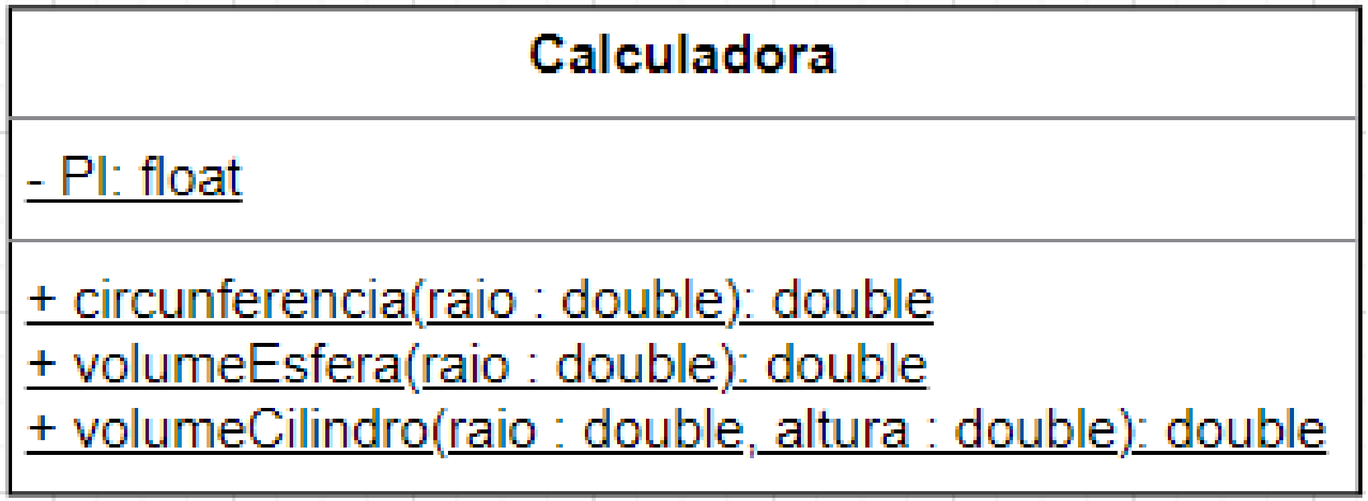
Inatel

Exercícios

1- Crie a classe Calculadora e utilize seus métodos na main sem criar uma instância da classe

- Circunferência: 2*pi*r
- Vol. Esfera: 4/3*pi*r^2
- Vol. Cilindro: pi*r^2*h

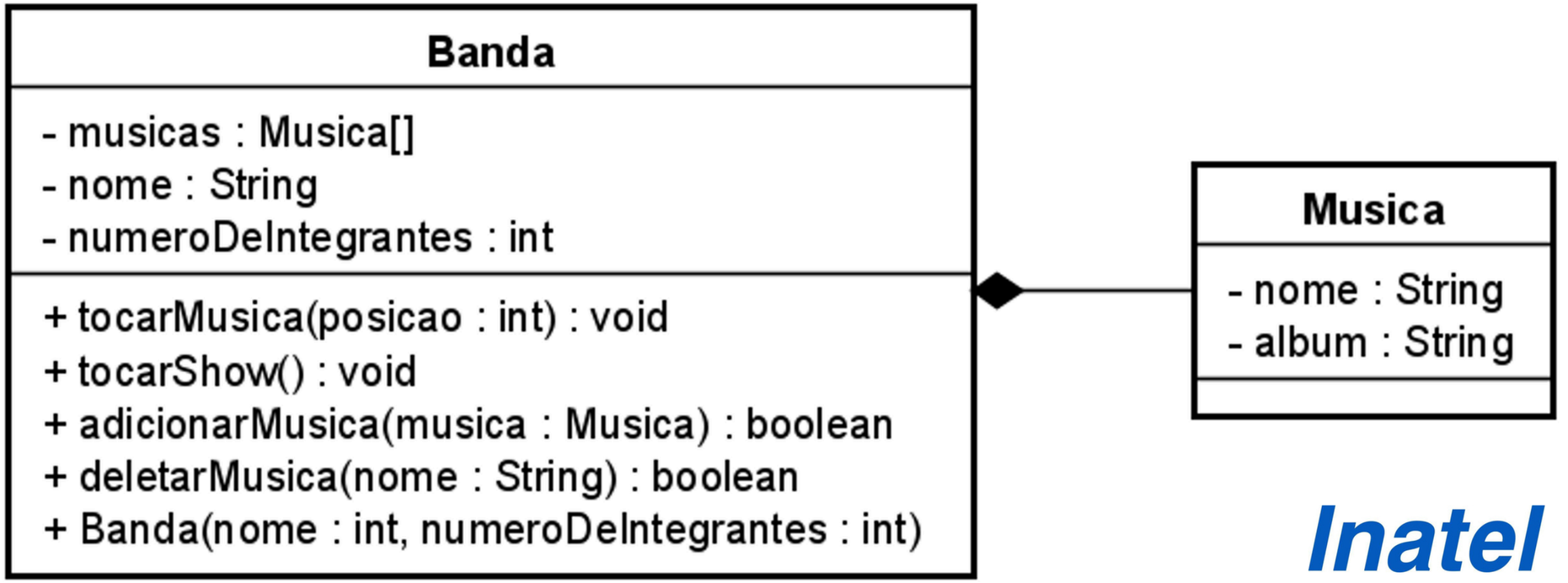
Para as potências, utilize:
Math.pow(raio, expoente)



Inatel

Exercícios

- **tocarMusica** recebe uma posição e deve imprimir as informações da música que está na array
- **tocarShow** precisa imprimir as informações de todas as músicas na array
- **adicionarMusica** e **deletarMusica** retornam true se a operação foi um sucesso
- Crie os getters e setters apenas quando forem necessários
- Faça todos os tratamentos necessários nas arrays e nas saídas
- Dica: em **deletarMusica** utilize “musicas[i].getNome().equals(nome)” para verificar se o nome do parâmetro é igual o nome da música no array. O método equals retorna True ou False
- Na Main, crie as instâncias necessárias e chame todos os métodos



Inatel