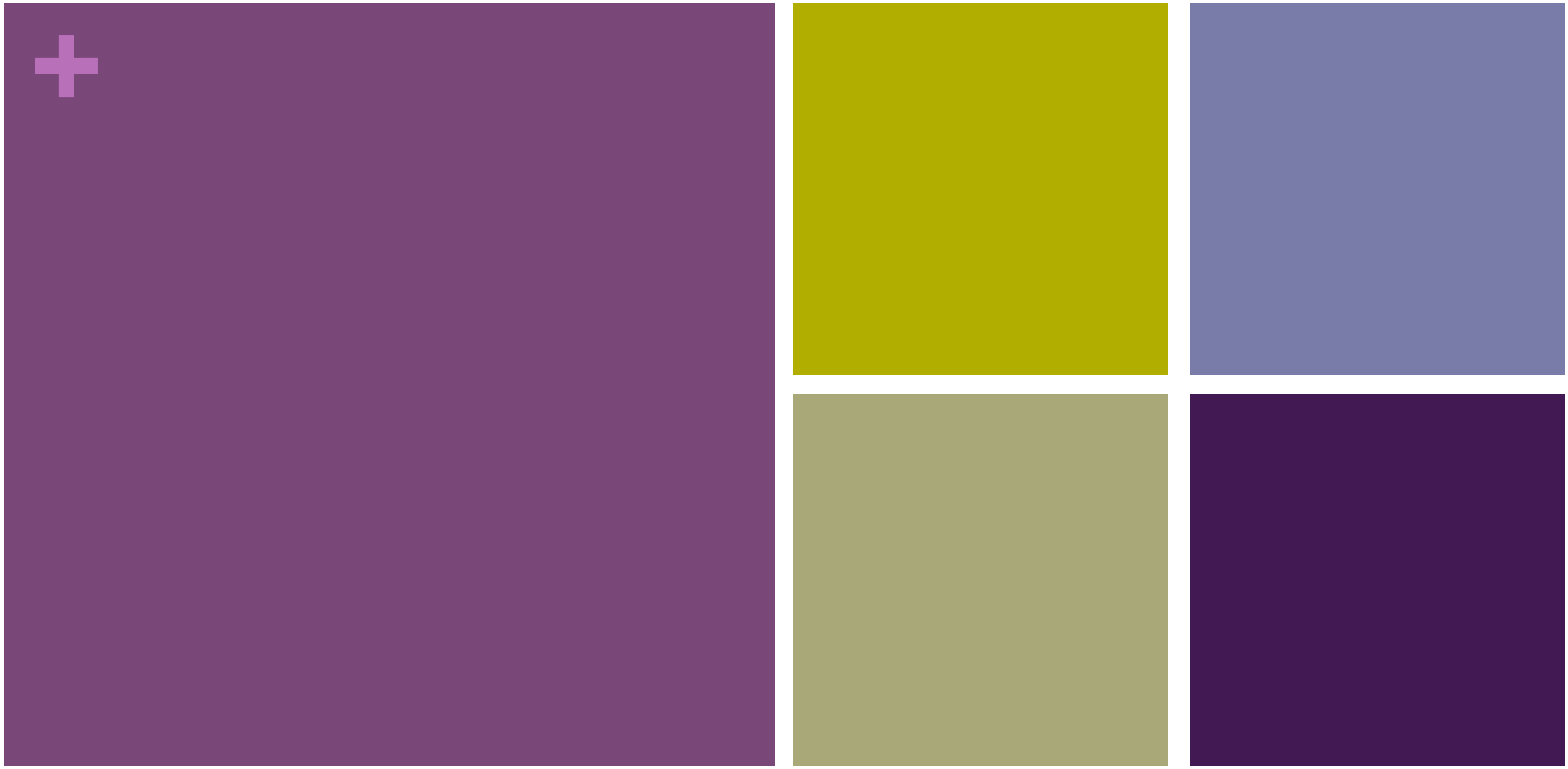


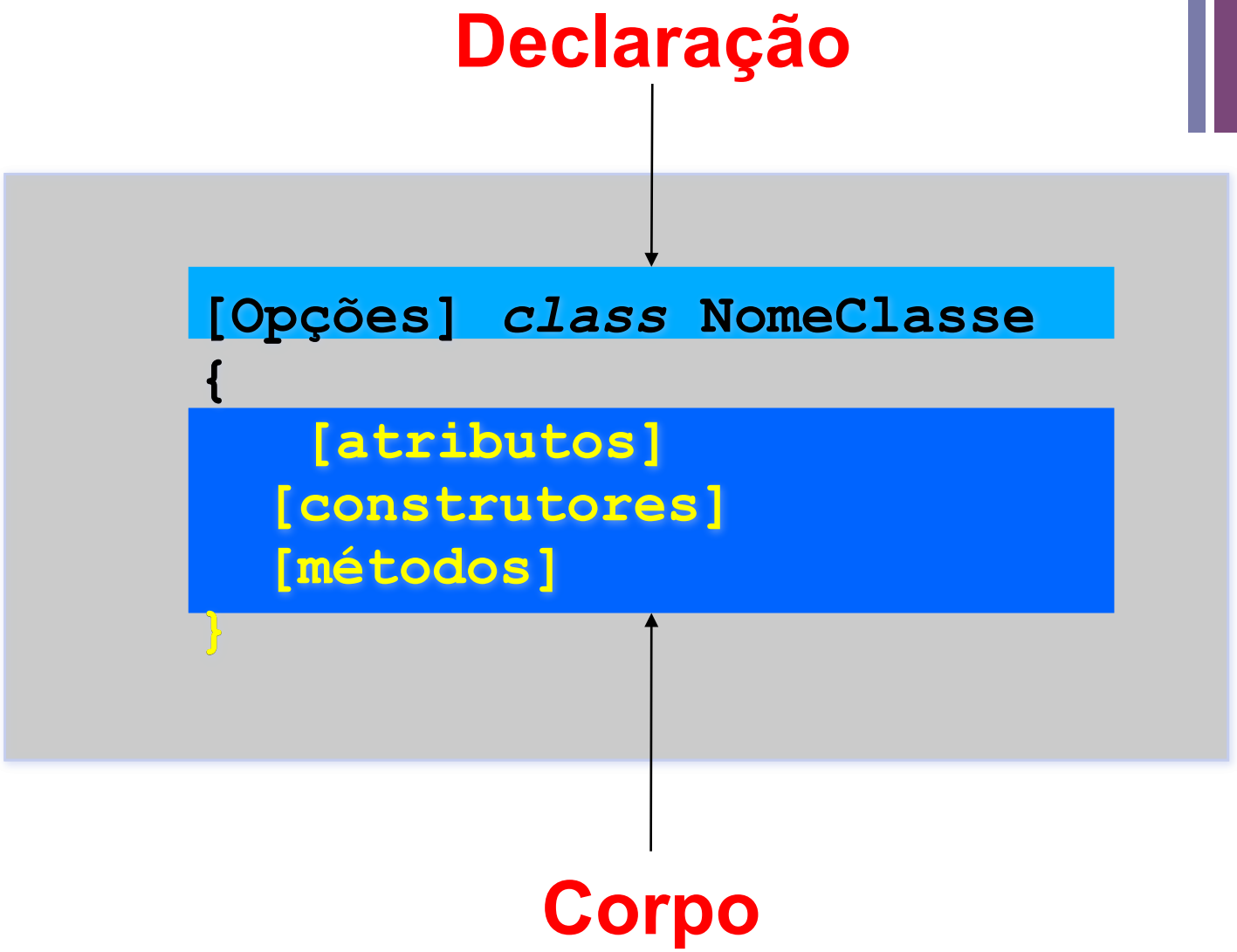
POO – Programação Orientada a Objetos

Classes em Java

+ Classes

- Para que a JVM crie objetos:
  - Ela precisa saber qual classe o objeto pertence
  - Na classe estão definidos os **atributos** e **métodos**
- Programamos classes e depois as usamos
  - Tudo em JAVA são classes
  - Menos os tipos primitivos
  - Uma classe define um novo tipo
- Componentes da classe
  - São os **membros** da classe

Definição de classe



Exemplo

```
public class Lampada{

  private boolean ligada;
  private double potencia;

  public Lampada(){
        ligada = false;
  }

  public void ligar(){
        ligada = true;
  }
  public void desligar(){
        ligada = false;
  }
  public boolean estaLigada(){
        return ligada;
  }
}
```

Atributos

Construtor

Métodos

O que fazer

- Edite o arquivo fonte
- Salve com a extensão **.java**
  - Se a classe for **public** o nome do arquivo **deve** ser o mesmo nome da classe com a extensão **.java**
- Compile com o javac
- Será criado o arquivo .class que contém a classe a ser usada
  - Este arquivo deverá estar no CLASSPATH da JVM
    - CLASSPATH é o lugar onde a JVM procura as classes
    - Variável de ambiente CLASSPATH
- Crie um programa que use a classe
  - Objetos dessa classe podem ser criados e manipulados

O que fazer

```
public class Main {
   public static void main(String[] args) {
     Scanner sc = new Scanner(System.in);
     Lampada l = new Lampada();
     int opcao = 0;
     do
     {
       System.out.println("0) Sair 1) Ligar 2) desligar 3) Mostrar ");
       System.out.print("Digite opção: "); opcao = sc.nextInt();
       switch (opcao){
         case 1: l.ligar(); break;
         case 2: l.desligar() ; break;
         case 3:
           if (l.estaLigada())
             System.out.println("Ligada");
           else
             System.out.println("Desligada");
           break;
       }

     } while (opcao != 0);
     System.out.println("Fim do programa!");
   }
}
```

Atributos

- São as variáveis de instância
  - Fazem parte de cada objeto (instância)
- Declarada fora dos métodos
- "Vivem" enquanto o objeto "viver"
- São pré-inicializadas
  - boolean ==> false
  - tipo aritmético ==> 0
  - char ==> '\u0000'
  - class ==> null

exemplo:

```
public int x;
private float y;
Ponto p1;
```

Obs: Todo objeto possui um identificador chamado **this**, *que é uma referência para o próprio objeto (proximo slide).*

this

- Todo objeto possui um atributo que é uma referência a ele mesmo
  - Usado para acesso a membros do proprio objeto
    - **this.membro**
  - Evita conflito
    - Com parâmetros de metodos, por exemplo
- Exemplo:

```
class Qualquer
{    int x, y;
public void mover(int x,int y){
          this.x = x;
          this.y = y;
}
}
```

+ Métodos

- Declaração de método:
  - <opcoes> tipoRetorno nomeMetodo (parametros)
  - public int metodo(float x)
- Passagem de parâmetros:
  - Deve ser informados o **tipo** e **identificador** dos parâmetros.
  - Funciona no método como uma variável normal
  - Passam o valor do identificador

+ Métodos

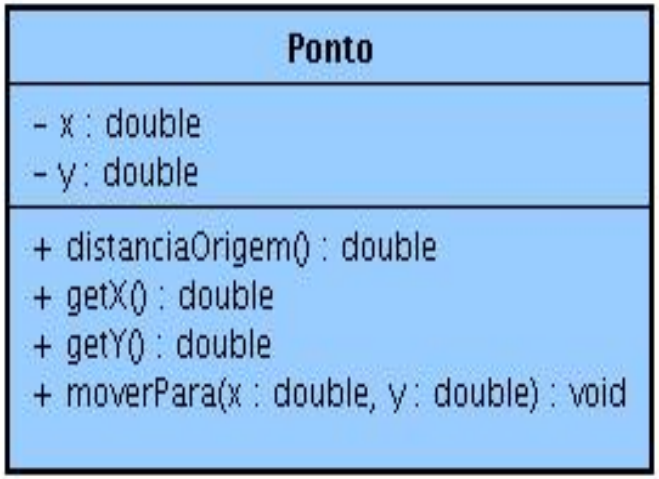
- Corpo do método:
  - Implementa as operações do método
  - Fica entre chaves ({})
  - Variáveis podem ser criadas
    - Ela é dita local
    - Não é pré-inicializada.
    - Só existe enquanto o método está em execução

Construtores

- **Mesmo** nome da classe
- Não possui retorno
- Podem ser **vários**
  - Diferença na quantidade e tipo dos parâmetros
- Construtor padrão é fornecido
  - Se não houver pelo menos um definido
  - Não possui parâmetros
- É chamado na execução do **new**

Outro exemplo

- Ponto
  - Plano cartesiano
  - Coordenadas X e Y
  - Pode ser movido de lugar
  - Podemos saber sua distância da origem




```
public class Ponto{
  private double x,y;
  public Ponto(){
    x = 0; y = 0;
  }
  public Ponto(double x, double y)
  {this.x = x, this.y = y;}

  public void moverPara(double x, double y){
    this.x = x;  this.y = y;
  }

  public double getX(){ return x;}

  public double getY(){ return y;}

  public double distanciaOrigem(){
    double distancia;
    distancia = (double)Math.sqrt(x*x +y*y);
    return distancia;
  }
  public String toString(){
    return "Ponto ("+x+","+y+")";
  }
}
```

+ Visibilidade

- Proteção de acesso
  - Proteger o interior da classe
- Explicitar o que usuários (da classe) precisa saber
- pode ser:
  - **private**: Apenas membros da classe têm acesso
  - **protected**: Membros da classe e subclasses
  - **public**: Todos têm acesso
  - **default**: Apenas membros do mesmo pacote

Proteção de acesso

- Interface
  - Visão **externa** da classe
  - **O que** os objetos da classes fazem
  - Definem o "contrato" da classe
    - O que o cliente precisa conhecer da classe
- Implementação
  - Visão **interna** da classe
  - **Como** os objetos fazem as operação
  - Representação interna
    - cliente não precisa (nem deve) conhecer a implementação
  - Realizam o contrato definifo pela interface

Elementos do modelo de objetos

- Abstração
  - Uma abstração denota as características essenciais de um objeto que o distingue de todas as outras espécies de objetos e assim provê limites conceituais bem definidos, sempre relativos à perspectiva de um observador.
- Encapsulamento
  - Encapsulamento é o processo de esconder todos os detalhes de um objeto que não contribuem para suas características essenciais

+ Proteção de acesso

- Atributos fazem parte a implementação
  - Declare-os como **private**
- Nem todos os métodos fazem parte da interface
  - Métodos que servem para auxiliar outros métodos
    - Declare-os **private**
- Deixe **public** apenas o que o cliente deve saber
  - Métodos da interface

Convenção de nomes

- Variáveis e Métodos:
  - Use minúsculas.
  - Se o nome consiste de várias palavras, concatene-as e use a primeira letra de cada uma delas em maiúsculo.

  - Variáveis: `raio` e `area`
  - Método: `calcularArea`

- Nomes de Classes:
  - Use a Primeira letra de cada palavra em maiúscula
  - Classe: `Circulo, Ponto, NumeroComplexo`

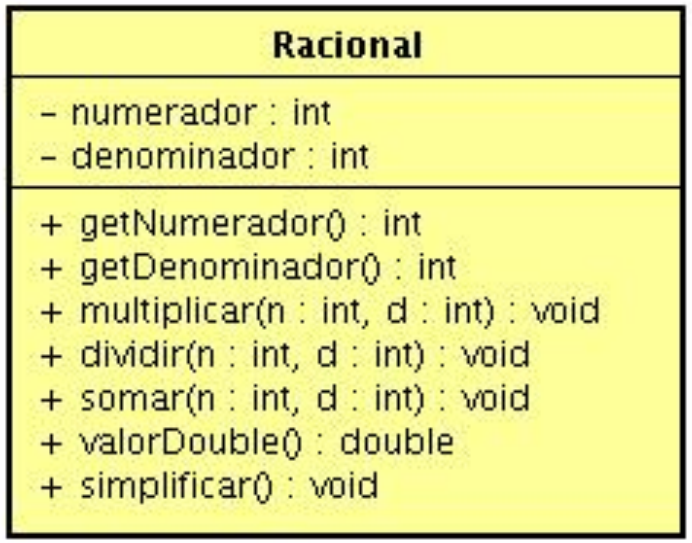
+ Exemplo

- // TODO Fazer um exemplo completo que use visibilidade

+ Dúvidas

Exercício

- Implemente a classe Racional
  - Representam uma fração
  - Dois construtores
    - Sem parâmetros
      - 1/1
    - Com dois parâmetros
      - numerador e denominador
  - Métodos que realizam as operações
    - Recebem numerador e denominador que representam a fração da operação
    - modificam a fração
- Crie um programa para testar sua classe



Exercicio

- Modifique os métodos que recebem numerador e denominador para que recem agora um objeto da classe Racional
  - Menos o construtor
- Modifique o programa que testa a classe