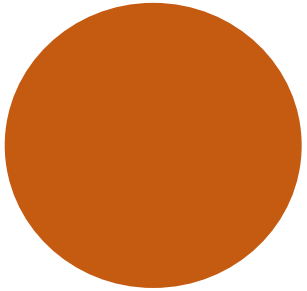
# JAVA2D

## Criando Objetos Simples

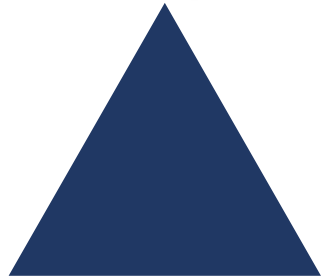
IDEA FACTORY

Rectângulo

Círculo



Triângulo



Essas figuras, muito provavelmente você já conhece

No mundo real, podemos entender que são figuras geométricas básicas ou simples

E não são tão diferentes em Java...

Vejamos os códigos!

# JAVA2D

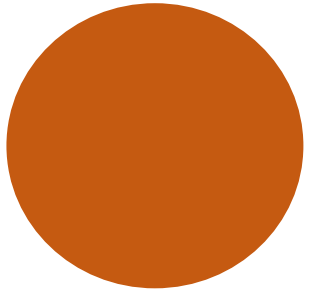
## Criando Objetos Simples

IDEA FACTORY

Rectângulo

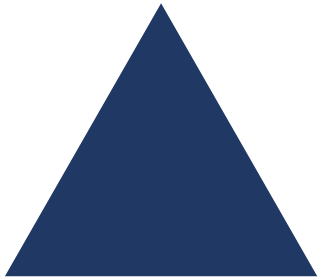
```
g.fillRect(10, 10, 120, 100);
```

Círculo



```
g.fillOval(10, 10, 120, 120);
```

Triângulo



E como é o triângulo?

IDEA FACTORY

# JAVA2D

## Criando Objetos Simples

- Introdução aos conceitos básicos

Para falarmos do triângulo, vamos fazer introdução aos conceitos

1. **Graphics** Classe que nos permite representar formas

   Em java, quando queremos exibir um componente visual, este componente é desenhado por esta classe.

   Esta classe acarreta métodos especiais que ajudam na visualização de componentes básicos, como **drawRect(), drawString(), fillRect, drawOval(), drawLine()…**

   Os métodos acima permitem-nos exibir figuras rectangulares, palavras, círculos, linhas… Existem mais métodos interessantes que esta classe acarrecta

IDEA FACTORY

# JAVA2D

## Criando Objetos Simples

- Introdução aos conceitos básicos

2. **Graphics2D** Vamos entender que é uma atualização da classe anterior

   Nesta classe conseguimos fazer tudo o que é possível na anterior, pois ela herda a classe **Graphics**

   Nesta classe, nós podemos inclusive criar **Transformações** na representação das formas, como escalar o objeto, mover, rodar e até cortar

   Esta classe também permite representar com melhor qualidade as formas gráficas

3. **Shape** Formas

   Esta classe serve para criarmos as formas propriamente ditas que posteriormente serão representadas pela classe **Graphics2D**

# JAVA2D

## Criando Objetos Simples

- Introdução aos conceitos básicos

IDEA FACTORY

### 3. **Shape** Formas

Esta classe serve para criarmos as formas propriamente ditas que posteriormente serão representadas pela classe **Graphics2D**

Aqui conseguimos definir melhor como queremos a nossa forma, quais transformações elas vão ter e inclusive operações

Algumas classes que herdam a **Shape** facilitam ainda mais o nosso trabalho:

**Rectangle2d**
**Ellipse2D**
**Area**
**CubicCurvid2D**
**RoundRectangle2D**
**Arc2D**
**Line2D**
**GeneralPath**

# JAVA2D

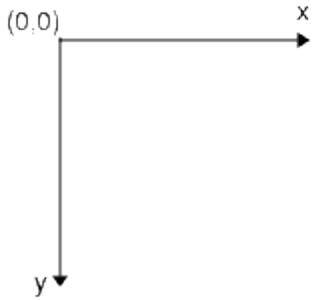
## Criando Objetos Simples

- Introdução aos conceitos básicos

IDEA FACTORY

### 4. **Point** Classe de localização (xy)

Em java2D, os gráficos são representedados em um determinado ponto, este ponto representa-se de




Toda a tela do nosso formulário tem um determinado ponto. O canto superior esquerdo, como mostra na figura representa as coordenadas x = 0, y = 0, e seguem os valores positivos para direita (x, valores positivos), para a esquerda (x, valores negativos), para baixo (y, valores positivos) e para cima (y, valores negativos)

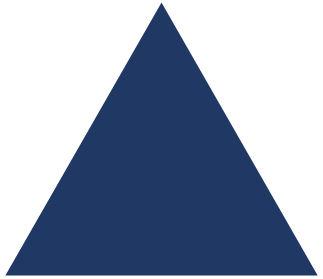
IDEA FACTORY

# JAVA2D

## Criando Objetos Simples

- Criando o Triângulo
  - Introdução ao GeneralPath

Triângulo



Não há uma maneira exata para criarmos um triângulo! Mas a melhor maneira é usando o Shape **GeneralPath**

Pensa no **GeneralPath** como um lápis. Para podermos usá-lo, você precisa mover o lápis à algum lugar da tela usando o método **moveTo()**

```
GeneralPath generalPath = new GeneralPath();
generalPath.moveTo(x, y);
```

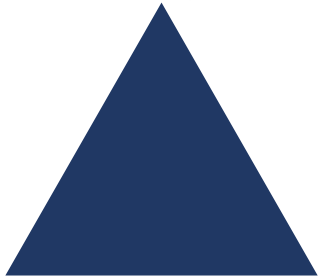
# JAVA2D

## Criando Objetos Simples

- Criando o Triângulo
  - Introdução ao GeneralPath

IDEA FACTORY

Triângulo



Se você pretende desenhar uma linha, deverá usar o método **lineTo()**

```
GeneralPath generalPath = new GeneralPath();
generalPath.moveTo(x, y);
generalPath.lineTo(500, 500);
```

Se você pretende desenhar uma curva, deverá usar o método **moveTo()** (este método é basiado na curva de Bézier)

```
GeneralPath generalPath = new GeneralPath();
generalPath.moveTo(x, y);
generalPath.lineTo(500, 500);
generalPath.curveTo(500, 500, 450, 300, 100, 100);
```

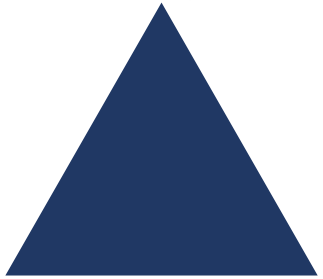
# JAVA2D

## Criando Objetos Simples

- Criando o Triângulo
  - Introdução ao GeneralPath

IDEA FACTORY

Triângulo



Para desenharmos este triângulo, seguem-se os códigos

Código 1

```
Graphics2D g2 = (Graphics2D) g;

GeneralPath lapis = new GeneralPath(); // O nosso lápis
Point coordenadas = new Point(500, 100); // coordenadas do lápis

g2.setPaint(new Color(32, 56, 100)); // cor do lápis

lapis.moveTo(coordenadas.x, coordenadas.y); // movendo o lápis
para a coordenadas
lapis.lineTo(coordenadas.x - 100, coordenadas.y + 100); //
Desenhando - 100px em x e + 100px para y
lapis.lineTo(coordenadas.x + 100, coordenadas.y + 100); //
Desenhando + 100px em x e + 100px para y
lapis.lineTo(coordenadas.x , coordenadas.y);  // Desenhando
mantendo as coordendas de forma inicial em x,y

g2.fill(lapis); // representado o desenho de forma preenchida
```

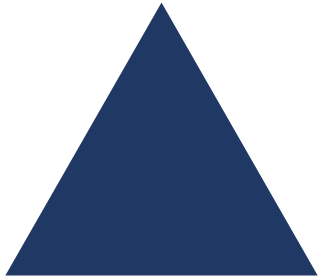
# JAVA2D

## Criando Objetos Simples

- Criando o Triângulo
  - Introdução ao GeneralPath

IDEA FACTORY

Triângulo



Para desenharmos este triângulo, seguem-se os códigos

Código 2

```
Graphics2D g2 = (Graphics2D) g;

GeneralPath lapis = new GeneralPath(); // O nosso lápis
Point coordenadas = new Point(500, 100); // coordenadas do lápis

g2.setPaint(new Color(32, 56, 100)); // cor do lápis

lapis.moveTo(coordenadas.x, coordenadas.y); // movendo o lápis
para a coordenadas
lapis.lineTo(coordenadas.x - 100, coordenadas.y + 100); //
Desenhando - 100px em x e + 100px para y
lapis.lineTo(coordenadas.x + 100, coordenadas.y + 100); //
Desenhando + 100px em x e + 100px para y
lapis.closePath();  // Unir este ponto ao ponto incial

g2.fill(lapis); // representado o desenho de forma preenchida
```

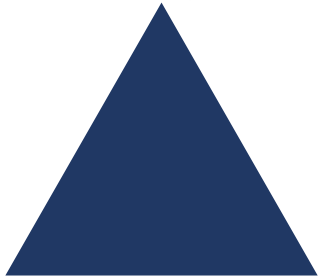
# JAVA2D

## Criando Objetos Simples

- Criando o Triângulo
  - Introdução ao GeneralPath

IDEA FACTORY

Triângulo



Para desenharmos este triângulo, seguem-se os códigos

Código 3

```
GeneralPath lapis = new GeneralPath(); // O nosso lápis
Point coordenadas = new Point(500, 100); // coordenadas do lápis

g2.setPaint(new Color(32, 56, 100)); // cor do lápis

lapis.moveTo(coordenadas.x, coordenadas.y); // movendo o lápis para a
coordenadas
lapis.lineTo(coordenadas.x -= 100, coordenadas.y += 100); //
Atualizando x - 100px e + 100px para y
lapis.lineTo(coordenadas.x += 200, coordenadas.y); // Atualizando +
200px em x e mantendo y
lapis.closePath(); // Unindo as extremidades

g2.fill(lapis); // representado o desenho de forma preenchida
```