Programação e Análise OO com UML e Java

Análise Orientada a Objetos utilizando diagramas UML

Mathias Santos de brito

UML - O que é?

- UML significa LINGUAGEM DE MODELAGEM UNIFICADA. A sigla vem da abreviação do termo em inglês, **U**nified **M**odeling **L**anguage.
- O UML é uma linguagem gráfica de modelagem, este constitui uma série de diagramas que podemos utilizar para descrever um Software durante a etapa de Análise.
- Vimos na aula passada um diagrama que descrivia as classes de um sistema de uma Vídeo Locadora e a de um Estoque. Aquele diagrama era um diagrama de Classes UML.
- Os diagramas servem para tornar as características do sistemas mais fáceis de compreender, provendo aos programadores farta documentação para que ele inicie a programação do Sistema.

Origem do UML

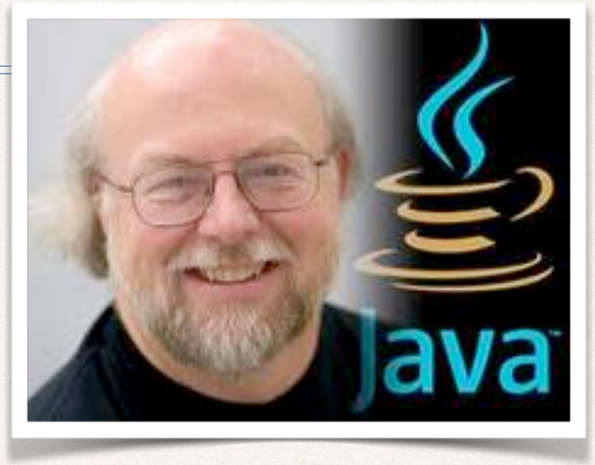
- Nos primórdios da Análise Orientada a Objetos não se existia diagramas padronizados para descrever o sistema na etapa de Análise.
- Cada empresas e pesquisadores possuíam um conjunto de diagramas e métodos.
- Se você trabalhava em uma empresa como Analista e mudasse de empresa você iria se deparar com diagramas que nunca viu, e iria ter que reaprender a trabalhar utilizando os diagramas da nova empresa.

Origem do UML

- Dois dos mais conhecidos pesquisadores da área (Booch e Jacobson), e criadores de duas famosas linguagens de modelagem, resolveram unir esforços para criar e encorajar o uso de uma linguagem padrão.
- Isso significava que, caso todos adotassem esta linguagem, você não teria tido problema quando mudou de uma empresa pra outra, no exemplo dado no Slide anterior.
- Isso significava também a união de esforços no intuito de se criar uma linguagem rica e eficiente.
- Nasce então o UML. O UML tornou-se rapidamente popular.


Java

Linguagem de Programação Java



- Java é uma linguagem de Orientada a Objetos.
- Tem fundamentos fortes das linguagens C e C++.
- Foi criada por James Arthur Gosling, quando trabalhava na Sun Microsystems em 1995.
- Java se tornou em partes muito famoso por ser uma linguagem que oferece um alto grau de portabilidade entre diferentes plataformas.
- Isso significa que Java foi uma das primeiras a se tornar popular dentre aquelas que permitiam rodar o seu programa em um computador com Unix, Linux, Windows, Mac OS X, sem a necessidade re recompilá-lo.

Maquina Virtual Java

- No geral se você escreve um programa em C para Windows, se quiser rodar no Linux terá que recompilá-lo, com um compilador específico pra o Linux.
- Isso vale também para o caso de ter que rodar o programa em uma Arquitetura de Computador diferente, por exemplo rodar um programa compilado para rodar em um processador Intel de 64 bits, não irá rodar em um processador Intel de 32 bits sem que se tenha que recompilá-lo.
- Isso se dá porque o arquivo binário escrito para um sistema ou arquitetura em princípio não é entendido por outro.
- O Java contornou esse problema desenvolvendo a Máquina Virtual Java (JVM - Java Virtual Machine). O arquivo gerado pelo processo de compilação em Java é feito para rodar sobre essa máquina virtual.

Maquina Virtual Java

- Então em princípio se você tiver a JVM instalada em seu computador, não importa o Sistema Operacional ou a Arquitetura do seu computador…
- O seu programa Java irá rodar sem que você necessite recompilar o seu código.
- Isso é o principal motivo pra a popularização do Java.
- Hoje muitas linguagens já são baseadas em Máquinas Virtuais como é o exemplo da plataforma .Net da Microsoft.

De volta ao UML - Tipos de Diagramas

Diagrama de Classes

Diagrama de Classes

- O diagrama de classes tem por objetivo identificar Classes que devem pertencer ao sistema, bem como as relações entre elas.
- Nas aulas passadas começamos um esboço de um diagrama de classes.
- Identificamos classes seus atributos e métodos, para um sistema de Vídeo Locadora e para um Estoque.
- Não sabiamos que aquele era um diagrama de classes, mas percebemos que aquela estrutura nos levava a pensar melhor sobre o problema.

Diagrama de Classes

- A representação de uma classe em um diagrama de Classes irá consistir de um retângulo divididos em 3 partes, vamos ver quais são, tome por base a representação gráfica da seguinte classe:

1. Primeira parte contêm o nome da Classe.

2. Segunda parte contêm o nome dos atributos e seus respectivos tipos.

3. Segunda parte contêm o nome dos métodos, seus argumentos entre parênteses e o tipo de retorno.

Diagrama de Classes: Nome

- O nome da classe em geral representa uma classe de objeto observado do mundo real.
- O nome da classe deve ter a sua primeira letra maiúscula.
- Caso o nome da classe seja composto utiliza-se a **Notação Camelo.**
  - **EsseEUmExemploDeNotacaoCamelo**
- Não use acentos ou caracteres especiais, apenas letras e números.

Diagrama de Classes: Atributos

- Os atributos são colocados na região logo abaixo do nome.
- Todo atributo tem um nome e um tipo. Este tipo pode ser um vetor também.
- Para descrever um atributo dê um nome a ele, depois insira o caractere : (dois pontos) seguido do tipo do atributo.
- Ao lado alguns exemplos.

Diagrama de Classes: Métodos

- Os métodos devem ser colocados na região logo abaixo dos atributos.
- Você deve inicialmente dar um nome, logo após entre parênteses listar os argumentos do método.
- Depois de dar o nome e listar os argumentos insira um : (dois pontos) e diga o tipo de retorno do método.
- Vamos analisar melhor um método que tem argumentos...

Diagrama de Classes: Métodos

- Vamos analisar o seguinte método:
  - calcularAreaDoRetangulo(base : floar, altura : float) : float
- Variáveis em UML serão diferentes do que estamos acostumado. Em C e em Java para declarar uma variável fazemos:
  - float base;
- Em UML faremos sempre
  - base : float
- Caso o método tenha mais de um argumento separe-os por virgula.

De volta ao Java - Programando uma Classe

Java - Programando uma Classe

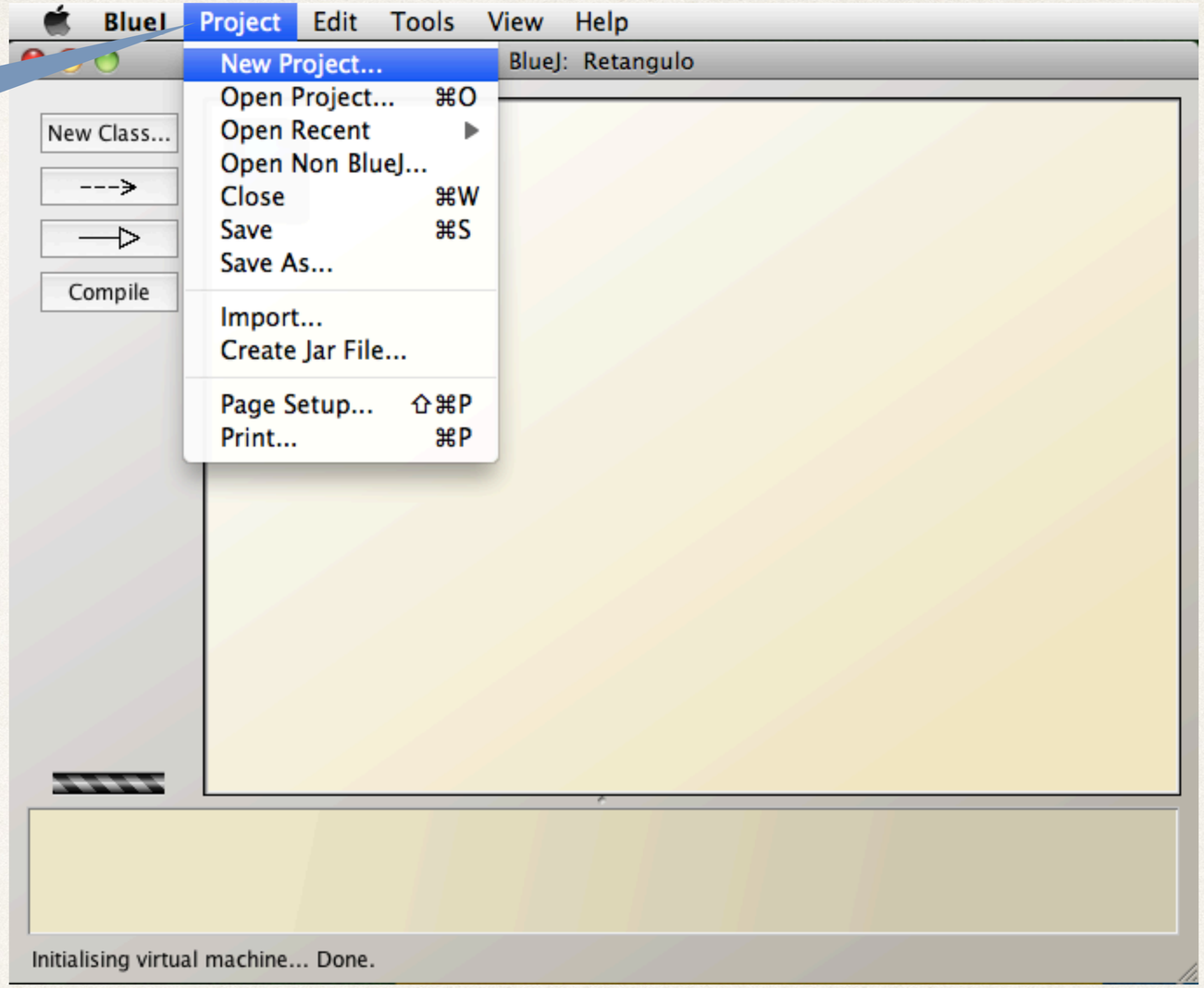
- Agora que já identificamos as classes, seus atributos e métodos é hora de criar o seu código em Java.
- Para isso usaremos o ambiente de programação **BlueJ.**
- Antes de voltarmos para o problema do Estoque, vamos pensar em uma Classe que modele um Retângulo.
- Obviamente que teremos dois atributos básicos, a base e a altura, e provavelmente podemos querer métodos que calculem a área do retângulo e o seu perímetro.

Criando Classes no BlueJ

Criando Classes no BlueJ

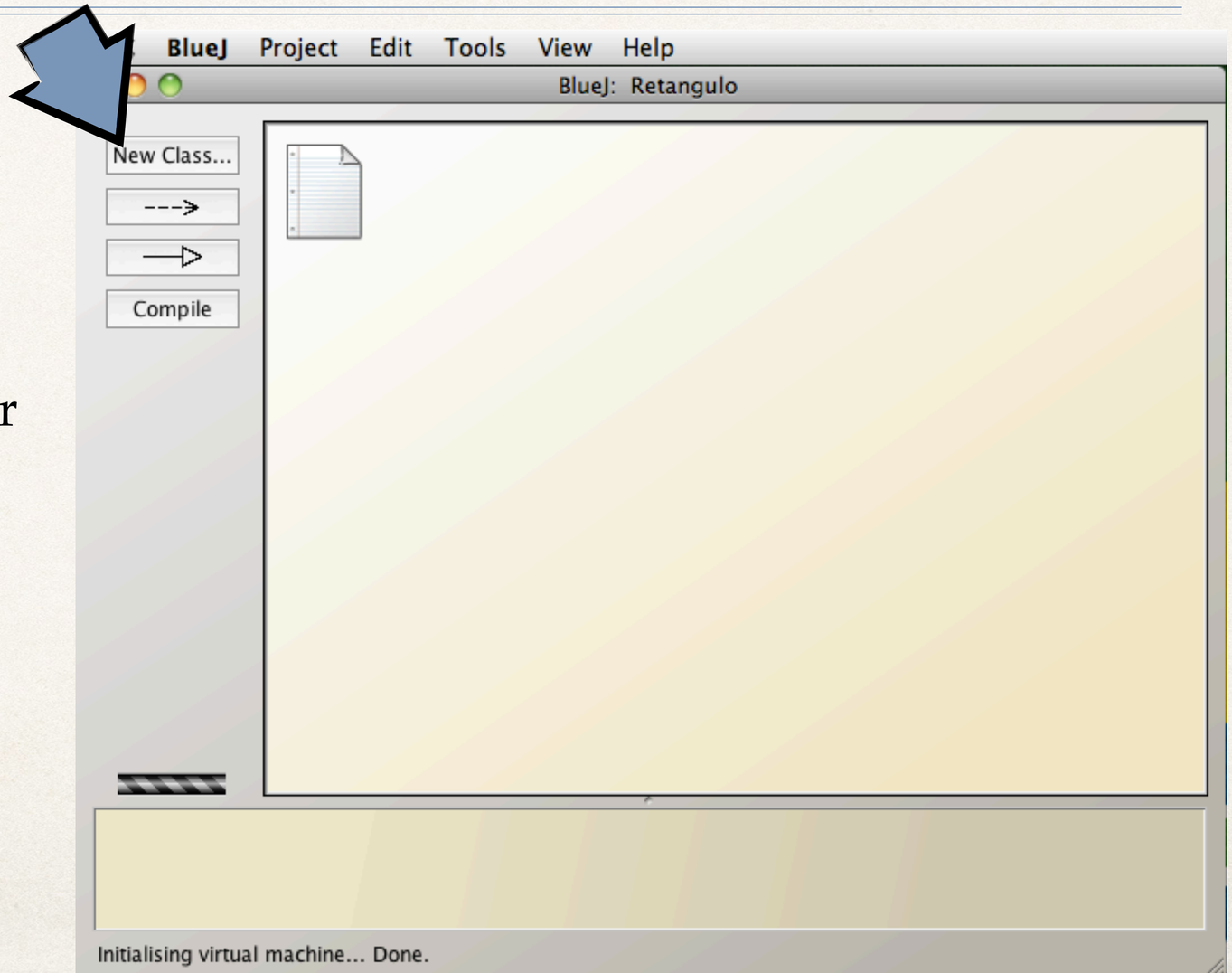
- Observem o BlueJ no próximos slides e os passos para se criar uma classe.
- Ao abrir o BlueJ primeiro criem um novo projeto, vamos chamá-lo de Formas Geométricas.
- Para isso faça o seguinte:

Clique em **Project** e logo após em **New Project...**



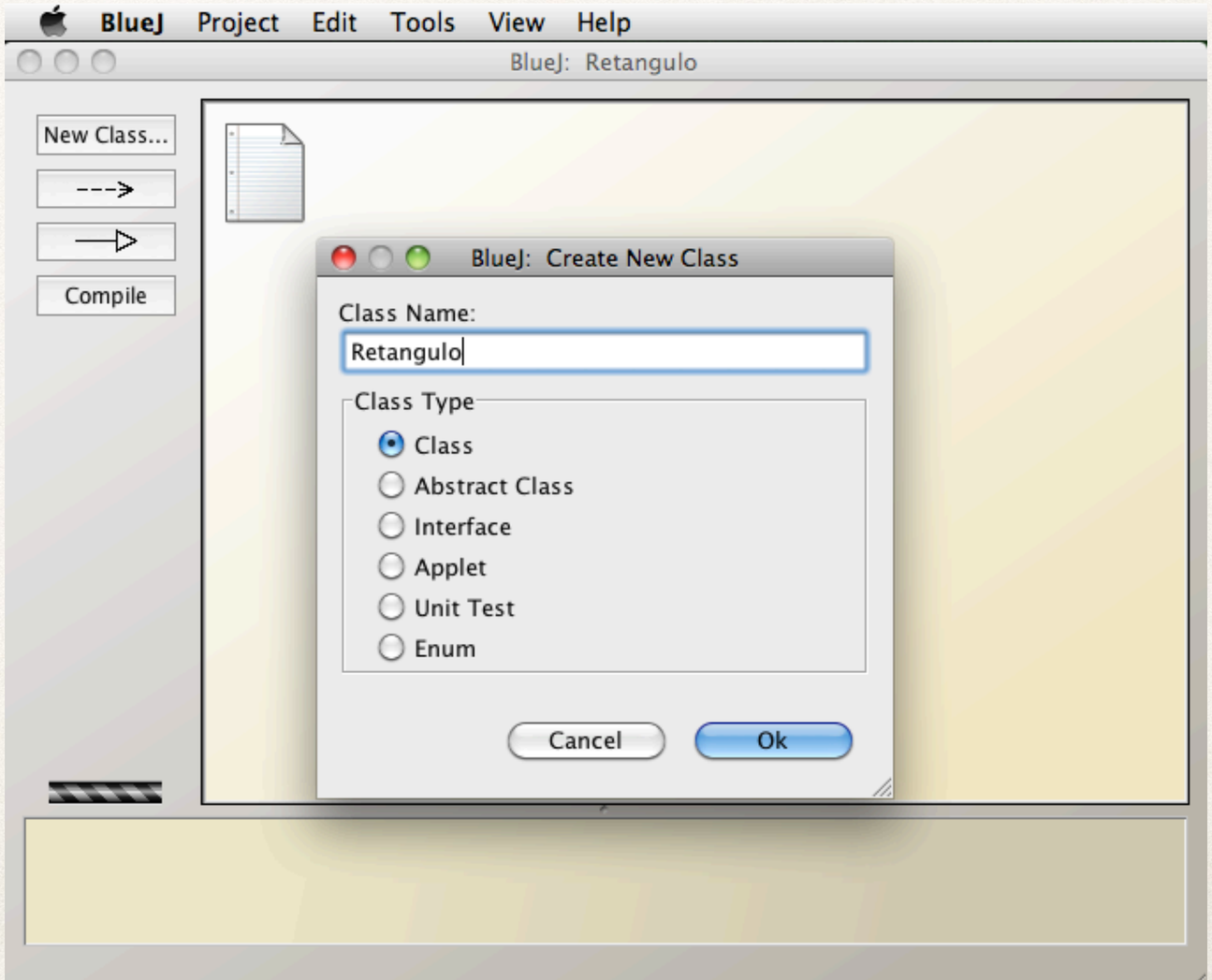
Criando Classes no BlueJ

- A seguinte tela irá aparecer…
- Clique em **New Class...** para criar uma nova classe.



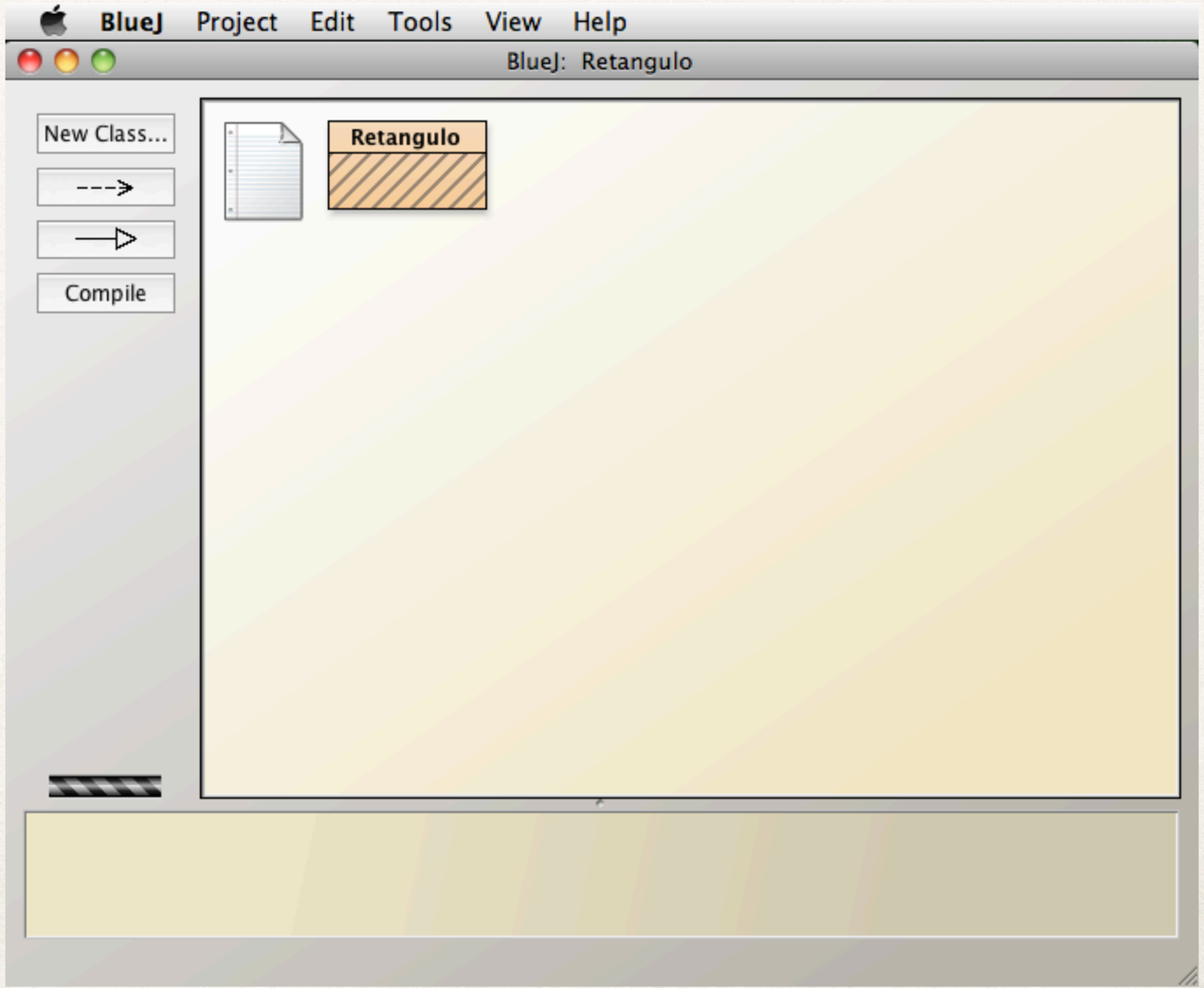
Criando Classes no BlueJ

- Dê um nome para a Classe e clique em OK.
- Neste caso chamamos a classe de Retângulo.



Criando Classes no BlueJ

- Ao clicar em Ok a sua classe irá aparecer no BlueJ.
- Observe a figura ao Lado...



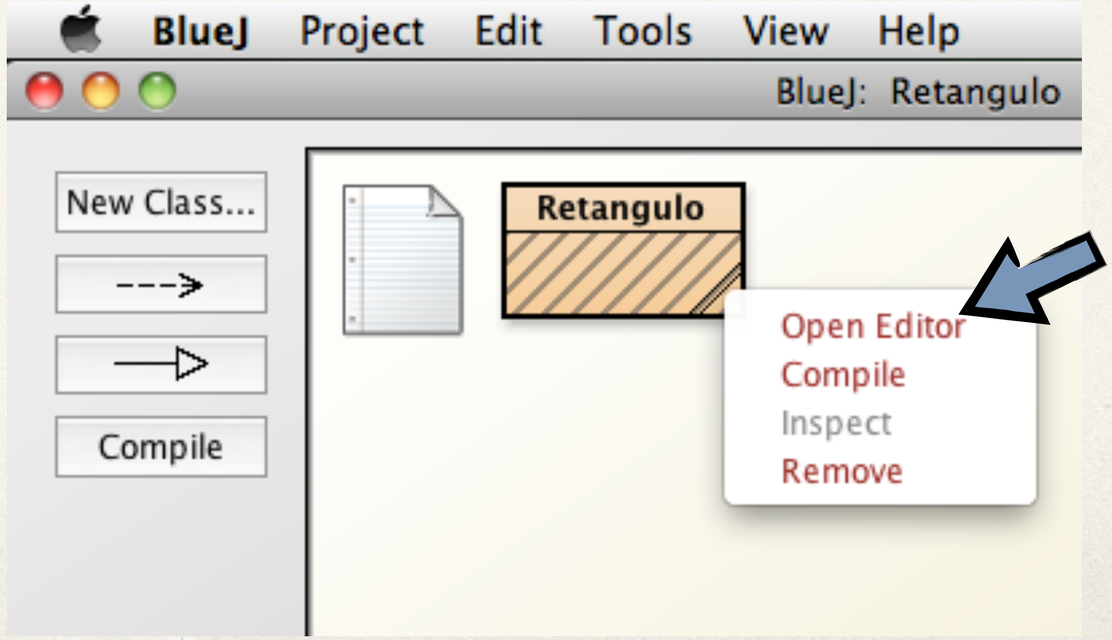
Criando Classes no BlueJ

- Quando criamos uma classe desta maneira no BlueJ, ele irá escrever uma série de código para nos auxiliar na programação da classe.
- Vamos ver então algumas características da linguagem Java e como efetivamente as classes são programadas...

Java -Programando uma Classe

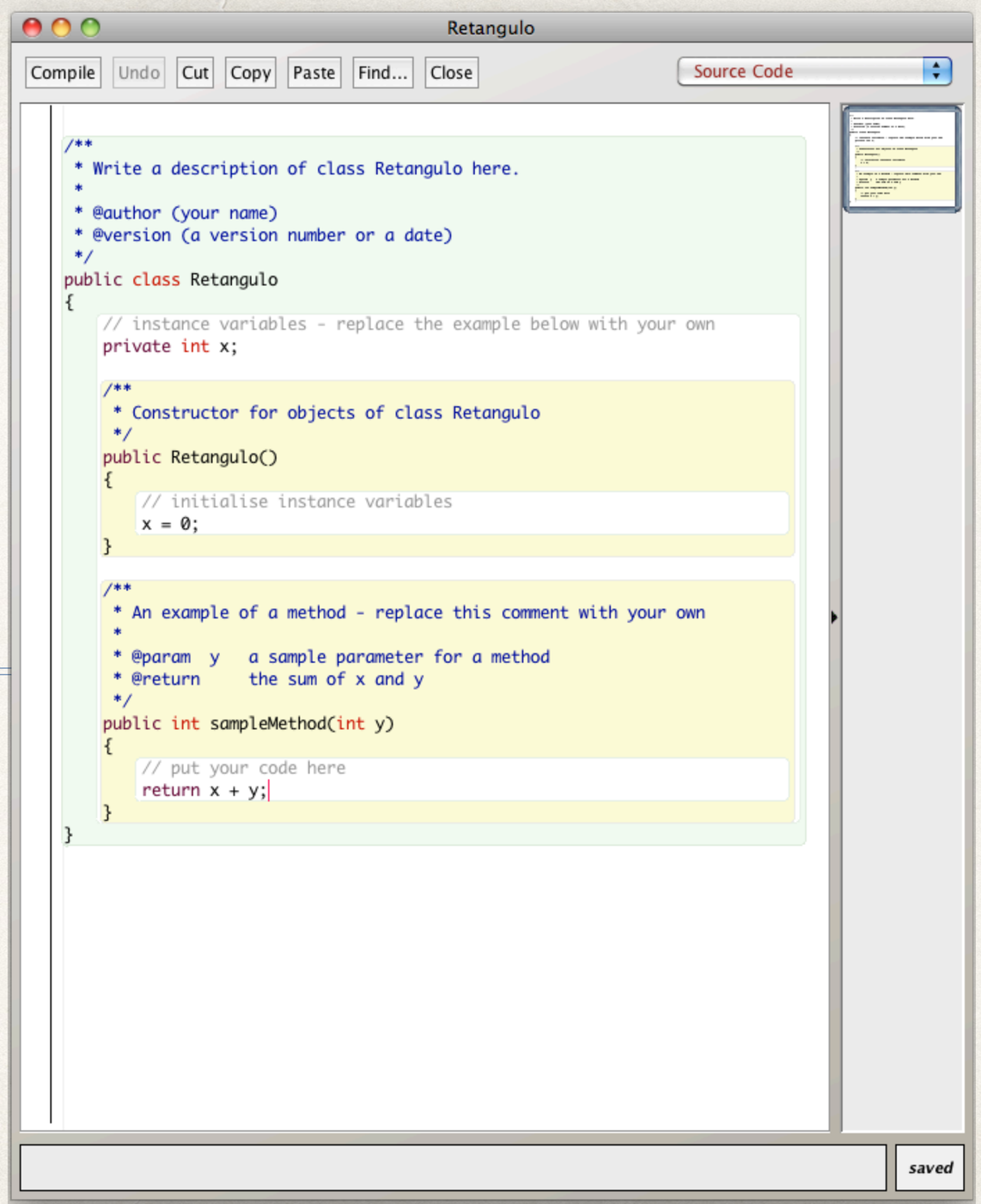
Usando o BlueJ

- Agora que criamos a classe utilizando o BlueJ precisamos programá-la, para isso clique com o botão direito na classe *Retangulo* e selecione a opção **Open Editor.**



Usando o BlueJ

Editor



Editor do BlueJ

- A princípio apague todo o código criado pelo BlueJ, exceto as linhas contendo...

public class *Retangulo*

{

}

Comentários em Java

Tipos de Comentários

- Em Java comentários podem ter três formas distintas, são elas:
  - Cometário do tipo JavaDoc
  - Comentários tipo C
  - Comentários tipo C++
- Todos estes tipos de Comentários são aceitos pelo Java, vamos ver cada um deles...

Comentário tipo Javadoc

- Abaixo exemplo de um comentário Javadoc, ele começa com /** e termina com um */
- Comentários do tipo Javadoc permitem diversas linhas...

```
/** Comentário Javadoc...
  * Aqui dentro você poderá utilizar algumass tags (espécie de comando)
  * Estas tags podem ser processadas para gerar a documentação do seu
  * código. Esse comentários devem vir imediatamente
  * à declaração de uma classe, método ou atributo. Abaixo exemplos de
tags...
  *
  * @author Mathias Brito
  * @version 0.1
  */
```

Comentários tipo C

- Os comentários tipo C começam com /* e terminam com */
- Este tipo de comentário permite múltiplas linhas, abaixo exemplo:

```
/*
    este é um comentário do tipo C que permite diversas linhas
    utiliza sempre que possível comentários...
    eles lhe ajudarão a manter a clareza do seu código...
*/
```

Comentários tipo C++

- Comentários tipo C++ irão tornar uma linha em comentários ao adicionar // no início da linha. Exemplo:

```
//Este é um comentário do tipo C++
//para criar comentários de várias linhas
//terá que colocar as barras no começo de cada linha
```

Definindo uma Classe

Definindo a Classe

- Veremos a partir de agora como efetivamente programar a nossa classe java. Lembrem-se iremos programar uma classe que represente um Retangulo.

- Declaramos uma classe da seguinte forma em Java:

```
1  public class Retangulo {
2
3          //Aqui dentro irão as características da classe
4
5  }
```

- A partir de agora nossos código terão suas linhas numeradas, entretando elas não fazem parte do código.

Definindo a Classe

```
1  public class Retangulo {
```

- A palavra reservada na linha 1 **public...**
  - A palavra reservada **public** em java é um modificador de acesso, veremos mais detalhes sobre modificadores de acesso em Java, por agora todas as nossas classes serão do tipo **public**
  - Tanto classes como atributos e métodos deverão ter um modificador de acesso.
- A palavra reservada na linha 1 **class...**
  - A palavra reservada **class** indica ao java que a partir de agora estaremos definindo uma Classe. Logo após esta palavra sempre colocaremos o nome da classe.
- Toda a definição da classe estará entre chaves { }
- As chaves estão nas linhas 1 e 5

Definindo a Classe

- Fazendo um paralelo com a aula passada onde criamos as classes no diagrama UML, temos que:

```
1  public class Retangulo {
2
3  //Aqui dentro...
4
5  }
```

=

Definindo os atributos da Classe

Definindo Atributos

- Quando modelamos as nossas classes em UML passamos boa parte do tempo tentando verificar quais seriam os atributos das Classes.
- Agora veremos como definir os atributos de uma classe que acabamos de criar.
- Note no próximo slide que a maneira com a qual declaramos estes atributos são em geral muito parecidas com a maneira de declarar variáveis em C.
- Analise o seguinte código...

Definindo Atributos

```
1  public class Retangulo {
2
3          private float base;
4          private float altura;
5
6  }
```

- Podemos afirmar aqui que nossa classe possui dois atributos, um chamado **base** que é do tipo **float**, e outro chamado **altura** do tipo **float.**

- Por enquanto sempre usaremos a palavra **private** para os nossos atributos que é um outro modificador de acesso.

Definindo Atributos

- Se compararmos nossa classe com os diagramas UML, podemos dizer que:

```
1  public class Retangulo {
2
3          private float base;
4          private float altura;
5
6  }
```

=

Retangulo
base : float
altura : float

Definindo métodos para a Classe

Definindo Métodos

- Como sabemos os métodos são as ações que a classe pode realizar.
- Estas ações em geral são solicitadas por algum código, em algum ponto do seu programa.
- Os métodos funcionam de forma muito semelhante às funções em C, porém elas pertencem a uma classe, veremos isso melhor na aula sobre encapsulamento.

Definindo Métodos

- A primeira etapa é identificar os métodos, em geral nesta etapa já deveríamos saber quais seriam estes métodos.
- Os métodos também devem ser identificados no momento da análise.
- Suponhamos que temos 4 métodos, calcularArea, calcularPerimetro, imprimir e setBaseEAltura.
- Como ficaríamos nosso sistema?
- Analise o seguinte código...

```
1  public class Retangulo {
2
3          private float base;
4          private float altura;
5
6          public float calcularArea() {
7                  return 0;
8          }
9
10         public float calcularPerimetro() {
11                 return 0;
12         }
13
14         public void imprimir() {
15                 System.out.println("A Altura é " + base);
16         }
17
18         public void setBaseEAltura(float b, float a) {
19                 base = b;
20                 altura = a;
21         }
22
23 }
```

O que temos de novo está entre as linhas 6 e 22

Definindo os métodos

- Temos 4 métodos, os seus cabeçalhos são:
- `public float calcularArea()`
- `public float calcularPerimetro()`
- `public void imprimir()`
- `public void setBaseEAltura(float b, float a)`

Definindo os métodos

- Se fizermos a comparação da nossa Classe implementada com um diagrama de Classes que o represente, teremos...

```
1  public class Retangulo {
2
3          private float base;
4          private float altura;
5
6          public float calcularArea() {
7                  return 0;
8          }
9
10         public float calcularPerimetro() {
11                 return 0;
12         }
13
14         public void imprimir() {
15                 System.out.println("A Altura é " + base);
16         }
17
18         public void setBaseEAltura(float b, float a) {
19                 base = b;
20                 altura = a;
21         }
22
23 }
```

=

Retangulo
base **: float** altura **: float**
calcularArea( ) **: float** calcularPerimetro( ) **: float** imprimir( ) **: void** setBaseEAltura(b **: float**, a **: float**) : **void**

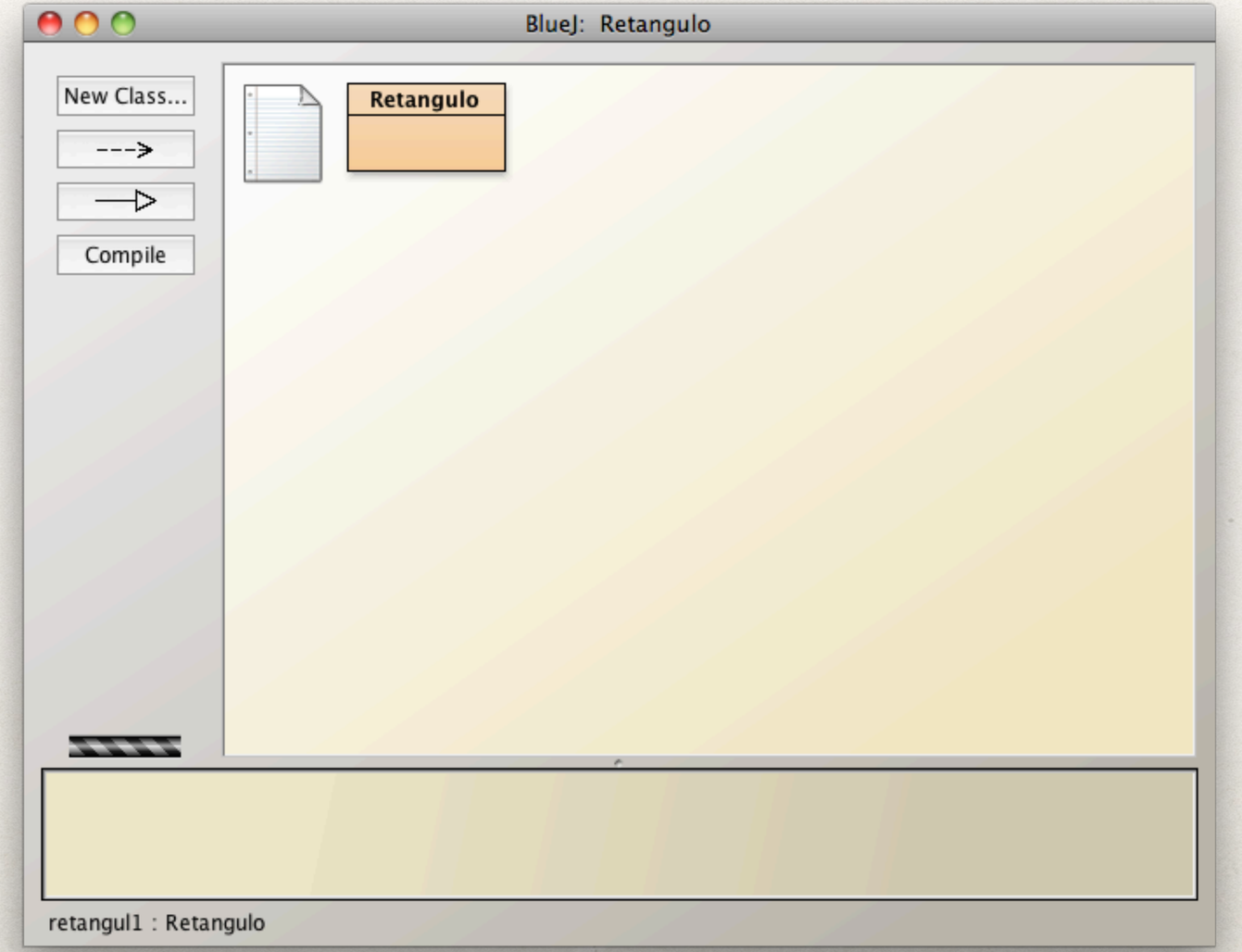
Definindo os métodos

- Todo método também terá um modificador de acesso.
- Este modificador por enquanto sempre será **public** para todos os métodos que criarmos. Note que usaremos sempre **private** para os atributos.
- Note também que para imprimir chamamos o método **println**, pertencente ao objeto **out** que está dentro da classe **System.**
- `System.out.println("A Altura é " + base);`
- Para imprimirmos um atributo, no exemplo **base**, basta fecharmos aspas colocar um + e escrever o nome do atributo.
- Se quiséssemos continuar escrevendo após **base** colocaríamos outro + e abriríamos outra **“**, escreveríamos o que quiséssemos e fecharíamos as aspas.
- `System.out.println("A Altura é " + base + " alguma coisa");`

Criando Objetos

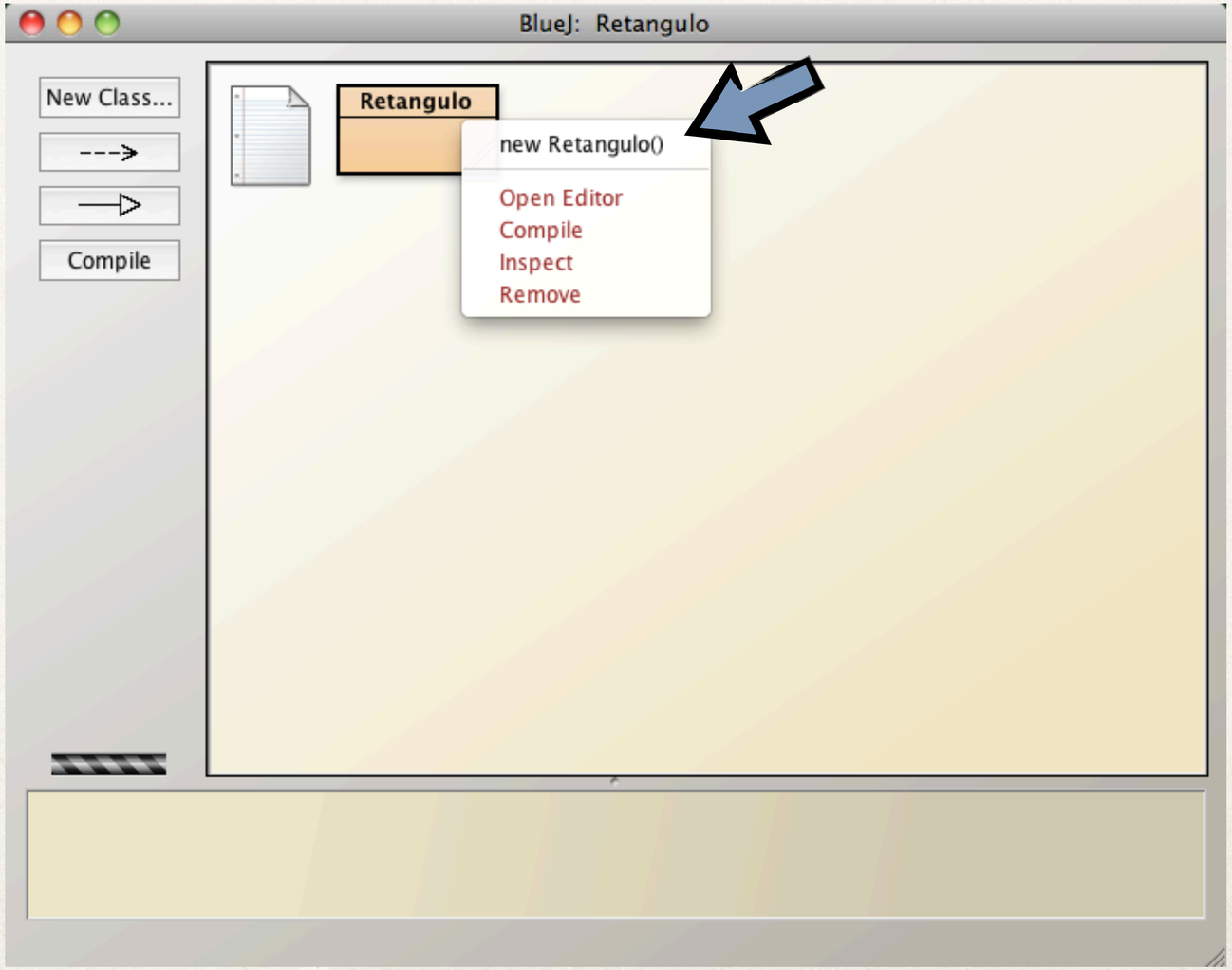
- Após termos digitado o código anterior no BlueJ devemos clicar em compilar e depois fechar a janela de edição.
- Seu código é salvo automaticamente antes de ser compilado.
- Ao retornar à tela principal do BlueJ, teremos o seguinte:

Criando Objetos



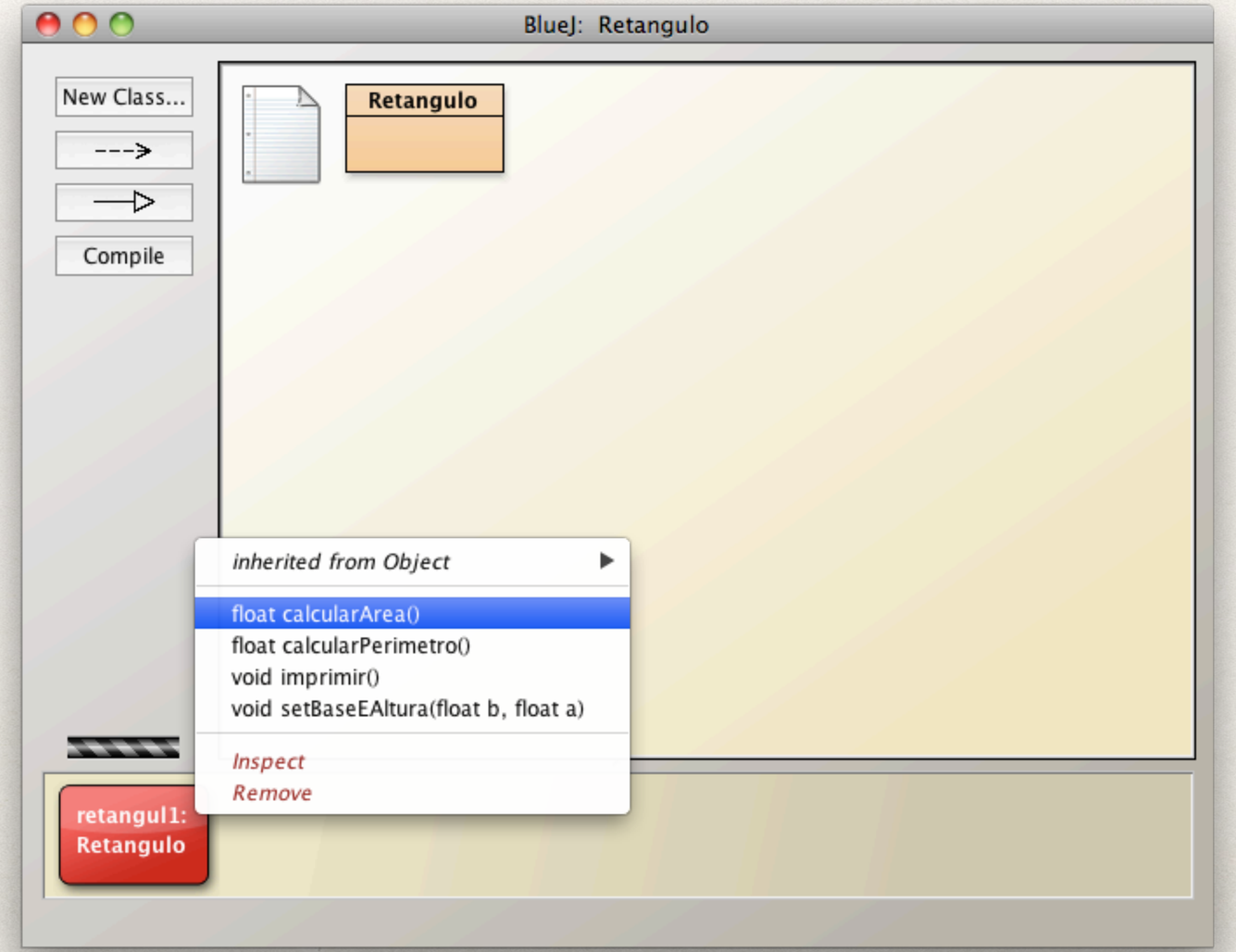
Criando Objetos

- Clique com o botão direito em Retangulo.
- Selecione a opção new Retangulo( ), para criar um novo objeto do tipo Retangulo.
- Selecione qualquer nome para o objeto clique Ok.



Criando Objetos

- O Objeto será criado e mostrado logo abaixo na Janela.
- Para executar os métodos simplesmente clique no método desejado.



Exercício

1) Agora que sabemos como implementar e manipular o objeto, termine a classe *Retangulo*. Implemente os métodos *calcularArea* e *calcularPerimetro* e faça com que a função imprimir além de imprimir base e altura também imprima os valores da Área e Perímetro chamando os métodos responsáveis.

   a) Perímetro do Retângulo = 2*(base + altura)

   b) Área do Retângulo = base * altura

2) Crie um novo projeto no BlueJ e tente implementar as classes que você identificou para o problema do Estoque visto na aula anterior.

Treine para a prova de quinta que vêm 09/09/2010