

# Noções Básicas de Java

2-1

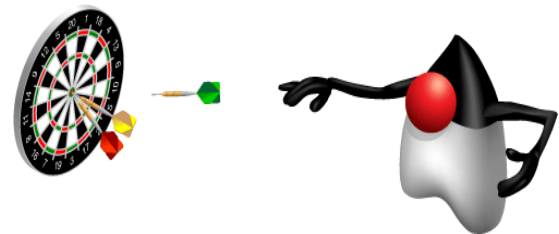
O Processo de Desenvolvimento do Software



# Objetivos

Esta lição abrange os seguintes objetivos:

- Entender o Modelo Espiral de desenvolvimento
- Reconhecer tarefas e subtarefas do Modelo Espiral
- Reconhecer o que acontece quando etapas são ignoradas
- Identificar recursos do software
- Entender como recursos são gradualmente implementados

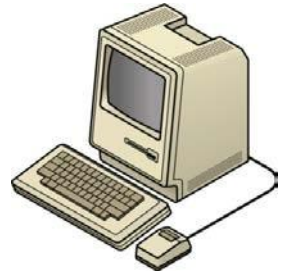


# Tópicos

- Apresentando o Modelo Espiral de Desenvolvimento
- Esquecendo Etapas no Modelo Espiral
- Avaliando o Software à medida que Ele é Desenvolvido



# Exercício 1, Parte 1



Seu amigo, Clinton, tem planos para o fim de semana. Leia o e-mail que ele enviou e considere quais etapas são necessárias para fazer com que esses planos aconteçam:

*Olá, amigo,*

*Vai ter uma apresentação especial da História do Computador no Museu da Cidade este mês. Estamos pensando em ir na sexta-feira às 17h. Você gostaria de ir conosco? Acho que o metrô seria a melhor opção para chegar lá.*

*Clinton*



# Exercício 1, Parte 2

Complete o gráfico criando pelo menos um item para cada seção.

## Requisitos

- O que o e-mail de Clinton está perguntando?

## Projetando um Plano

- O que você precisa considerar antes de sair?

## Testando

- Como você sabe se o plano funcionou?

## Implementando o Plano

- Quais ações você toma?

# Sexta-feira no Museu



Você deve ter escrito algo semelhante ao seguinte:

## Requisitos

- O que o e-mail de Clinton está perguntando?
  - **Estar no Museu da Cidade às 17h na sexta-feira.**

## Projetando um Plano

- O que você precisa considerar antes de sair?
  - **Marcar um horário de encontro na estação de metrô do campus antes das 17h.**
  - **Consultar os mapas do metrô e das ruas.**

## Testando

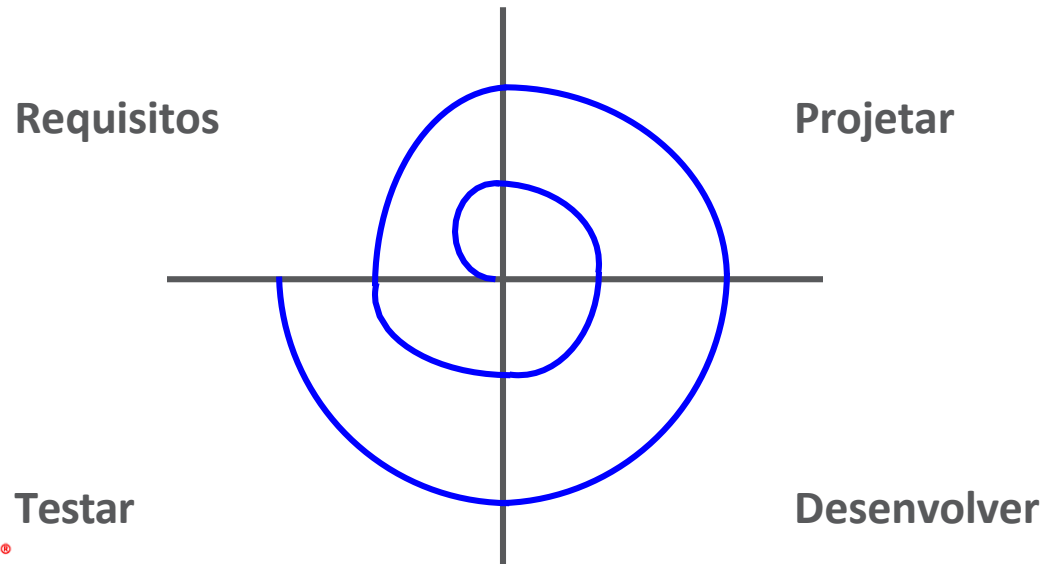
- Como você sabe se o plano funcionou?
  - **Você desceu na estação correta?**
  - **Os nomes das ruas e dos prédios eram os que você esperava?**
  - **Você viu alguns computadores?**

## Implementando o Plano

- Quais ações você toma?
  - **Pegue o metrô da linha vermelha em direção à South Station.**
  - **Ande três quarteirões para leste.**

# Apresentando o Modelo Espiral de Desenvolvimento

- O desenvolvimento de um software requer um processo de reflexão semelhante.
- Isso é representado pelo **Modelo Espiral**.
- Existem outros modelos, mas o Modelo Espiral é o que melhor reflete o que você estará fazendo neste curso.

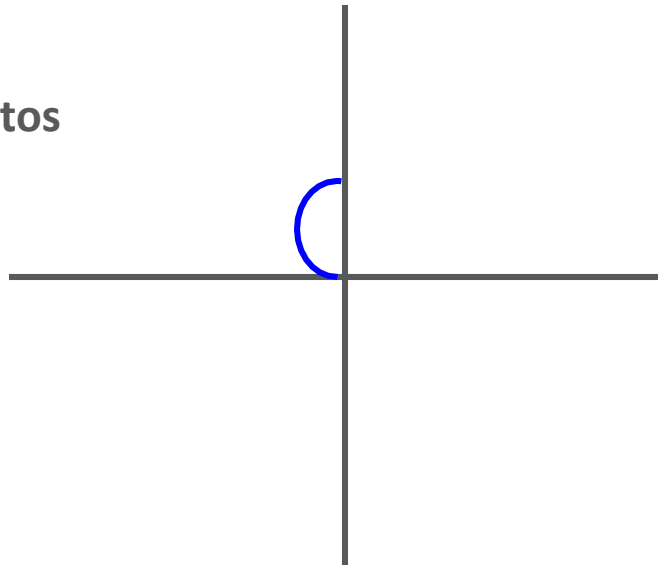


# Requisitos

Leia todas as instruções atentamente:

- O que seu programa deve fazer?
- Que problemas ele está tentando resolver?
- Que recursos seu programa deve ter?

Requisitos

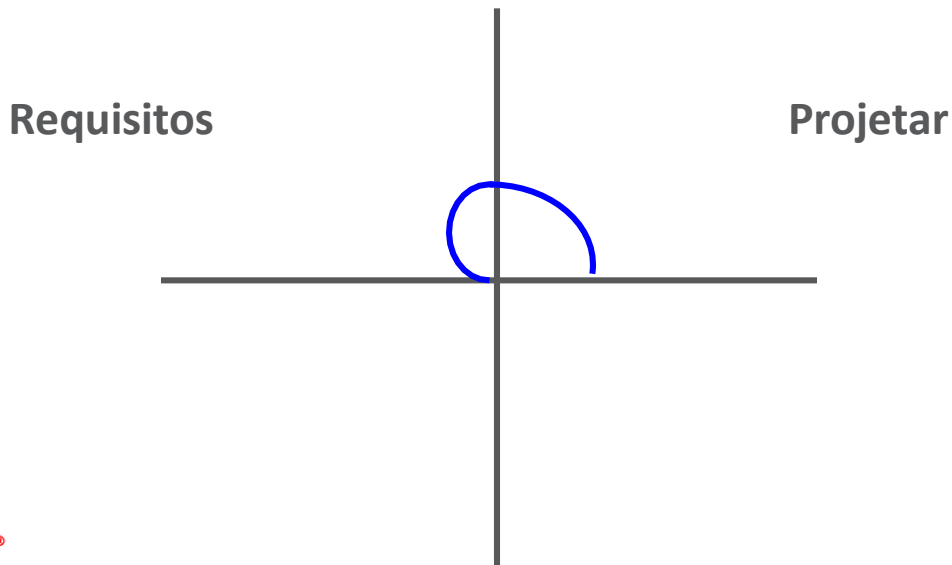




# Projetar

Planeje sua abordagem:

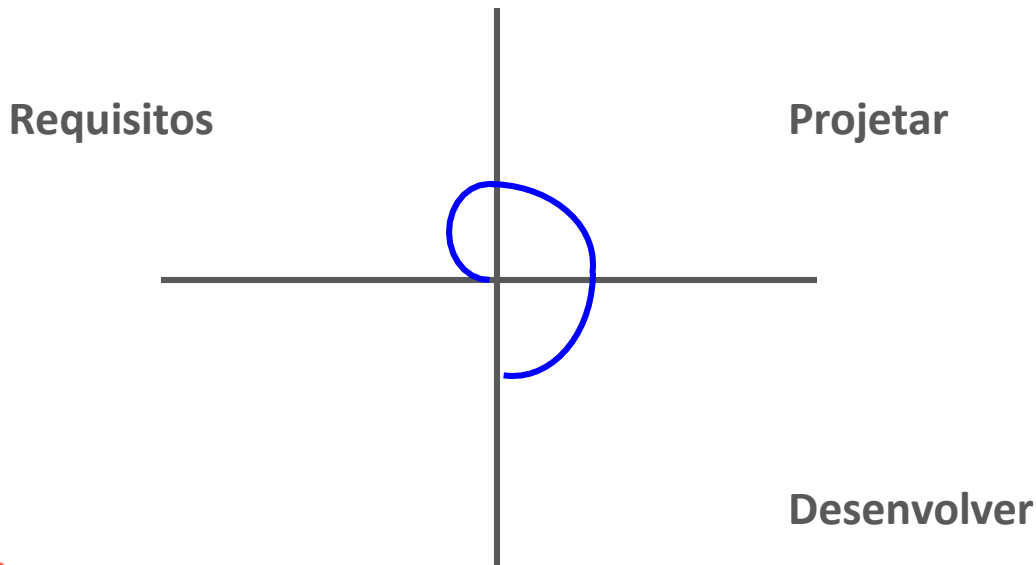
- Seu programa precisa modelar dados ou comportamentos?
- Partes específicas do programa precisam ser concluídas antes de você poder iniciar outras partes?



# Desenvolver

Inicie a codificação:

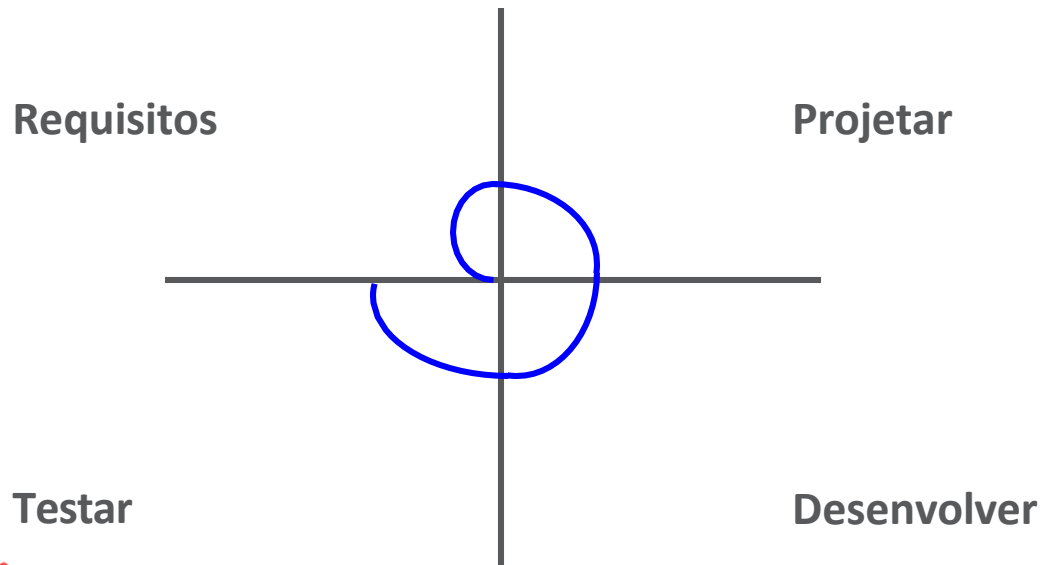
- Crie uma versão simplificada do seu programa.
- Foque em um número pequeno de recursos simples ou importantes.



# Testar

Teste seu código:

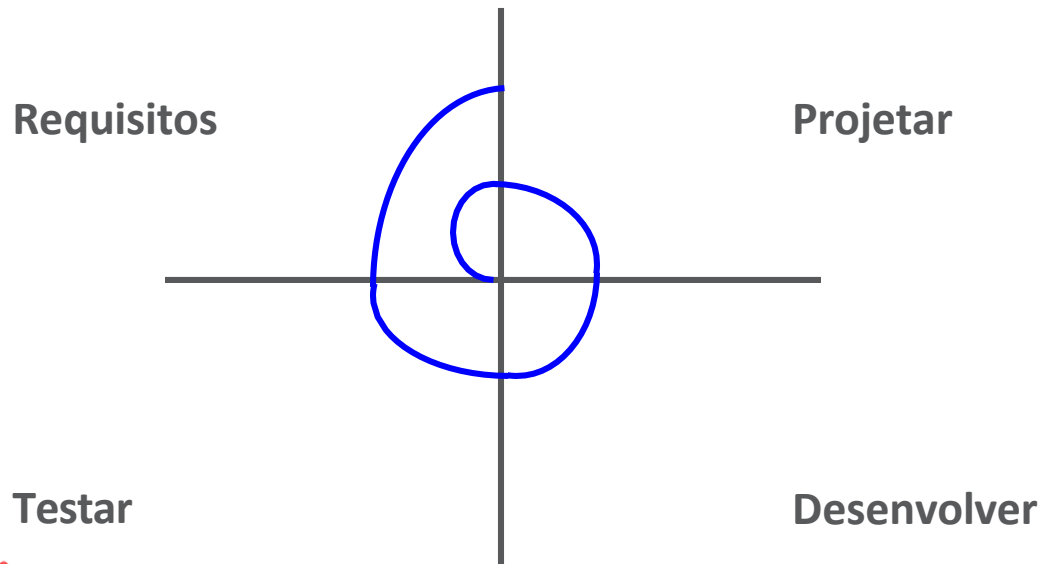
- O programa fornece os resultados esperados?
- Algum cenário produz resultados indesejados?
- Dependendo do impacto, esses bugs podem precisar ser corrigidos.



# Iteração de Requisitos

Verifique os requisitos novamente:

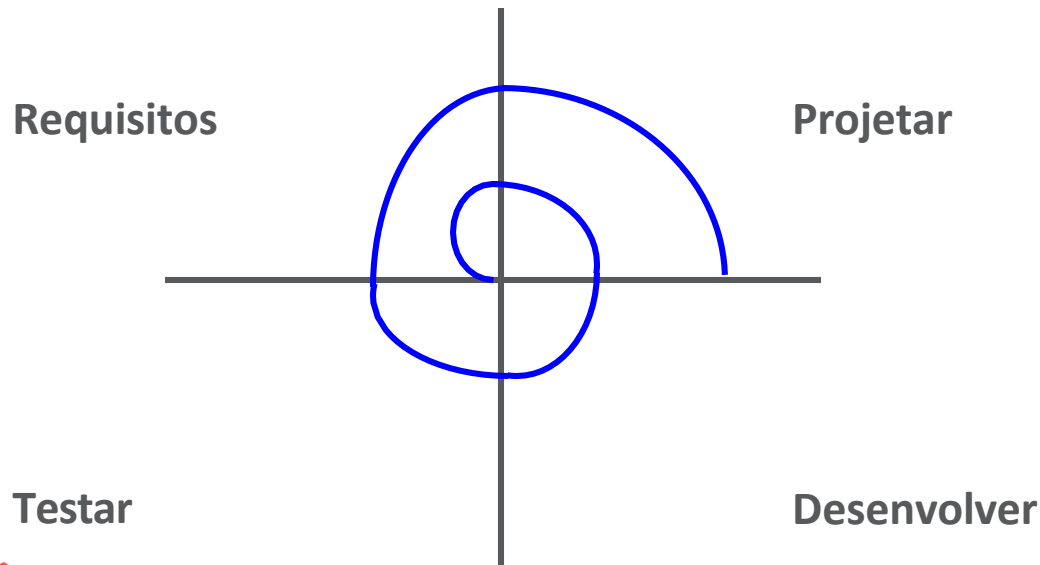
- O comportamento do programa corresponde aos requisitos?
- Existem requisitos ou recursos adicionais para serem construídos?
- Alguns requisitos precisam ser alterados?



# Iteração do Design

Planeje suas alterações:

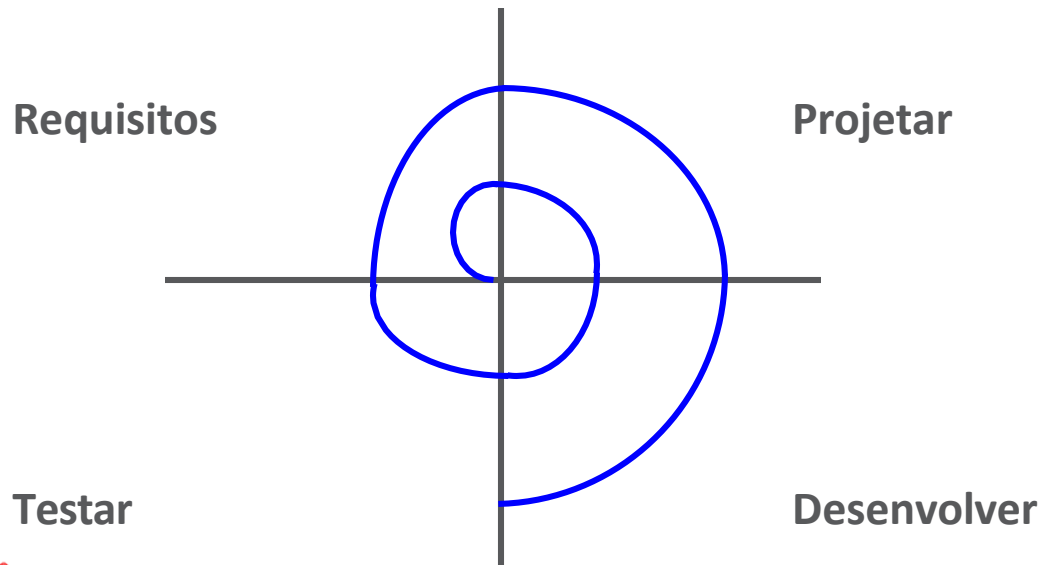
- Como você deve modelar recursos adicionais?
- É necessário alterar o design atual para suportar melhor a expansão dos recursos atuais ou para adicionar novos recursos?



# Iteração do Desenvolvimento

Continue desenvolvendo:

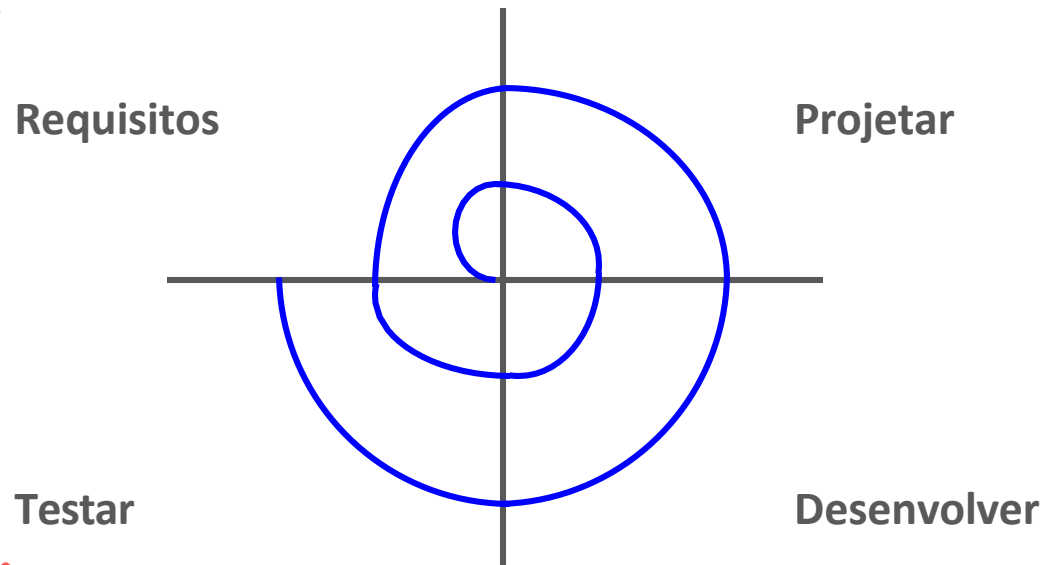
- Adicione novos recursos.
- Se necessário, modifique ou aprimore os recursos existentes.



# Mais Testes

Continue testando:

- O novo código funciona como você esperava?
- O código antigo continuará a funcionar corretamente?
- Dependendo da severidade, os bugs precisam ser corrigidos



# Desenvolvendo, Testando e Corrigindo

O processo de desenvolvimento, teste e correção de bugs às vezes é frustrante:

- Normalmente, o código não funciona.
- Bugs inesperados aparecem.
- As soluções parecem difíceis e evasivas.





# Programar é como Montar Quebra-cabeças

- Pode levar algum tempo...
  - Pensando
  - Experimentando
  - Pesquisando e repetindo
- Mas é muito gratificante...
  - Ver seu código finalmente funcionando (ou se comportando um pouco melhor).
  - Ver seu programa evoluir e tornar-se mais robusto.
  - Perceber que você está mais habilidoso.
  - Encontrar maneiras divertidas de produzir bugs.



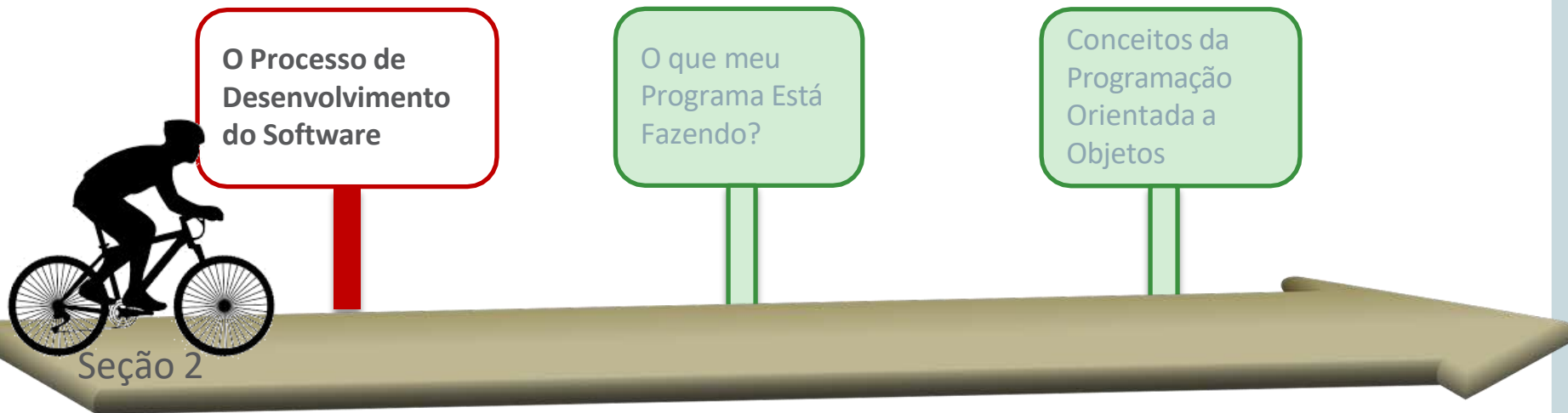
# Como Pesquisar

Você continua confuso depois de tantas correções? Existem muitos recursos para ajudar você a progredir:

- Anotações das aulas e exercícios rápidos já feitos
  - Eles usam os comandos e as técnicas que você está procurando?
- Documentação do Oracle Java
  - Descreve os comandos Java disponíveis.
  - <http://docs.oracle.com/javase/8/docs/api/index.html>
- Internet
  - Pode ser que outras pessoas tenham feito perguntas semelhantes às suas.
  - Você pode descobrir exemplos úteis ou novos comandos.
  - Mas suas soluções devem ser de sua autoria, e não um código copiado.

# Tópicos

- Apresentando o Modelo Espiral de Desenvolvimento
- Esquecendo Etapas no Modelo Espiral
- Avaliando o Software à medida que Ele é Desenvolvido



## Exercício 2, Parte 1

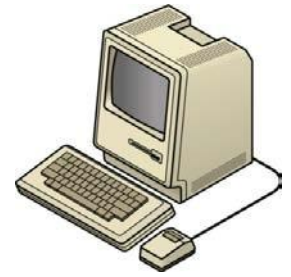


Veja a seguir novamente o e-mail que Clinton enviou, caso você precise consultá-lo neste exercício.

*Olá, amigo,*

*Vai ter uma apresentação especial da História do Computador no Museu da Cidade este mês. Estamos pensando em ir na sexta-feira às 17 h. Você gostaria de ir conosco? Acho que o metrô seria a melhor opção para chegar lá.*

*Clinton*



## Exercício 2, Parte 2

Complete este gráfico. Imagine o que aconteceria com a sua noite no museu se uma etapa específica fosse esquecida:

**Requisitos**

**Projetando um Plano**

**Testando**

**Implementando o Plano**

# Sexta-feira Esquecida



Você deve ter escrito algo semelhante ao seguinte:

## Requisitos

- Você já tem compromisso na sexta-feira.

## Projetando um Plano

- Todo mundo está no metrô, mas ninguém sabe para onde está indo.
- Você fica horas andando de metrô, mas nunca chega ao museu.

## Testando

- Você passa pelo museu.
- Você chega no edifício errado.
- O museu está fechado.

## Implementando o Plano

- Apesar de ser um plano incrível, ninguém vai ao museu.
- Clinton está chateado.

**ORACLE®**

Academy

# Esquecendo Etapas no Modelo Espiral

Da mesma forma, coisas ruins podem acontecer quando determinada etapa do Modelo Espiral é esquecida.

## Requisitos

- O programa funciona, mas não resolve o problema correto.
- Estão faltando recursos.

## Projetar

- O código está desorganizado.
- Os bugs são difíceis de serem corrigidos.
- Os recursos são difíceis de serem aprimorados.

## Teste

- O programa continua travando.
- O programa fornece resultados incorretos.
- Os usuários ficam frustrados.
- Os usuários não param de rir.

## Desenvolvimento

- Não há um programa.

# Tópicos

- Apresentando o Modelo Espiral de Desenvolvimento
- Esquecendo Etapas no Modelo Espiral
- Avaliando o Software à medida que Ele é Desenvolvido





# O que É um Recurso de Software?

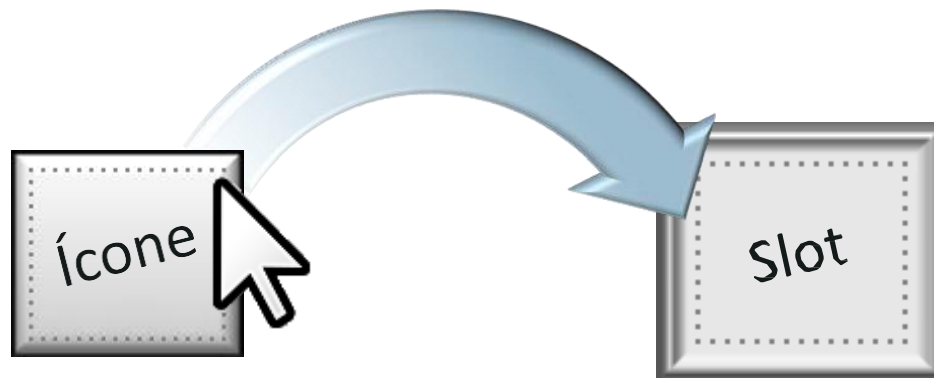
- Pense em um **recurso** como:
  - Algo que um programa pode fazer
  - Algo que você pode fazer com um programa
- Exemplos:
  - Imprimir um texto
  - Reproduzir um som
  - Calcular um valor
  - Arrastar e soltar um ícone
  - Dar uma pontuação alta em uma classificação semanal on-line
  - Um novo tipo de inimigo em um videogame

**ROAR! Sou um inimigo!  
Vou te morder!**



# Implementando um Recurso

- Alguns recursos são mais fáceis de serem implementados:
  - Você pode codificá-los em algumas linhas simples.
  - Por exemplo, imprimir um texto na janela de saída do NetBeans.
- Alguns recursos são difíceis de serem implementados.
  - Eles baseiam-se em uma combinação de outros recursos.
  - Por exemplo, ser capaz de "arrastar e soltar" um ícone.



# Implementando o Recurso "Arrastar e Soltar"

- Um recurso "arrastar e soltar" requer vários recursos menores:
  - Adicionar um gráfico à tela
  - Encontrar a posição do mouse
  - Detectar um clique no botão do mouse
  - Detectar a liberação do botão do mouse
  - Alterar a posição do gráfico
- A implementação de apenas um desses itens pode parecer uma grande conquista.

# Noções Básicas de Java

2-2

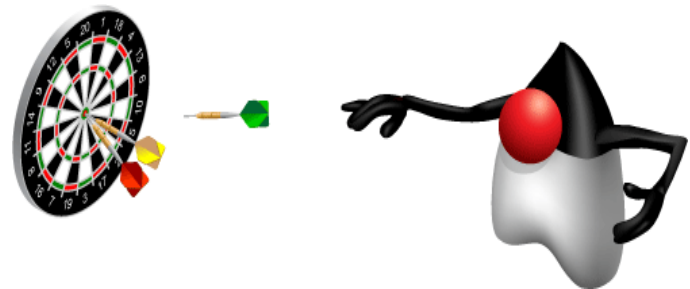
O que meu Programa Está Fazendo?



# Objetivos

Esta lição abrange os seguintes objetivos:

- Entender como o Java é lido linha por linha
- Configurar e usar pontos de interrupção
- Terminar instruções com pontos e vírgulas (;)
- Organizar o código usando espaço em branco e outras convenções
- Criar comentários



# Tópicos

- Pontos de interrupção
- Espaço em branco e {Chaves}
- Comentários
- O Método Main



# Lendo um Programa Linha por Linha

- Cada linha em um programa é lida uma vez.

```
1 System.out.println("Linha 1");  
2 System.out.println("Linha 2");  
3 System.out.println("Linha 3");  
4 System.out.println("Linha 4");  
5 System.out.println("Linha 5");
```

- Neste exemplo...
  - A linha 1 é lida...
  - Depois a linha 2...
  - Depois a linha 3...
  - Depois a linha 4...
  - Depois a linha 5...

# Lendo Linha por Linha

- O Java é lido principalmente linha por linha.
- Mas existem alguns pontos adicionais a serem considerados.
- Vamos investigar usando...
  - Um ponto de interrupção
  - Outros recursos do NetBeans





# Pontos de interrupção

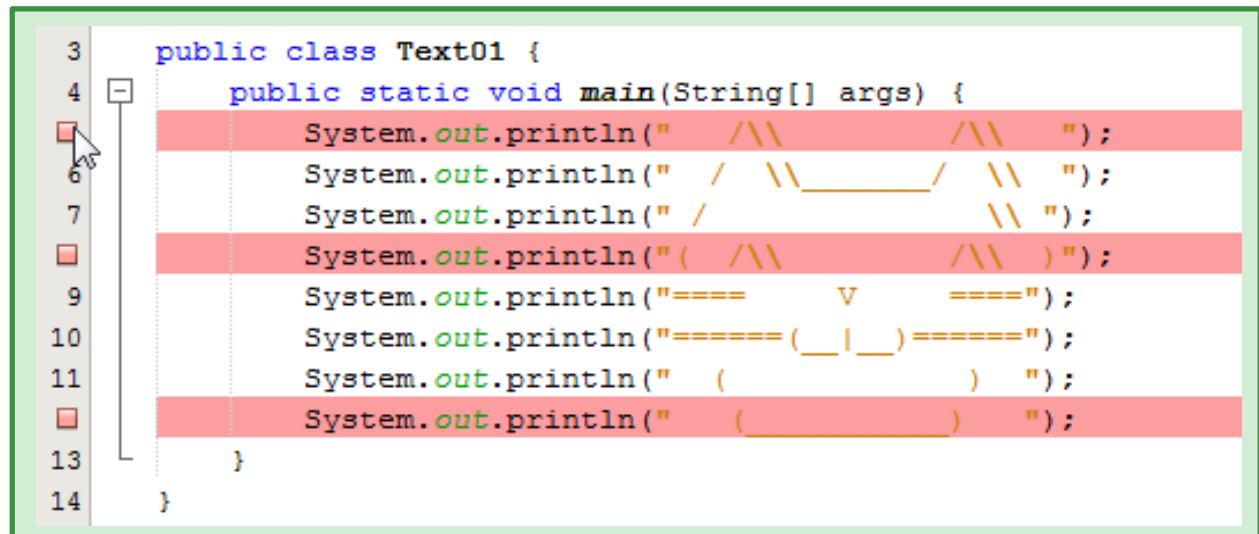
- Definir um ponto de interrupção no seu código para
  - Pausar a execução do código.
  - Verificar o estado atual do programa.
  - Ajudar na depuração.
- Os pontos de interrupção afetam a execução do código...
  - Quando o código é executado com o depurador.
- Os pontos de interrupção não podem afetar a execução do código...
  - Quando o código é executado normalmente.



# Definindo a Animação de um Ponto de Interrupção

Para definir um ponto de interrupção...

- Posicione o cursor sobre um número na margem esquerda.
- Clique em ... e você tem um ponto de interrupção!
- Clique novamente para remover um ponto de interrupção.
- Você pode definir vários pontos de interrupção.



```
3 public class Text01 {  
4     public static void main(String[] args) {  
5         System.out.println("  /\\"  
6         System.out.println(" /  \\"  
7         System.out.println(" /  
8         System.out.println("(  /\\"  
9         System.out.println("====  V  =====");  
10        System.out.println("===== ( _ | _ ) =====");  
11        System.out.println(" (                ) ");  
12        System.out.println(" (                ) ");  
13    }  
14 }
```

# Exercício 1, Parte 1



- Importe e abra o projeto Text01.
- Defina um ponto de interrupção na Linha 5 (a linha com a primeira instrução de impressão).
- Execute o programa normalmente.
  - Os pontos de interrupção não devem interferir no programa.



# Tópicos

- Pontos de interrupção
- Espaço em branco e {Chaves}
- Comentários
- O Método Main

O Processo de  
Desenvolvimento  
do Software

O que meu  
Programa Está  
Fazendo?

Conceitos da  
Programação  
Orientada a  
Objetos

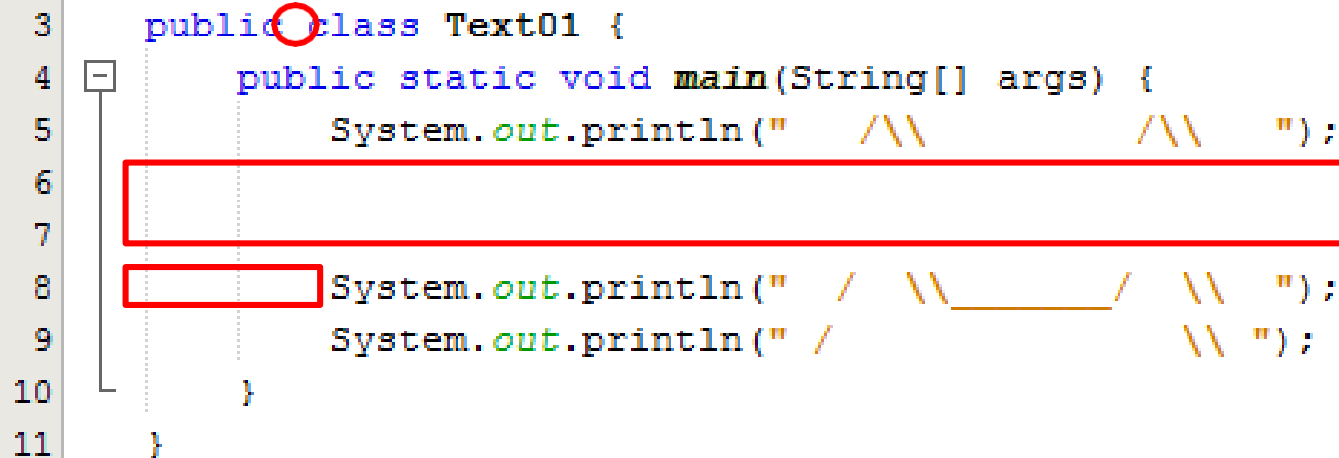
Seção 2



# Espaço em branco

O espaço em branco é qualquer espaço sem código:

- Espaço entre palavras
- Linhas em branco
- Recuo antes de uma linha de código



The screenshot shows a Java code editor with the following code:

```
3 public class Text01 {  
4     public static void main(String[] args) {  
5         System.out.println("  /\\"  
6           
7           
8         System.out.println(" /  \\"  
9         System.out.println(" /  \\"  
10    }  
11 }
```

Annotations in the image:

- A red circle highlights the space between `public` and `class` on line 3.
- A red rectangle highlights the empty line between lines 6 and 7.
- A red rectangle highlights the indentation (spaces) at the start of line 8.

# Efeitos do Espaço em Branco

- O espaço em branco ajuda a manter o código organizado.
- O espaço em branco não afeta o modo como o código é executado.
- Você pode usar o espaço em branco da maneira que preferir.
- Mas o recuo apropriado é **bastante** recomendado porque ele...
  - Facilita a leitura
  - Evita erros durante a programação

Ahh! Código desorganizado!



# Recuo e Chaves

- Insira um recuo de mais uma tabulação (4 espaços) após uma chave de abertura ( { ).
- Pare o recuo de mais uma tabulação (4 espaços) antes de uma chave de fechamento ( } ).
- O código dentro de chaves denomina-se bloco de código.
  - Quando adicionar uma chave de abertura ( { ) ...
  - Você acabará precisando de uma chave de fechamento ( } ).
  - Um erro comum é incluir uma chave sem correspondência ou esquecer de incluir uma chave.

# Exemplo de Animação em Bloco

```
public class Example
{
    public static void main(String[] args){
        System.out.println("Inner code");
        System.out.println("Inner code");
        {
            System.out.println("Inner-inner code");
        }
    }
}
```



# Ajuda de Recuo do IDE

- Um IDE pode...
  - Codificar com cores o escopo de um bloco (Greenfoot, BlueJ)
  - Definir automaticamente um recuo logo após uma chave
  - Realçar uma chave correspondente (como mostrado abaixo)
- Alguns comandos Java requerem chaves, embora você possa sempre adicionar mais.

```
public class Example
{
    public static void main(String[] args){
        System.out.println("Inner code");
        System.out.println("Inner code");
        {
            System.out.println("Inner-inner code");
        }
    }
}
```

# Tópicos

- Pontos de interrupção
- Espaço em branco e {Chaves}
- Comentários
- O Método Main

O Processo de  
Desenvolvimento  
do Software

O que meu  
Programa Está  
Fazendo?

Conceitos da  
Programação  
Orientada a  
Objetos

Seção 2



# Comentários

- Os programas com espaços bem organizados podem ficar muito grandes e tornarem-se difíceis de ler.
- Você pode adicionar comentários ao código para...
  - Fornecer uma explicação ou informações adicionais ao programador (**Código de comentário**)
  - Desativar o código e impedir que ele seja executado sem apagá-lo (**Código marcado para ser ignorado**)

Ahh! O que todo esse código está fazendo?



# Adicionando Comentários ao Código

- Comentários em uma única linha...
  - Começar com duas barras `//`
  - Terminar quando a linha termina
- Comentários em várias linhas...
  - Começar com uma barra e um asterisco `/*`
  - Terminar com um asterisco e uma barra `*/`

```
//Um comentário em uma única linha termina automaticamente quando a
linha termina
System.out.println("Esta linha é impressa");

/*Um comentário em várias linhas...
continua por muitas linhas...
System.out.println("Esta linha não é impressa");
até aparecer um asterisco com uma barra*/
System.out.println("Esta linha é impressa");
```

# Lendo Linha por Linha

- Podemos analisar um pouco mais o código.
- Vamos investigar usando...
  - Blocos de código
  - Comentários
  - Pontos de interrupção
  - Outros recursos do NetBeans



# O Fluxo do Programa

1. Todos os programas Java começam no método main.
2. Nenhum outro código é executado a menos que seja chamado.

2) E depois passe para cá

1) Comece aqui.

```
public class Text03 {  
    public static void drawLegs() {  
        System.out.println("    ||    ||    ");  
        System.out.println("    ||    ||    ");  
        System.out.println("    (||)  (||)  ");  
    }  
  
    public static void main(String[] args) {  
        System.out.println("    /\\"      /\\"    ");  
        System.out.println("  /  \\"      /\\"    ");  
        System.out.println(" /      \\"      );  
        System.out.println("(  /\\"      /\\"  )");  
        System.out.println("====      V      ===");  
        System.out.println("===== ( _ | _ ) =====");  
        System.out.println(" (              ) ");  
        System.out.println(" (              ) ");  
        drawLegs();  
    }  
}
```

# Tópicos

- Pontos de interrupção
- Espaço em branco e {Chaves}
- Comentários
- O Método Main

O Processo de  
Desenvolvimento  
do Software

O que meu  
Programa Está  
Fazendo?

Conceitos da  
Programação  
Orientada a  
Objetos

Seção 2



# O Método Main

- O método main é um bloco de código especial.
- Todos os programas Java começam no método main.
- Seus programas só devem ter um método main.
- Os métodos serão analisados mais detalhadamente na próxima lição.
  - drawLegs() é um exemplo de método.

```
public static void main(String[] args) {  
    //Seu programa começa aqui.  
}
```



# Resumo

- Erros comuns:

- Ponto e vírgula ausente (;)

```
System.out.println("Miau")
```

- {Chaves} sem correspondência

```
{  
    System.out.println("Miau");  
}
```

- Mantenha o código organizado usando:

- Espaço em branco
- Chaves ( { } )
- Comentários

# Noções Básicas de Java

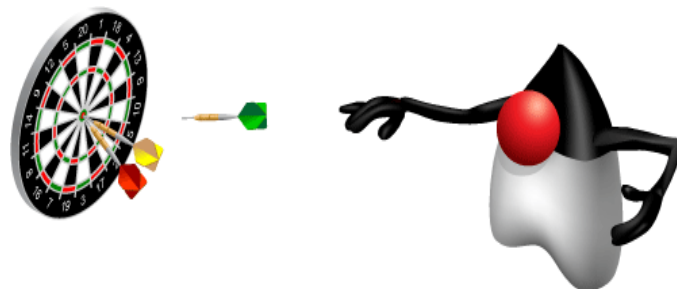
## 2-3 Introdução aos Conceitos da Programação Orientada a Objetos



# Objetivos

Esta lição abrange os seguintes objetivos:

- Fazer a distinção entre programação procedural e programação orientada a objetos
- Entender uma **classe** como um plano gráfico de um **objeto**
- Entender como uma classe é usada para criar **instâncias** de objeto
- Modelar objetos como uma combinação de...
  - **Propriedades** (campos de dados)
  - **Comportamentos** (métodos)



# Tópicos

- Linguagens Orientadas a Objetos x Linguagens Procedurais
- Classes, Instâncias, Propriedades e Comportamentos
- Convertendo para uma Sintaxe Java

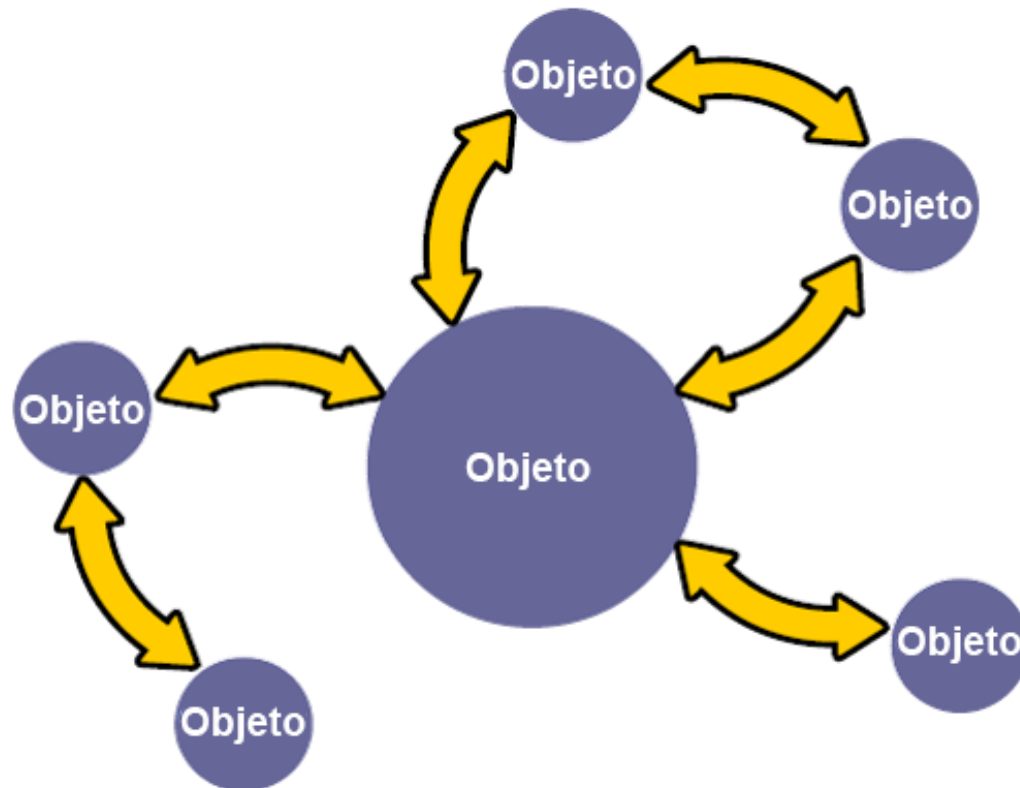


# O Java Pode Fazer Mais!

- Linguagens Procedurais...
  - Leem uma linha por vez.
  - A linguagem **C** é procedural.
- Linguagens orientadas a objetos...
  - Leem uma linha por vez.
  - Modelam objetos por meio do código.
  - Enfatizam a interação do objeto.
  - Permitem uma interação sem uma ordem prescrita.
  - **Java** e **C++** são linguagens orientadas a objetos.

# Programação Orientada a Objetos

- Interação de objetos
- Nenhuma sequência prescrita



# Tópicos

- Linguagens Orientadas a Objetos x Linguagens Procedurais
- Classes, Instâncias, Propriedades e Comportamentos
- Convertendo para uma Sintaxe Java



Seção 2

# Exemplo

- **Propriedades:**

- Nome
- Idade
- Raça
- Comida Favorita



- **Comportamentos:**

- Miar
- Brincar
- Lavar
- Comer
- Caçar



# Classes e Instâncias

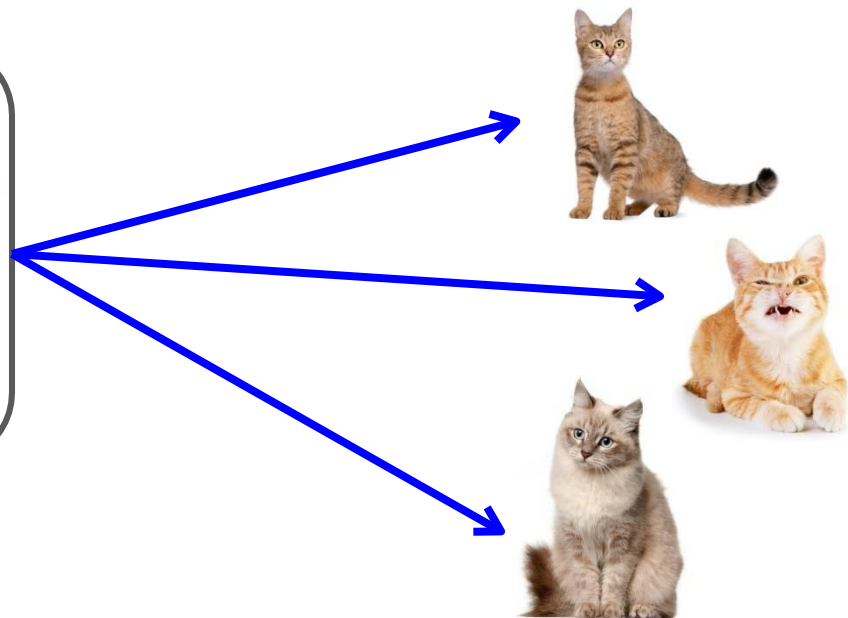
A combinação de propriedades e comportamentos é...

- Denominada **classe**
- Um plano gráfico ou uma receita para um objeto
- Usada para criar **instâncias** do objeto

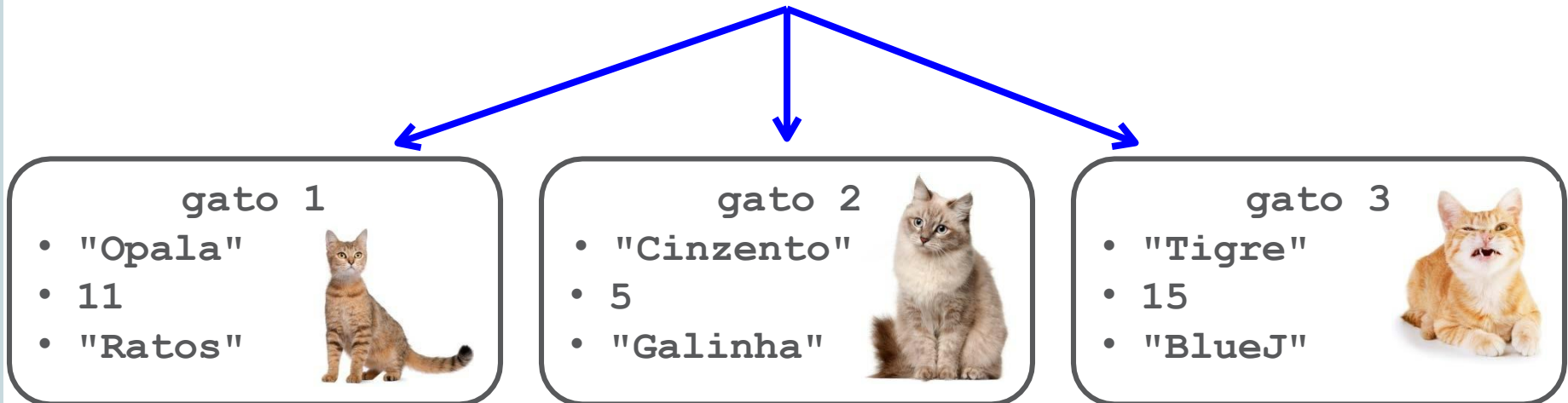
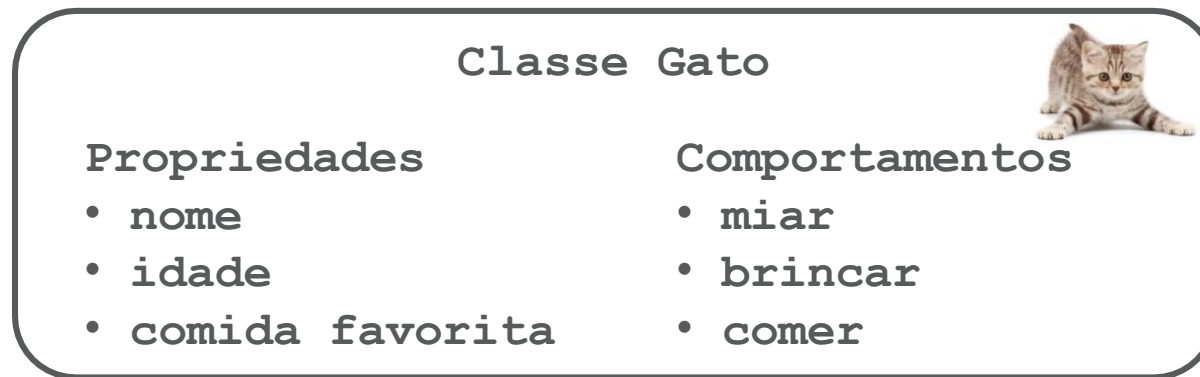
**Instâncias do objeto**

**Classe Gato**

- **Propriedades**
- **Comportamentos**



# Criando Novas Instâncias de um Plano Gráfico

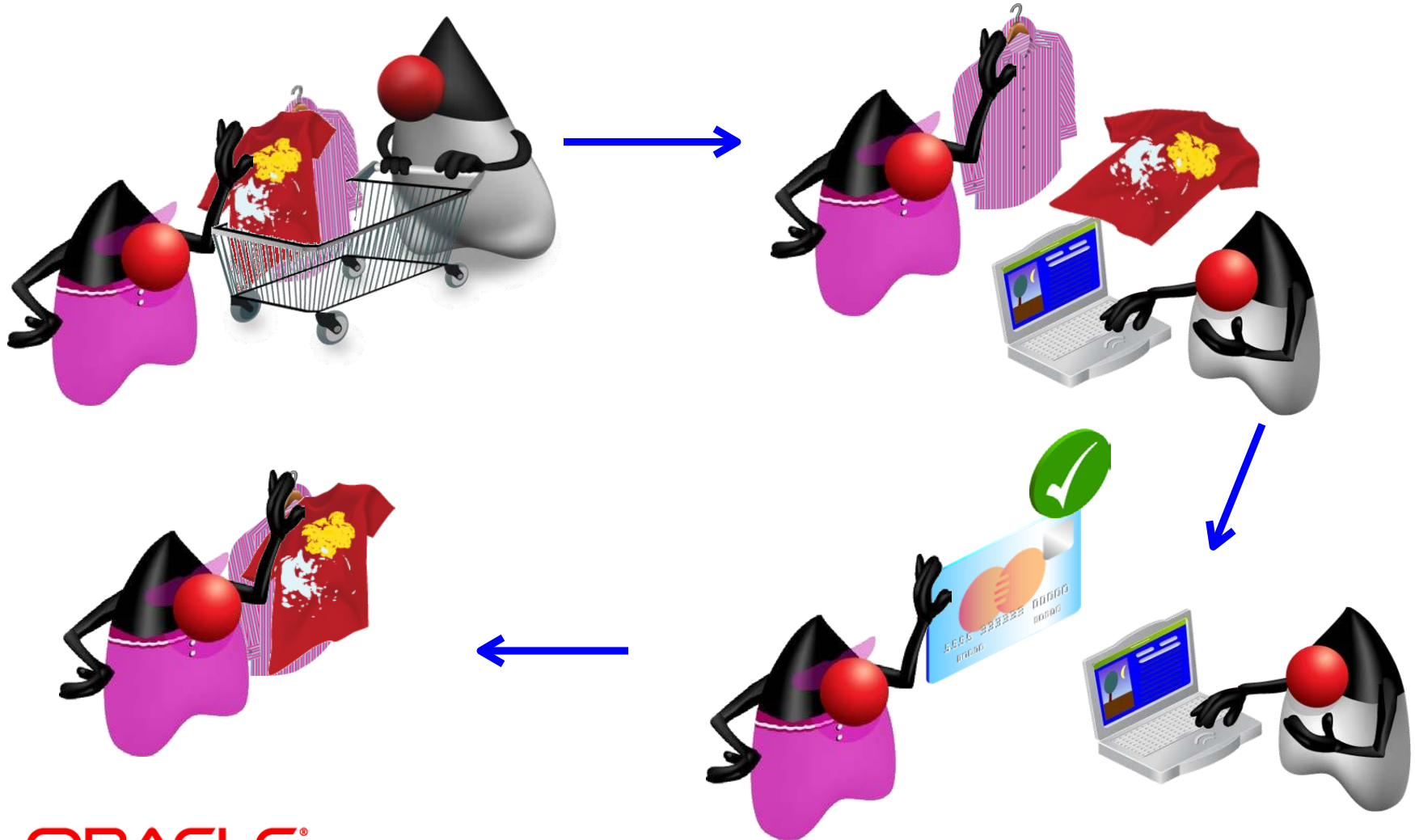


**Todas as instâncias do gato compartilham a capacidade de miar, brincar e comer.**

# Estratégia Orientada a Objetos

- Como você escreve programas para obter esse nível de flexibilidade?
- Quando você tem uma ideia ou um requisito para um programa...
  - Considere que tipo de objetos pode existir nesse programa.
  - Considere as propriedades e os comportamentos desses tipos de objetos.
  - Considere o modo como os objetos interagem.

# Loja de Compras On-line Duke's Choice



# Características dos Objetos

- Os objetos são físicos ou conceituais.
- Os objetos têm **propriedades**:
  - Tamanho
  - Preço
  - Cor
- Os objetos têm **comportamentos**:
  - Comprar
  - Colocar o item no carrinho
  - Pagar



Físico:  
camisa



Conceitual:  
conta on-line



O valor da propriedade  
Cor é vermelho.

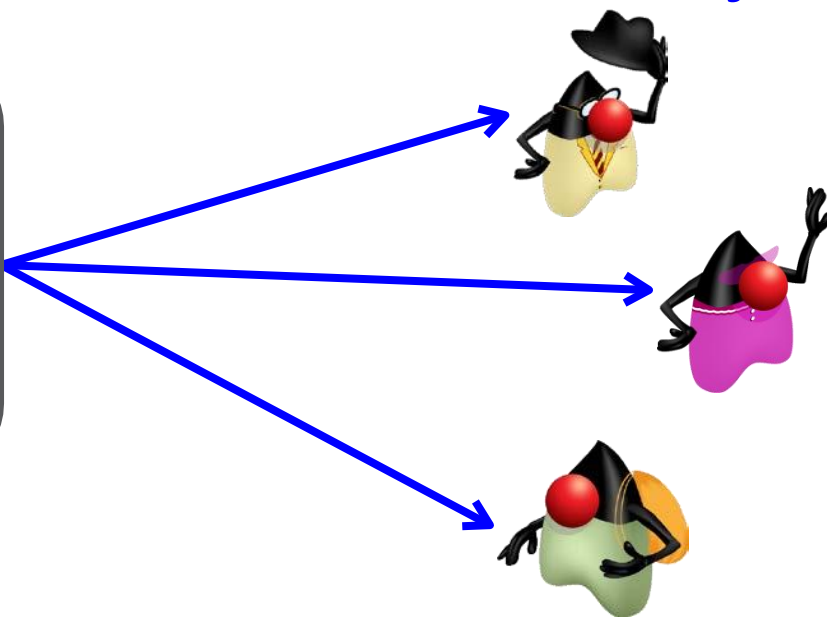


Sra. Duke

# Classes e Instâncias

- Lembre-se de que uma classe...
  - É um plano gráfico ou uma receita para um objeto
  - Descreve as propriedades e os comportamentos de um objeto
  - É usada para criar instâncias do Objeto

## Instâncias do objeto



# Exercício 2, Parte 1

Dado o cenário a seguir, quais objetos você poderia modelar para concluir seu programa?

Crie um programa para uma máquina de contagem de moedas. Essa máquina deve medir, contar e classificar moedas com base no respectivo tamanho ou valor. Ela também deve imprimir um recibo.

- Liste pelo menos 3 objetos.

- 1.
- 2.
- 3.



# Exercício 2, Parte 2

- |  |   |   |
|--|---|---|
| <ul style="list-style-type: none"><li>• Escolha um objeto da Parte 1.</li><li>• Quais propriedades e comportamentos desse objeto você poderia incluir no seu programa?</li></ul> | <ul style="list-style-type: none"><li>• Propriedades:<ol style="list-style-type: none"><li>1.</li><li>2.</li><li>3.</li></ol></li></ul> | <ul style="list-style-type: none"><li>• Comportamentos:<ol style="list-style-type: none"><li>1.</li><li>2.</li><li>3.</li></ol></li></ul> |
|--|---|---|



# Tópicos

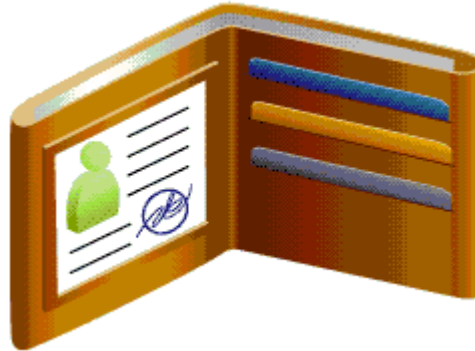
- Linguagens Orientadas a Objetos x Linguagens Procedurais
- Classes, Instâncias, Propriedades e Comportamentos
- Convertendo para uma Sintaxe Java



# Propriedades e Comportamentos da Classe Customer

- Propriedades:

- Nome
- Endereço
- Idade
- Número do pedido
- Número do cliente



- Comportamentos:

- Comprar
- Definir o endereço
- Adicionar um item ao carrinho
- Pedir um desconto
- Exibir detalhes do cliente

# Convertendo para uma Sintaxe Java

```
1 public class Customer {  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11 }
```

# Terminologia Java

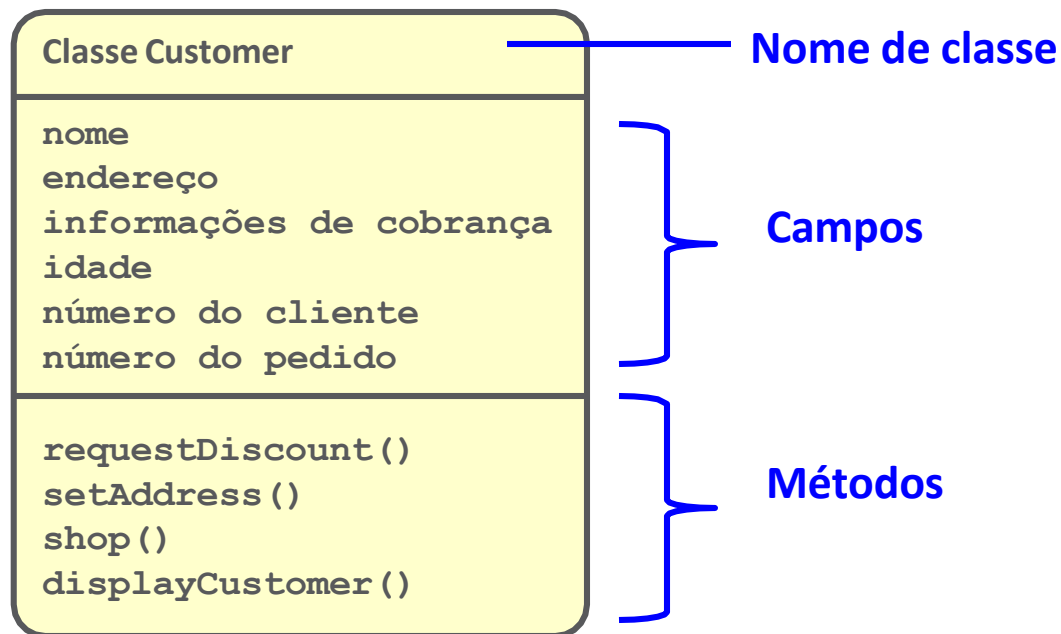
## Declaração de classe

```
1 public class Customer {  
2     public String name = "Junior Duke";  
3     public int custID = 1205;  
4     public String address;  
5     public int orderNum;  
6     public int age;  
7  
8     public void displayCustomer() {  
9         System.out.println("Cliente: "+name);  
10    }  
11 }
```

**Campos**  
(Propriedades)  
(Atributos)

**Métodos**  
(Comportamentos)

# Modelando Propriedades e Comportamentos



# Campos de Dados

- Os **Campos** ou **Campos de Dados** são a terminologia Java oficial. Eles também são denominados:
  - Propriedades
  - Atributos
  - Membros de Dados
- O Java tem maneiras particulares de representar os dados.
  - A Seção 3 fará uma análise mais detalhada dos dados.
  - Usaremos o método main para essa investigação.
  - Por enquanto, é adequado incluir um grande volume de código no método main.
  - MAS não recomendamos em hipótese alguma o uso de um método main muito grande. A Seção 4 mostrará como evitar esse cenário.