
ORACLE
Academy

Java Fundamentals

7-4

Herança

ORACLE
Academy

Objetivos

- Esta aula abrange os seguintes tópicos:
  - Demonstrar e explicar os diagramas de classe UML (Unified Modeling Language)
  - Usar a palavra-chave extends para herdar uma classe
  - Comparar e contrastar superclasses e subclasses
  - Descrever como a herança afeta o acesso de um membro
  - Usar super para chamar um construtor de superclasse
  - Usar super para acessar membros da superclasse
  - Criar uma hierarquia de classes de vários níveis

Visão geral

- Esta aula abrange os seguintes tópicos:
  - Reconhecer quando os construtores são chamados em uma hierarquia de classes
  - Demonstrar compreensão da herança com o uso de applets
  - Reconhecer alterações corretas do parâmetro em um applet existente

O que é Herança?

- Herança é uma ferramenta simples, porém poderosa, de linguagens orientadas por objetos que permitem que as classes herdem métodos e campos de outras classes
- Herdar significa receber ou obter algo do seu antecessor ou pai
- No Java, o conceito de herança é semelhante à genética
  - Os genes e características genéticas são passados de pai para filho
  - Consequentemente, os filhos normalmente se parecem e agem como seus pais

Mais Informações sobre Herança

- Para obter mais informações sobre herança, visite:
  - http://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

Superclasse versus Subclasse

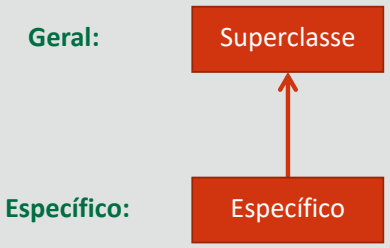
- As classes podem derivar ou evoluir de classes pai, o que significa que elas contêm os mesmos métodos e campos que seus pais, mas podem ser consideradas uma forma mais especializada de suas classes pai
- A diferença entre uma subclasse e uma superclasse é a seguinte:

Superclasse	**Subclasse**
A classe mais genérica da qual outras classes derivam seus métodos e dados.	A classe mais específica que é derivada ou herdada de outra classe (a superclasse).

A superclasse, às vezes, é denominada a classe de base.

Superclasse versus Subclasse

- Superclasses contêm métodos e campos que são passados para todas as suas subclasses
- Subclasses:
  - Herdam métodos e campos de suas superclasses
  - Podem definir métodos ou campos adicionais que a superclasse não tem



Exemplo de Herança

- Criar uma classe Forma com uma variável, cor e um método que retorna a cor
  - Criar uma classe Retângulo que herda a variável e o método de Forma, e pode ter seus próprios métodos e variáveis

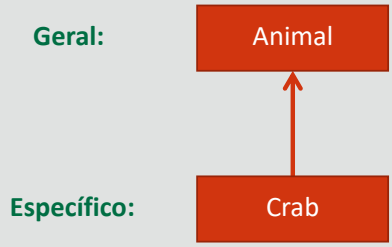
```
Shape (superclass)
public String color
public String getColor()
```

```
Rectangle (subclass)
public String color
public String getColor()

//Rectangle-only data
public int length
public int width
public int getLength()
public int getWidth()
```


A API do Java tem uma interface Forma e uma classe Retângulo. Os exemplos contidos nos slides não se referem à API.

Exemplo de Superclasse vs. Subclasse

- Considere as classes Animal e Crab no Greenfoot
- Animal é um termo mais genérico do que Crab e pode se aplicar a mais criaturas do que apenas a Crab
- Um Crab é um tipo de Animal, e esse Crab se aplica a um tipo específico de Animal
- Portanto, Crab é a subclasse e Animal é a superclasse



A subclasse deve atender ao teste "É um" com a superclasse. Um caranguejo "É um" animal. Esse conceito é abordado no slide 26.

Método move da Classe Crab

- De onde vem o método move() na classe Crab?
- Não há código visível que mostre a lógica do método move() na classe Crab

```
public class Crab extends Animal
{
   public void act()
   {
      move(1);
   }//fim do método act
}//fim da classe Crab
```

Método move Herdado

- Apesar de o código não estar escrito na classe Crab, sabemos que um objeto Crab pode chamar o método move()
- Portanto, o código deve ser herdado da superclasse, Animal, da seguinte forma:

```
public class Animal
{
   public void move(int d)
   {
      //Logic for move()
   }//fim do método move
}//fim da classe Animal
```

Palavra-chave extends

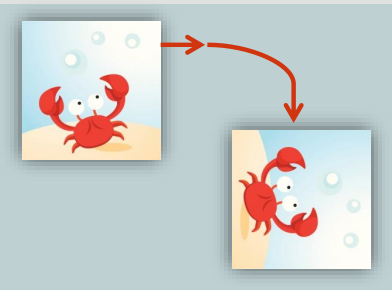
- Em Java, você pode escolher de quais classes você deseja herdar usando a palavra-chave extends
- A palavra-chave extends permite designar a superclasse que tem métodos que você deseja herdar, ou cujos métodos e dados você deseja estender
- Por exemplo, para herdar métodos da classe Forma, use extends quando a classe Retângulo for criada

```
public class Rectangle extends Shape
{
   //code
}//fim da classe Rectangle
```

Exemplo de Palavra-chave extends

- Queremos que a classe Crab estenda os métodos e dados da classe Animal e herde métodos como move(), turn(), etc
  - Como a classe Animal é estendida, você pode chamar os métodos move() e turn(), apesar deles não aparecerem dentro do código da classe Crab

```
public class Crab extends Animal
{
   public void act()
   {
      move(1);
      turn(90);
   }//fim do método act
}//fim da classe Crab
```



A Regra de Herança Simples

- Herança simples significa que você não pode declarar ou estender mais de uma superclasse por classe
- O código a seguir não fará compilação:

```
public class Crab extends Animal, Crustacean, ...
```

Estendendo Mais de Uma Classe

- Por que não podemos estender mais de uma classe?
  - Como as superclasses passam seus métodos e dados para todas as suas subclasses, e para as subclasses de suas subclasses, não é necessário estender mais de uma classe

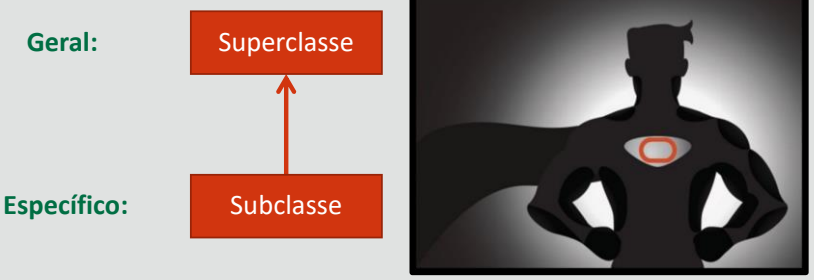
Interfaces, às vezes, são usadas para implementar o comportamento que compensa a falta de várias Encapsuladas. As interfaces são abordadas com mais detalhes em Programação Java.

Mais Sobre Herança

- Herança é uma via de mão única
  - As subclasses herdam de superclasses, mas as superclasses não podem acessar ou herdar métodos e dados de suas subclasses
  - Assim como pais não herdam características genéticas, como cor de cabelo ou cor do olho, de seus filhos

Object: A Maior Superclasse

- Toda superclasse estende implicitamente a classe Object
- Objeto:
  - É considerado o maior e mais genérico componente de qualquer hierarquia
  - É a única classe que não tem uma superclasse
  - Contém métodos muito genéricos que toda classe herda



ORACLE Academy

Os métodos do Objeto, geralmente, são substituídos, se forem usados.

Exemplo de Objeto 1

- O objeto contém métodos que podem ser usados em todas as classes (como toString() ou equals())
- Por exemplo, após criar uma classe e construir uma instância dela, você pode chamar o método toString() no seu objeto?

```
A_Class class = new A_Class();
class.toString();
```

- Sim.
  - Mesmo que você não tenha escrito o método toString(), ainda será possível chamar este método, porque ele foi herdado de Object

Exemplo de Objeto 2

- O class.toString() é legal se A_Class explicitamente estender Another_Class, uma superclasse?
- Sim.
- Isso também é legal, pois a superclasse de A_Class estende Object

```
A_Class class = new A_Class();
class.toString();
```

Por que Usar Herança?

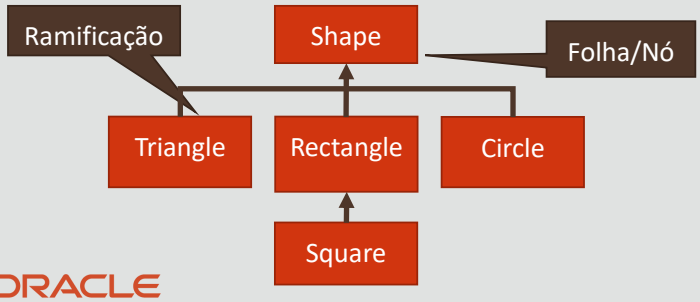
- O benefício principal da herança é a reutilização de código
- Herdar métodos de uma superclasse dá à sua classe acesso ao código e aos dados da superclasse
- Você não precisará escrever o código duas vezes, o que economiza tempo e otimiza seu código
- Além disso, ocorrem poucos erros

Hierarquias de Herança

- Em muitas situações, é comum classificar conceitos como hierarquias
  - Uma hierarquia é uma forma de categorizar a relação entre ideias, conceitos ou coisas com o componente mais geral ou abrangente na parte superior e o componente mais específico, ou com o escopo mais restrito, na parte inferior
  - As hierarquias são um conceito útil em se tratando de herança, e podem ser usadas para modelar e organizar a relação entre superclasses e subclasses

Diagramas em Árvore

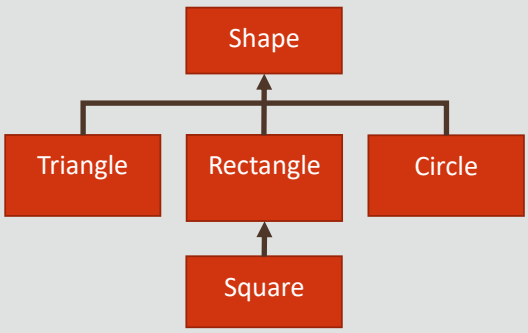
- As hierarquias podem ser organizadas em diagramas de árvore
  - Profissionais de informática normalmente se referem a árvores tendo folhas e ramificações, ou se referem às "folhas" como nós
  - Por exemplo, as formas podem ser categorizadas por diferentes propriedades, como o número de lados



ORACLE Academy

Diagramas em Árvore

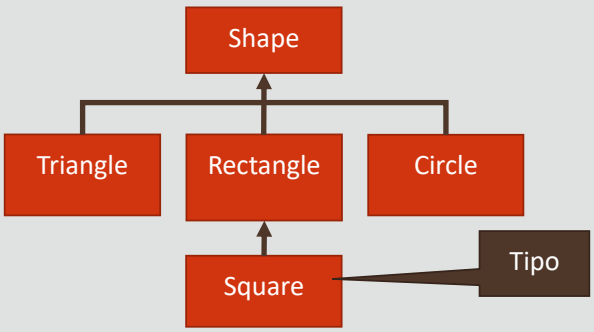
- Note que Círculo, Triângulo e Retângulo têm um número diferente de lados portanto, eles são ramificações diferentes na árvore
- Somente os nós com as mesmas propriedades ocupam a mesma ramificação



ORACLE Academy

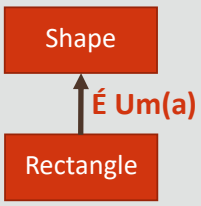
Tipo de Nós Pais

- Tudo que estiver abaixo de um nó na árvore é um tipo de nó pai
- Sabemos que Quadrado é um tipo de Retângulo
- Triângulo, Retângulo e Círculo são todos tipos de formas



Hierarquias de Herança: “É Um(a)”

- Com hierarquias de classes, você pode usar a frase “is-a” para descrever uma relação hierárquica
- Um nó de uma ramificação pode ser considerado do mesmo tipo que o nó da raiz
- Exemplo: Um Rectangle “is-a” Shape (Retângulo "é-um(a)" Forma), pois tem todas as propriedades de uma forma
- Para modelar relações entre as classes, usamos UML



Unified Modeling Language: UML

- Os profissionais de informática modelam hierarquias de herança usando uma linguagem de modelagem chamada Unified Modeling Language ou UML
- UML é uma forma de descrever as relações entre as classes de um sistema, ou a representação gráfica de um sistema
- A UML foi desenvolvida por Grady Booch, James Rumbaugh e Ivar Jacobson, e é padronizada de forma que possa ser compreendida em vários idiomas

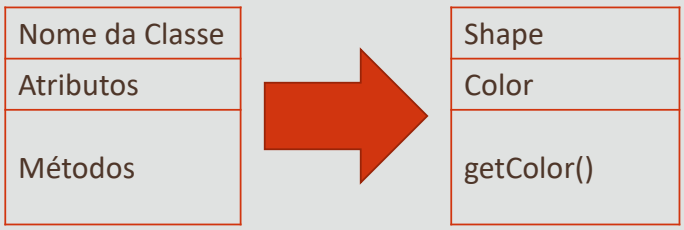
Componentes Básicos da UML

- As hierarquias de herança podem ser modeladas simplesmente com a UML
- Alguns componentes simples são necessários para começar:
  - Diagrama de classe:
    - Mostra o nome da classe, e quaisquer dados ou métodos importantes dentro da classe
  - Setas e linhas:
    - Mostra a relação de uma classe com outras classes

O UML é muito mais útil para modelagem do que é mostrado neste curso.

Diagrama de Classe na UML

- Uma classe pode ser desenhada como uma caixa que contém o nome da classe, variáveis de instâncias e métodos
- As classes também podem ser desenhadas como caixas simples com apenas o nome da classe, embora a inclusão de métodos seja útil
- Não é necessário incluir cada atributo ou aqueles que representam conjuntos de dados (como arrays) Inclua somente os atributos mais úteis



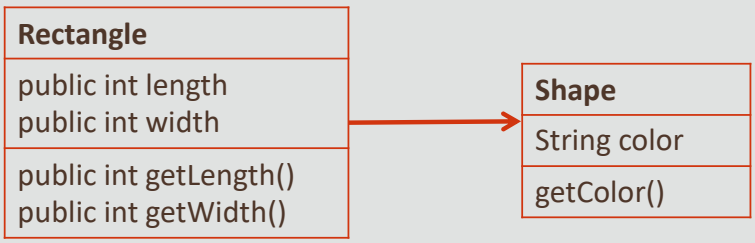
Mais Detalhes Sobre UML

- UML é uma ferramenta útil para você planejar como organizar hierarquias de classe com vários níveis
- Você deve usar a UML para projetos de codificação
- Para obter informações mais detalhadas sobre UML, visite:
  - http://www.oracle.com/technetwork/developer-tools/jdev/gettingstartedwithumlclassmodeling-130316.pdf

Mostrando Herança na UML

- Os diagramas de classe são conectados usando linhas com setas
- As linhas de conexão variam de acordo com a relação entre as classes
- Para a herança, uma linha reta e uma seta triangular são usadas para representar a relação "é-um(a)"

Relação	Símbolo
Herança	



Encapsulamento

- Encapsulamento é um conceito fundamental em programação orientada por objetos

Encapsulamento significa incluir algo em uma cápsula ou container, como colocar uma carta em um envelope. Em programação orientada por objetos, o encapsulamento inclui, ou embrulha, os trabalhos internos de uma instância/objeto Java

Como Funciona o Encapsulamento

- Em programação orientada por objetos, o encapsulamento inclui, ou embrulha, os trabalhos internos de uma instância/objeto Java
- As variáveis de dados, ou campos, são ocultos do usuário do objeto
- Os métodos podem fornecer acesso aos dados privados ou trabalhar com os dados, mas os métodos ocultam a implementação
- O encapsulamento dos seus dados os impede de serem modificados pelo usuário ou outras classes, para que os dados não sejam corrompidos

Como o Encapsulamento é Usado

- O encapsulamento pode ser usado para proteger dados confidenciais, como informações pessoais, evitando que os dados sejam alterados, exceto dentro do escopo da própria classe
- Os dados são protegidos, e a implementação é oculta declarando modificadores de acesso em variáveis e métodos
- Os modificadores de acesso (público, privado, protegido, “padrão”) são palavras-chave que determinam se outras classes podem acessar os dados ou métodos da classe

Modificadores de acesso

- Os programadores podem personalizar a visibilidade de seus dados e métodos com vários níveis de modificadores de acesso

Modificador de Acesso	Acessado por:
`public`	Qualquer classe de qualquer pacote
`private`	Somente a outros métodos dentro da própria classe
`protected`	Todas as subclasses e todas as classes do mesmo pacote
"default"	Qualquer classe do pacote Na verdade, quando nenhuma palavra-chave é especificada A palavra padrão NÃO é usada

Declarando Modificadores de Acesso

- A regra geral para declarar modificadores de acesso é que quaisquer dados que você queira proteger contra alterações feitas por outras classes, ou dados confidenciais, devem ser declarados privados
- Isso inclui variáveis
- Os métodos são geralmente declarados como públicos para que outras classes possam usá-los
- No entanto, os métodos podem ser declarados privados quando forem para ser usados somente pela própria classe

Exemplo de Declaração de Modificadores de Acesso

- Se a classe Shape (Forma) contém dados de cor, os dados desta classe devem ser privados

```
public class Shape {
   private String color;
}//end class Shape
```

Acesso de Membros

- Use a palavra-chave private para ocultar dados que somente a classe deve poder alterar (Este é o modificador de acesso recomendado)
- Se o acesso aos dados for necessário, um método de obtenção deverá ser escrito para conseguir isso

```
public class Shape {
    private String color; //the color of the Shape

  //Method which returns the color
  public String getColor() {
    return color;
    }//fim do método getColor
}//fim da classe Shape
```

Com a classe Shape, não queremos que os objetos alterem a cor desta Forma. Para conseguir isso, escreva 'private' na declaração da variável.

O método getColor() retorna a cor. Qualquer método destinado a acessar dados privados deve ser público.

Acesso de Membros

- Se as variáveis privadas tiverem que ser (ou puderem ser) alteradas, um método de definição deverá ser escrito

```
public class Shape {
   //the color of the Shape
   private String color;
   //Method which returns the color
   public String getColor() {
      return color;
   }//fim do método getColor
   //Method to change the color
   public void setColor(String c){
      color = c;
   }//fim do método setColor
}//fim da classe Shape
```

Se a cor precisar ser alterada, um método de definição é criado.

Usar Público ou Protegido para Acessar Dados?

- Se quiséssemos ter a capacidade de alterar a cor da variável da classe Forma de fora do código da classe, poderíamos definir a cor da variável da String para ser pública ou protegida
- No entanto, é recomendável que as variáveis da classe sejam declaradas como privadas

```
public class Shape {
   protected String color;
}//fim da classe Shape
```

```
public class Shape {
   public String color;
}//fim da classe Shape
```

Alterando a Cor

- Se a variável for declarada como pública, o código que estende ou cria um objeto Shape poderá alterar a cor sem usar um método de acesso, como setColor()

```
//exemplo para estender a classe forma, e alterar a cor
super.color = "Blue";

//exemplo para criar um objeto Shape e alterar a cor
Shape s1 = new Shape();
s1.color = "Blue";
```

Não recomendado

Não recomendado

Acesso de Membros e Herança

- Como estes modificadores de acesso afetam a herança?
- Com o encapsulamento, nem as subclasses podem acessar métodos e variáveis privadas
- Os modificadores públicos e protegidos fornecem acesso a métodos e variáveis da superclasse

Modificador de Acesso	Acessado por:
`public`	Todas as classes
`private`	Somente a própria classe
`protected`	Todas as subclasses e todas as classes do mesmo pacote
"default"	Se nenhuma palavra-chave for especificada, as variáveis de membros poderão ser acessadas por qualquer classe do pacote

Estendendo a Classe Shape

- Como a classe Shape não é uma classe específica, podemos estendê-la criando classes mais específicas, como Rectangle e Square
- Começaremos criando uma classe Rectangle que estende a classe Shape

Herdando Construtores

- Embora uma subclasse herde todos os métodos e campos de uma classe pai, ela não herda construtores
- Você pode:
  - Escrever seu(s) próprio(s) construtor(es)
  - Usar o construtor padrão
    - Se você não declarar um construtor, um construtor no-argument padrão será fornecido
    - Se você declarar seu próprio construtor, o construtor padrão não será mais fornecido

Usando a Palavra-chave super em um Construtor

- Ao criar um objeto Rectangle, você terá que definir a cor do Rectangle
- Se a cor da variável for privada na superclasse Shape, como fazer para defini-la?
- Para construir uma instância de uma subclasse, normalmente é mais fácil chamar o construtor da classe pai

Usando a Palavra-chave super em um Construtor

- A palavra-chave super é usada para chamar o construtor de uma classe pai
- Ela deve ser a primeira instrução do construtor
- Se ela não for fornecida, uma chamada padrão para super() será implicitamente inserida para você
- A palavra-chave super também pode ser usada para chamar um método da classe pai ou para acessar o campo (não privado) de uma classe pai

É por isso que sempre fornecer um construtor padrão é uma prática recomendada ao sobrecarregar construtores. Se você não fornecer um na superclasse, a instanciação da subclasse poderá falhar se a chamada padrão para super() for usada.

Exemplo do Uso da Palavra-chave Super

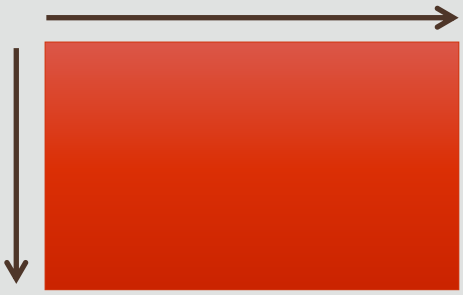
```
public class Rectangle extends Shape
{
    private int length;
    private int width;

    //Construtor
    public Rectangle(String color, int length, int width)
    {
        super(color);
        this.length = length;
        this.width = width;
    }//fim construtor
}//fim da classe Rectangle
```

Isso chama o construtor de Shape, que inicializa a cor da variável.

Adicionando métodos da Classe Rectangle

- Os métodos de Rectangle que seriam úteis são:

```
public int getWidth()
public int setWidth(int width)
public int getHeight()
public int setHeight(int height)
public int getArea()
```



Métodos da Classe Rectangle

- Como Square é um tipo de Rectangle, ou estende a classe Rectangle, ele herdará todos os métodos da superclasse Rectangle:

```
public int getWidth()
public int setWidth(int width)
public int getHeight()
public int setHeight(int height)
public int getArea()
```

Configurar a Classe

- Este código configura a classe
- Use a palavra-chave extends para herdar os métodos de Rectangle

```
public class Square extends Rectangle {

}//fim da classe Square
```

Escrever o Construtor

- Para escrever o construtor, considere os valores que precisam ser inicializados
- Se usarmos o construtor super de Rectangle, precisamos repassar os valores: String color, int length e int width
- Nosso construtor Square requer esses valores como parâmetros se quisermos chamar o construtor super

```
public class Square extends Rectangle {
    public Square(String color, int length, int width) {

    }//fim consrutor
}//fim da classe Square
```

O comprimento e a largura não são iguais em um quadrado?

Parâmetro de Tamanho

- Embora Quadrados sejam um tipo de Retângulo, eles têm uma propriedade única, que é comprimento = largura
- Acomode esta característica exigindo apenas um parâmetro de tamanho que defina tanto os valores da largura quanto os do comprimento

```
public class Square extends Rectangle {

    public Square(String color, int size) {
        super(color, size, size);
    }//fim construtor
}//fim da classe Square
```

Em vez de repassar dois parâmetros diferentes de comprimento e largura, podemos repassar um parâmetro de tamanho duas vezes, o que definirá os valores do comprimento e da largura (localizados na classe Rectangle) como sendo iguais.

Variáveis Exclusivas da Subclasse

- E as variáveis exclusivas que se aplicam somente a Quadrados e não a Retângulos?
- Por exemplo, um recurso que nos informa se devemos ou não preencher um Square
- Adicione um valor booleano na lista de parâmetros para adicionar esta variável exclusiva da classe Square:

```
public class Square extends Rectangle {
   private boolean isFilled;
   public Square(String color, int size, boolean isFilled){
        super(color, size, size);
        this.isFilled = isFilled;
   }//fim construtor
}//fim da classe Square
```

A variável isFilled é exclusiva da classe Square e é um exemplo de como as subclasses podem conter mais métodos ou campos do que suas superclasses.

Personalizar Métodos

- Como um Quadrado tem os mesmos valores para altura e largura, devemos personalizar os métodos setWidth(int width) e setHeight(int height) para que ambos sejam atualizados quando o método for chamado
- Use a palavra-chave super para chamar os métodos setLength() e setWidth() da superclasse e defini-los para o valor do parâmetro repassado para o método

```
 public int setWidth(int width){
     super.setLength(width);
     super.setWidth(width);
}//fim do método setWidth
```

Subclasse Square

- O produto final ficará da seguinte forma:

```
public class Square extends Rectangle {
  private boolean isFilled;

  public Square(String color, int size, boolean isFilled){
     super(color, size, size);
     this.isFilled = isFilled;
  }//fim construtor

  public void setLength(int length) {
     super.setLength(length);
     super.setWidth(length);
  }//fim do método setLength
  public void setWidth(int width) {
     super.setWidth(width);
     super.setLength(width);
  }//fim do método setWidth
  public boolean getIsFilled() {
     return isFilled;
  }//fim do método getIsFilled
}//fim da classe Square
```

Herança e Applets

- Os applets Java são outro exemplo do uso de herança
- Applet Java é um programa Java baseado na Web que pode ser incorporado em um navegador da Web
- A classe Applet pode ser estendida para criar applets especiais usando alguns dos principais métodos da classe Applet

Documentação Java para a Classe Applet

- Visite a Documentação Java da classe Applet para saber mais
- Para obter toda a documentação, visite:
  - http://docs.oracle.com/javase/8/docs/api/
- Para obter apenas a documentação da classe Applet, visite:
  - http://docs.oracle.com/javase/8/docs/api/java/applet/Applet.html

Criando Applets

- Para criar um applet, você pode usar todos os principais métodos da classe Applet e personalizar esses métodos para adequar às necessidades específicas do seu applet

Criando Applets

- Por exemplo, para criar um applet que desenhe Formas, comece configurando a herança com a palavra-chave extends:

```
public class DrawShapes extends Applet {
   ...
}//fim da classe DrawShapes
```

- Agora nossa classe applet DrawShapes herdará métodos de Applet que podemos personalizar para criar o applet

Exemplo de Applet

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Graphis2D;
import java.awt.Rectangle;

public class RectangleApplet extends Applet{

   public void paint(Graphics g){
      Graphics2D g2 = (Graphics2D)g;
      Rectangle testRectangle = new Rectangle(5,10,20,30);
      g2.draw(testRectangle);
   }//fim do método paint

}//fim da classe RectangleApplet
```

Terminologia

- Os principais termos usados nesta lição foram:
  - Modificadores de acesso
  - Classe filha
  - padrão
  - Encapsulamento
  - extends
  - Hierarquia
  - Herança
  - Relação “is-a”

Terminologia

- Os principais termos usados nesta lição foram:
  - Classe pai
  - private
  - protegido
  - public
  - Subclasse
  - super
  - Superclasse
  - Linguagem de Modelamento Unificado (UML)

Visão geral

- Nesta aula, você deverá ter aprendido a:
  - Demonstrar e explicar os diagramas de classe UML (Unified Modeling Language)
  - Usar a palavra-chave extends para herdar uma classe
  - Comparar e contrastar superclasses e subclasses
  - Descrever como a herança afeta o acesso de um membro
  - Usar super para chamar um construtor de superclasse
  - Usar super para acessar membros da superclasse

Visão geral

- Nesta aula, você deverá ter aprendido a:
  - Criar uma hierarquia de classes de vários níveis
  - Reconhecer quando os construtores são chamados em uma hierarquia de classes
  - Demonstrar compreensão da herança com o uso de applets
  - Reconhecer alterações corretas do parâmetro em um applet existente


ORACLE
Academy