## **O que é herança?**

Herança é um recurso que permite a uma classe se beneficiar de outra, incorporando a sua estrutura. Imagine que você tem uma classe A e, agora, precisa definir uma classe B muito parecida com A. Nós poderíamos copiar a estrutura de A em B e só então definir o que B tem de específico. Em vez disso, podemos utilizar o recurso de herança para criar uma relação entre essas duas classes, em que B herda variáveis e métodos que existem em A, implicitamente, ou seja, o código-fonte de A não está reescrito em B, mas B pode fazer uso do que foi definido em A. A partir desse ponto, podemos definir o que é específico para B.

A herança nos permite ter classes que representam certa entidade do mundo real em diversos níveis de [abstração](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=abstra%C3%A7%C3%A3o). Abstração, nesse caso, é a medida do quanto uma entidade se afasta da sua forma concreta. Note este exemplo: um cachorro é um tipo de mamífero, que, por sua vez, é um tipo de animal. Mas, quando nos referimos a um cachorro, não o chamamos de animal ou mamífero porque existe uma forma mais específica de tratá-lo; logo, o chamamos de cachorro. Esse conceito também vale na Programação Orientada a Objetos.  
 Estrutura organizacional de herança.   
  
Essa relação “ser” é a base do conceito de herança. Perceba que, quando dizemos que uma entidade X é do mesmo tipo de outra entidade Y, estamos dizendo que X tem características comuns a Y, porém que não é igual a Y. Cabe a X possuir características específicas que o torne diferente de Y. Encontraremos, em diversos momentos, esse tipo de situação quando estivermos escrevendo nossas classes.

A relação “ser” permite que criemos uma hierarquia, chamada Hierarquia de heranças, na qual uma classe herda uma determinada característica de outra classe, essa outra classe herda de uma terceira classe e assim por diante. É comum os desenvolvedores utilizarem a notação de classe pai (classe que fornece características) e classe filha (classe que recebe características). Na verdade, cada linguagem de programação define sua própria nomenclatura para essa relação. Por definição, uma classe pode herdar características de várias outras classes, porém a Linguagem Java não permite esse modelo, que é chamado de herança múltipla.  
  
 Quando houver dúvida sobre qual relacionamento é mais adequado entre duas classes, basta utilizar o termo “é um” ou o termo “tem um”. Caso o termo “é um” seja mais adequado, devemos realizar herança entre as classes, por exemplo, entre as classes Animal e Cachorro, em que Cachorro “é um” Animal. No entanto, quando o termo “tem um” for mais adequado, devemos instanciar um objeto de uma classe dentro da outra. Por exemplo, uma relação entre as classes Carro e Pneu, em que um Carro “tem um” Pneu. Observe que o termo “é um” não é adequado entre Carro e Pneu, pois um Carro não é um Pneu, e que o termo “tem um” não é adequado entre Animal e Cachorro.  
  
 A herança é um recurso valioso para a engenharia de software. Quando bem aplicada, ela permite que classes, previamente desenvolvidas e testadas, sejam reutilizadas, evitando que existam códigos-fontes duplicados e que organizam melhor nosso projeto.

## **Aplicando herança em Java**

Em Java, utilizaremos a palavra-chave extends para definir a relação de herança entre duas classes. Essa definição será feita sempre na classe que herda características. Essa classe é chamada de [subclasse](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=subclasse). A classe que fornece características é chamada de superclasse. Podemos dizer, então, que a [subclasse](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=subclasse) é uma especialização da superclasse. Atentemo-nos para o exemplo a seguir:

public class A{

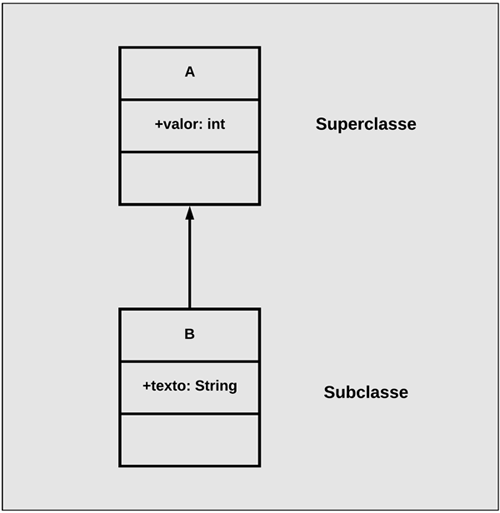
public int valor;

}

public class B extends A{

public String texto;

}

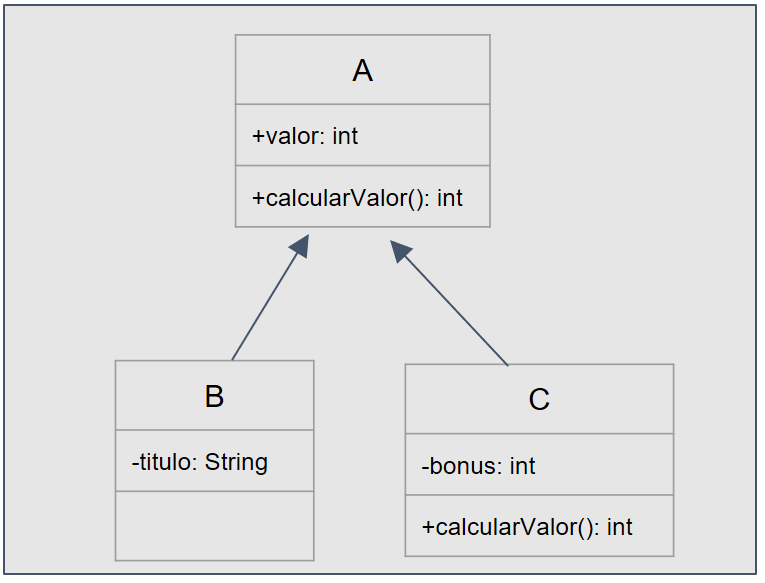


Perceba que A é a superclasse e B é [subclasse](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=subclasse) de A. Assim, B herda características de A. A classe B possui dois membros: valor do tipo int e texto do tipo String. O membro valor não aparece explicitamente no código-fonte de B, mas, devido à herança criada com o uso da palavra-chave extends, esse membro foi herdado de A por B. Na imagem, temos a representação formal da herança entre as duas classes, simbolizada pela seta, no sentido da [subclasse](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=subclasse) para a superclasse.

## **Visibilidade e acesso aos membros herdados**

Quando dizemos que uma [subclasse](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=subclasse) herda características da sua superclasse, estamos nos referindo aos seus membros, como variáveis, construtores e métodos. Porém, cada membro da superclasse pode ter sua visibilidade definida pelas palavras-chaves public, protected ou private ou, ainda, ter visibilidade padrão (default), quando não possui um modificador de acesso explícito. No quadro a seguir, resumimos a visibilidade dos membros da superclasse dentro da [subclasse](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=subclasse). Note:

|  |  |
| --- | --- |
| **MEMBRO DA SUPERCLASSE**  **(Variável, construtor ou método)** | **VISIBILIDADE DA** [**SUBCLASSE**](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=SUBCLASSE) |
| private | Invisível  (não é acessível) |
| protected | Visível |
| public | Visível |
| padrão  (default) | Visível (se estiver no mesmo  pacote) |

Antes de saber como acessar um membro herdado, devemos saber como a Máquina Virtual Java (JVM) busca por membros em uma hierarquia de heranças. Por exemplo, na hierarquia de heranças apresentada no quadro a seguir, B e C são subclasses de A. Se instanciarmos um objeto do tipo B (instanciado a partir da classe B) e, a partir dele, chamarmos o método calcularValor(), a JVM irá procurar primeiro pelo método em B. Se não encontrá-lo, irá subir na hierarquia de heranças para buscar na superclasse A pelo método calcularValor().  
 Representação da Herança das classes B e C a partir da classe A. 

Se instanciarmos um objeto do tipo C e, a partir dele, chamarmos o método calcularValor(), a JVM encontrará uma versão local desse método em C e, em vez de procurar pelo método em A, utilizará essa versão. Isso ocorre porque a versão desse método em C deve ser mais específica para C, ou seja, mais adaptada para as necessidades de C do que a versão definida em A. Chamamos de sobrescrita ([override](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=override)) a ação de definir, nas subclasses, versões mais específicas para métodos e construtores herdados. Não é possível sobrescrever variáveis.

A JVM se comportará da forma descrita anteriormente quando buscar por variáveis, construtores e métodos. Mas, digamos que o objeto, mesmo possuindo uma versão personalizada de um método ou construtor, deseja utilizar a versão definida pela sua superclasse. Para isso, utilizaremos a palavra-chave super. Saberemos com mais detalhes o uso da palavra-chave super mais adiante.

OBS: Quando sobrescrevemos métodos, o cabeçalho do método da [subclasse](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=subclasse) deve ter o mesmo nome e os mesmos parâmetros do método na superclasse.

Considere uma superclasse Pessoa definida no exemplo a seguir:

public void Pessoa{

public void cumprimentar (){

System.out.println("Olá”);

}

}

A [subclasse](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=subclasse) Aluno do quadro a seguir sobrescreve o método cumprimentar de Pessoa.

public void Aluno extends Pessoa{

public void cumprimentar (){

System.out.println("Olá colega”);

}

}

Observe que o cabeçalho do método cumprimentar das duas é igual: public void cumprimentar(). Porém, no exemplo do quadro a seguir, dado para a classe Aluno, ela não está realizando a sobrescrita do método cumprimentar.

public void Aluno extends Pessoa{

public void cumprimentar (String cumprimento){

System.out.println("Olá” + cumprimento);

}

}

Perceba que os cabeçalhos são diferentes. Enquanto o método cumprimentar da classe Pessoa é definido sem o recebimento de parâmetros, o método cumprimentar da classe Aluno recebe um parâmetro do tipo String e isso os diferencia.

## **Utilização da** [**palavra reservada**](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=palavra%20reservada) **super**

Quando queremos utilizar versões de métodos e construtores definidos na superclasse, utilizamos a palavra-chave super, acompanhada de ponto, antes do identificador do membro buscado. Atentemo-nos para o exemplo a seguir:

public class C extends A {

private int bonus;

public C (int valor, int bonus){

super();

this.bonus = bonus;

}

public int calcularValor(){

return super.calcularValor() + bonus;

}

public static void main(String args[]){

C objetoTipoC = new C(10, 50);

System.out.println(objetoTipoC.calcularValor());

}

}

public class A {

public int valor;

public A(){

this.valor = 0;

}

public int calcularValor(){

return valor;

}

}

Considerando a hierarquia de heranças, sabemos que C herda o método calcularValor() da classe A. Porém, escolhemos sobrescrever esse método em C para somar o valor da variável bônus. Para aproveitar o método calcularValor() definido em A, utilizamos a palavra super seguida de ponto e o nome do método que queremos utilizar para orientar a JVM a buscar esse método na superclasse, ou seja, em A. De forma parecida, no construtor de C, utilizamos super seguido de parênteses. Neste caso, super faz referência ao construtor da classe A, que não recebe parâmetros.

Quando definimos uma superclasse sem construtores default, ou seja, que não recebe nenhum parâmetro, somos obrigados a sobrescrever ao menos um dos construtores da superclasse nas subclasses. Nesse caso, a primeira linha do construtor sobrescrito deve ser a “chamada”, ou seja, a invocação do construtor da superclasse, utilizando a palavra super e referenciando, dentro dos parênteses, os parâmetros necessários, como no exemplo a seguir:

public class Pessoa {

private String nome;

public Pessoa (String nome){

this.nome = nome;

}

}

public class Funcionario extends Pessoa {

private String profissao;

public Funcionario(String nome, String profissao){

super(nome);

this.profissao = profissao;

}

}