Herança

- Um dos conceitos fundamentais em programação orientada aos objectos é a herança
- Quando utilizada correctamente permite a reutilização de código e facilita o desenho de software
- A herança permite derivar uma nova classe a partir de outra já existente
- À classe já existente dá-se o nome de super classe
- À nova classe chama-se subclasse
- Também se pode usar o conceito pai-filho para descrever esta relação
- Uma subclasse herda características da sua super classe (herda as variáveis de instância e os métodos nela definidos)

Herança

- A relação de herança deve criar uma relação é-um (is-a), significando que a subclasse é uma versão mais específica da super classe
- Exemplo: dicionário é-um livro, pelo que uma classe que represente um dicionário pode ser subclasse de uma classe mais genérica que represente um livro
- Em Java a relação de herança é estabelecida usando a palavra reservada extends

```
class dicionario extends livro {
// conteúdo da classe
}
```

Herança

- Exemplo

```
// Classe Livro será uma super classe
class Livro {
    protected int paginas = 1500;
    public void page_message () {
        System.out.println ("Número de páginas: " + paginas);
    }
}
// Classe Dicionario herda os métodos e variáveis da classe Livro
class Dicionario extends Livro {
    private int entradas = 52500;
    public void entradas_message () {
      System.out.println ("Número de entradas: " + entradas);
      System.out.println ("Média por página:" + entradas/paginas);
    }
}
```


Herança

- Exemplo

```
class Palavras {
    public static void main (String[] args) {
            Dicionario larousse = new Dicionario ();
             //Dicionario herda este método de Livro
            larousse.page_message();
            larousse.definition_message();
    }
}
```

- Notar que não foi criado explicitamente um objecto da classe Livro
- No entanto, a criação de um objecto da classe Dicionario implica também a criação de um objecto da classe Livro, já que esta é a sua super classe

Herança

- Como já foi visto, a utilização de modificadores de visibilidade serve para controlar o acesso aos métodos e variáveis de uma classe
- O modificador public indica que a variável ou método pode ser acedida a partir de qualquer classe
- O modificador private indica que a variável ou método só pode ser acedida a partir da própria classe
- Isto significaria que seria necessário declarar os métodos e variáveis como public para que fossem herdados
  - Mas isto viola os princípios do encapsulamento...
- Podemos usar o modificador protected para indicar que os métodos e variáveis podem ser acedidos a partir de subclasses, mas não das restantes classes

Herança

- Apesar de terem visibilidade public, os construtores não são herdados
- Por vezes é necessário invocar o construtor da super classe a partir da subclasse, por forma a inicializar devidamente as variáveis herdadas
- A referência super pode ser usada para referenciar a super classe e é frequentemente usada para invocar o seu construtor

Herança

- Exemplo

```
class Livro {
  protected int paginas;
  //  Construtor, inicializa paginas com um valor dado
    public Livro (int num_paginas) {
        paginas = num_paginas;
     }
    public void page_message () {
        System.out.println ("Número de páginas: " + paginas);
  }
 }  // classe Livro
```


Herança

- Exemplo (cont.)

```
class Dicionario extends Livro {
    private int entradas;
// Construtor de dicionario. Chama o construtor de Livro para
// inicializar o número de páginas
    public Dicionario (int num_paginas, int num_entradas) {
       super (num_paginas);
       entradas= num_entradas;
     }  // construtor Dicionario
    public void definition_message () {
       System.out.println ("Número de entradas: " + entradas);
       System.out.println (" Média por página: " + entradas/paginas);
  }
 }  // classe Dicionario
```

Herança

- Exemplo (cont.)

```
class Words2 {
public static void main (String[] args) {
     Dicionario larousse = new Dicionario (1500, 52500);

     larousse.page_message();
     larousse.definition_message();
   }
}
```


Herança

- Um membro herdado por uma subclasse pode ser invocado directamente, tal como se tivesse sido nela declarado
- Mas os membros não herdados estão também definidos na subclasse, podendo ser utilizados indirectamente através dos métodos herdados da super classe

Herança

- Uma subclasse pode sobrepor um dos seus métodos à definição de um método herdado da sua super classe (overriding)
- Ou seja, uma subclasse pode redefinir um método que herdou
- O novo método tem que ter a mesma assinatura que o herdado, mas pode ter conteúdo diferente
- O tipo de objecto usado na chamada determina qual dos métodos é usado

Herança

- Exemplo

```
class Pensamento {
  public void mensagem() {
    System.out.println ("Aproxima-se a Queima das Fitas...");
    }
  } // class Pensamento

class Conselho extends Pensamento {
  public void mensagem() {
    System.out.println ("Atenção, Junho está à porta....");
  }
} // class Conselho

class Mensagens {
  public static void main (String[] args) {
    Pensamento diversao = new Pensamento();
    Conselho exames = new Conselho();
    diversao.mensagem();
    exames.mensagem();
  }
} // class Mensagens
```

Herança

- Vimos já que a palavra reservada super referencia a super classe da classe onde é utilizada
- Vimos também que esta palavra reservada pode ser utilizada para invocar o construtor da super classe a partir do construtor da subclasse
- Podemos agora generalizar e dizer que super pode ser usado para invocar qualquer método da super classe
- Esta capacidade só é útil quando ao método da super classe que se pretende invocar foi sobreposto outro na subclasse
- Assim, para explicitar que se quer executar o método da super classe podemos fazer super.método (parametros);

Herança

- O estabelecimento de relações de herança permite a criação de hierarquias de classes, uma vez que nada impede uma subclasse de ser, por sua vez, super classe de uma nova classe
- De igual modo nada impede que várias classes derivem da mesma super classe
- A definição da forma como a hierarquia de classes existente deve ser expandida para resolver um dado problema é uma questão central em programação orientada a objectos
- Geralmente aceita-se que os comportamentos comuns a um conjunto de classes devem ser colocados tão acima quanto possível na hierarquia

Herança

- Em Java a raiz da hierarquia de classes é ocupada pela classe Object
- Isto significa que todos os objectos são também derivados desta classe (ainda que longinquamente em muitos casos)
- Quando nada é dito (não existe extends na definição da classe) é assumido que a classe descende de Object
- A classe Object contém alguns métodos (como toString()) que são herdados por todos os objectos
- Para que estes métodos tenham comportamentos específicos de uma classe ela terá que sobrepor a sua versão à herdada de Object

Herança

- Vimos já que uma referência pode referir-se a um objecto da sua classe
- Por exemplo, imaginando uma classe Ferias:
  ```
  Ferias dia = new Ferias ();
  ```
- Se existir uma classe Verao derivada a partir de Ferias, é legítimo fazer:
  ```
  Ferias dia;
  dia = new Verao ();
  ```

Herança

- Ou seja, uma referência pode referir a sua própria classe ou qualquer outra classe relacionada com ela por relações de herança
- Como regra geral diz-se que se pode usar um objecto de uma subclasse onde um objecto da sua super classe possa ser utilizado
- Embora o inverso também seja verdadeiro, essa situação é menos comum e normalmente menos útil