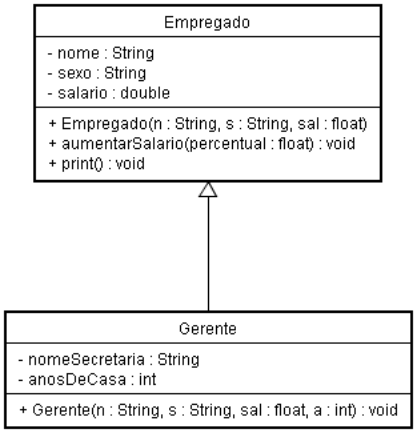
Nome: _____ Data: ____/____/____

Herança

Herança:

Um tipo de objeto (classe) pode ter subtipos (subclasse). Quando isso ocorre entra em ação o mecanismo da herança. Uma subclasse herda toda a estrutura de dados e todos os comportamentos de sua superclasse.



A subclasse "Gerente" herda toda a estrutura da superclasse "Empregado", dessa forma, só é necessário definir na classe "Gerente" os dados e o comportamento específico dessa classe, todo o restante já está definido em "Empregado".

Implementação da superclasse:

```
3  public class Empregado {
4
5      private String nome;
6      private String sexo;
7      private float salario;
8
9      public Empregado(String n, String s, float sal) {
10         nome = n;
11         sexo = s;
12         salario = sal;
13     }
14
15     public void aumentarSalario(float percentual) {
16         salario = salario * (1 + percentual / 100);
17     }
18
19     public void print() {
20         System.out.println(nome + " - " + sexo + " - " + salario);
21
22     }
23 }
```

Implementação da subclasse:

```
3  public class Gerente extends Empregado {
4
5      private String nomeSecretaria;
6      private int anosDeCasa;
7
8      public Gerente(String n, String s, float sal, int a) {
9          super(n, s, sal);
10         anosDeCasa = a;
11     }
12
13     public void aumentarSalario(float percentual) {
14         float bonus = percentual + (0.5f * anosDeCasa);
15         super.aumentarSalario(bonus);
16     }
17 }
```

```
3  public class TesteGerente {
4
5      public static void main(String[] args) {
6
7          Empregado empregado = new Empregado("Fulana", "F", 1000.0f);
8          empregado.aumentarSalario(10);
9          empregado.print();
10
11         Gerente gerente = new Gerente("Ciclano", "M", 1000.0f, 5);
12         gerente.aumentarSalario(10);
13         gerente.print();
14     }
15 }
```

Sobrescrita de métodos (override):

A sobrescrita de métodos ocorre quando uma subclasse possui o mesmo método que já existe em sua superclasse (**os métodos possuem a mesma assinatura**). A sobrescrita de métodos também é conhecida como anulação de método, uma vez que, o método da superclasse deixa de existir a partir da subclasse; passando a existir somente o método da subclasse.

Obs.: Observe as classes "Empregado" e "Gerente". A classe Empregado definiu o método "aumentarSalario", esse método foi sobrescrito na classe "Gerente". **Para que o método seja sobrescrito as assinaturas devem ser iguais, se as assinaturas forem diferentes e somente o nome dos métodos forem iguais, ocorreu uma sobrecarga e não uma sobrescrita**.

Modificadores de visibilidade:

A acessibilidade (ou visibilidade) de uma classe, método ou atributo de uma classe é a forma como esse elemento pode ser visto e utilizado por outras classes. Esse conceito faz parte do encapsulamento e é importantíssimo dentro da orientação a objetos. A visibilidade de uma classe, ou membro de uma classe é determinada pelo modificador de visibilidade que é utilizado na sua definição. A tabela abaixo esquematiza a visibilidade de elementos quando consideradas situações de implementação (codificação) e criação de objetos:

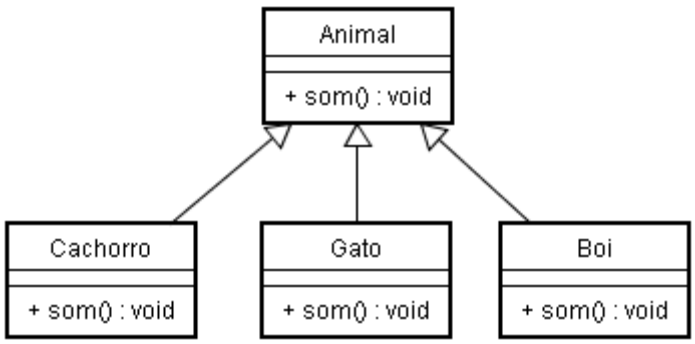
Modificador	**Implementação da classe**	**Instância da classe**
private (privado)	sim	não
protected (protegido)	sim	não
package (pacote)	sim	sim
public (público)	sim	sim

Polimorfismo:

É importante entender o que acontece quando a chamada de um método é aplicada a objetos de vários tipos em uma hierarquia de heranças. Lembre-se que em POO mensagens são enviadas para os objetos pedindo-lhes que realizem certas operações. Ao enviar uma mensagem que pede para que uma subclasse utilize certo método, a subclasse verifica se ela possui um método com essa assinatura. Caso ela possua o método, ele é usado. Se o método não existir na subclasse, essa passa o processamento a sua superclasse, que por sua vez também verifica se possui um método com essa assinatura. Caso exista nessa classe, o método é executado senão o processo continua até que seja encontrado um método que tenha essa mesma assinatura. Se em nenhuma das classes da hierarquia existir um método que tenha a mesma assinatura do solicitado, então um erro de compilação é gerado.

A capacidade de um objeto decidir qual método vai aplicar a si mesmo, dependendo da posição que está na hierarquia de heranças é chamado de polimorfismo. Para que o polimorfismo possa acontecer, é fundamental um mecanismo conhecido como ligação tardia (Late Binding). Esse mecanismo permite que a definição de qual método será realmente usado ocorra durante a execução do programa.

Exemplo:



Implementação da classe Animal:

```
3 public class Animal {
4
5     public void som(){}
6 }
```

Implementação da classe Cachorro:

```
3 public class Cachorro extends Animal {
4
5     public void som() {
6         System.out.println("O cachorro late!");
7     }
8 }
```

Implementação da classe Gato:

```
3 public class Gato extends Animal {
4
5     public void som() {
6         System.out.println("O gato mia!");
7     }
8 }
```

Implementação da classe Boi:

```
3 public class Boi extends Animal {
4
5     public void som() {
6         System.out.println("O boi mugi!");
7     }
8 }
```

Testando o Polimorfismo:

```
5     public static void main(String[] args) {
6
7         Animal[] animais = new Animal[3];
8
9         animais[0] = new Cachorro();
10        animais[1] = new Gato();
11        animais[2] = new Boi();
12
13        for (int i = 0; i < 3; i++) {
14            animais[i].som();
15        }
16    }
17 }
```

Conversão de tipos (Cast):

Da mesma forma que às vezes é necessária à conversão de um tipo primitivo em outro, pode ser que surja a necessidade de converter a referência de um objeto de uma classe para outra. Assim como na conversão entre tipos primitivos, esse processo é chamado de conversão explícita ou casting. A sintaxe para a conversão entre objetos é semelhante à sintaxe para a conversão entre tipos primitivos:

```
double x = 3.05;
int y = (int) x;


Cachorro cao = (Cachorro) animais[0];
```

Para que a conversão de tipos seja possível:

- Deve ser feita dentro de uma hierarquia de classes
- É recomendado o uso do operador **instanceof** para verificar a hierarquia e a validade da conversão.

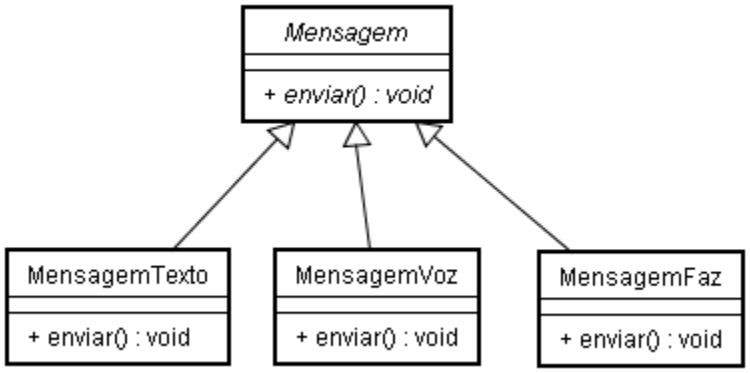
Na verdade a conversão explícita não é recomendada. Ao invés de usá-la, deve-se verificar a possibilidade do uso do polimorfismo.

A única razão para se fazer uma conversão explícita é usar um método que seja único à subclasse.

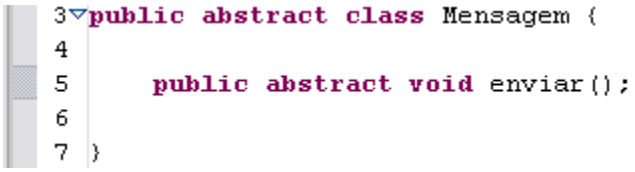
As conversões explícitas são mais usadas em contêineres como a classe Vector, pois os elementos de um objeto do tipo Vector são do tipo Object e, depois que colocamos um objeto dentro dele, precisamos da conversão explícita para buscá-lo.

Classes abstratas:

Ao subir na hierarquia de heranças, as classes tornam-se mais genéricas e, provavelmente, mais abstratas. Em algum ponto, a classe ancestral torna-se tão geral que acaba sendo vista mais como um modelo para outras classes do que uma classe com instâncias específicas que serão usadas. Considere um sistema de mensagens eletrônicas que integre e-mail, faz e viva-voz.



Para se criar uma classe abstrata, basta colocar a palavra reservada **abstract** antes da palavra reservada **class**.



```
3  public abstract class Mensagem {
4
5      public abstract void enviar();
6
7  }
```

Classes abstratas não podem ser instanciadas, somente servem de modelo para outras classes. Métodos abstratos devem, obrigatoriamente, serem implementados nas subclasses que estendem da classe abstrata. Uma classe abstrata pode ter métodos concretos (não abstratos) e métodos abstratos. Porém, para que uma classe possua métodos abstratos ela deve ser declarada abstrata. Ou seja, se uma classe tiver um único método abstrato ela deve necessariamente ser abstrata.