PILARES

Herança: Mecanismo que permite que novas classes sejam criadas a partir de classes já existentes. As subclasses herdam todos os métodos e atributos de suas superclasses (também conhecidas como classe mãe, classe pai ou classe base). Na herança há uma interdependência onde a existência de um depende da do outro.

- Atenção: construtores não são herdados. Pertencem somente a sua própria classe.

A herança é expressada pela palavra “extends”. Um sinônimo para a palavra extends é usar o "herda", ou ainda "é um".

public class Gerente extends Funcionario {
// Ou seja, Gerente tem, e sabe fazer, tudo que Funcionário tem, e faz.
}

No java, não existe herança múltipla! pode-se estender somente de uma classe.

- Atenção: a herança se baseia nos pilares de reutilização de código e do polimorfismo. Por isso, é recomendada quando há a combinação dessas duas necessidades. Se a necessidade for somente a reutilização de código, use a

composição! (teoricamente sem uso de herança pois cada classe existe independentemente). Se quiser somente uma solução pura de polimorfismo, utilize a interface.

Polimorfismo: Permite que referências de tipos de classes mais genéricas referenciam objetos mais específicos (desde que pertençam à mesma hierarquia).

Ao criarmos um objeto, ele nasce e morre com o mesmo tipo, ao lado esquerdo podemos ter uma referência de outros tipos, graças ao polimorfismo. Isso significa que podemos colocar, também, as interfaces que ele implementa. Ou seja, um objeto nunca irá mudar o seu tipo, mas o tipo de referência para este objeto é que pode variar (genérico, específico, interface).

Polimorfismo significa "várias formas", comportamentos diferentes para um mesmo objeto.

O polimorfismo pode ser promovido através de classes abstratas e interfaces.

- Classe abstrata: é apenas o conceito da classe. Significa que não podem ser instanciadas. Para instanciar, devemos criar primeiro uma classe filha não abstrata. Para criar uma classe abstrata usa-se a palavra "abstract" e esta está **sempre** relacionada com herança. Em uma classe abstrata podemos ter métodos concretos e abstratos.

- Método abstrato: Significa que não há implementação padrão no método (sem corpo), cada classe filha obrigatoriamente precisa implementar o seu. Um método abstrato define apenas a assinatura (visibilidade, retorno, nome do método e parâmetros). Isso significa que delegamos a chamada - o método não foi embora, mas a implementação, que era concreta, agora foi delegada.

  Classes e métodos abstratos consomem menos memória e por conta disso melhoram o desempenho do nosso programa.

Interface: É uma classe abstrata e genérica por padrão, por isso não é necessário escrever a palavra “abstract”. Funciona como uma espécie de contrato. As classes que implementam a interface (que assinam o contrato) são obrigadas a implementar os métodos abstratos definidos nela. É possível implementar mais de uma interface (separadas com **,**) simultaneamente (diferentemente de uma classe abstrata “normal” que só pode ser estendida uma vez.). Isso porque, como nesta modalidade não há nada concreto (todos os métodos são sempre abstratos e sempre públicos), não corremos o risco de incorrermos em uma duplicidade de métodos.

A interface é expressada pela palavra “implements”.

public abstract interface Autenticavel

public class Administrador implements Autenticavel

podemos ler também como: "a classe Administrador assinou o contrato Autenticavel".

- Temos polimorfismo quando uma classe estende de outra ou também quando uma classe implementa uma interface.

Existem dois tipos de polimorfismo que são conhecidos como sobrecarga (overload) e sobreposição (override).

- Sobrecarga (overload) consiste em permitir, dentro da mesma classe, mais de um método com o mesmo nome. Entretanto, eles necessariamente devem possuir argumentos diferentes para funcionar. (A sobrecarga não leva em conta a visibilidade ou retorno do método, apenas os parâmetros e não depende da herança).
- Sobreposição (override) permite reescrever um método em uma subclasse que possua um comportamento diferente do método de mesma assinatura na superclasse. É necessário que os métodos tenham a mesma assinatura (tipo de retorno, nome do método, tipos

e quantidades de parâmetros), mas com implementações diferentes.

Encapsulamento: Adiciona uma camada de segurança à aplicação. Através da possibilidade de mudar a visibilidade de um atributo tornando-o privado, isso significa que ele não pode ser lido ou modificado, a não ser na própria classe. Utilizando métodos (getters e setters) para retornar e setar valor às propriedades evitando-se assim o acesso direto ao atributo.