CLASSES ABSTRATA

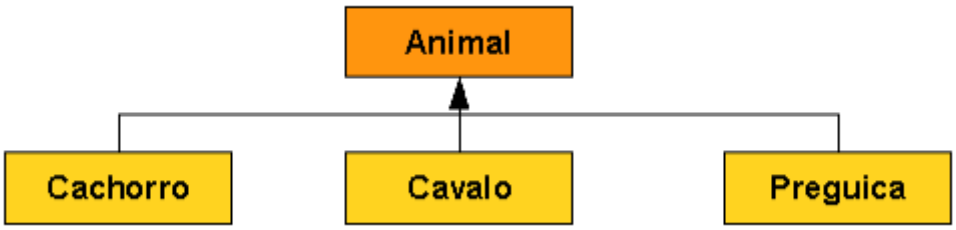
Classes Abstratas

Em Java, temos um tipo especial de classe chamado classe abstrata. Este tipo de classe possui uma característica muito específica, que é o de não permitir que novos objetos sejam instanciados a partir desta classe. Por este motivo, as classes abstratas possuem o único propósito de servirem como super classes a outras classes do Java.

Até este momento todas as classes que havíamos desenvolvido eram classes concretas, isto é todas as nossas classes podem originar objetos. Classes concretas são estruturas definidas e prontas para instanciarem objetos.

Uma classe abstrata se comporta de forma diferente de uma classe concreta. Uma classe abstrata nunca pode ser instanciada de forma direta, seu maior propósito, a razão da sua existência, é ser estendida.

Para que serve esta classe então? Imagine a seguinte situação, nós temos uma hierarquia de classe exemplo abaixo:



Todos os animais (cachorro, cavalo e preguiça) estendem da classe Animal, no entanto, no nosso, programa a classe Animal por si só não representa nenhum tipo de animal. Isto é, esta classe representa apenas um animal genérico e ela existe apenas para ser estendida dando origem a um animal de fato. Nesta situação é mais adequado que seja feita a mudança na classe Animal a fim de torná-la abstrata e inviabilizar a sua instanciação por qualquer parte do programa. Isto é realizado adicionando-se o modificador *abstract*.

Em Java definimos uma classe como abstrata utilizando a palavra-chave *abstract* na declaração da classe, por exemplo:

```
package soulcode.abstrata;
/**
* Classe Abstrata representando uma Pessoa.
*/
public abstract class Pessoa {
	private String nome;
	public Pessoa(String nome) {
		this.nome = nome;
	}
```

```
	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}
}
```

Na linha 5 estamos criando uma classe abstrata através da palavra-chave *abstract*.

Uma classe abstrata é desenvolvida para representar entidades e conceitos abstratos, sendo utilizada como uma classe pai, pois não pode ser instanciada. Ela define um modelo (template) para uma funcionalidade e fornece uma implementação incompleta - a parte genérica dessa funcionalidade - que é compartilhada por um grupo de classes derivadas. Cada uma das classes derivadas completa a funcionalidade da classe abstrata adicionando um comportamento específico.

Uma classe abstrata normalmente possui métodos abstratos. Esses métodos são implementados nas suas classes derivadas concretas com o objetivo de definir o comportamento específico. O método abstrato define apenas a assinatura do método e, portanto, não contém código assim como feito nas Interfaces.

Uma classe abstrata pode também possuir atributos e métodos implementados, componentes estes que estarão integralmente acessíveis nas subclasses, a menos que o mesmo seja do tipo `private`.

Como todo método abstrato precisa ser implementado pela classe filha, então não pode ser private, pois não seria visível na subclasse.

Neste exemplo, criaremos uma classe abstrata chamada `Tela`. Nesta classe implementaremos alguns métodos que devem ser utilizados por todas as telas do computador de bordo de um veículo, como: `setTitulo()` para informar o título da tela e `imprimir()`, que imprime as informações na tela.

Nesta classe definiremos também a assinatura de um método abstrato chamado `obterInformacao()`, que deve ser implementado pelas classes filhas.

```
package material.abstrata;
/**
* Classe abstrata que possui os métodos básicos para
* as telas do computador de bordo de um veiculo.
*/
public abstract class Tela {
	private String titulo;
```

```
	public void setTitulo(String titulo) {
		this.titulo = titulo;
	}

	public abstract String obterInformacao();

	public void imprimir() {
		System.out.println(this.titulo);
		System.out.println("\t" + obterInformacao());
	}
}
```

Agora criaremos a classe `TelaKilometragem`, que será uma subclasse da classe abstrata `Tela`. Esta classe será utilizada para mostrar a km atual percorrida pelo veículo.

```
package soulcode.abstrata;

/**
* Tela que mostra a kilometragem percorrida por um veiculo.
*/
public class TelaKilometragem extends Tela {
	/* Atributo que guarda o valor da km atual do veiculo. */
	int km = 0;


	/**
	* Construtor que iniciliza o titulo da tela.
	*/
	public TelaKilometragem() {
		/* Atribui o valor do titulo desta tela. */
		super.setTitulo("Km Atual");
	}

	/**
	* Implementa o método abstrato da classe Tela
	* neste método buscamos a km atual do veiculo.
	*
	* @return Texto com a km atual.
	*/
	@Override
	public String obterInformacao() {
		km += 10;
		return String.valueOf(km) + " km";
	}
}
```

Vamos, agora, criar uma classe para testar a nossa aplicação. Neste teste, criaremos um objeto da classe `TelaKilometragem` e chamar o método `imprimir()`, que esta classe herdou da classe `Tela`.

```
package soulcode.abstrata;
/**
* Classe utilizada para testar a Tela Kilometragem.

 */
public class TesteTelaKm {
	public static void main(String[] args) {
		TelaKilometragem tk = new TelaKilometragem();
		tk.imprimir();

		System.out.println("\n—----------\n");

		tk.imprimir();
	}
}
```

Quando definimos `Tela tk = new TelaKilometragem()`, estamos guardando um objeto do tipo `TelaKilometragem` dentro de uma variável do tipo `Tela`. Podemos fazer isso porque toda `TelaKilometragem` é uma subclasse de `Tela`.

Quando chamamos o método `tk.imprimir()`, estamos chamando o método que foi implementado na classe abstrata Tela, mas note que o método `imprimir()` usa o método `obterInformacao()`, que foi implementado na classe `TelaKilometragem`. Em tempo de execução, a JVM verifica o tipo do objeto e chama o método que foi implementado nele, isto chama-se **Polimorfismo**.

Temos a seguinte saída no console:

```
Km Atual
		10 km
-----------

Km Atual
		20 km
```

Segue abaixo um quadro comparativo entre Interface e Classe Abstrata:

Classe Abstrata	**Interface**
Pode possui atributos de instância	Possui apenas constantes
Possui métodos de diversas visibilidade e métodos implementados ou abstratos	Todos os métodos são *public* e *abstract*
É estendida por classes (sub-classes)	É implementada por classes
Uma subclasse só pode estender uma única classe abstrata	Uma classe pode implementar mais de uma *interface*
Hierarquia de herança com outras classes abstratas	Hierarquia de herança com outras interfaces