Capítulo 4 – Polimorfismo

1. Polimorfismo: “Vamos nos adaptar”

1.1 Aprendendo a prever o futuro

O encapsulamento permite construir componentes de software independentes e a herança permite reutilizar e estender esses componentes. Entretanto, ainda falta algo. O software está sempre mudando. Se os usuários exigem nova funcionalidade, erros aparecem ou o software precisa ser integrado em novos ambientes, a única constante é a mudança. O ciclo de vida do software não termina quando você distribui um produto. Você precisa de software que possa se adaptar às necessidades futuras. Não seria ótimo, se você pudesse escrever software ‘à prova do futuro’?

Um software à prova do futuro se adapta aos requisitos futuros sem alterações e permite que você faça alterações e adicione novos recursos facilmente. A POO utiliza o conceito de polimorfismo para permitir que softwares a prova do futuro sejam escritos.

1.2 O que é Polimorfismo

Polimorfismo significa ter ‘muitas formas’, que significa um único nome representando um código diferente, selecionado por algum mecanismo automático. “Um nome, vários comportamentos”.

O Polimorfismo não é um pensamento novo para nós. Ele está contido em nosso dia a dia, principalmente na linguagem. Veja os exemplos:

1. Ontem sai para **dançar** com uns amigos, mas acabamos **dançando** porque não conseguimos encontrar um lugar que nos agradasse.
2. José **cantou** a noite inteira no Karaoke e João **cantou** a noite inteira a namorada de José.
3. *O cachorro do João* não foi passear ontem.

Pensando mais em objetos e funcionalidades, pense agora no termo *abrir*, por exemplo. Você pode abrir uma porta, uma caixa, uma janela e uma conta bancária. A palavra *abrir* pode ser aplicada a muitos objetos do mundo real sendo que cada objeto interpreta ‘abrir’ de sua própria maneira. Porém, você pode simplesmente dizer ‘abrir’, para descrever a ação.

Uma linguagem polimórfica é a que suporta polimorfismo (Actionscript, Java), já a linguagem monomórfica não suporta polimorfismo (Pascal, ASP).

O Flash ainda não suporta recursos avançados de polimorfismos, só aceitando funções de sobreposição, veremos adiante.

Observe o exemplo a seguir:

Actionscript

Obs: Cada classe abaixo será um arquivo diferente com o respectivo nome.

(ObjetoPersonalidade.as)

```
class ObjetoPersonalidade
{
       public function comportamento():String
```

```
	{
		return "Eu sou um objeto.";
	}
	public function fale()
	{
		trace(comportamento());
	}
}
```

(ObjetoPessimista.as)

```
class ObjetoPessimista extends ObjetoPersonalidade {
	public function comportamento():String {
		return "Nada é tão ruim que não possa piorar.";
	}
}
```

(ObjetoOtimista.as)

```
class ObjetoOtimista extends ObjetoPersonalidade {
	public function comportamento():String {
		return "Amanhã sempre será um dia melhor.";
	}
}
```

(ObjetoIntrovertido.as)

```
class ObjetoIntrovertido extends ObjetoPersonalidade {
	public function comportamento():String {
		return "Cuti...";
	}
}
```

(ObjetoExtrovertido.as)

```
class ObjetoExtrovertido extends ObjetoPersonalidade {
	public function comportamento():String {
		return "E aí!!! Vamos agitar...";
	}
}
```

(Arquivo FLA para teste)

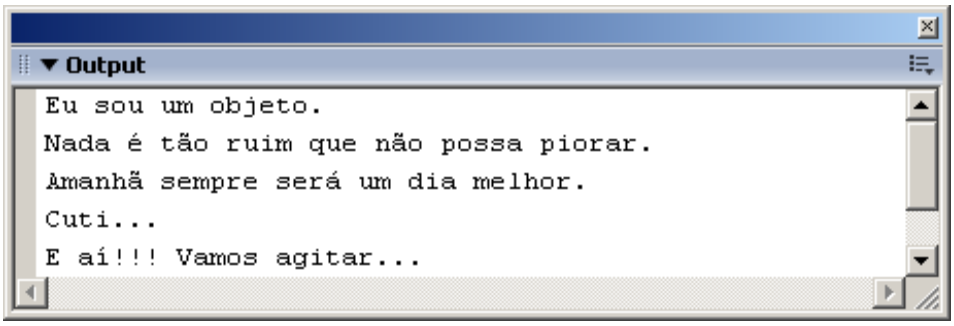
```
var personalidade: ObjetoPersonalidade = new ObjetoPersonalidade();
var pessimista: ObjetoPessimista        = new ObjetoPessimista();
var otimista: ObjetoOtimista            = new ObjetoOtimista();
var introvertido: ObjetoIntrovertido = new ObjetoIntrovertido();
var extrovertido: ObjetoExtrovertido   = new ObjetoExtrovertido();

var personalidades = [
	personalidade,
	pessimista,
	otimista,
	introvertido,
	extrovertido
];

for(i=0; i<5; i++)
{
	var persona:ObjetoPersonalidade  = personalidades[i];
	persona.fale();
}
```

Java

```
public class ObjetoPersonalidade
{
	public String comportamento()
	{
		return "Eu sou um objeto.";
	}
	public void fale()
	{
		System.out.println(comportamento());
	}
}
public class ObjetoPessimista extends ObjetoPersonalidade
{
	public String comportamento()
	{
		return "Nada é tão ruim que não possa piorar.";
	}
}
public class ObjetoIntrovertido extends ObjetoPersonalidade {
	public String comportamento()
	{
		return Cuti...";
	}
}
public class ObjetoExtrovertido extends ObjetoPersonalidade {
	public String comportamento()
	{
		return "E aí!!! Vamos agitar...";
	}
}
public class ObjetoOtimista extends ObjetoPersonalidade {
	public String comportamento()
	{
		return "Amanhã sempre será um dia melhor.";
	}
}

public class ExecutaObjetosComPersonalidade {

	public static void main(String[] args) {
		ObjetoPersonalidade personalidade = new ObjetoPersonalidade();
		ObjetoPessimista pessimista       = new ObjetoPessimista();
		ObjetoOtimista otimista           = new ObjetoOtimista();
		ObjetoIntrovertido introvertido   = new ObjetoIntrovertido();
		ObjetoExtrovertido extrovertido   = new ObjetoExtrovertido();

		ObjetoPersonalidade[] personalidades = {

				personalidade,
				pessimista,
				otimista,
				introvertido,
				extrovertido
		};
		for(int i=0; i<5; i++)
		{
			ObjetoPersonalidade persona = personalidades[i];
			persona.fale();
		}
	}
}
```

Essas classes formam uma hierarquia de herança muito simples. A classe base, *ObjetoPersonalizado* declara o método *fale()*. Esse método é redefinido pelas subclasses filhas. A hierarquia forma relacionamentos com capacidade de substituição entre os progenitores.

Resultado do arquivo teste.



Os filhos de *ObjetoPersonalizado* são instanciados um a um e depois armazenados em um vetor de *ObjetoPersonalizado*. É aqui que o polimorfismo está fazendo o seu papel. Embora as instâncias de *ObjetoPersonalizado* estejam com a 'casca' de seu pai, ou seja, a interface de métodos visíveis é a de seu pai, a implementação é cabível a instância. Quando se percorre o vetor e se chama a mensagem *fale()* cada instância envia uma mensagem diferente, como você pode verificar no exemplo acima.

Você pode explorar o polimorfismo para adicionar nova funcionalidade em seu sistema, a qualquer momento. Você pode adicionar novas classes, que tenham funcionalidades ainda não imaginadas quando o programa foi escrito pela primeira vez – isso sem ter que mudar o código já existente.

Note também que em OO o conceito de polimorfismo está diretamente ligado aos conceitos de encapsulamento e herança, onde um conceito completa o outro para que o "sistema" (paradigma) funcione na teoria e na prática.

1.3 Tipos de Polimorfismo

1. Sobreposição
2. Polimorfismo de inclusão
3. Polimorfismo paramétrico
4. Sobrecarga

Existem Várias Tipos de Polimorfismo, porém, o único aceito pelo Flash é o Polimorfismo por sobreposição.

Sobreposição

Quando ocorre a sobreposição de um método de uma classe filha em cima do método da classe pai, não quer dizer que o método da classe pai foi destruído. A classe filha

herda sempre o método como um método recursivo da classe pai, ou seja, nada impede que o método da classe filha utilize o método da classe herdada.

Lembre-se do exemplo dado em herança *TwoDimensionalPoint* e *ThreeDimensionalPoint*.

1.4 Perguntas - Exercício

1) Como o polimorfismo e a sobreposição trabalham juntos?

Ao terminar de responder as questões acima, envie-as em um arquivo texto ao seu tutor.