



PROGRAMAÇÃO ORIENTADA A OBJETOS

Engenharia de Software II

07/03/2003 – Fabiano Reese Righetti (fabiano@cascavel.pm.org)

PROGRAMAÇÃO ORIENTADA A OBJETOS (POO)

Este tipo programação pode ser considerado uma extensão da programação estruturada ou se preferir programação modular, tendo em vista várias melhorias como maior agilidade, redução de complexidade (conseqüentemente aumenta a produtividade), facilidade de manutenção, diminuição de custos, reutilização de códigos, etc.

Tudo começou na década de 60 na Noruega com a criação da linguagem Simula (Simula-67) cujo nome vem da palavra "Simulação" onde tinha o conceito de similaridade com o mundo real. Depois começou a surgir outras linguagens como Simula-68 e Smalltalk criada pela Xerox sendo esta a primeira totalmente orientada a objeto e que popularizou o termo, isto em 1970. Depois desta grande evolução foram criadas várias outras linguagens como C/C++ (da AT&T em 1980), Object Pascal (da Apple em 1986) e Java (da Sun em 1990). Hoje em dia os conceitos Orientação a Objeto também é utilizado em Sistemas Operacionais, Banco de Dados, etc. A POO emprega os seguintes conceitos básicos objeto, abstração, encapsulamento, classe, herança e polimorfismo.

1. Classe

Uma classe é formada por atributos e métodos no qual representa um conjunto de objetos, definindo então as características e as funcionalidades do mesmo.

```
public class NomeDaClasse
{
    // Atributos.
    ...

    // Métodos construtores.
    ...

    // Métodos set() e get().
    ...

    // Métodos de trabalho.
    ...
}
```

Existem ainda dois tipos de classes, a chamada "especificação" (abstrata) e a "final". A classe especificação ocorre quando nesta só é especificado o que as classes filhas devem ter, sem implementar nenhuma funcionalidade.

```
public class abstract Especialista
{
    ...

    public abstract float a ();
    public abstract float b ();
    public abstract float c ();

    ...
}
```

A classe final como o próprio nome já diz são as classes terminais/finais que através dela não podem ser criadas outras classes, ou seja, não podem ser herdadas ou se preferir especializadas.

```
public final class Terminal
{
    ...
}
```

- public** A classe é "vista" por todos.
- protected** Restringe a apenas classe do mesmo pacote poder vê-la.
- abstract** É apenas uma definição do método (declaração/cabeçalho).
- final** Significa que a classe não pode ser herdada

1.1. Atributos

São as características do objeto, ou seja, são as propriedades, estes podendo ser public, protected, private ou final, com característica de ser atributos da classe ou do objeto.

```
class Ponto2D
{
    // Atributos do objeto.
    private float x, y;

    // Atributos da classe.
    public static final float z = 8.456124234; // Constante.

    ...
}
```

- public** Vai ser de livre acesso podendo ser alterado diretamente pelos Objetos.
- protected** Significa que estes atributos só poderão ser alterados pela classe pai ou pelas classes filhas.
- private** São os atributos que só podem ser alterados pelos métodos da classe pai.
- static** Significa que este atributo pertence somente a classe.
- final** É um valor que não pode ser alterado pela aplicação.

Um exemplo de atributo da classe é o Math.PI que é declarado por **public static final**:

```
float meioPI = Math.PI/2;
```

1.2. Métodos

É o comportamento que o objeto vai ter, ou seja, que funcionalidade podem ser aplicada a seus atributos locais ou da classe. Estes métodos podem ser: **public**, **protected**, **private** ou **final**. Existem quatro tipos de métodos:

O primeiro deles é o método construtor que é chamado sempre quando um objeto é criado, tendo a função de inicializar os atributos do objeto.

```

class Ponto2D
{
    ...

    public Ponto2D ()
    {
        this.x = this.y = 0;
    }

    ...
}

```

O segundo são os métodos set() e get() no qual são responsáveis por determinar a "visibilidade" dos atributos tendo a funcionalidade de setar (set) e retornar (get) os valores dos atributos.

```

class Ponto2D
{
    ...

    public void setX (float x)
    {
        this.x = x;
    }

    public void setY (float y)
    {
        this.y = y;
    }

    public float getX ()
    {
        return(this.x);
    }

    public float getY ()
    {
        return(this.y);
    }

    ...
}

```

O terceiro são os métodos de trabalho no qual se divide em dois, métodos do objeto e métodos da classe. Os métodos do objeto são os responsáveis em trocar mensagens com o objeto desempenhando alguma funcionalidade. Os métodos da classe (static) vão ser chamados pela classe e não mais pelos objetos, trocando então mensagens com classe.

```

class Ponto2D
{
    ...

    // Método do Objeto.
    public void algumaCoisa ()
    {

```

```

        return(1);
    }

    // Método da Classe.
    public static void distancia (Ponto2D p, Ponto2D q)
    {
        float a = p.getX() - q.getX(),
              b = p.getY() - q.getY();

        return(Math.pow( (a * a) + (b * b), 0.5));
    }
    ...
}

```

O quarto são os métodos finais no qual tem a característica de não poder ser redefinido em classes derivadas.

```

class Ponto2D
{
    ...

    public static final void distancia (Ponto2D p, Ponto2D q)
    {
        float a = p.getX() - q.getX(),
              b = p.getY() - q.getY();

        return(Math.pow( (a * a) + (b * b), 0.5));
    }
    ...
}

```

- public** O método pode ser utilizado por qualquer objeto e classe.
- protected** Só pode ser acessado pelas classes relacionadas.
- private** Somente objetos/métodos da própria classe podem manipulá-lo.
- static** Defini um método da classe.
- final** Proibi a redeclaração do método em classes derivadas.

1.3. Herança

É o mecanismo que permite você "agrupar" as características comuns pertencentes outras classes, ou seja, as classes filhas vão herdar as os atributos e métodos da classe pai podendo também acrescentar.

```

public class Pai
{
    // Atributos.
    private float x;

    // Métodos construtores.
    public Pai ()
    {
        this.x = 0;
    }
}

```

```

    }

    public Pai (float x)
    {
        this.x = x;
    }

    ...
}

public class Filha extends Pai
{
    // Atributos.
    private float y;

    // Métodos construtores.
    public Filha ()
    {
        super();
        this.y = 0;
    }

    public Filha (float x, float y)
    {
        super(x);
        this.y = y;
    }

    ...
}

```

extends Significa que vai “estender” uma classe (herdar).
super Serve para chamar o construtor da classe pai.

1.3.1. Especialização

Isto ocorre quando você herda uma classe e a partir desta você adiciona novos atributos métodos fazendo assim uma especialização.

1.3.2. Generalização

É quando a partir de várias classes é criada uma outra classe contendo apenas os atributos e métodos comuns entre as classes.

1.4. Polimorfismo

Este conceito é muito interessante, pois trata de ter métodos com o mesmo nome, mas com funções diferentes.

1.4.1. Estático

Polimorfismo estático ou se preferir sobrecarga, este métodos tem o mesmo nome, mas possuem parâmetros diferentes, ou seja, a sua funcionalidade difere. Isto é possível porque é verificado em tempo de compilação qual será o método executado.

```
class Ponto2D
```

```

{
    ...

    public Ponto2D ()
    {
        this.x = this.y = 0;
    }

    public Ponto2D (float x, float y)
    {
        this.x = x;
        this.y = y;
    }

    public Ponto2D (Ponto2D p)
    {
        this.x = p.getX();
        this.y = p.getY();
    }

    ...
}

```

1.4.2. Dinâmico

Trata-se de que vai ser definido qual método chamar em tempo de execução. Significa que as classes filhas derivadas da classe pai vão ter os mesmos métodos (cabeçalho), mas sendo especializados para cada classe.

```

public abstract class Pai
{
    ...

    public abstract float a ();
    public abstract float b ();
    public abstract float c ();
}

public class Filha extends Pai
{
    ...

    public float a ()
    {
        ...
    }

    public float b ()
    {
        ...
    }

    public float c ()
    {
        ...
    }
}

```

abstract É apenas uma definição do método (declaração/cabeçalho) e através deste não é possível criar objetos. Se uma classe possui um método abstract obrigatoriamente a classe se torna abstract.

2. Objeto

São instâncias das classes, sendo através deles que efetuado todos o processamento do programa seguindo os conceitos das classes, ou seja, os objetos são criados através das classes para através dele realizar o trabalho.

```
int vetor[] = new int[100];  
NomeDaClasse teste = new NomeDaClasse();
```

Tipos:

boolean	Valor lógico true/false.
char	Caractere.
String	Conjunto de caracteres.
byte	Inteiro 8 bits (-128 á +127).
short	Inteiro 16 bits (-32768 á +32767).
int	Inteiro 32 bits (-2147483648 á +2147483647).
long	Inteiro 64 bits (-9223372036854775808 á +9223372036854775807).
float	Real 32 bits com nove dígitos significativos.
double	Real 64 bits com dezoite dígitos significativos.

2.1. Encapsulamento

Cada objeto contém internamente toda a informação necessária para sua manipulação, tornando-se uma agregação de informações (encapsulamento).

2.2. Ocultamento de Informações

Para manipular um objeto você não precisa saber como os resultados são manipulados/criados, é necessário saber somente os cabeçalhos/assinaturas dos métodos ou ainda atributos públicos que se possa manipular. Criando então um nível de abstração alto para o programador, ou seja, ele não precisa se preocupar em como obter o resultado e sim somente interpretar o mesmo.