

Paradigma e linguagem orientada a objetos

Fabio Takeshi Matsunaga

Universidade Estadual do Paraná

7 de Abril de 2015

Conteúdo

Missão da aula de hoje

- Breve introdução a paradigmas de programação
- Programação estruturada/Procedural X Orientada a objetos
- Por que estudar orientação a objetos?
- Qual a importância deste paradigma de programação e por que ela está presente na maioria das coisas?
- Principais conceitos
- Uma breve apresentação de exemplos e implementações
- Atividades a serem desenvolvidas em sala e para próxima aula

Paradigmas de Programação – o que são?

- A forma como um programador enxerga um programa e a sua execução para solução de um problema
- Define a forma como o programador vai abstrair um problema e estruturar seu programa
- Dependendo do paradigma adotado, algumas restrições de implementações são estabelecidas, como tipo de estrutura de controle, abstrações e tipos de dados utilizados
- Uma linguagem pode adotar mais de um paradigma

Paradigmas de Programação

- Não existe um melhor paradigma ou um universal
- Cada categoria de problema adota um paradigma
 - A chave de fenda para apertar parafuso
- É importante saber alguns paradigmas para encontrar bugs específicos em um determinado programa

Exemplos de paradigmas e linguagens

- **Programação imperativa** (C, Python, Java)
 - Descrição passo a passo de como um programa deve funcionar
 - Definição de um conjunto de comandos e instruções
 - Comparável a uma “receita de bolo”
- **Programação declarativa** (SQL, Prolog)
 - O programador basicamente descreve o que o código faz e não como
 - Não usa variáveis nem atribuições

Exemplos de paradigmas e linguagens

- **Programação procedural** (C, C++, Java, Python)
 - Paradigma derivado da programação imperativa
 - Comandos organizados em grupos denominados procedimentos ou funções
 - Os procedimentos podem ou não ter parâmetros ou retornos
- **Programação funcional** (Lisp, Haskell, Scheme)
 - Uso de funções matemáticas puras, para criar novos objetos a partir dos existentes, sem uso de atribuições e alteração dos argumentos
 - Existe interação entre as funções onde uma função serve de parâmetro para outra
 - Não se declara variáveis

Exemplos de paradigmas e linguagens

- **Programação estruturada** (C, Pascal)
 - Programas descritos através de mecanismos básicos (sequência, seleção, iteração e modularização)
- **Orientada a Objetos** (Smaltalk, C++, Java)
 - Programas descritos através objetos que trocam mensagens entre si

Aplicações de alguns paradigmas

- **Computação científica:** FORTRAN, Scilab, Matlab
- **Aplicações comerciais:** COBOL
- **Lógica Matemática e IA:** LISP, Prolog
- **Software básico e fins didáticos:** C, C++, Python
- **Desenvolvimento de sistemas e aplicações:** Java, Python

Linguagem estruturada X Orientada a objetos

- Programação estruturada (ou imperativa) – útil para sistemas de médio porte, sistemas embarcados e para programadores com alto conhecimento de hardware
 - Desempenho melhor!!!
 - Sem muita complexidade na execução dos códigos
- A complexidade de aplicações mais modernas e avançadas motivaram a criação de novos paradigmas
 - Linguagens de níveis mais altos
 - O alto desempenho das máquinas deixou o desempenho das linguagens em segundo plano

Linguagem estruturada X Orientada a objetos

- **Programação estruturada**

- Estrutura sequencial
- Estrutura de decisão
- Iteração (repetição)
- Subprogramas (procedimentos e funções)

- **Orientação a objetos**

- Uso de estruturas de dados (objetos)
- Interação entre objetos por trocas de mensagens

Linguagem estruturada X Orientada a objetos



Resolução de um problema (atividade)

Calcular a área e o perímetro de um retângulo e mostrar as respostas na interface de um sistema

Programação estruturada

- 1 Leia h = altura do retângulo
- 2 Leia l = valor da largura do retângulo
- 3 $\text{Área} = h * l$
- 4 $\text{Perímetro} = 2 * (h + l)$
- 5 Imprima Área
- 6 Imprima Perímetro

Programação Orientada a Objetos

- ❶ Procurar objetos
- ❷ Determinar as características de cada objeto
 - Atributos
 - Ações
- ❸ Determinar como cada objeto irá interagir com outro

Programação Orientada a Objetos

- ❶ Procurar objetos – Retângulo
- ❷ Determinar as características de cada objeto
 - Atributos – altura e largura
 - Ações – calcular área e perímetro
- ❸ Determinar como cada objeto irá interagir com outro – objeto "Janela" ou o programa principal

Orientação a Objetos

- Termo criado por Alan Kay no início da década de 1960 que definiu um novo paradigma de linguagens de programação
- Paradigma criado para facilitar a simulação do mundo real dentro do computador
- Características
 - Modelar os objetos na forma que o computador possa entendê-lo
 - Comunicação e interação entre esses objetos, através do envio de mensagens
 - Definição de propriedades e ações a serem tomadas

Linguagens de Programação Orientada a Objetos

- Java, C++, Ruby, Python, entre outros
- Linguagem C++
 - Linguagem derivada de C
 - Procedural e orientada a objetos
 - Tem mais liberdade com uso da memória e hardware

Conceitos básicos

- Classe
- Encapsulamento
- Herança
- Polimorfismo
- Instância de objetos

Classe

- Tipo definido de um determinado objeto
 - Atributos
 - Métodos

Exemplo de uma classe

```
class Funcionario {  
    public:  
        char nome[50];  
        long int cpf;  
        float salario;  
  
        Funcionario() {  
            salario = 1000;  
        }  
  
        void aumentoSalario() {  
            salario = salario+salario*0.1;  
        }  
};
```

Exemplo de uma classe

```
int main()
{
    Funcionario f; // Instancia de classe

    printf("Salario atual: %f \n",f.salario);
    f.aumentoSalario(); // Chamada de um metodo
    printf("Novo salario: %f \n",f.salario);
}
```

Saída

```
Salario atual: 1000.000000
Novo salario: 1100.000000
```

Exemplo de uma classe

Manipulação livre dos dados, sem proteção e segurança

```
f.salario = 50000;  
printf("Novo salario: %f \n",f.salario);
```

Saída

Novo salario: 50000.000000

Encapsulamento

- Define a separação e proteção de dados internos de um objeto
- Usado para impedir acesso direto aos atributos do objeto
- Exemplo: dados pessoais de um cliente. Definição do que é público ou privado

Versão do código anterior encapsulado

```
class Funcionario {  
    private:  
        char nome[50];  
        long int cpf;  
        float salario;  
  
    public:  
        Funcionario() {  
            salario = 1000;  
        }  
  
        void aumentoSalario() {  
            salario = salario+salario*0.1;  
        }  
};
```


Erro

Erro ao processar o código: não permitida a chamada direta dos atributos.

```
int main()
{
    Funcionario f; // Instancia de classe

    printf("Salario atual: %f \n",f.salario);
    f.aumentoSalario(); // Chamada de um metodo
    printf("Novo salario: %f \n",f.salario);

    f.salario = 50000;
    printf("Novo salario: %f \n",f.salario);
}
```

Getters e Setters

Uso dos *getters* e *setters*

public:

```
Funcionario() {  
    salario = 1000;  
}
```

```
char* getNome() {  
    return nome;  
}
```

```
long int getCPF() {  
    return cpf;  
}
```

```
float getSalario() {  
    return salario;  
}
```

```
void setNome(char *n) {  
    strcpy(nome,n);  
}
```

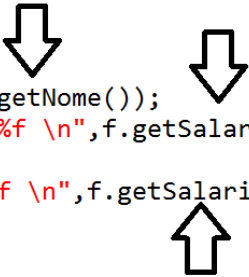
```
void setCPF(long int n) {  
    cpf = n;  
}
```

```
void aumentoSalario() {  
    salario = salario + salario*0.1;  
}
```

Chamada dos métodos

Versão correta das chamada dos métodos e acesso aos atributos.

```
int main()
{
    Funcionario f;
    f.setNome("Fabio");
    printf("Nome: %s \n",f.getNome());
    printf("Salario atual: %f \n",f.getSalario());
    f.aumentoSalario();
    printf("Novo salario: %f \n",f.getSalario());
}
```



Definição de vários funcionários

Repetição e redundância de dados e métodos.

```
class Gerente {
private:
    char nome[50];
    long int cpf;
    float salario;
    int id;
    int senha;
public:
    int getID() {
        return id;
    }
    int getSenha() {
        return senha;
    }
    char* getNome() {
        return nome;
    }
    long int getCPF() {
        return cpf;
    }
    float getSalario() {
        return salario;
    }
    void setNome(char *n) {
        strcpy(nome,n);
    }
    void setCPF(long int n) {
        cpf = n;
    }
    void aumentoSalario() {
        salario = salario + salario*0.1;
    }
};
```

```
class Secretaria {
private:
    char nome[50];
    long int cpf;
    float salario;
    int ramal;
public:
    int getRamal() {
        return ramal;
    }
    int setRamal(int r) {
        ramal = r;
    }
    char* getNome() {
        return nome;
    }
    long int getCPF() {
        return cpf;
    }
    float getSalario() {
        return salario;
    }
    void setNome(char *n) {
        strcpy(nome,n);
    }
    void setCPF(long int n) {
        cpf = n;
    }
    void aumentoSalario() {
        salario = salario + salario*0.1;
    }
};
```

Herança

- Recurso para uma classe herdar atributos e métodos de uma classe
- **Superclasse:** classe pai
- **Subclasse:** classe filho ou herdada
- Vantagem: reutilização de código

Herança em C++

Estrutura de uma herança:

class classe-derivada : **public** classe-base

```
class Secretaria : Funcionario {  
    private:  
        int ramal;  
    public:  
        int getRamal() {  
            return ramal;  
        }  
        int setRamal(int r) {  
            ramal = r;  
        }  
};
```

```
class Gerente : Funcionario {  
    private:  
        int id;  
        int senha;  
    public:  
        int getID() {  
            return id;  
        }  
        int getSenha() {  
            return senha;  
        }  
        void setID(int i) {  
            id = i;  
        }  
        void setSenha(int s) {  
            senha = s;  
        }  
};
```

Instância das subclasses

Chamada de um método da classe **Funcionario** através de um objeto da classe **Secretaria**.

Chamada também dos métodos próprios da classe **Secretaria**.

```
Secretaria s;  
s.setNome("Maria");  
s.setRamal(1234);  
printf("Nome: %s \n",s.getNome());  
printf("Ramal: %d \n",s.getRamal());
```

Saída

Nome: Maria

Ramal: 1234

Chamada dos métodos de uma subclasse

Um objeto da superclasse não pode acessar os dados de uma subclasse

```
Funcionario f;  
f.setRamal(123);
```

```
Secretaria s;  
s.setNome("Maria");  
s.aumentoSalario()
```


Chamada dos métodos de uma subclasse

Uso de polimorfismo

Métodos virtuais (**virtual**) para definir um método que pode ser reimplementado por uma subclasse

```
class Funcionario {  
    protected:  
        char nome[50];  
        long int cpf;  
        float salario;  
    public:  
        Funcionario() {  
            char* getNome() {  
                long int getCPF() {  
                    float getSalario() {  
                        void setNome(char *n) {  
                            void setCPF(long int n) {  
  
                            virtual void aumentoSalario() {  
                                salario = salario + salario*0.1;  
                            }  
                        }  
                    }  
                }  
            }  
};
```

```
class Gerente : public Funcionario {  
    private:  
        int id;  
        int senha;  
    public:  
        int getID() {  
            int getSenha() {  
                void setID(int i) {  
                    void setSenha(int s) {  
  
                    void aumentoSalario() {  
                        salario = salario + salario*0.3;  
                    }  
                }  
            }  
};
```

Instância das subclasses

Uso de um método com o mesmo nome para diferentes objetos. A instância de uma classe é encarregada de reconhecer qual método deve ser chamado.

```
Funcionario f;  
f.aumentoSalario();  
printf("Novo salario (funcionario): %f \n",f.getSalario());  
  
Gerente g;  
g.aumentoSalario();  
printf("Novo salario (gerente): %f\n",g.getSalario());
```

Saída

```
Novo salario (funcionario): 1100.000000  
Novo salario (gerente): 1300.000000
```

Resumindo ...

Revisão

- Importância do paradigma de orientação a objetos
- Linguagem C++
- Classes e instância de classes
- Encapsulamento
- Herança
- Polimorfismo
- Exemplos de uso no código-fonte

Atividade

1) Defina uma classe denominada Ser Vivo (SerVivo) e todas as subclasses possíveis que podem ser herdadas dela, utilizando o conceito de Herança de Classe vista na aula. Estabeleça todos os atributos e métodos possíveis deixando claro o que é geral de um Ser Vivo e o que é específico de cada subclasse.

2) Para próxima aula, pesquisar e descrever linguagens que utilizem o paradigma de orientação a objetos. Também as respectivas plataformas e ambientes de desenvolvimento (IDEs) para tais linguagens.

Bibliografia recomendada

- Sebesta, R. W. (2003). Conceitos de Linguagens de Programação. Bookman.

Obrigado! Fabio Takeshi Matsunaga ftakematsu@gmail.com