Programação Imperativa e Orientada a Objeto

Paradigmas de Programação – BCC/UFRPE
Lucas Albertins – lucas.albertins@deinfo.ufrpe.br

Agenda

- Paradigma Imperativo
- Arquitetura de um programa imperativo-procedural
- Paradigma Orientado a Objetos
- Objeto e Classe
- Herança
- Vinculação Dinâmica
- Decisões de Projeto
- Implementando Construções OO

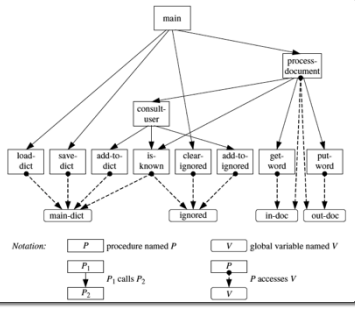
Paradigma Imperativo

- Termo "Imperativo" porque o programa dá ordens
- Nasceu na década de 50 e predominou até a década de 90
- Variáveis e atribuições como abstrações de manipulação de memória
- Boa parte dos softwares que estão rodando pelo mundo foram construídos de acordo com o paradigma imperativo
- Abre portas para outros paradigmas mais modernos (orientação a objeto)

Paradigma Imperativo

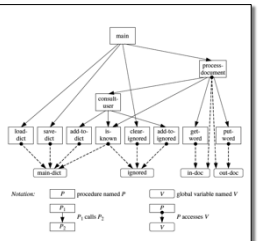
- Conceitos chave
  - Variáveis
    - Modela estado do mundo real
  - Comandos
    - Ordens
- Subdivisões
  - Procedural (Estruturado)
    - Abstração sobre um conjunto de comandos
    - Separação de interesses – comportamento e algoritmo do procedimento
    - Exemplos de linguagens: C e Pascal
  - Orientação a Objeto: Java, C++, C#

Arquitetura de um programa Imperativo procedural

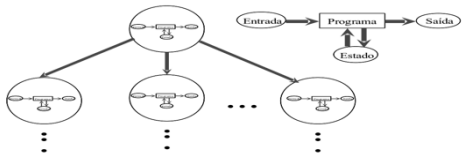


Arquitetura de um programa Imperativo - procedural

- Problemas
  - Manutenção
  - Legibilidade
- Solução?
  - Abstração de Dados
  - Orientação a objeto



O Paradigma Orientado a Objetos



Orientação a Objeto

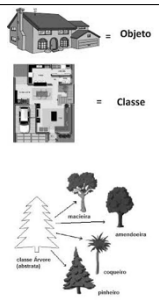
- Derivação do paradigma Imperativo/Procedural
- Início: Smalltalk em 1980
- Algumas suportam programação procedural e orientada a dados: C++ e Ada 95+
- Algumas não suportam outros paradigmas mas usam estruturas imperativas: Java e C#
- Algumas são puramente OO: Smalltalk e Ruby

Orientação a Objeto

- Conceitos chave:
  - Tipos Abstratos de Dados (TADs)
    - Classes
  - Objetos
  - Herança
  - Vinculação dinâmica (Dynamic binding)
    - Polimorfismo

Objeto e Classe

- Classe: Família de objetos
  - Forma de Encapsulamento de um TAD
- Objeto: Forma natural de modelar entidades do mundo real e do mundo virtual
  - Possuem variáveis (atributos) e operações (métodos) que manipulam tais variáveis
  - Instância de uma classe



Herança

- Aumento de produtividade pode vir de reuso
  - TADs são difíceis de reusar
  - TADs estão sempre no mesmo nível
- Herança permite a definição de novas classes a partir de classes existentes
  - Herda-se dados e operações
- Resolve o problema de reuso com pequenas mudanças e define hierarquia de classes
  - Classe pai: superclasse
  - Classe filha: subclasse

Herança

- Ocultação de Informação pode alterar mecanismos de herança
  - Uma classe pode esconder entidades das suas subclasses
  - Uma classe pode esconder entidades dos seus clientes enquanto permite que suas subclasses as vejam
- Além disto subclasses podem:
  - Sobrecarregar métodos herdados
  - Adicionar variáveis e/ou métodos aqueles herdados da classe pai
- Herança Simples vs Herança Múltipla

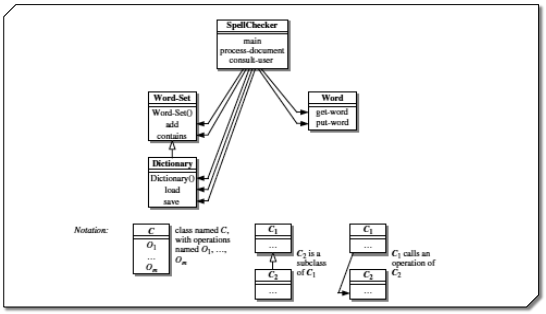
Relacionamentos entre classes

- Classes podem se relacionar através de:
  - Dependência
    - Uma classe chama operações de outra classe
  - Herança
    - Uma classe é subclasse de outra classe
  - Confinamento
    - Objetos de uma classe contém objetos de uma outra classe
      - Composição
      - Agregação

Relacionamentos entre classes

- Um dos problemas de manutenabilidade de código é o alto acoplamento entre classes
- Componentes públicos de uma classe geram alto acoplamento
- Herança também é um relacionamento de alto acoplamento
  - Mudanças nas superclasses têm impacto nas subclasses
  - Mesmo para componentes privados podem ser manipulados através da invocação de métodos pelas subclasses

Arquitetura de um programa orientado a objeto



Vinculação Dinâmica (Polimorfismo)

- Objetos de uma subclasse é tratado como um objeto da superclasse
  - Ex: Uma coleção de objetos de diferentes classes
- Uma variável polimórfica pode referenciar (ou apontar para) objetos de sua classe ou objetos de qualquer uma classe descendente
- A vinculação a um método a partir de uma variável polimórfica é feito de forma dinâmica
- Extensão de sistemas de forma mais fácil

Outros conceitos

- Classe Abstrata
  - Um método abstrato não contém implementação
  - Uma classe abstrata tem no mínimo um método abstrato
  - Uma classe abstrata não pode ser instanciada
- Tipos de Variáveis
  - Variáveis de classe
  - Variáveis de instância
- Tipos de Métodos
  - Métodos de Classe
  - Métodos de Instância

Decisões de Projeto

- Exclusividade de objetos
- Subclasses são subtipos?
- Herança simples ou múltipla
- Alocação e Liberação de Objetos
- Vinculação Estática e Dinâmica
- Classes aninhadas
- Inicialização de objetos

Exclusividade de Objetos

- Tudo é um objeto: Smalltalk, Ruby
  - Vantagem: elegância e purismo
  - Desvantagem: operações mais lentas em objetos simples
- Adiciona objetos a um sistema de tipos completo: C++, C#
  - Vantagem: operações rápidas em objetos simples
  - Desvantagem: resulta num sistema de tipo confuso (dois tipos de entidades)
- Inclui um sistema de tipo imperativo para primitivos mas o resto são objetos: Java
  - Vantagem: Operações rápidas em tipos simples
  - Desvantagem: Ainda possui confusão entre os dois sistemas de tipo

Subclasses são subtipos?

- Existe um relacionamento "é-um" entre um objeto da classe pai e um objeto da subclasse?
  - Se uma classe derivada também é uma classe pai, então seus objetos devem se comportar da mesma forma dos da classe pai
  - Subclasses podem só adicionar variáveis e métodos, como também sobrescrever métodos de forma compatível (quantidade de parâmetros, tipos, retorno)
- Caso uma subclasse possa sobrescrever um método de forma incompatível ou mudar a forma de acesso a elementos da classe pai, então ela não é um subtipo

Subclasses são subtipos?

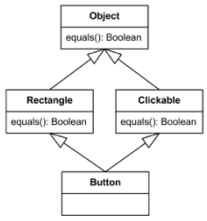
- Exemplo C++

```
class base_class {
      private:
            int a;
            float x;
      protected:
            int b;
            float y;
      public:
            int c;
            float z;
};
class subclass_1 : public base_class {. . .};
class subclass_2 : private base_class {. . .};
```

- E Em Java? Ruby?

Herança Simples e Múltipla

- Herança Múltipla permite que uma nova classe herde de duas ou mais classes
- Vantagem: Algumas vezes é conveniente
- Desvantagem:
  - Potencial ineficiência – vinculação dinâmica pode custar mais com herança múltipla
  - Colisões de nomes entre as diversas classes pai

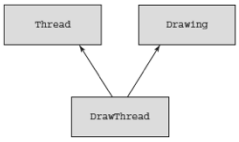


Herança Simples e Múltipla

- C++

```
class Thread { . . . };
class Drawing { . . . };
class DrawThread : public Thread, public Drawing { . . . };
```

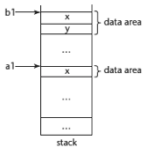
- E Java?



Alocação e Liberação de Objetos

- Onde os objetos são alocados?
  - Pilha de execução? – C++, C#
  - Heap? – Smalltalk, Java, C++, C#
  - *Object Slicing* – caso se use uma pilha, objetos podem ser cortados ao se usar herança

```
class A {
      int x;
      . . .
};
class B : A {
      int y;
      . . .
}
…
a1 = b1
```



Alocação e Liberação de Objetos

- A liberação é implícita ou Explícita?
- Java, Smalltalk usam liberação implícita
- C++ e C# usam construtores e destrutores

Vinculação Estática e Dinâmica

- As ligações entre as chamadas dos métodos e suas especificações são dinâmicas?
  - Se não são você perde as vantagens de vinculação dinâmica
  - Se todas são o programa pode se tornar ineficiente
- Smalltalk
  - Todas as vinculações são dinâmicas
    - Procura o objeto que a mensagem deve ser enviada, se não achar, procura a superclasse, até o classe topo que não têm superclasse
    - Em geral é 10 vezes mais lenta que C++

Vinculação Estática e Dinâmica

- C++
  - Vinculação estática
    - Variáveis de valor (estáticas ou dinâmicas na pilha)
    - Funções em geral definidas na linguagem (sem a palavra **virtual**)
  - Vinculações dinâmicas:
    - Variáveis de ponteiro ou referências são polimórficas
    - funções com a palavra reservada **virtual** (sem definição – classes abstratas)

Vinculação Estática e Dinâmica

```
class Shape {
  public:
    virtual void draw() = 0;
    ...
};
class Circle : public Shape {
  public:
    void draw() { ... }
  ...
};
class Rectangle : public Shape {
  public:
    void draw() { ... }
  ...
};
class Square : public Rectangle
 {
  public:
    void draw() { ... }
  ...
};
```

```
Square* sq = new Square;
Rectangle* rect = new Rectangle;
Shape* ptr_shape;
ptr_shape = sq;  // points to a Square
ptr_shape ->draw();  // Dynamically
                     // bound to draw in Square
rect->draw();  // Statically bound to
               // draw in Rectangle

 Square sq; // Allocates a Square object
from the stack
  Rectangle rect;  // Allocates a Rectangle
object from the stack
  rect = sq;   // Copies the data member
values from sq object
  rect.draw();  // Calls the draw from
Rectangle
```

Vinculação Estática e Dinâmica

- Java
  - Todas as mensagens são dinamicamente vinculadas aos métodos, ao menos que um método seja **final** (não pode ser sobrescrito)
  - Vinculação estática também acontece se os métodos são **static** ou **private**, não permitindo sobrescrita
- Ruby
  - Todas as variáveis não têm tipo, assim sendo polimórficas

Classes Aninhadas

- Se uma classe é necessário somente para uma única classe, então ela não precisa ser definida para ser vista por outras classes
- Que elementos da classe pai devem ser visíveis na classe aninhada e vice-versa
- Java
  - Classes aninhadas não são visíveis para outras classes no pacote, exceto para a classe pai
  - Classes aninhadas não-estáticas (*innerclasses*) acessa membros da classe pai, se estática não acessa
- C#: classes aninhadas se comportam como classes aninhadas estáticas de Java

Inicialização de Objetos

- Objetos são inicializados com valores quando eles são criados?
  - Inicialização implícita ou explícita?
- Como os membros da superclasse são inicializados quando um objeto de uma subclasse é criado?
- Java e C++
  - Toda classe tem ao menos um construtor
  - Se a classe não declara, o compilador adiciona um
  - Membros da classe pai são inicializados quando o objeto da subclasse é criado

Implementando Construções OO

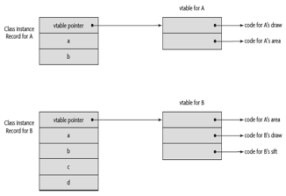
- Armazenamento de dados de instâncias
  - Registros de Instâncias de Classes (RIC) armazenam o estado de um objeto
    - Estático (construído em tempo de compilação
  - Se uma classe tem pais, as variáveis de instância da subclasse são adicionadas ao RIC da classe pai
  - Como RIC é estático, ele é usado como um modelo para criação de instâncias

Implementando Construções OO

- Vinculação Dinâmica de chamadas de método

```
public class A {
  public int a, b;
  public void draw() { . . . }
  public int area() { . . . }
}

public class B extends A {
  public int c, d;
  public void draw() { . . . }
  public void sift() { . . . }
}
```



- E com herança múltipla?

Leitura Adicional

- Capítulo 12 – Suporte a programação OO. SEBESTA, R. W. Conceitos de Linguagens de Programação. 9ª ED. BOOKMAN, 2011.
- Próxima aula
  - Programação Orientada a Aspectos