Paradigmas de Linguagens de Programação

Tipos de Dados Abstratos

Cristiano Lehrer, M.Sc.

ybadoo

Conceito de Abstração

- O conceito de abstração é fundamental em programação.
- Quase todas as linguagens suportam abstração de processos, através de subprogramas:
  - Exemplo em Java:
    - `public void sort(int vetor[])`
- Quase todas as linguagens de programação projetadas desde 1980 tem suporte à abstração de dados com algum tipo de módulo.

ybadoo

Encapsulamento

- Motivação original:
  - Grandes programas tem duas necessidades especiais:
    - Algum tipo de organização, além da simples divisão em subprogramas.
    - Algum tipo de compilação parcial:
      - Unidades de compilação são menores que o programa inteiro.
- Solução óbvia:
  - Um agrupamento de subprogramas que são logicamente relacionados em uma unidade que pode ser compilada separadamente:
    - São chamados encapsulamento.

Exemplo de Mecanismos de Encapsulamento

- Subprogramas aninhados em alguma linguagem similar ao ALGOL:
  - Pascal
- FORTRAN 77 e C:
  - Arquivos contendo um ou mais subprogramas podem ser compilados independentemente.
- FORTRAN 90, C++, Ada (e outras linguagens contemporâneas):
  - Separadamente em módulos compiláveis.

Abstração de Dados (1/3)

- Um tipo abstrato de dados é um tipo de dados definido pelo usuário que satisfaz a duas condições:
  - A representação e as operações em objetos do tipo são definidos em uma unidade sintática simples:
    - Outras unidades podem também criar objetos daquele tipo.
  - A representação de objetos de tipo é oculto das unidades do programa que usam esses objetos, assim as únicas operações possíveis são aquelas que provêm da definição do tipo.

Abstração de Dados (2/3)

- Vantagens:
  - As mesmas referentes à encapsulamento:
    - Organização do programa, alterabilidade (tudo que estiver associado com uma estrutura de dados está junto), e compilação separado.
  - Confiabilidade:
    - Ao esconder as representações dos dados, o código do usuário não pode acessar diretamente objetos do tipo. O código do usuário não pode depender da representação dos dados, permitindo que tal representação seja alterada sem afetar o código do usuário.

Abstração de Dados (3/3)

- Tipos Primitivos (*built-in*) são tipos abstratos de dados:
  - Exemplo: tipo **int** no C
    - A representação é oculta.
    - Todas as operações são *built-in*.
    - Programas do usuário podem definir objetos do tipo **int**.
    - Tipos de dados definidos pelo usuário devem ter as mesmas características dos TAD *built-in*.

Requisitos da Linguagem para Abstração de Dados

- Uma unidade sintática na qual a definição de tipo seja encapsulada.
- Um método de fazer nomes de tipos e cabeçalhos de subprogramas visíveis a clientes, ao mesmo tempo que esconde as verdadeiras definições.
- Algumas operações primitivas devem fazer parte do processador da linguagem (usualmente apenas atribuições e comparações para igualdade desigualdade):
  - Algumas operações são normalmente necessárias, mas devem ser definidas por quem especifica o tipo.
  - Ex.: Construtores e destrutores.

Exemplo (1/2)

- Tipo de dado abstrato deve ser construído para uma pilha que tem as seguintes operações abstratas:
  - `create(pilha)` → cria e possivelmente inicializa um objeto da pilha.
  - `destroy(pilha)` → desaloca o armazenamento da pilha.
  - `empty(pilha)` → uma função predicada (ou booleana) que retorna *true* (verdadeiro) se a pilha especificada estiver vazia, e *false* (falso) se não estiver.
  - `push(pilha, elemento)` → empurra o elemento especificado para a pilha especificada.
  - `pop(pilha)` → remove o elemento do topo da pilha especificada.
  - `top(pilha)` → retorna uma cópia do elemento do topo da pilha especificada.

Exemplo (2/2)

- Um cliente do tipo pilha poderia ter uma sequência de código como a seguinte:

```
create(STK1);
push(STK1, COR1);
push(STK1, COR2);
if(not empty(STK1))
    then TEMP := top(STK1);
```

C++ (1/5)

- Baseado no tipo struct do C e nas classes do Simula 67.
- A classe é o dispositivo de encapsulamento.
- Todas as instâncias de uma classes compartilham uma só cópia das funções-membro.
- Cada instância de uma classe possui sua própria cópia dos membros de dados.
- Instâncias podem ser estáticas, dinâmicas na pilha ou dinâmicas no *heap*.

C++ (2/5)

- Ocultamento da informação:
  - Cláusulas private para entidades ocultas.
  - Cláusulas public para entidades da interface.
  - Cláusulas protected para herança.

C++ (3/5)

- Construtores:
  - Funções para inicializar os membros de dados das instâncias (não criam os objetos).
  - Podem também fazer alocação de armazenamento se parte do objeto é dinâmico no *heap*.
  - Implicitamente chamados quando uma instância é criada.
  - Podem ser chamados explicitamente.
  - O nome é o mesmo do nome da classe.

C++ (4/5)

- Destrutores:
  - Funções para limpeza após uma instância ser destruída.
  - Usualmente apenas retomada de espaço de armazenamento no *heap*.
  - Implicitamente chamados quando o tempo de vida do objeto acaba.
  - Podem ser chamados explicitamente.
  - Nome é o mesmo do nome da classe, precedido de um til (~).

ybadoo

C++ (5/5)

- Funções ou classes friend:
  - Provêm acesso a membros privados a algumas unidades ou funções não relacionadas (necessário em C++).
- Avaliação do suporte do C++ a TAD:
  - Classes são similares a pacotes do Ada ao prover TAD.
  - Diferença: pacotes são encapsulamento, enquanto que classes são tipos.

Java

- Similar ao C++, exceto que:
  - Todos os tipos definidos pelo usuário são classes.
  - Todos os objetos são alocados no *heap* e acessados através de variáveis de referência.
- Entidades individuais nas classes tem modificadores do controle de acesso (private ou public), ao invés de cláusulas.
- Java possui um segundo mecanismo de *scoping*, o package score, que pode ser usado no lugar de friends.
- Todas as entidades em todas as classes num pacote que não tem acesso aos modificadores do controle de acesso, são visíveis através do pacote.