ESTRUTURAS DE DADOS BÁSICAS EM JAVA

José de Siqueira

UFMG - ICEx - DCC

1º semestre de 2005

O Tipo Abstrato de Dados Pilha

- O TAD pilha tem quase as mesmas operações apresentadas anteriormente:
  1. *empilha(o)*: insere o objeto *o* no topo da pilha.
     Entrada: objeto. **Saída:** nenhuma.
  2. *desempilha()*: retira o objeto no topo da pilha e o roetorna; ocorre erro se a pilha estiver vazia.
     Entrada: nenhuma. **Saída:** objeto.
  3. *tamanho():* retorna o número de objetos na pilha.
     Entrada: nenhuma. **Saída:** inteiro.
  4. *vazia():* Retorna um booleano indicando se a pilha está vazia.
     Entrada: nenhuma. **Saída:** booleano.
  5. *topo():* Retorna o objeto no topo da pilha, sem retirá-lo; ocorre um erro se a pilha estiver vazia.
     Entrada: nenhuma. **Saída:** objeto.

Uma interface para pilhas em Java

- Em Java, já existe a classe para o TAD pilha: `java.util.Stack`.
- Os métodos disponíveis nesta classe, entre outros, são: `push(obj)`, `pop()`, equivalentes a *empilha(o)* e *desempilha()* e `peek()`, equivalente a *topo(), tamanho()* e *vazia()*.
- Os métodos `pop() e peek()` lançam a exceção `StackEmptyException` se a pilha estiver vazia quando eles são chamados.
- Apesar de já existir esta classe em Java, aqui estamos interessados em aprender como projetar e implementar uma pilha e uma fila em Java.
- A implementação de um TAD em Java envolve dois passos:
  1. a definição de uma *API (Application Programming Interface)* que descreve os nomes dos métodos que o TAD oferece, como eles são declarados e como são usados.
  2. uma ou mais implementações concretas dos métodos descritos na interface (API) associada com o TAD.

```
public interface Pilha {
   /* retorna o número de itens na pilha */
   public int tamanho();
   /* retorna true se a pilha está vazia, false senão */
   public boolean vazia();
   /*retorna, sem removê-lo, o item do topo da pilha;
   lança StackEmptyException se a pilha estiver vazia*/
   public Object topo()
        throws StackEmptyException;
   /* insere um item, passado em parâmetro, no topo
   da pilha */
   public void empilha(Object element);
   /* remove e retorna o item no topo da pilha; lança
   StackEmptyException se a pilha estiver vazia*/
   public Object desempilha()
        throws StackEmptyException;
}
/* Exceções lançadas quando se tenta usar as operações
em uma pilha vazia são tratadas aqui*/
public class StackEmptyException extends
                                   RuntimeException {
   public StackEmptyException (String erro) {
      super(erro);
   }
}
```

Uma implementação baseada em vetores

- Nesta implementação baseada em vetores, como o vetor é alocado estaticamente, ao tentar empilhar um objeto em uma pilha cheia, devemos lançar uma exceção StackFullExcpetion.
- Esta exceção não foi definida no TAD Pilha por ser específica desta implementação.

```
/* Implementação da interface Pilha usando um
vetor de tamanho fixo. Uma exceção é lançada ao
tentar empilhar um objeto em uma pilha cheia. */
public class PilhaComVetor implements Pilha
{
   /* Tamanho máximo fixo do vetor usado como
   pilha */
   public static final int CapacidadeMax =
   1000;
   /* Capacidade da pilha */
   private int Capacidade;
   /* Vetor usado como pilha */
   private Object P[ ];
   /* índice do elemento do topo da pilha */
   private int topo = -1;
```

```
/* inicia a pilha para usar um vetor com tamanho
máximo CapacidadeMax */
public PilhaComVetor() {
   this(CapacidadeMax);
}
/* inicia a pilha para um arranjo com o tamanho
fornecido; o parâmetro é o tamanho do vetor */
public PilhaComVetor(int tam) {
   Capacidade = tam;
   P = new Object[Capacidade];
}
public int tamanho() {
   return(topo + 1);
}
public boolean vazia() {
   return(topo < 0);
}
public void empilha(Object obj) throws
                              StackFullException {
   if (tamanho() == Capacidade)
      throw new StackFullException(‘‘Pilha
                                      cheia!’’);
   P[++topo] = obj;
}
```

```
    public Object desempilha() throws
    StackEmptyException {
      Object elemento;
      if (vazia())
         throw new StackEmptyException(‘‘Pilha
                                        vazia!’’);
      elemento = P[topo];
      P[topo] = null; // Libera P[topo] para a
                                 // coleta de lixo
      topo--;
      return elemento;
    }
}
```

- A pilha declarada acima é genérica, pois os elementos são instâncias da classe `Object` de Java.
- Pode-se armazenar qualquer objeto na pilha, pois todas as classes Java herdam da classe `Object`.
- Assim, podemos empilhar objetos das classes `Integer, Estudante` ou até mesmo `Planetas`.
- No entanto, ao desempilhar, é preciso fazer uma conversão para a classe específica a que o objeto realmente pertence, como mostra o trecho de programa abaixo:

```
public static Integer[]
                    inverte(Integer[] a) {
   PilhaComVetor P = new
   PilhaComVetor(a.length);
   Integer[] b = new Integer[a.length];
   for (int i = 0; i < a.length; i++)
      P.empilha(a[i]);
   for (int i = 0; i < a.length; i++)
      b[i] = (Integer) (P.desempilha());
   return b;
}
```

O TAD fila em Java

- As operações do TAD fila são:
  1. *insere(o)*: insere o obejto *o* no fim da fila.
     Entrada: objeto. **Saída:** nenhuma.
  2. *retira(o)*: retira e retorna o objeto do início da fila. Lança uma exceção se a fila estiver vazia.
     Entrada: nenhuma. **Saída:** objeto.
  3. *tamanho()*: retorna o número de objetos na fila.
     Entrada: nenhuma. **Saída:** inteiro.

4. *vazia()*: retorna um booleano indicando se a fila está vaiza ou não.
   Entrada: nenhuma. **Saída:** booleano.
5. *frente()*: retorna o objeto no início da fila, sem retirá-lo.Lança uma exceção se a fila estiver vazia.
   Entrada: nenhuma. **Saída:** objeto.

```
public interface Fila {
    /* retorna o número de itens na fila */
    prublic int tamanho();
    /* retorna true se a fila estiver vazia, false senão */
    public boolean vazia();
    /* retorna o item à frente na fila; lança
    QueueEmptyException se a fila estiver vazia */
    public Object frente() throws
    QueueEmptyException;
    /* insere elemento no final da fila */
    public void insere(Object item);
    /* remove e retorna o item à frente da fila; lança
    QueueEmptyException se a fila estiver vazia */
    public Object retira() throws
                                QueueEmptyException;
```

Implementação de filas com arranjos

- Para implementar uma fila em um arranjo de tamanho N, é melhor utilizar uma fila circular.
- Para isso, temos dois apontadores i e f que indicam o início e o fim da fila.
- A fila está vazia quando $i = f$ e f indica a próxima posição livre.
- **Problema:** O que acontece quando $f = N$? O que fazer neste caso?
- A implementação da circularidade é simples se o incremento for feito como $(i + 1) \bmod N$ ou $(f + 1) \bmod N$.
- **Problema**: como distinguir que a fila está cheia?
- Por exemplo, deixando sempre uma casa vazia entre o fim e o início.
- Ou inserindo, no máximo, $N - 1$ elementos.

```
Algoritmo tamanho();
   retorna (N-i+f) mod N;

Algoritmo vazia();
   retorna (i=f);

Algoritmo frente();
   se vazia() então lançar uma
                              QueueEmptyException;
   retorna F[i];

Algoritmo retira();
   se vazia() então lançar uma
                             QueueEmptyException ;
   aux ← F[i];
   F[i] ← null;
   i ← (i+1) mod N;
   retorna aux;

Algoritmo insere(o);
   se tamanho() = N - 1 então lançar uma
                                QueueFullException;
   F[f] ← o;
   f ← (f+1) mod N;
```

Listas encadeadas em Java

```
public class Nó {
    // Variáveis de instância
    private Object item;
    private Nó prox;
    // Construtores simples
    public Nó() {
       this(null,null);
    }
    public Nó(Object i, Nó n) {
       item = i;
       prox = n;
    }
    // Métodos de acesso
    Object retItem() {
       return item;
    }
    Nó retProx() {
       return prox;
    }
    // Modificadores
    void posItem(Object novoItem) {
       item = novoItem;
    }
    void posProx(Nó novoNó) {
       prox = novoNó;
    }
}
```

Implementação de uma pilha com listas encadeadas

```
public class PilhaEncadeada implements Pilha {
   private Nó topo; // referência para o nó do
                                          // topo
   private int tam; // número de itens na pilha
   public PilhaEncadeada() {
      topo = null;
      tam = 0;
   }
   public int tamanho() {
      return tam;
   }
   public boolean vazia() {
      return (topo == null);
   }
   public void empilha(Object item) {
      Nó v = new Nó(); // cria um novo nó
      v.posItem(item);
      v.posProx(topo); // encadeia o novo nó
      topo = v;
      tam++;
   }
```

```
public Object topo() throws
StackEmptyException {
   if (vazia())
      throw new StackEmptyException(‘‘Pilha
                                  está vazia.’’);
   return topo.retItem();
}
public Object desempilha() throws
StackEmptyException {
   if (vazia())
      throw new StackEmptyException(‘‘Pilha
                                  está vazia.’’);
   Object aux = topo.retItem();
   topo = topo.retProx; // aponta para o
                                    // próximo nó
   tam--;
   return aux;
   }
}
```

Implementação de uma fila com listas encadeadas

- Apresentamos apenas a implementação de dois métodos principais.
- As declarações de classe e os outros métodos para a implementação de filas com listas encadeadas são deixadas como exercício.

```
public void insere(Object obj) {
   Nó nó = new Nó();
   nó.posItem(obj);
   nó.posProx(null); // o nó inserido será o do
                                  // final da lista
   if (tam == 0) //inserção em uma fila vazia
      primeiro = nó
   else
      último.posProx(nó); // insere nó no final
                                          // da fila
   último = nó; // aponta para o nó inserido
   tam++;
}
```

```
public Object retira() throws
                    QueueEmptyException {
   Object obj;
   if (tam == 0)
      throw new QueueEmptyException
                    (``Fila está vazia.);
   obj = primeiro.retItem();
   primeiro = primeiro.retProx();
   tam--;
   if (tam == 0)
      último = null // a fila ficou
                                  // vazia
   return obj;
}
```