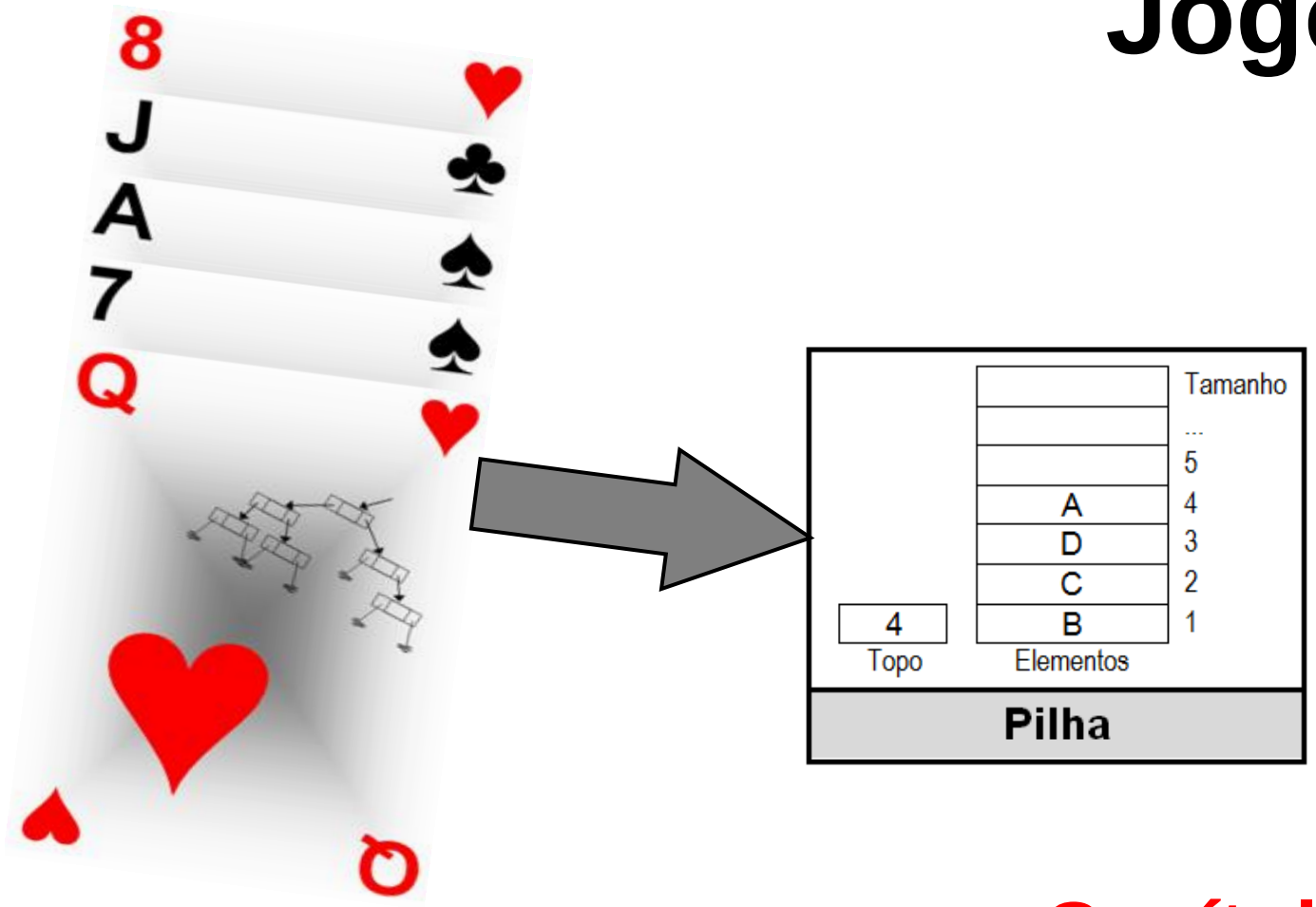
Estruturas de Dados com Jogos



Capítulo 2

Pilhas - Alocação Sequencial e Estática

Seus Objetivos neste Capítulo

- Entender o que é e para que serve uma estrutura do tipo Pilha;
- Entender o significado de Alocação Sequencial e Alocação Estática de Memória, no contexto do armazenamento temporário de conjuntos de elementos;
- Desenvolver habilidade para implementar uma estrutura do tipo Pilha, como um Tipo Abstrato de Dados - TAD, com Alocação Sequencial e Estática de Memória;
- Desenvolver habilidade para manipular Pilhas através dos Operadores definidos para o TAD Pilha;
- Iniciar o desenvolvimento do seu primeiro jogo.

O Que É uma Pilha?

Pilha de Pratos	Pilha de Cartas
 iStochPhoto	

Definição de Pilha

Definição: Pilha

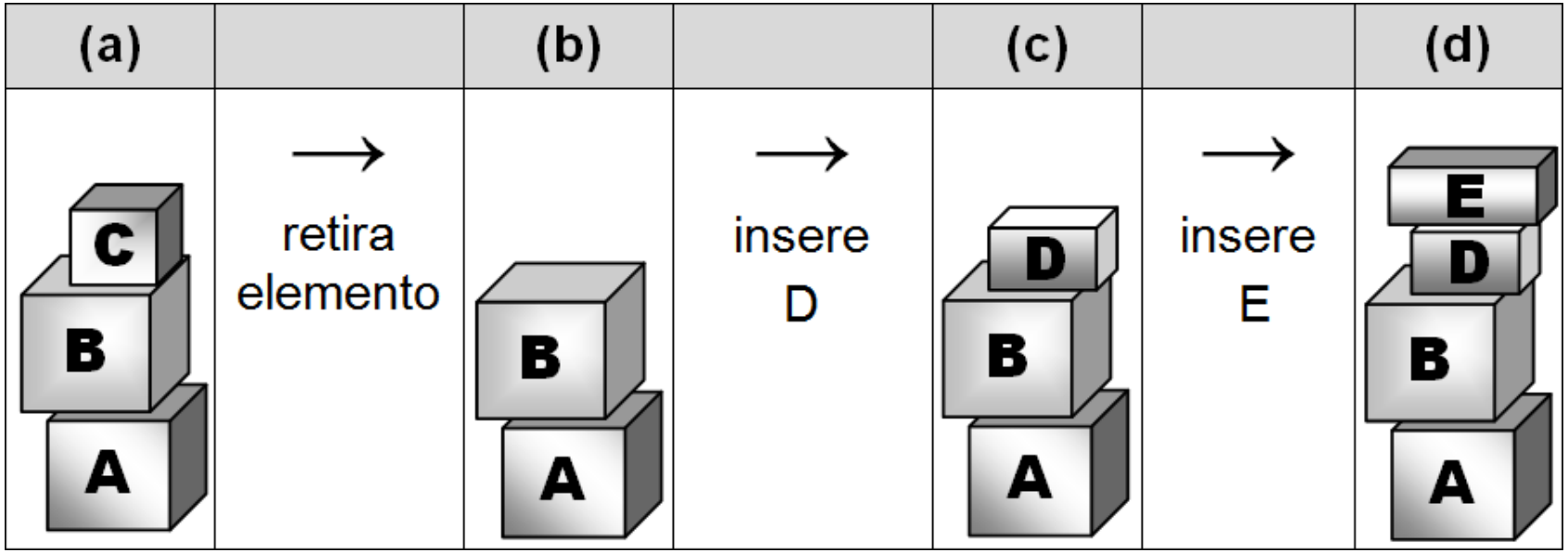
Pilha é uma estrutura para armazenar um conjunto de elementos, que funciona da seguinte forma:

- Novos elementos entram no conjunto sempre no topo da Pilha;
- O único elemento que pode ser retirado da Pilha em um dado momento, é o elemento do topo.

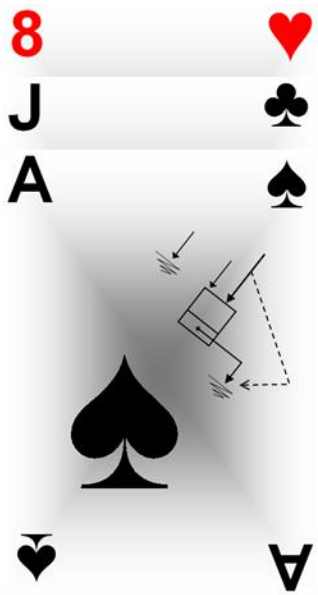
Do Inglês: *Stack*, L.I.F.O.

Uma Pilha (em Inglês: *Stack*) é uma estrutura que obedece o critério *Last In, First Out* - L.I.F.O. ➔ o último elemento que entrou no conjunto será o primeiro elemento a sair do conjunto.

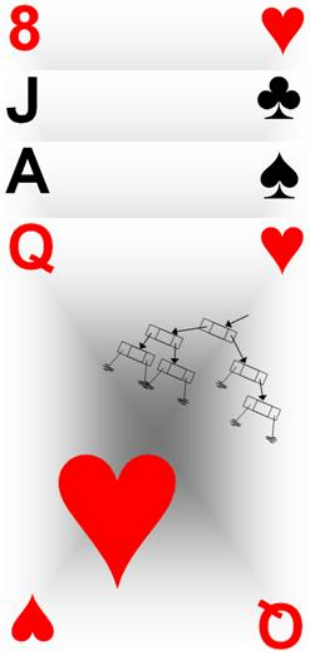
Retirando e Inserindo Elementos em uma Pilha



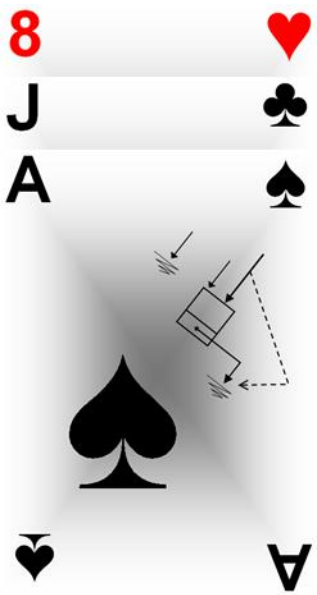
Inserindo e Retirando Cartas em uma Pilha



→
Insere
Q ♥

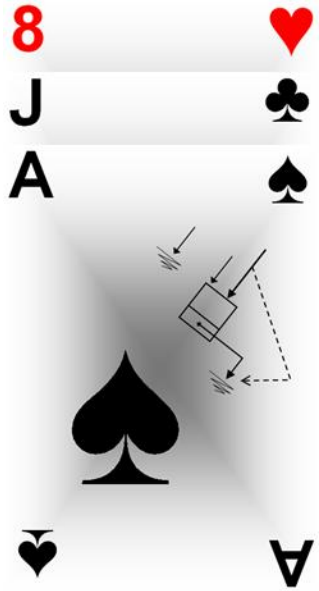


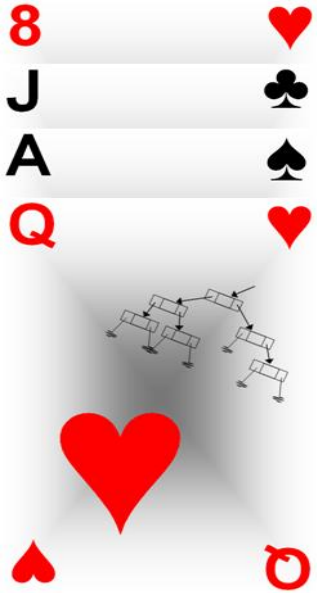
→
Retira
Carta



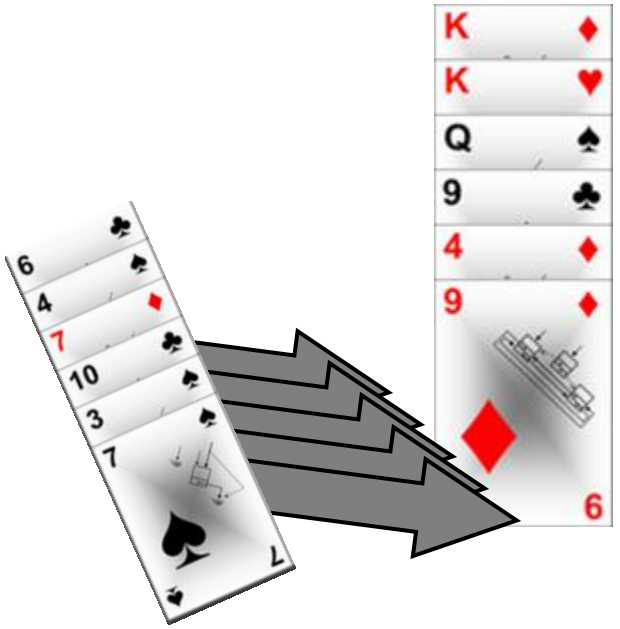
Operações de uma Pilha

- Empilha (P, X, DeuCerto)
- Desempilha (P, X, DeuCerto)
- Vazia (P)
- Cheia(P)
- Cria (P)





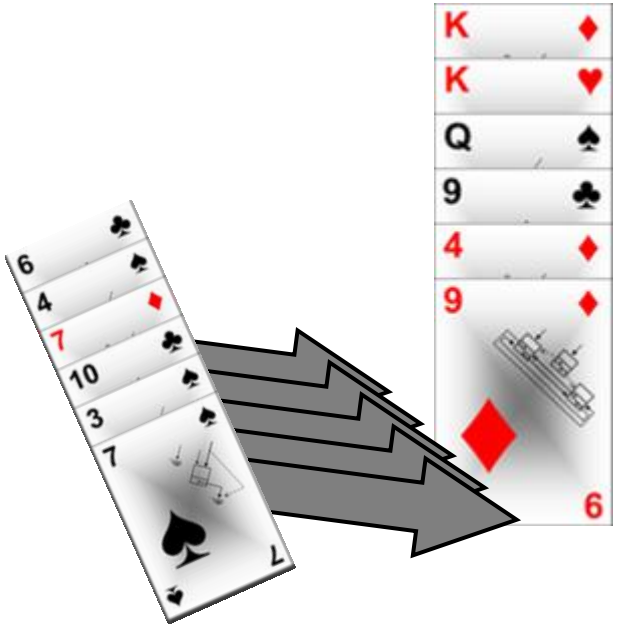
Exercício 2.1
Transfere Elementos de uma Pilha para Outra



TransfereElementos (parâmetros por referência **P1**, **P2** do tipo Pilha)

/* trasfere todos os elementos de P1 para P2 */

Exercício 2.1
Transfere Elementos de uma Pilha para Outra

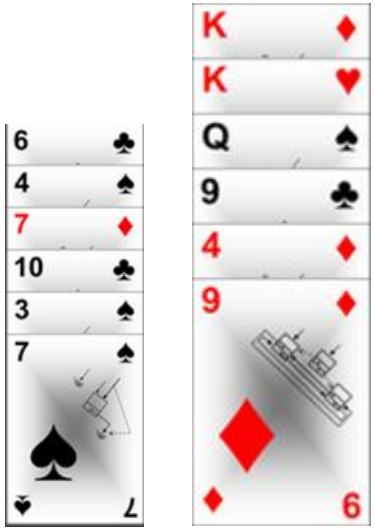


TransfereElementos (parâmetros por referência **P1**, **P2** do tipo Pilha) {
/* trasfere todos os elementos de P1 para P2 */

Variável ElementoDaPilha do tipo Inteiro;
Variável DeuCerto do tipo Boolean;

```
Enquanto (Vazia(P1) == Falso) Faça {          // enquanto P1 não for vazia
        Desempilha(P1, ElementoDaPilha, DeuCerto); // desempilha de P1
        Empilha(P2, ElementoDaPilha, DeuCerto); }      // empilha em P2
}
```

Exercícios



Exercício 2.2 Mais Elementos?

• Boolean MaisElementos (parâmetros por referência P1, P2 do tipo Pilha);

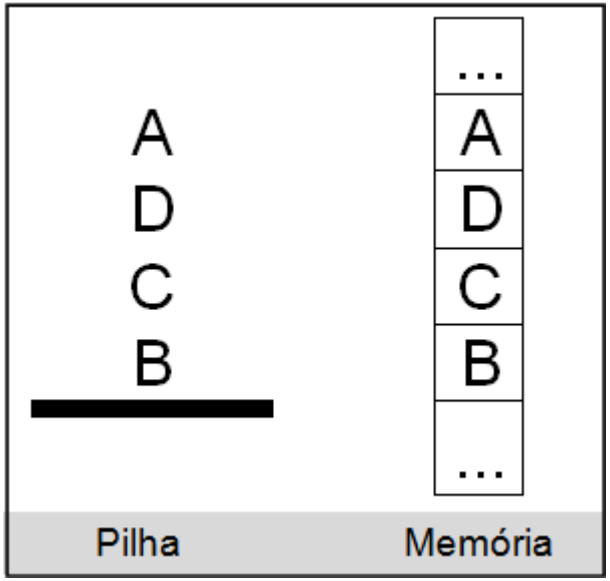
Exercício 2.3 Algum Elemento Igual a X?

• Boolean AlgumElementoIgualaX (parâmetro por referência P do tipo Pilha, parâmetro X do tipo inteiro);

Exercício 2.4 As Pilhas São Iguais?

• Boolean Iguais (parâmetros por referência P1, P2 do tipo Pilha);

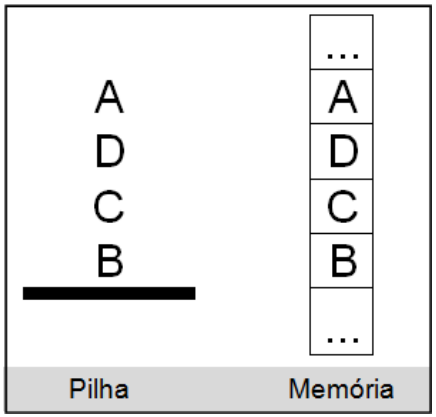
Alocação Sequencial



Definição: Alocação Sequencial de Memória para um Conjunto de Elementos

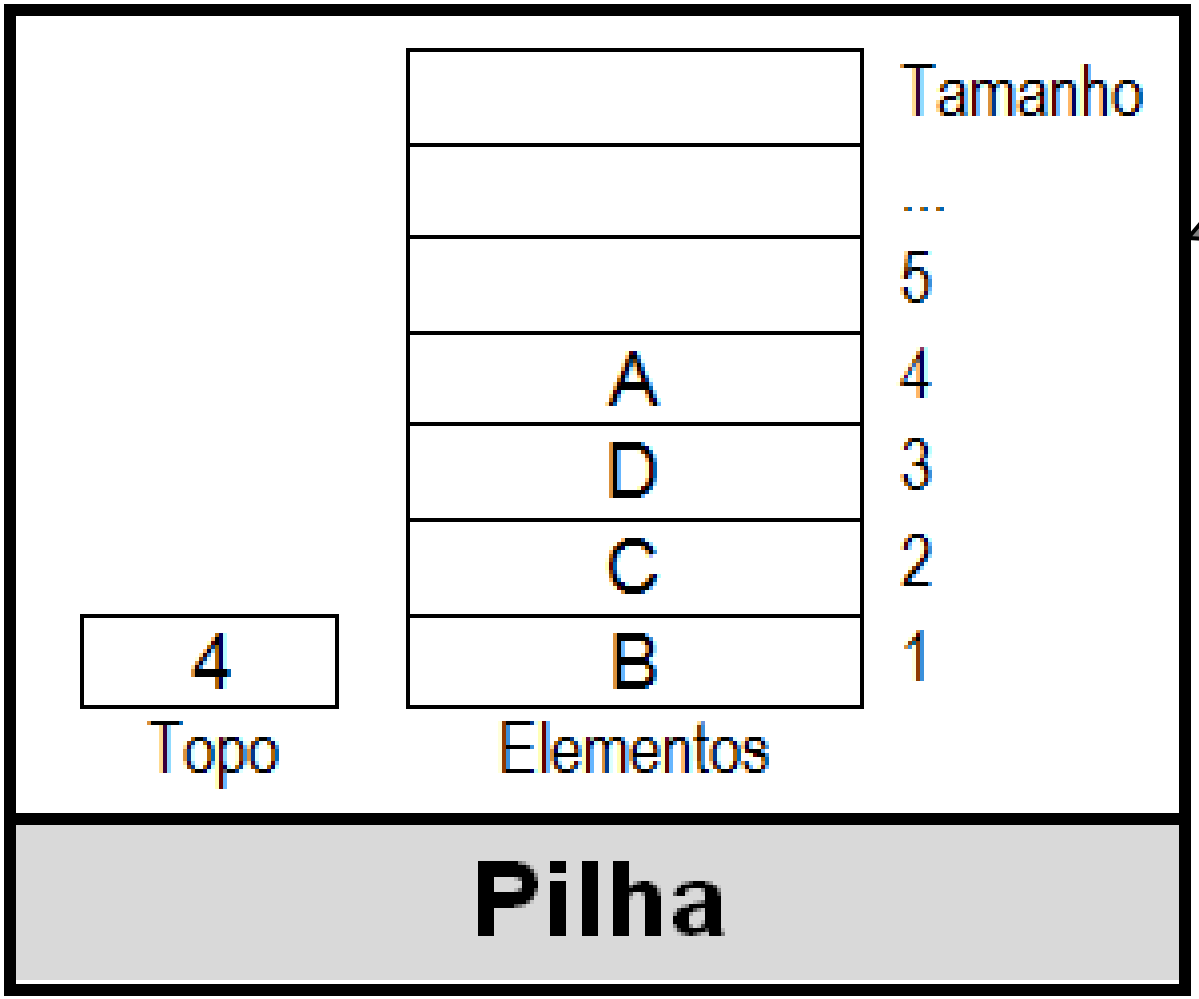
Os elementos ficam, necessariamente, em posições de memória adjacentes, ou ainda, "em sequência".

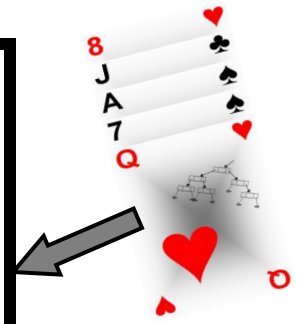
Definição: Alocação Estática de Memória para um Conjunto de Elementos



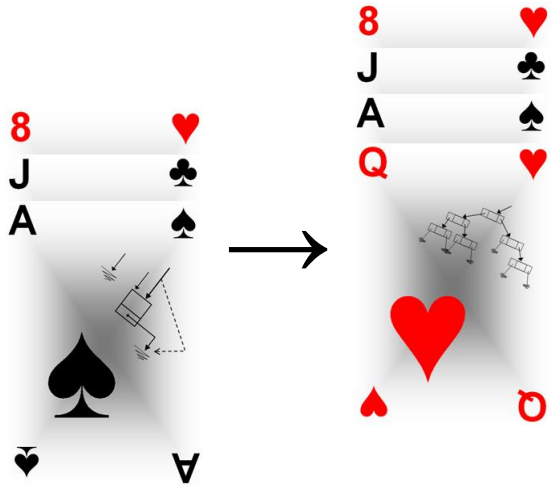
- O espaço de memória para todos os elementos que poderão fazer parte do conjunto é dimensionado previamente;
- Esse espaço de memória permanecerá reservado durante toda a execução do programa, mesmo que não estiver sendo efetivamente utilizado.

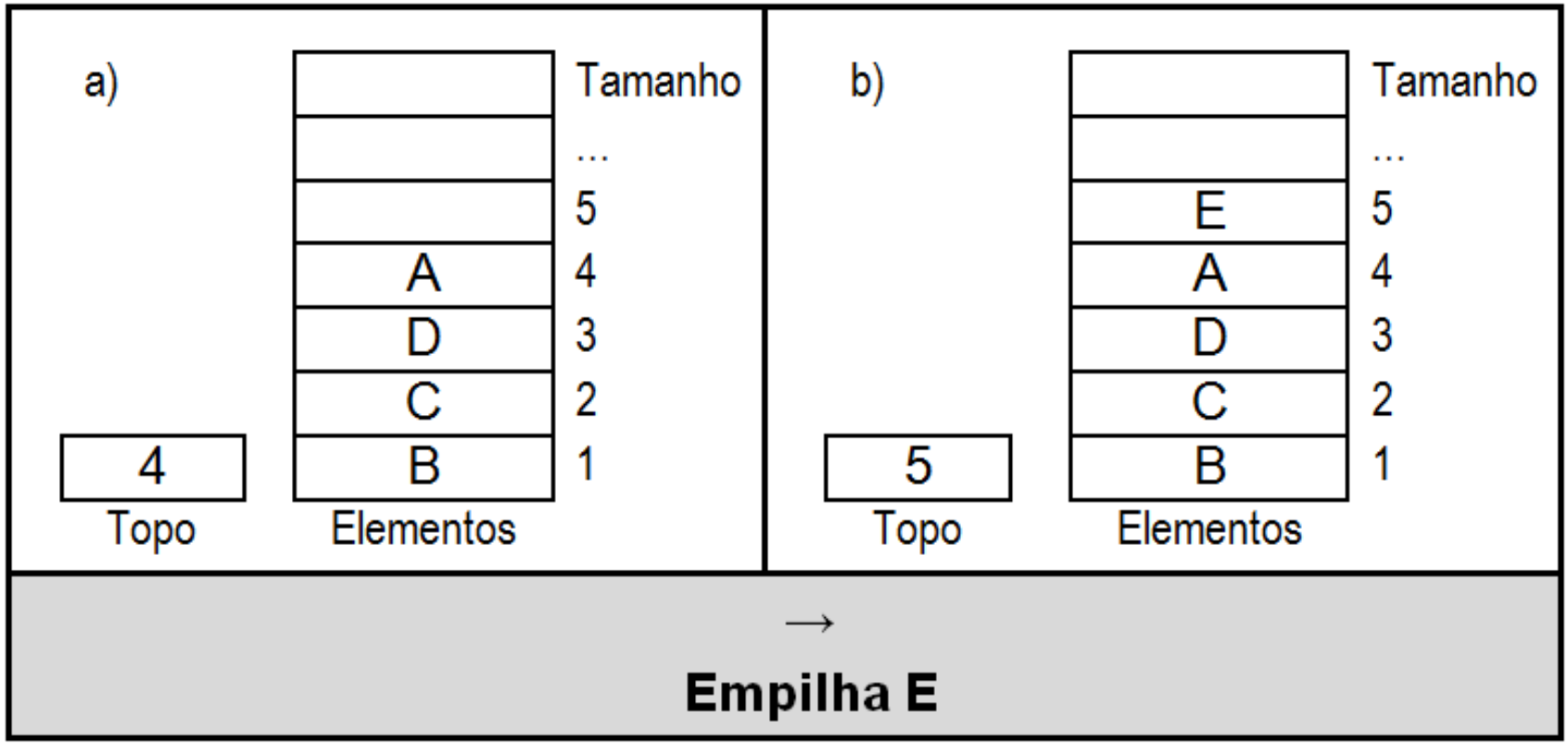
Pilha com Alocação Sequencial e Estática de Memória



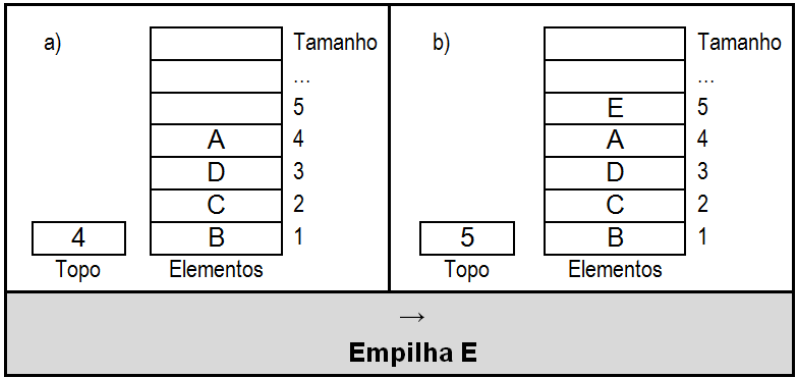


Exercício 2.5
Operação Empilha





Exercício 2.5
Operação Empilha

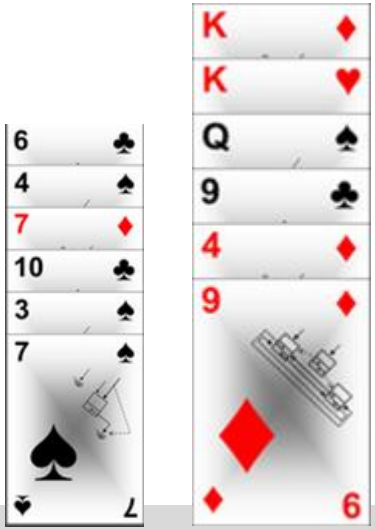


Empilha (parâmetro por referência **P** do tipo Pilha, parâmetro **X** do tipo Char, parâmetro por referência **DeuCerto** do tipo Boolean) {

/* Empilha o elemento X, passado como parâmetro, na Pilha P também passada como parâmetro. O parâmetro DeuCerto deve indicar se a operação foi bem sucedida ou não.*/

```
Se (Cheia(P) == Verdadeiro)                    // se a Pilha P estiver cheia...
Então     DeuCerto = Falso              // ... não podemos empilhar
Senão { P.Topo = P.Topo + 1;                    // incrementa o Topo da Pilha P
P.Elementos[ P.Topo ] = X;          // armazena o valor de X no Topo da Pilha
          DeuCerto = Verdadeiro; }          // a operação deu certo
}
```

Exercícios



Exercício 2.6 Operação Desempilha

• Desempilha(parâmetro por referência P do tipo Pilha, parâmetro por referência X do tipo Char, parâmetro por referência DeuCerto do tipo Boolean);

Exercício 2.7 Operação Cria

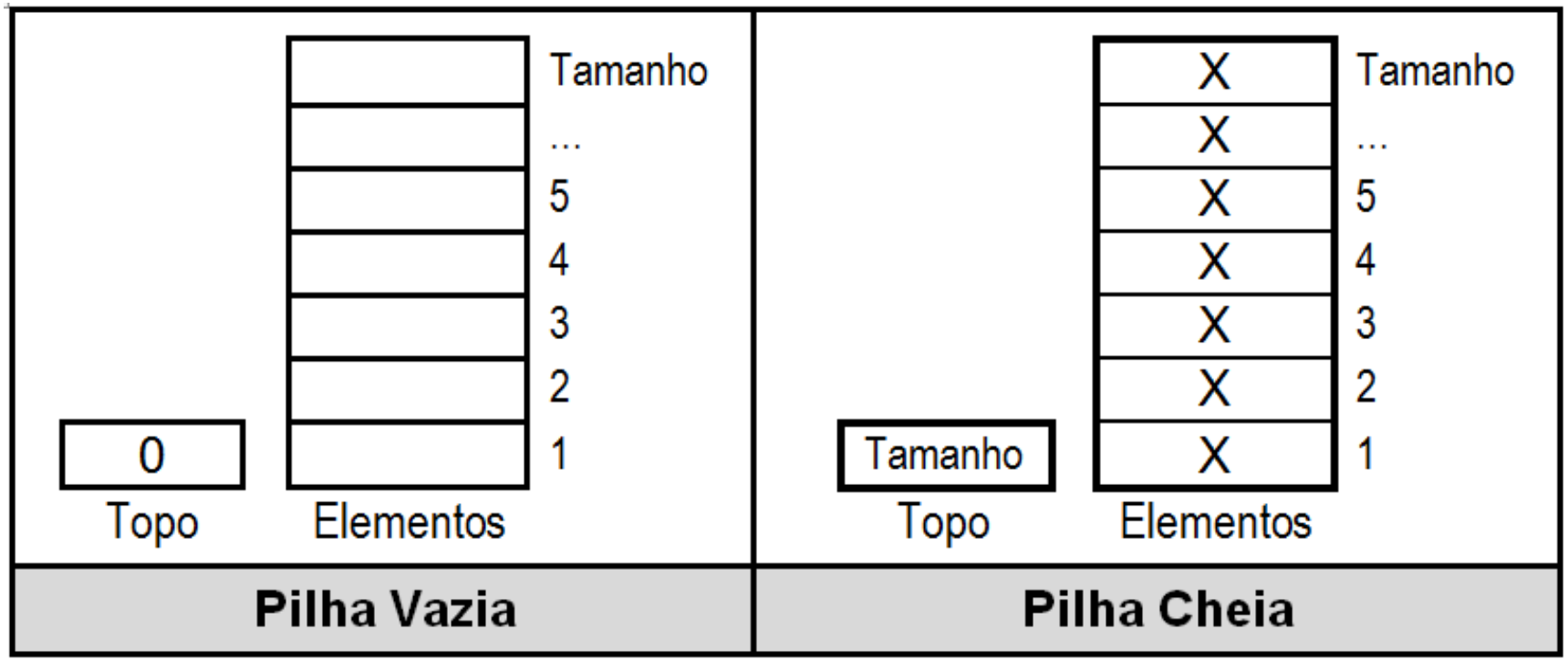
• Cria (parâmetro por referência P do tipo Pilha);

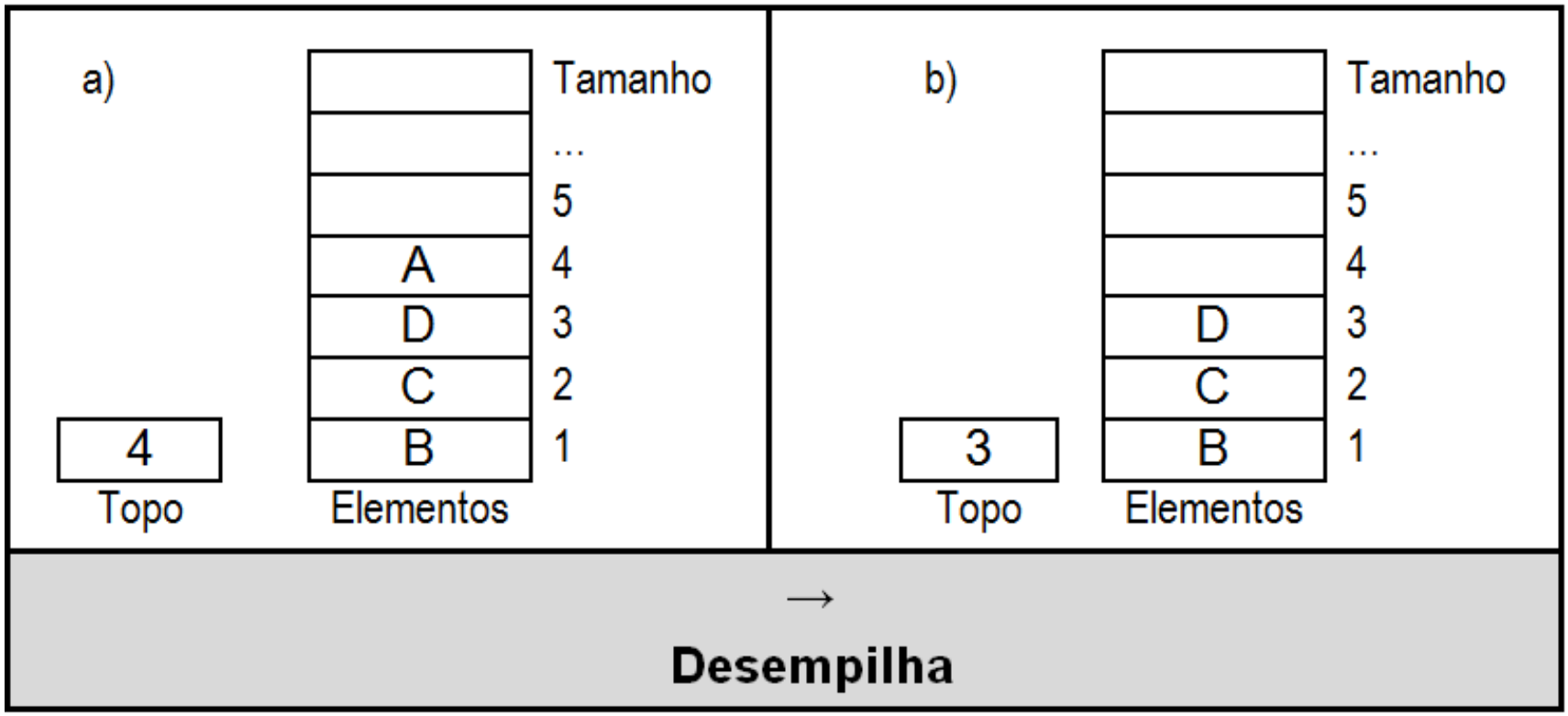
Exercício 2.8 Operação Vazia

• Boolean Vazia (parâmetro por referência P do tipo Pilha);

Exercício 2.9 Operação Cheia

• Boolean Cheia (parâmetro por referência P do tipo Pilha);


Tamanho
...
5
4
3
2
1
0
Topo
Elementos
Pilha Vazia
X
X
X
X
X
X
Tamanho
...
5
4
3
2
1
Tamanho
Topo
Elementos
Pilha Cheia


a)
Tamanho
...
5
A
4
D
3
C
2
B
1
4
Topo
Elementos
b)
Tamanho
...
5
4
D
3
C
2
B
1
3
Topo
Elementos
→
Desempilha

Abrir a TV ou Não Abrir a TV?

Exercício 2.10 Duas Estratégias para Elemento do Topo

Char ElementoDoTopo (parâmetro por referência P do tipo Pilha, parâmetro por referência DeuCerto do tipo Boolean) {

/* Retorna o valor do elemento do Topo da Pilha P, caso a Pilha não estiver vazia. Caso a Pilha estiver vazia, DeuCerto retorna Falso. */

Char **ElementoDoTopo** (parâmetro por referência **P** do tipo Pilha, parâmetro por referência **DeuCerto** do tipo Boolean) {

/* Retorna o valor do elemento do Topo da Pilha P, caso a Pilha não estiver vazia. Caso a Pilha estiver vazia, DeuCerto retorna Falso. */

Variável X do tipo Char; /* X armazenará o valor do elemento do Topo */

/* **primeira solução**: abre a TV com uma chave de fenda para aumentar o volume */
Se (Vazia (P)==Verdadeiro)
Então DeuCerto = Falso
Senão { **X = P.Elementos[P.Topo];** /* pega o valor do elemento do Topo */
DeuCerto = Verdadeiro;
Retorne X; };

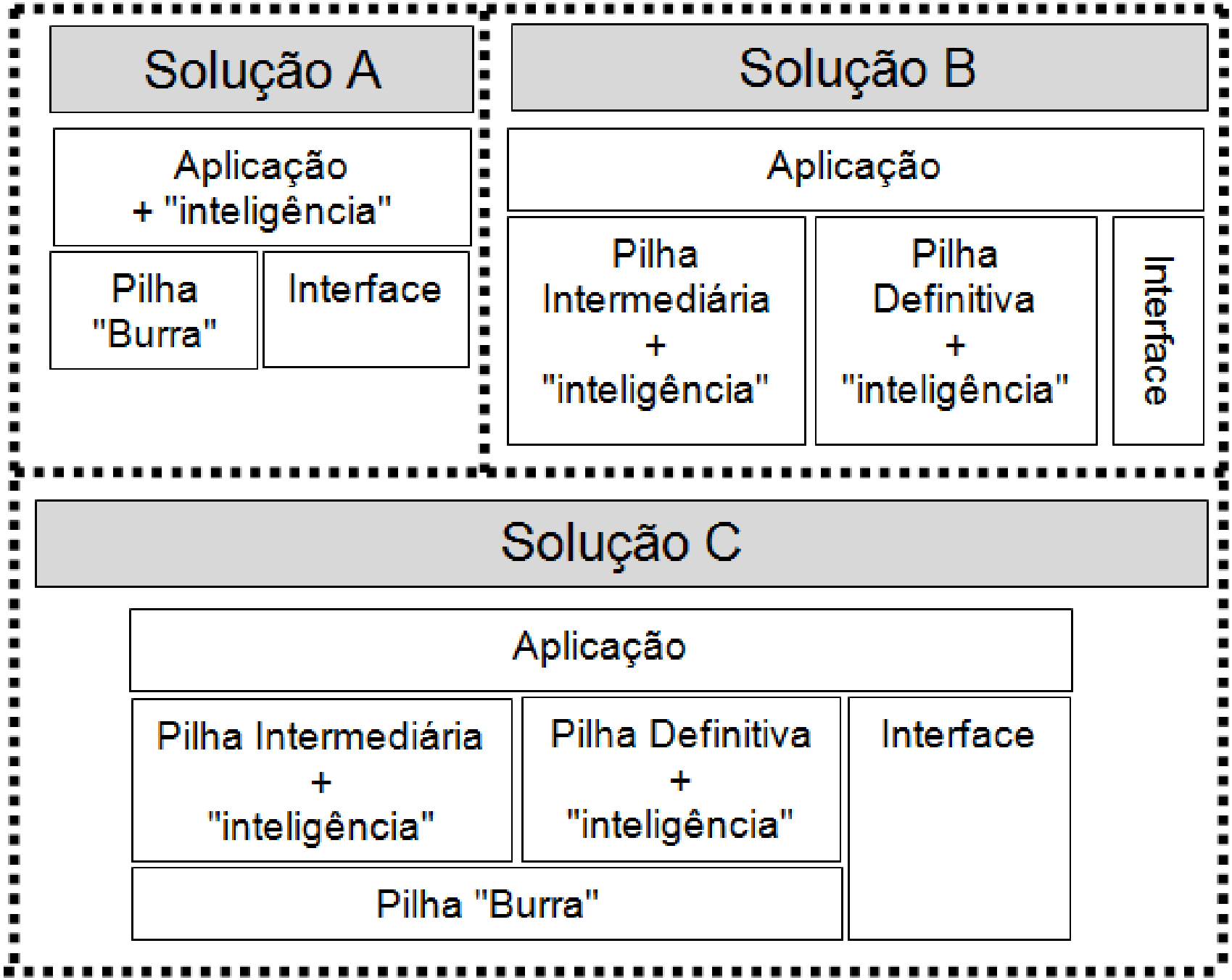
/* **segunda solução**: aumenta o volume pelo botão de volume */
Se (Vazia (P)==Verdadeiro)
Então DeuCerto = Falso
Senão { **Desempilha (P, X, DeuCerto);** /* pega o valor do elemento do Topo */
Empilha (P, X, DeuCerto);
DeuCerto = Verdadeiro;
Retorne X; }
}

Operações Primitivas e Não Primitivas

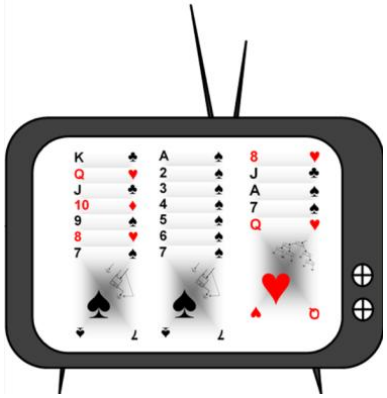
Operações Primitivas de um Tipo Abstrato de Dados são aquelas que só podem ser implementadas através de uma solução dependente da estrutura de armazenamento.

Operações Não Primitivas de um Tipo Abstrato de Dados são aquelas que podem ser implementadas através de chamadas a Operações Primitivas, possibilitando uma implementação independente da estrutura de armazenamento.

Projeto *FreeCell*: Pilha Burra ou Pilha Inteligente?


Solução A
Aplicação
+ "inteligência"
Pilha
"Burra"
Interface
Solução B
Aplicação
Pilha
Intermediária
+
"inteligência"
Pilha
Definitiva
+
"inteligência"
Interface
Solução C
Aplicação
Pilha Intermediária
+
"inteligência"
Pilha Definitiva
+
"inteligência"
Interface
Pilha "Burra"

Diretrizes de Projeto



- Para manipular as Pilhas de Cartas, projete e utilize um TAD Pilha (seja uma Pilha Burra, seja uma Pilha Inteligente);
- A Aplicação em si e o TAD Pilha devem estar em unidades de software independentes, e em arquivos separados;
- A Aplicação (e outros módulos) deve(m) manipular o TAD Pilha exclusivamente através dos seus Operadores Primitivos - Empilha, Desempilha, Vazia, Cria e Cheia;
- Inclua no código do TAD Pilha exclusivamente ações pertinentes ao armazenamento e recuperação das informações sobre as Pilhas de Cartas.

Avanço de Projeto

Exercício 2.11 Arquitetura de Software
Exercício 2.12 Implementar uma Pilha Burra
Exercício 2.13 Pilha Burra como uma Classe
Exercício 2.14 Carta do Baralho
Exercício 2.15 Pilha Intermediária Inteligente
Exercício 2.16 Pilha Definitiva Inteligente
Exercício 2.17 Definir Regras, Escolher Nome
Exercício 2.18 Iniciar o Projeto da Interface

Comece a Desenvolver Seu *FreeCell* Agora!

Faça um jogo legal! Faça um jogo que tenha a sua cara! Faça seu jogo ficar atrativo; distribua para os amigos; disponibilize publicamente; faça seu jogo bombar! Aprender a programar pode ser divertido!

Estruturas de Dados com Jogos

Aprender a programar pode ser divertido!