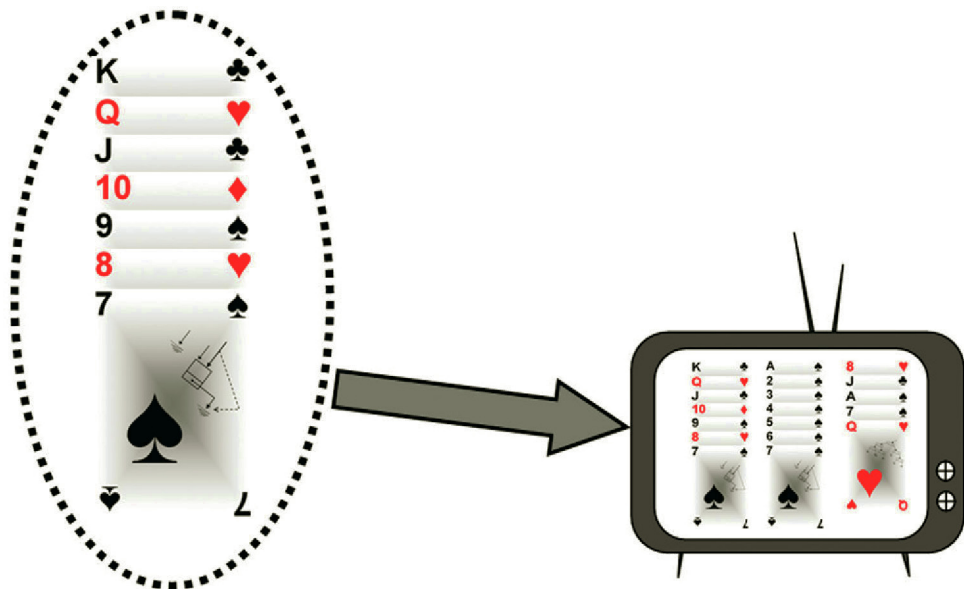


Capítulo 1

Tipos Abstratos de Dados



Seus objetivos neste capítulo

- Entender o conceito de Tipos Abstratos de Dados e o modo de utilizá-lo no desenvolvimento de programas.
- Perceber que o uso de Tipos Abstratos de Dados dá ao software maior portabilidade, maior potencial para reutilização, reduz custos de desenvolvimento e de manutenção.
- Conscientizar-se quanto à importância de adotar uma estratégia que agregue portabilidade e reusabilidade aos jogos que você desenvolverá.

1.1 Queremos desenvolver um jogo. E agora?

Suponha que você e mais dois amigos tenham decidido desenvolver um jogo no mundo virtual — um software. E agora? Qual o primeiro passo?

Quando queremos desenvolver um software, o primeiro passo é decidir *o que* desenvolver. O ciclo de vida tradicional do desenvolvimento de software envolve as fases de análise, projeto, implementação, teste e manutenção ([Figura 1.1](#)). Na fase de análise, determinamos *o que* o software precisa fazer, quais problemas deverá resolver e quais informações deverá manipular.



Figura 1.1 Fases do desenvolvimento de software.

Na fase de projeto, o objetivo é definir *como* o software será desenvolvido. Estamos falando de uma definição geral, não detalhada. Na fase de projeto, escolhemos a arquitetura do software, definimos quais serão os principais módulos e, com isso, podemos dividir o trabalho entre os desenvolvedores — cada membro da equipe fica responsável pela implementação de um dos módulos.

As outras fases referem-se à implementação (codificação em uma linguagem de programação), teste (para garantir que o software funcione segundo o esperado) e manutenção (ajuste ou evolução nas funcionalidades do software, ao longo de seu período de utilização). Para uma leitura complementar sobre o ciclo de vida tradicional do desenvolvimento de software, consulte [Sommerville \(2003\)](#) e [Pressman \(1995\)](#).

Vamos supor que você e seus dois amigos aceitaram o Desafio 1 do livro *Estruturas de dados com jogos* e decidiram desenvolver um jogo como o *FreeCell*. Já sabem *o que* desenvolver e estão agora na fase de projeto, definindo *como* desenvolver. E vocês estão decididos a escolher a melhor estratégia para desenvolver esse jogo. É precisamente nesse contexto, na fase de projeto do software, em busca da melhor estratégia, que você precisa conhecer e utilizar o conceito de Tipos Abstratos de Dados.

1.2 Definição de Tipos Abstratos de Dados

Um Tipo Abstrato de Dados é formado por uma coleção de **Dados** a serem armazenados e um conjunto de **Operações** ou ainda **Operadores** que podem ser aplicados para manipulação desses Dados ([Langsam, Augenstein e Tenenbaum, 1996](#); [Celes, Cerqueira e Rangel, 2004](#)). Toda manipulação desse conjunto de Dados, para fins de armazenamento e recuperação, deve ser realizada exclusivamente através dos Operadores.

Definição — Tipo Abstrato de Dados (TAD):

Um Tipo Abstrato de Dados é constituído por um conjunto de **Dados** a serem armazenados e por um grupo de **Operadores** que podem ser aplicados para manipulação desses Dados.

Manipulação dos Dados armazenados:

O armazenamento e a recuperação dos Dados devem ser realizados **exclusivamente** através dos Operadores do TAD.

Em que momento identificar e projetar um TAD:

Na fase de projeto do software.

Figura 1.2 Definição de Tipo Abstrato de Dados.

Em que momento devem ser identificados e projetados os Tipos Abstratos de Dados? Na fase de projeto do software, ou seja, no momento em que definimos uma visão geral, não detalhada, de *como* o software será desenvolvido.

1.3 Exemplo: *FreeCell*

Como o conceito de Tipos Abstratos de Dados pode ser utilizado no projeto do *FreeCell*, o jogo do Desafio 1 que queremos desenvolver? Temos no *FreeCell* algum conjunto de Dados a serem armazenados? Sim! Temos, por exemplo, os valores e os naipes das cartas e também a sequência dessas cartas nas *Pilhas Intermediárias*.

Um dos Operadores para manipulação desses Dados tem por objetivo retirar a carta que está no topo da pilha. Um segundo Operador poderia ser colocar uma carta no topo da pilha, caso o valor da carta estiver na sequência correta. Nessas *Pilhas Intermediárias*, só é possível inserir cartas no topo da pilha em ordem decrescente e em cores alternadas — pretas sobre vermelhas e vermelhas sobre pretas.

A partir da definição do TAD Pilha Intermediária do *FreeCell*, o aplicativo *FreeCell* como um todo deve realizar o armazenamento das cartas nas *Pilhas Intermediárias* de modo abstrato, ou seja, sem se preocupar com os detalhes de como esses dados são efetivamente armazenados. A recuperação dos dados também deve ocorrer de modo abstrato. Funcionaria como se as *Pilhas Intermediárias* fossem *caixas-pretas*, com uma operação para retirar a carta que está no topo da pilha e outra para colocar uma carta no topo da pilha, caso a sequência estiver correta. Os demais módulos do programa simplesmente pediriam “Retire a carta que está no topo desta pilha” ou “Coloque esta carta no topo daquela pilha”, e a *caixa-preta* que cuida disso daria um jeito de atender as solicitações.


TAD Pilha Intermediária do <i>FreeCell</i>		
	Coleção de Dados a Serem Armazenados	Operadores Para Manipulação
	Para cada uma das <i>Pilhas Intermediárias</i> : <ul style="list-style-type: none"> As cartas que estão na pilha - valor e naipe de cada carta; A sequência das cartas na pilha. 	<ul style="list-style-type: none"> Retira a carta que está no topo da pilha; Coloca uma carta no topo da pilha, se o valor estiver na sequência correta.

Figura 1.3 Exemplo de TAD Pilha Intermediária do *FreeCell*.

Operações e parâmetros	Funcionamento
Desempilha (Pilha, Carta, DeuCerto)	Retira a Carta que está no topo da Pilha passada como parâmetro. Se a operação for bem-sucedida, o parâmetro DeuCerto retornará o valor Verdadeiro, e o parâmetro Carta retornará o valor da carta retirada do topo da Pilha. Caso não houver carta a ser desempilhada, o parâmetro DeuCerto retorna o valor Falso.
EmpilhaNaSequência (Pilha, Carta, DeuCerto)	Insere a Carta passada como parâmetro na Pilha passada como parâmetro, caso o valor da Carta estiver na sequência correta para uma <i>Pilha Intermediária</i> — ordem decrescente e em cores alternadas. O parâmetro DeuCerto retornará o valor Verdadeiro se a Carta foi empilhada corretamente, e o valor Falso, caso contrário.
EmpilhaSempre (Pilha, Carta)	Insere a Carta passada como parâmetro no topo da Pilha passada como parâmetro, mesmo se a Carta não estiver na sequência correta para uma <i>Pilha Intermediária</i> .

Figura 1.4 Operações do TAD Pilha Intermediária.

Suponha que, na fase de projeto, tenha sido definido o TAD Pilha Intermediária do *FreeCell* e suas operações detalhadas conforme mostra a [Figura 1.4](#).

Suponha também que outro projetista de software tenha ficado encarregado de implementar o TAD Pilha Intermediária do *FreeCell*. Coube a você desenvolver a operação para transferir uma carta entre duas *Pilhas Intermediárias*, caso a *Carta* retirada da *Pilha de Origem* estiver na sequência correta na *Pilha de Destino*, conforme as regras do *FreeCell*. Como você desenvolveria essa operação sem ter a menor ideia quanto aos detalhes da implementação da Pilha Intermediária?

Exercício 1.1 Transfere Carta

Desenvolva uma operação para transferir uma *Carta* entre duas *Pilhas Intermediárias* do *FreeCell*. A transferência só poderá ser realizada se a *Carta* retirada da *PilhaOrigem* estiver na sequência correta na *PilhaDestino*, conforme as regras de funcionamento do *FreeCell*. Se a *Carta* não for efetivamente transferida, deve retornar à *PilhaOrigem*. Essa operação deve ser implementada com o uso dos operadores do TAD Pilha Intermediária do *FreeCell*, conforme especificado na [Figura 1.4](#).

TransfereCarta (parâmetros por referência PilhaOrigem, PilhaDestino do tipo PilhaIntermediária, Parâmetro por referência DeuCerto do tipo Boolean);

/* Transfere uma carta da Pilha Origem para a PilhaDestino, caso a carta estiver na sequência correta na PilhaDestino. O parâmetro DeuCerto retornará o valor Verdadeiro se uma carta for efetivamente transferida, e o valor Falso caso contrário */

A [Figura 1.5](#) apresenta um algoritmo conceitual para a operação, que transfere uma carta entre duas *Pilhas Intermediárias*. Note que a manipulação dos Dados armazenados nas *Pilhas Intermediárias* é feita exclusivamente através dos Operadores definidos na especificação do TAD ([Figura 1.4](#)). Devido a isso, para desenvolver essa solução não foi

```
TransfereCarta(parâmetros por referência PilhaOrigem, PilhaDestino do tipo
PilhaIntermediária, parâmetro por referência DeuCerto do tipo Boolean) {

    /* Transfere uma carta da Pilha Origem para a PilhaDestino, caso a carta estiver na
    sequência correta na PilhaDestino. O parâmetro DeuCerto retornará o valor Verdadeiro se
    uma carta for efetivamente transferida, e o valor Falso caso contrário */

    Variável Carta do tipo Carta-do-Baralho;
    Variável ConseguiuRetirar do tipo Boolean;
    Variável ConseguiuEmpilhar do tipo Boolean;

    /* Tenta retirar Carta do topo da PilhaOrigem */
    Desempilha(PilhaOrigem, Carta, ConseguiuRetirar);
    Se (ConseguiuRetirar == Verdadeiro)
    Então { /* empilha na Pilha Destino, se estiver na sequência correta */
        EmpilhaNaSequência(PilhaDestino, Carta, ConseguiuEmpilhar);
        Se (ConseguiuEmpilhar == Verdadeiro)
        Então DeuCerto = Verdadeiro;
        Senão { /* carta não está na sequência correta e deve retornar à PilhaOrigem */
            EmpilhaSempre(PilhaOrigem, Carta);
            DeuCerto = Falso;
        };
    };
    Senão DeuCerto = Falso;
}
```

Figura 1.5 Algoritmo conceitual — Transfere Carta.

preciso conhecer detalhes sobre a implementação das Pilhas Intermediárias. Não é legal programar assim?

As Operações do TAD Pilha Intermediária, conforme constam na [Figura 1.4](#), e o algoritmo conceitual da [Figura 1.5](#) têm por objetivo apenas exemplificar o conceito de Tipos Abstratos de Dados. A concepção das Operações de uma Pilha deverá ser aprimorada a partir da apresentação de novos conceitos no Capítulo 2.

1.4 Qual a melhor maneira de aumentar o volume da televisão?

Temos duas estratégias alternativas para aumentar o volume de uma televisão. Primeira alternativa: podemos aumentar o volume da televisão acionando o botão do volume, no controle remoto ou na própria televisão. Na segunda alternativa, pegamos uma chave de fenda, abrimos a televisão, mexemos com a chave de fenda em um componente eletrônico e aumentamos o volume “na marra”. Qual alternativa você acha mais interessante: botão de volume ou chave de fenda?

Em uma analogia, podemos considerar um Tipo Abstrato de Dados como uma televisão. Podemos desenvolver programas aumentando o volume através do botão da televisão ou podemos desenvolver programas aumentando o volume com uma chave de fenda. Aumentar o volume pelo botão da televisão significa identificar Tipos Abstratos de Dados, definir Dados a serem armazenados, definir Operadores aplicáveis a esses Dados e, a partir de então, só mexer nesses Dados através dos Operadores do TAD. Os Operadores do TAD, em nossa analogia, equivalem aos botões da televisão ([Figura 1.6](#)).

Qual o papel de um TAD no projeto de software? Um Tipo Abstrato de Dados é um *modelo abstrato* do armazenamento e manipulação de determinado conjunto de Dados



	Operadores do TAD ou Botões da TV
	<p>Pilha de Cartas:</p> <ul style="list-style-type: none">• Retira a carta que está no topo da pilha;• Coloca uma carta no topo da pilha, se o valor estiver na sequência correta.
	<p>TV:</p> <ul style="list-style-type: none">• Aumenta o volume;• Diminui o volume;• Muda de canal (1 canal acima);• Muda de canal (1 canal abaixo).

Figura 1.6 Analogia: Operadores do TAD e botões da televisão.

de um programa. Um TAD é uma *caixa-preta*. O que é visível externamente a essa *caixa-preta* é a sua funcionalidade, ou seja, as Operações definidas para o TAD. Os detalhes de implementação ficam escondidos.

O principal propósito desse *modelo abstrato* chamado Tipo Abstrato de Dados é simplificar. Por exemplo, uma vez definido o TAD Pilha de Cartas, ao construir a lógica da aplicação devemos abstrair, nos despreocupar ou simplesmente esquecer como o armazenamento das cartas é efetivamente implementado. Devemos simplesmente acionar as operações da Pilha de Cartas, ou os “botões da televisão”, para construir a lógica da aplicação. É mais simples programar assim, não é?

É possível alterar os dados de uma Pilha de Cartas sem ser por intermédio de uma operação definida na especificação do TAD Pilha de Cartas? Sim, é possível. Mas, fazendo isso, estamos optando pela estratégia da chave de fenda, ou seja, estamos abrindo a televisão com uma chave de fenda para aumentar o volume. Utilizar o botão de volume é uma estratégia melhor!

1.5 O que é um bom programa?

Não, um bom programa não é simplesmente um programa que funciona. Na verdade, um programa que não funciona ainda não é um programa. Vamos detalhar a pergunta: temos dois programas e os dois funcionam. Um deles é um programa bom e o outro é um programa ruim. Quais são as possíveis características do programa que é bom?

Você sabe o que significa *portabilidade* de código? Resumidamente, **portabilidade** é a capacidade de um código (trecho de programa) executar em diferentes plataformas de hardware e software. Assim como podemos carregar um computador portátil de um lugar para o outro, podemos carregar um software portátil de uma plataforma para outra, e esse software continuará funcionando. Pense, por exemplo, em um editor de texto. Como ele consegue executar a operação de imprimir, mesmo se você trocar a sua impressora? *Portabilidade* de software: capacidade de executar em diferentes plataformas.

E o termo **reusabilidade**, você sabe o que significa? *Reusabilidade* de código ou software significa reutilizar um software já desenvolvido, em uma segunda situação. Ou seja, você desenvolve o software para uma necessidade e o aproveita (reutiliza) para satisfazer uma segunda necessidade.

Portabilidade de software: capacidade de executar em diferentes plataformas de hardware e software.

Reusabilidade de software: capacidade de aproveitar (reutilizar) um software já desenvolvido, para satisfazer uma segunda necessidade.

Figura 1.7 Portabilidade e reusabilidade de software.

Outro critério que pode ser utilizado para diferenciar programas bons de programas ruins é a **eficiência** — um programa que executa mais rápido ou utiliza menos memória. Se você tivesse que escolher entre um código com maior portabilidade e reusabilidade, e um código que execute milésimos de segundo mais rápido, o que escolheria?

Existem circunstâncias específicas nas quais certamente é necessário priorizar a eficiência. Por exemplo, quando precisamos garantir que o tempo de resposta de um software satisfaça critérios muito rígidos, para que não ocorram grandes desastres. Mas, na maior parte das circunstâncias, as opções de projeto devem priorizar o que resulta em custo mais baixo no desenvolvimento e na manutenção de software.

Portabilidade e reusabilidade são características de um software que resultam em custo mais baixo de desenvolvimento e manutenção.

1.6 Vantagens da utilização de Tipos Abstratos de Dados

Podemos apontar as seguintes vantagens da utilização do conceito de Tipos Abstratos de Dados:

- **É mais fácil programar, sem se preocupar com detalhes de implantação.** Por exemplo, no momento de transferir dados de uma *Pilha de Cartas* para outra, você não precisa se preocupar em saber como a *Pilha* é efetivamente implementada. Você precisa apenas saber utilizar as operações que colocam ou retiram cartas em uma *Pilha*.
- **É mais fácil preservar a integridade dos dados.** Apenas as operações do Tipo Abstrato de Dados alteram os dados armazenados. Suponha que as operações do TAD *Pilha de Cartas* estejam corretas, não corrompendo os dados nem ocasionando perda de dados. Se você alterar os Dados pelos Operadores do TAD, os Dados certamente serão preservados. Mas, se você não conhece muito bem a implementação do TAD *Pilha de Cartas* e decide alterar os Dados *com a chave de fenda* (ou seja, sem usar os botões da televisão), é possível que faça alguma besteira e acabe corrompendo os dados.
- **Maior independência e portabilidade de código.** Alterações na implementação de um TAD não implicam em alterações nas aplicações que o utilizam. Vamos supor que

implementamos um TAD *Pilha de Cartas* utilizando determinada técnica de implementação, mas depois decidimos mudar a técnica de implementação. Se uma aplicação que usa o TAD tiver sido implementada exclusivamente através dos Operadores do TAD, essa aplicação continuará funcionando, ainda que a implementação do TAD *Pilha de Cartas* seja completamente alterada. O que mudou foi a implementação do TAD, mas seus Operadores continuam os mesmos! Logo, a aplicação continuará funcionando.

- **Maior potencial de reutilização de código.** Pode-se alterar a lógica de um programa sem necessidade de reconstruir as estruturas de armazenamento. Um mesmo TAD *Pilha de Cartas* pode ser utilizado no desenvolvimento do *FreeCell*, no desenvolvimento de variações do *FreeCell* e também no desenvolvimento de outras aplicações que utilizam pilhas de cartas.

Se adotarmos uma estratégia de desenvolvimento com o uso de Tipos Abstratos de Dados, ou seja, aumentando o volume da televisão sempre pelos botões e nunca com a chave de fenda, o desenvolvimento será mais fácil, os dados estarão mais seguros, o código terá um potencial maior para executar em diferentes plataformas e para ser reutilizado em outras aplicações. Como consequência, teremos custo menor de desenvolvimento e manutenção. Software bom, bonito e barato!

Vamos adotar Tipos Abstratos de Dados no desenvolvimento de nossos jogos?

Software bom, bonito e barato

O uso do conceito de Tipos Abstratos de Dados aumenta a portabilidade e o potencial de reutilização do software. Em consequência disso, o custo de desenvolvimento e manutenção é reduzido.

Consulte no Banco de Jogos

Adaptações do *FreeCell*



<http://www.elsevier.com.br/edcomjogos>

Exercícios de fixação

Exercício 1.2 O que é um Tipo Abstrato de Dados (TAD)? Como os dados armazenados em um TAD devem ser manipulados? Em que momento um TAD deve ser identificado e definido?

Exercício 1.3 Como é possível desenvolver um programa utilizando um TAD *Pilha de Cartas*, por exemplo, sem conhecer detalhes de sua implementação?

Exercício 1.4 O que é portabilidade de código? O que é reusabilidade de código?

Exercício 1.5 Faça uma pesquisa sobre portabilidade e reusabilidade de software. Converse com os colegas sobre o que você considera importante sobre isso. Você concorda que portabilidade e reusabilidade são, na maioria das situações, características mais importantes para um programa do que executar milésimos de segundo mais rápido?

Exercício 1.6 Quais as vantagens de programar utilizando o conceito de TAD? Explique com exemplos.

Exercício 1.7 Consulte em um dicionário o significado do termo “abstrato”. Consulte as palavras “abstrato”, “abstrair” e “abstração”.

Referências e leitura adicional

- Celes, W.; Cerqueira, R.; Rangel F. L. *Introdução a estruturas de dados*. Rio de Janeiro, Elsevier; 2004. p. 126.
- Langsam, Y.; Augenstein, M.J.; Tenenbaum, A.M. *Data Structures Using C and C++*. 2nd ed. Upper Saddle River. New Jersey: Prentice Hall, 1996. p. 13.
- Pressman, R. *Engenharia de software*. 3. ed. São Paulo: Makron Books, 1995. p. 32-35.
- Sommerville R. I. *Engenharia de software*. 6. ed. São Paulo: Addison Wesley, 2003. p. 35-38.