1 Comece com as especificações

Todo projeto de desenvolvimento profissional de software começa com uma especificação e este não será exceção. Você desenvolverá um clássico jogo chamado Vá Pescar! Algumas pessoas jogam com regras ligeiramente diferentes. Então eis aqui um lembrete das que você usará:

- ★ O jogo começa com um baralho de 52 cartas. Cinco são dadas para cada jogador. As restantes são chamadas de monte. Cada jogador, na sua vez, pergunta por um **valor** ("alguém tem sete?"). Qualquer outro jogador que tenha cartas com aquele valor deve entregá-las. Se ninguém tiver uma, o jogador precisa "ir pescar" no monte, pegando uma carta dele.
- ★ O objetivo do jogo é fazer "livros", que são conjuntos completos de todas as quatro cartas com o mesmo valor. O jogador com a maior quantidade deles no fim do jogo é o vencedor. Tão logo um jogador tenha montado um livro, ele o coloca virado para cima na mesa para que todos os demais jogadores possam ver que livros cada um tem.
- ★ Quando um jogador coloca um livro na mesa, isso pode fazer com que ele fique sem cartas. Se for o caso, ele tem que pegar mais cinco do monte. Se sobraram menos de cinco nela, ele tem que pegar todas. O jogo acaba assim que o monte acabar. O vencedor é então aquele que tiver mais livros.
- ★ Para essa versão computadorizada de "vá pescar", deve existir dois jogadores programados e um jogador humano. Todos os turnos começam com o jogador humano selecionando uma das cartas em sua mão, que deverá ser visível sempre para ele. Ele faz isso escolhendo uma de suas cartas e indicando que ele vai perguntar por um valor. Os dois jogadores programados então devem, em seguida, pedir seus valores também. O resultado de cada turno deve ser exibido. Isso se repetirá até que um dos jogadores vença.
- ★ O jogo deve cuidar de todos os detalhes de troca de cartas e montagem de livros automaticamente. Uma vez que um jogador tenha vencido, o jogo termina. O nome do vencedor deve então ser exibido (ou dos vencedores, se houver empate). Nenhuma outra ação pode ser feita – o jogador deve reiniciar o programa para começar outro jogo.

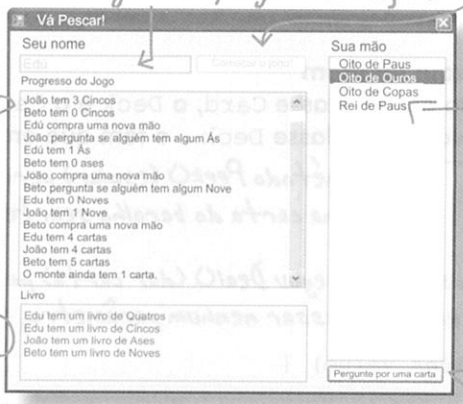
Se você não souber o que está desenvolvendo antes de começar, como saberá quando terminou? É por isso que a maior parte dos projetos profissionais de desenvolvimento de software começa com uma especificação que lhe diz o que vai ser feito.

2 **Crie o formulário**

Crie o formulário para o jogo Vá Pescar. Ele deve ter um controle de caixa de listagem para a mão dos jogadores, duas caixas de texto para o progresso do jogo e um botão para que o jogador pergunte por uma carta. Para jogar, o usuário tem que selecionar uma das cartas de sua mão e clicar o botão para perguntar se os jogadores computadorizados têm cartas com aquele valor.

Esse controle de caixa de texto deve ter sua propriedade Name alterada para textName. Nessas imagens do formulário ele aparece desabilitado, mas ele deve ser ativado quando o programa começar.

Atribua à propriedade Name deste botão o valor buttonStart (botão iniciar). Ele está desabilitado nessa imagem, mas ele começa ativo. Ele deve ser desativado quando o jogo começa.



Estes são controles TextBox chamados textProgress e textBooks.

A mão atual do jogador é mostrada num controle ListBox chamado listHand (lista mão). Você pode atribuir esse nome a ele usando sua propriedade Name.

Iguale a propriedade ReadOnly (apenas leitura) das duas caixas de texto para True (verdadeiro) - isso fará com que elas sejam caixas de texto só para leitura.

Atribua buttonAsk (botão perguntar) para a propriedade Name deste botão e False (falso) para a sua propriedade Enabled (ativo). Isso vai desabilitá-lo, ou seja, ele não pode ser pressionado. O formulário deve ativá-lo assim que o jogo começar.

Exercício Longo (continuação)

3 **Eis o código do formulário**
Digite-o exatamente como você o vê aqui. O restante do código a ser escrito você mesmo deve interagir com ele.

```
public partial class Form1 : Form {
  public Form1() {
    InitializeComponent();
  }
  private Game game;
  private void buttonStart_Click(object sender, EventArgs e) {
    if (String.IsNullOrEmpty(textName.Text)){
      MessageBox.Show("Please enter your name", "Can't start the game yet");
      return;
    }
    game = new Game(textName.Text, new string[] { "Joe", "Bob" }, textProgress);
    buttonStart.Enabled = false;
    textName.Enabled = false;
    buttonAsk.Enabled = true;
    UpdateForm();
  }
  private void UpdateForm() {
    listHand.Items.Clear();
    foreach (String cardName in game.GetPlayerCardNames())
      listHand.Items.Add(cardName);
    textBooks.Text = game.DescribeBooks();
    textProgress.Text += game.DescribePlayerHands();
    textProgress.SelectionStart = textProgress.Text.Length;
    textProgress.ScrollToCaret();
  }
```

Esta é a única classe com a qual o formulário interage. Ela executa todo o jogo.

A propriedade Enabled ativa ou desativa um controle no formulário.

Quando você começa um novo jogo, ele cria uma nova instância da classe Game, ativa o botão pergunte, desativa o iniciar o jogo e repinta o formulário.

Esse método limpa e repovoa a caixa de listagem que mantém a mão do jogador, e então atualiza as caixas de texto.

Usar SelectionStart() (início da seleção) e ScrollToCaret() (role até o cursor). Assim, a caixa rolará até exibir a parte final de seu texto, de modo que se ele for muito grande aparecerá só o seu final.

A linha com SelectionStart move o cursor da caixa de texto até o final. Uma vez que tenha feito isso, o método ScrollToCaret() rola a caixa até a posição do cursor.

```
  private void buttonAsk_Click(object sender, EventArgs e) {
    textProgress.Text = "";
    if (listHand.SelectedIndex < 0) {
      MessageBox.Show("Please select a card");
      return;
    }
    if (game.PlayOneRound(listHand.SelectedIndex)) {
      textProgress.Text += "The winner is... " + game.GetWinnerName();
      textBooks.Text = game.DescribeBooks();
      buttonAsk.Enabled = false;
    } else
      UpdateForm();
  }
}
```

O jogador seleciona uma das cartas e clica no botão pergunte para ver se algum dos outros jogadores tem uma carta com o mesmo valor da selecionada. A classe Game joga um turno usando o método PlayOneRound() (jogar um turno).

4 **Você vai precisar deste código aqui também**
Você vai precisar do código escrito para a classe `Card`, a `Deck` e a `CardComparer_byValue`. Mas você deverá adicionar mais alguns métodos na classe `Deck`... e tem que entendê-los para por usá-los.

```
public Card Peek(int cardNumber) {
    return cards[cardNumber];
}
public Card Deal() {
    return Deal(0);
}
public bool ContainsValue(Card.Values value) {
    foreach (Card card in cards)
      if (card.Value == value)
        return true;
    return false;
}
```

O método Peek() (espiar) permite que você dê uma espiada numa carta do baralho sem ter de pegá-la para você.

Alguém sobrecarregou Deal() (dar carta) para deixá-lo um pouco mais fácil de ler. Se você não passar nenhum parâmetro, ele dá a carta do topo do baralho.

O método ContainsValue() procura pelo baralho todo por cartas um valor e retorna true se encontrar alguma. Você pode adivinhar como usará esse método nesse jogo?

```
public Deck PullOutValues(Card.Values value) {
    Deck deckToReturn = new Deck(new Card[] { });
    for (int i = cards.Count - 1; i >= 0; i--)
      if (cards[i].Value == value)
        deckToReturn.Add(Deal(i));
    return deckToReturn;
}
public bool HasBook(Card.Values value) {
    int NumberOfCards = 0;
    foreach (Card card in cards)
      if (card.Value == value)
        NumberOfCards++;
    if (NumberOfCards == 4)
      return true;
    else
      return false;
}
public void SortByValue() {
    cards.Sort(new CardComparer_byValue());
}
```

Você usará o método PullOutValues() (retirar valores) quando fizer o código para conseguir um livro de cartas do baralho. Ele procura por qualquer carta com um determinado valor, tira-as do baralho e retorna um baralho novo sem aquelas cartas.

O método HasBook() (tem livro) checa um baralho para ver se ele contém um livro de quatro cartas de qualquer valor que seja passado como parâmetro. Ele retorna true se existir um livro no baralho e false se não.

O método SortByValue() ordena o baralho usando a classe Comparer_byValue.

5 **Agora vem a parte DIFÍCIL: desenvolva a classe Player**

Existirá uma instância de Player (jogador) para cada um dos três jogadores. Elas serão criadas pelo tratador de eventos do botão `buttonStart`.

Olhe atentamente para cada comentário - eles dizem a você o que o método deve fazer. Sua tarefa é desenvolvê-los.

```
public class Player
{
  private string name;
  public string Name { get { return name; } }
  private Random random;
  private Deck cards;
  private TextBox textBoxOnForm;
  public Player(String name, Random random, TextBox textBoxOnForm) {
    // o construtor para a classe Player inicializa quatro campos private, e entao
    // adiciona uma linha no controle TextBox no formulario que diz "Joao entrou
    // no jogo" - mas use o nome no campo privado e nao se esqueca de
    // adicionar uma quebra ("\r\n") no fim de cada linha adicionada na caixa de texto
  }
  public List<Card.Values> PullOutBooks() { } // veja o código no final da página
  public Card.Values GetRandomValue() {
    // esse metodo retorna uma valor aleatorio - mas devem ser um que exista no baralho!
  }
  public Deck DoYouHaveAny(Card.Values value) {
    // neste metodo um oponente pergunta se o jogador tem cartas de um certo valor
    // usando Deck.PullOutValues() para retirar os valores. Adicione uma linha na
    // caixa de texto que diz "Joao tem 3 Seis" - use o novo metodo estatico Card.Plural()
  }
  public void AskForACard(List<Player> players, int myIndex, Deck stock) {
    // Aqui temos uma versao sobrecarregada de AskForACard() - escolha um valor
    // aleatorio do baralho usando GetRandomValue() e pergunte por ele usando AskForACard()
  }
  public void AskForACard(List<Player> players, int myIndex, Deck stock, Card.Values value) {
    // pergunte para outros jogadores se eles tem um dado valor. Primeiro adicione
    // uma linha na caixa de texto que diz "Joao pergunta se alguem tem alguma Rainha",
    // por exemplo. Entao itere pela lista de jogadores passada como parametro e para
    // cada um deles pergunte se ele tem o valor (usando seu método DoYouHaveAny).
    // Ele deve passar a você um baralho - adicione este ao seu. Mantenha registro
    // de quantas cartas foram adicionadas. Se forem zero, teremos que pegar uma do
    // monte (que também foi passado como parametro), e nesse caso uma linha
    // "joao tem que pegar uma carta da pilha" deve ser adicionada
  }
  // Eis aqui uma propriedade e alguns metodos pequenos que ja foram escritos para voce
  public int CardCount { get { return cards.Count; } }
  public void TakeCard(Card card) { cards.Add(card); }
  public string[] GetCardNames() { return cards.GetCardNames(); }
  public Card Peek(int cardNumber) { return cards.Peek(cardNumber); }
  public void SortHand() { cards.SortByValue(); }
}
```

Ainda não terminou - vire a página!

Exercício Longo (continuação)

Esse método Peek() acrescentado à classe Deck será útil. Ele permite ao programa olhar uma das cartas do baralho dado um índice, mas diferente de Deal(), ele não remove a carta.

```
public List<Card.Values> PullOutBooks() {
  List<Card.Values> Books = new List<Card.Values>();
  for (int i = 1; i <= 13; i++) {
    Card.Values value = (Card.Values)i;
    int howMany = 0;
    for (int card = 0; card < cards.Count; card++)
      if (cards.Peek(card).Value == value)
        howMany++;
    if (howMany == 4) {
      Books.Add(value);
      for (int card = cards.Count - 1; card >= 0; card--)
        cards.Deal(card);
    }
  }
  return Books;
}
```

Você terá que desenvolver DUAS versões sobrecarregadas do método AskForACard(). A primeira será usada pelos oponentes quando eles perguntarem por valores - ela procura nas próprias mãos por uma carta para originar a pergunta. A segunda será usada quando o jogador perguntar por uma carta. As duas perguntam para TODOS os demais jogadores (tanto computadorizados quanto humanos) por qualquer carta que tenha o mesmo valor.

6 **Você precisará adicionar o seguinte método na classe Card**
É um método estático que recebe um valor e retorna-o no plural - dessa forma um "cinco" retornará "cincos", mas um "seis" deve retornar "seis". Uma vez que ele é estático, deve ser chamado diretamente com o nome da classe - `Card.Plural()` - e não de uma instância.

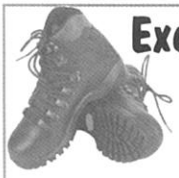
```
public partial class Card {
  public static string Plural(Card.Values value) {
    if (value == Values.Six)
      return "Sixes";
    else
      return value.ToString() + "s";
  }
}
```

7 **O resto do trabalho: desenvolva a classe Game**
O formulário mantém uma instância de Game. Ela gerencia o jogo. Examine atentamente como ela é usada no formulário.

```
public class Game {
   private List<Player> players;
   private Dictionary<Card.Values, Player> books;
   private Deck stock;
   private TextBox textBoxOnForm;
   public Game(string playerName, string[] opponentNames, TextBox textBoxOnForm) {
      Random random = new Random();
      this.textBoxOnForm = textBoxOnForm;
      players = new List<Player>();
      players.Add(new Player(playerName, random, textBoxOnForm));
      foreach (string player in opponentNames)
           players.Add(new Player(player, random, textBoxOnForm));
      books = new Dictionary<Card.Values, Player>();
      stock = new Deck();
      Deal();
      players[0].SortHand();
}
private void Deal() {
      // sera aqui o metodo A PARTIR DO QUAL o jogo comeca - ele sera chamado apenas no
      // inicio do jogo. Ele embaralha o monte, dá cinco cartas para cada jogador e
      // usa um laco foreach para chamar o metodo PullOutBooks() de cada um.
}
public bool PlayOneRound(int selectedPlayerCard) {
      // Execute um turno do jogo. O parametro sera a carta que o jogador selecionou
      // na sua mao - recupera seu valor. Entao itere por todos os jogadores e chame o
      // metodo AskForACard() de cada um, começando pelo jogador humano (que deve
      // estar no indice zero na lista de jogadores. Certifique-se de que ele pergunte pelo valor
```

```
        // da carta que selecionou). Entao chame PullOutBooks() - se ele retornar true,
        // o jogador ficou sem cartas e precisa comprar uma nova mao. Depois
        // que todos jogaram, ordene a mao do jogador humano (para que ela apareca bem
        // arrumada no formulario). Verifique se o monte ainda tem cartas. Se nao tem,
        // apague o texto na caixa e escreva "O monte esta sem cartas. O jogo acabou!" e
        // retorne true. Se nao for o caso, o jogo ainda nao acabou, logo retorne false.
    }
    public bool PullOutBooks(Player player) {
        // Monte um livro para um jogador. Retorne true se o jogador ficar sem cartas. Se
        // nao for o caso, retorne false. Cada livro deve ser adicionado ao dicionario Books.
        // Um jogador fica sem cartas quando usar todas as suas restantes para montar um
        // livro - e assim tentar ganhar o jogo.
    }
    public string DescribeBooks() {
        // Retorne uma longa string que descreve os livros de todos, composta examinando
        // o dicionario Books. "Joao tem um livro de seis. (quebra de linha) Edu tem um livro
        de Ases."
    }

    public string GetWinnerName() {
        // este metodo sera chamado no final do jogo. Ele usa seu proprio dicionario
        // (Dictionary<string, int> winners) para determinar quantos livros cada jogador
        // acrescentou no dicionario Books. Primeiro ele usa um laco foreach em Book.Keys
        // foreach(Card.Values value in Books.Keys) - para povoar seu dicionario winners
        // (vencedores) com a quantidade de livros de cada um. Entao ele itera por esse
        // dicionário para encontrar o maior numero de livros, o que determina o vencedor.
        // E finalmente ele faz uma ultima passagem por winners para montar uma string
        // com o nome dos vencedores ("Joao e Edu", por exemplo). Se temos um unico
        // vencedor,  ele deve retornar uma string como "Edu com tres livros". Se temos mais
        // de um, ele deve retornar algo como "um empate entre Joao e Beto com 2 livros"
    }
    // Eis aqui dois pequenos metodos que ja foram escritos para você
    public string[] GetPlayerCardNames() {
      return players[0].GetCardNames();
    }
    public string DescribePlayerHands() {
      string description = "";
      for (int i = 0; i < players.Count; i++) {
        description += players[i].Name + " has " + players[i].CardCount;
        if (players[i].CardCount == 1)
            description += " card.\r\n";
        else
            description += " cards.\r\n";
      }
      description += "The stock has " + stock.Count + " cards left.";
      return description;
    }
```



Exercício Longo

Aqui estão os métodos da classe Game.

```
public class Game {
  private void Deal() {
    stock.Shuffle();
    for (int i = 0; i < 5; i++)
      foreach (Player player in players)
        player.TakeCard(stock.Deal());
    foreach (Player player in players)
      PullOutBooks(player);
  }
  public bool PlayOneRound(int selectedPlayerCard) {
   Card.Values cardToAskFor = players[0].Peek(selectedPlayerCard).Value;
    for (int i = 0; i < players.Count; i++) {
      if (i == 0)
        players[0].AskForACard(players, 0, stock, cardToAskFor);
      else
        players[i].AskForACard(players, i, stock);
      if (PullOutBooks(players[i])) {
        textBoxOnForm.Text += players[i].Name + " drew a new hand\r\n";
        int card = 1;
        while (card <= 5 && stock.Count > 0) {
          players[i].TakeCard(stock.Deal());
          card++;
        }
      }
    players[0].SortHand();
      if (stock.Count == 0) {
        textBoxOnForm.Text = "The stock is out of
cards. Game over!\r\n";
        return true;
      }
    }
    return false;
  }
  public bool PullOutBooks(Player player)
  {
    List<Card.Values> BooksPulled =
player.PullOutBooks();
    foreach (Card.Values value in BooksPulled)
      books.Add(value, player);
    if (player.CardCount == 0)
      return true;
    return false;
  }
  public string DescribeBooks() {
    string whoHasWhichBooks = "";
    foreach (Card.Values value in books.Keys)
      whoHasWhichBooks += books[value].Name + " has a book of "
        + Card.Plural(value) + "\r\n";
    return whoHasWhichBooks;
  }
  public string GetWinnerName() {
    Dictionary<string, int> winners = new Dictionary<string, int>();
    foreach (Card.Values value in books.Keys) {
      string name = books[value].Name;
      if (winners.ContainsKey(name))
        winners[name]++;
      else
        winners.Add(name, 1);
    }
```

O método Deal() é chamado quando o jogo começa. Ele embaralha o monte e então dá cinco cartas para cada jogador. Então ele monta qualquer livro que por acaso os jogadores já tenham.

Tão logo o jogador clique no botão pergunte por uma carta, o jogo chama AskForACard() com essa carta. Então ele chama o mesmo método para cada oponente.

Depois que o jogador ou um oponente pergunta por uma carta, o jogo monta qualquer livro que ele possa ter feito. Se o jogador ficar sem cartas, ele compra uma nova mão de cinco cartas do monte.

Depois que o turno termina, o jogo ordena a mão do jogador, para se certificar que as cartas apareçam em ordem no formulário. Então ele checa para ver se o jogo acabou. Se é o caso, PlayOneRound() retorna true.

PullOutBooks() olha as cartas de um jogador para ver se ele conseguiu quatro do mesmo valor. Se for o caso, elas são adicionadas ao dicionário de livros. E se não sobrar nenhuma carta depois disso, ela retorna true.

O formulário precisa mostrar uma lista de livros, então ele usa DescribeTheBooks() para transformar um dicionário de livros de um jogador em texto.

Depois que a última carta foi comprada, o jogo precisa determinar quem ganhou. E o que GetWinnerName() faz. E ele usa um dicionário chamado winners para fazer isso. O nome de cada jogador é uma chave no dicionário, e o valor de cada chave é o número de livros que aquele jogador conseguiu durante o jogo.

```
    int mostBooks = 0;
    foreach (string name in winners.Keys)
      if (winners[name] > mostBooks)
        mostBooks = winners[name];
    bool tie = false;
    string winnerList = "";
    foreach (string name in winners.Keys)
      if (winners[name] == mostBooks)
      {
        if (!String.IsNullOrEmpty(winnerList))
        {
          winnerList += " and ";
          tie = true;
        }
        winnerList += name;
      }
    winnerList += " with " + mostBooks + " books";
    if (tie)
      return "A tie between " + winnerList;
    else
      return winnerList;
  }
}
```

Em seguida o jogo itera pelo dicionário para determinar o número de livros que o jogador com a maior quantidade deles tem. Ele coloca esse valor numa variável chamada mostBooks.

Agora que nós sabemos qual jogador tem mais livros, o método pode criar uma string que indica o vencedor (ou vencedores).

Aqui estão os métodos da classe Game.

```
public Player(String name, Random random, TextBox textBoxOnForm) {
    this.name = name;
    this.random = random;
    this.textBoxOnForm = textBoxOnForm;
    this.cards = new Deck( new Card[] {} );
    textBoxOnForm.Text += name + " has just joined the game\r\n";
}
public Card.Values GetRandomValue() {
    Card randomCard = cards.Peek(random.Next(cards.Count));
    return randomCard.Value;
}
public Deck DoYouHaveAny(Card.Values value) {
    Deck cardsIHave = cards.PullOutValues(value);
    textBoxOnForm.Text += Name + " has " + cardsIHave.Count + " "
        + Card.Plural(value) + "\r\n";
  return cardsIHave;
}
public void AskForACard(List<Player> players, int myIndex, Deck stock) {
  Card.Values randomValue = GetRandomValue();
  AskForACard(players, myIndex, stock, randomValue);
}
public void AskForACard(List<Player> players, int myIndex,
    Deck stock, Card.Values value) {
  textBoxOnForm.Text += Name + " asks if anyone has a " + value + "\r\n";
  int totalCardsGiven = 0;
  for (int i = 0; i < players.Count; i++) {
    if (i != myIndex) {
        Player player = players[i];
        Deck CardsGiven = player.DoYouHaveAny(value);
        totalCardsGiven += CardsGiven.Count;
        while (CardsGiven.Count > 0)
          cards.Add(CardsGiven.Deal());
    }
  }
  if (totalCardsGiven == 0) {
    textBoxOnForm.Text += Name + " must draw from the stock.\r\n";
    cards.Add(stock.Deal());
  }
}
```

Aqui está o construtor da classe Player. Ele atribui valores para os campos privados e adiciona uma linha à caixa de texto de progresso dizendo que o jogador se juntou ao jogo.

O método GetRandomValue() usa Peek() para olhar uma carta aleatória na mão do jogador.

DoYouHaveAny() usa o método PullOutValues() para retirar e retornar todas as cartas que sejam equivalentes ao parâmetro.

Existem dois métodos AskForACard() sobrecarregados. Este é usado pelos oponentes - ele pega uma de suas cartas aleatoriamente e chama o outro método AskForACard().

Este método AskForACard() itera por todos os jogadores (exceto aquele que está perguntando), chamando o método DoYouHaveAny(), e adiciona qualquer carta que seja entregue na mão do jogador que chama.

Se nenhuma carta foi entregue, o jogador tem que comprar uma do monte usando o método Deal().

E ainda MAIS outros tipos de coleções...

`List` e `Dictionary` são duas das **coleções genéricas nativas** que são parte do Framework .NET. Listas e dicionários são bastante flexíveis - você pode acessar qualquer dos dados em qualquer ordem. Mas algumas vezes você precisa restringir a forma como um programa trabalha com os dados porque a coisa que você está representando dentro do seu programa funciona dessa forma no mundo real. Para situações como essas, você pode usar uma **Queue** (fila) ou **Stack** (pilha). Essas são as outras duas coleções genéricas que se parecem com listas, mas são especialmente boas para se certificar de que os dados sejam processados seguindo uma ordem.

Existem ainda mais outros tipos de coleções - mas estes são os que você provavelmente vai encontrar mais vezes.

Use uma fila quando o primeiro objeto que você armazenou será o primeiro a ser usado, como por exemplo:

- ★ Carros movendo-se por uma rua estreita de mão única;
- ★ Pessoas numa fila;
- ★ Clientes esperando para serem atendidos por um setor de suporte por telefone;
- ★ Qualquer outra coisa que precise ser tratada na base do primeiro-a-chegar, primeiro-a-atender.

Uma fila é primeiro-a-entrar, primeiro-a-sair, o que quer dizer que o primeiro objeto colocado nela é o primeiro que se retira para usar.

Use uma pilha quando você sempre quiser usar o objeto mais recente, como por exemplo:

- ★ Mobília colocada num caminhão de mudanças;
- ★ Uma pilha de livros onde se quer ler aquele que foi acrescentado mais recentemente;
- ★ Uma pirâmide humana onde os mais acima devem descer primeiro (imagine a bagunça se alguém embaixo sair antes!).

Uma pilha é primeiro-a-entrar, último-a-sair - o primeiro objeto que entra na pilha é o último que sai dela.

Coleções genéricas são uma parte importante do Framework .NET

Elas são realmente úteis - tanto que o IDE automaticamente adiciona o seguinte comando no topo de qualquer classe adicionada a um projeto:

```
using System.Collections.Generic;
```

Quase todo projeto grande inclui algum tipo de coleção genérica, porque seus programas precisam armazenar dados. E quando você estiver lidando com grupos de coisas similares no mundo real, elas quase sempre caem numa categoria que corresponde bastante bem a um dos tipos de coleções.

Uma fila é como uma lista que deixa que você coloque objetos no final e use os do começo. Uma pilha só deixa você acessar o último objeto colocado nela.

Você pode, no entanto, usar foreach para iterar pelos elementos de uma pilha ou fila, porque elas implementam IEnumerable!