IFRN

Tecnologia em Análise e Desenvolvimento de Sistemas

Mais sobre Classes em C#

Prof. Gilbert Azevedo

Conteúdo

- Métodos e Encapsulamento
- Métodos Acessadores
- Referência This
- Passagem de Parâmetros
- DateTime, TimeSpan
- Membros Estáticos
- Classes Estáticas
- Exercício: Agenda de Contatos

Métodos e Encapsulamento

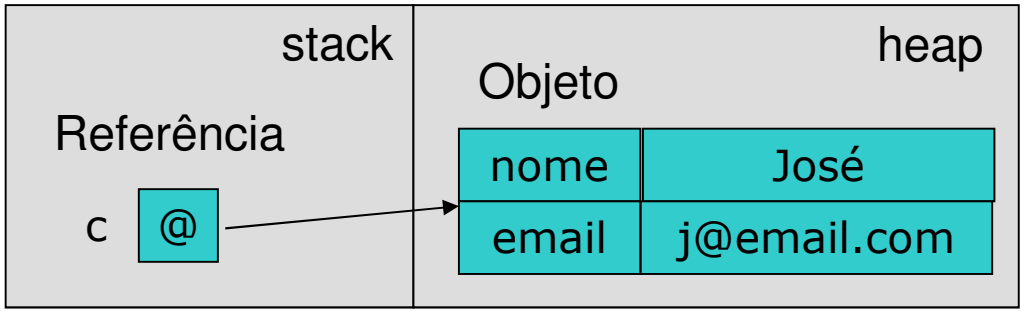
- São sequências nomeadas de instruções
- Os métodos podem receber dados (parâmetros), retornar dados (retorno) e ter um controle de acesso
  - [accessModifier] returnType MethodName (parameterList)
  - {
  - // Instruções
  - }
- Para chamar um método, utiliza-se a sintaxe
  - MethodName(argumentList)
- Encapsulamento
  - Atributos da classe, em geral, são "escondidos" do mundo exterior com um modificador *private*

Métodos Acessadores

- Métodos Acessadores são utilizados para definir e recuperar valores dos atributos de um objeto
  - class Contato
  - {
  - private string nome;
  - private string email;
  - public void SetNome(string aNome) { nome = aNome; }
  - public string GetNome() { return nome; }
  - public void SetEmail(string aEmail) { email = aEmail; }
  - public string GetEmail() { return email; }
  - }

Chamando Métodos Acessadores

- Invocando os métodos acessadores
  - class Program {
  - static void Main(string[] args) {
  - Contato c = new Contato();
  - c.SetNome("José");
  - c.SetEmail("j@email.com");
  - Console.WriteLine("{0} – {1}", c.GetNome(),
  - c.GetEmail());
  - Console.ReadKey();
  - }

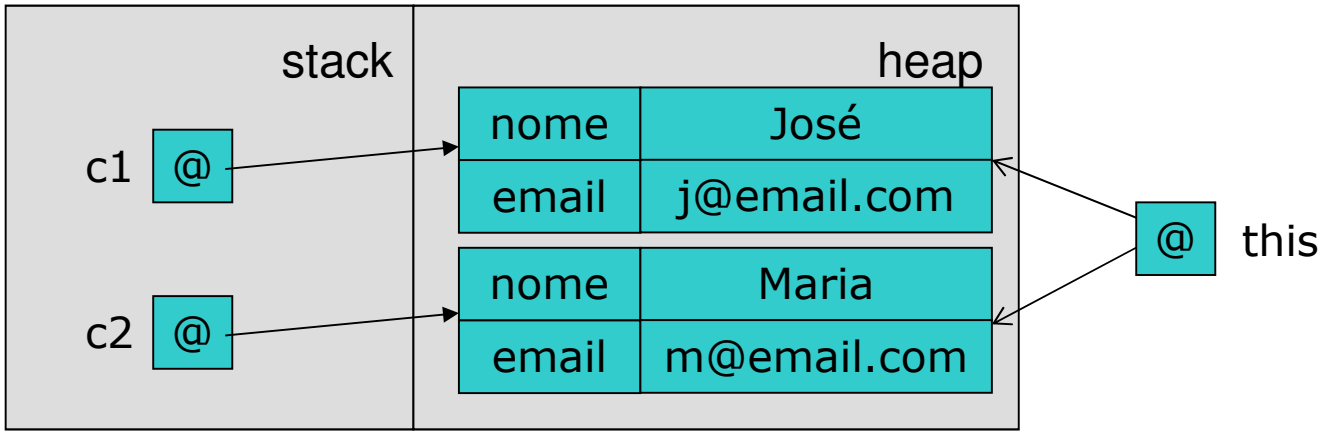


Referência This

- A palavra reservada this é uma referência para o objeto que invoca o método da classe.
  - class Contato {
  - private string nome;
  - private string email;
  - public void SetNome(string nome) { this.nome = nome; }
  - public string GetNome() { return this.nome; }
  - public void SetEmail(string email) { this.email = email; }
  - public string GetEmail() { return this.email; }
  - }
- Parâmetros de um método tem precedência em relação aos atributos da classe.

Referência This

```
class Program {
    static void Main(string[] args) {
        Contato c1 = new Contato();
        Contato c2 = new Contato();
        c1.SetNome("José");                    // this = c1
        c1.SetEmail("j@email.com");
        c2.SetNome("Maria");                   // this = c2
        c2.SetEmail("m@email.com");
}
```



Passagem de Parâmetros

- Os parâmetros de métodos no C# são classificados em três grupos:
  - Parâmetros por Valor
    - Forma padrão, nenhuma sinalização é necessária
  - Parâmetros por Referência
    - Sinalizados com o prefixo *ref*
  - Parâmetros de Saída
    - Sinalizados com o prefixo *out*

Passagem por Valor

- Passagem por valor (padrão)
  - public void Metodo1(int a, int b) {
  - int c = a;
  - a = b;
  - b = c;
  - }
- Ao invocar o método, os valores são copiados na memória
  - int x = 10, y = 20;
  - **Metodo1(x, y);**
  - Console.WriteLine(x); // Escreve 10
  - Console.WriteLine(y); // Escreve 20
- Alterar o parâmetro, não modifica o argumento

Passagem por Referência

- Passagem por referência (sinalizador por ref)
  - public void Metodo2(**ref** int a, **ref** int b) {
  - int c = a;
  - a = b;
  - b = c;
  - }
- Ao invocar o método, uma referência para a variável é criada
  - int x = 10, y = 20;
  - **Metodo2(ref x, ref y);**
  - Console.WriteLine(x); // Escreve 20
  - Console.WriteLine(y); // Escreve 10
- Alterar o parâmetro, modifica o argumento

Parâmetros de Saída

- Parâmetros de saída (sinalizados por out)
  - public void Metodo3(**out** int a, **out** int b) {
  - a = 10;
  - b = 20;
  - }
- Ao invocar o método, uma referência para a variável é criada, mas não precisa ser iniciada.
  - int x, y;
  - **Metodo3(out x, out y);**
  - Console.WriteLine(x); // Escreve 10
  - Console.WriteLine(y); // Escreve 20
- O valor do parâmetro deve ser setado no método e altera o argumento fornecido.

DateTime

- DateTime – Estrutura que representa um instante no tempo, expressando uma data e uma hora do dia
- Algumas Propriedades
  - Day, Month, Year – Retorna dia, mês ano da data/hora (int)
  - DayOfWeek – Dia da semana (DayOfWeek)
  - DayOfYear – Dia do ano (int)
  - Hour, Minute, Second – Retorna hora, minuto, segundo da data/hora (int)
  - Now – Retorna o dia e hora atual
  - Today – Retorna o dia atual (zero hora)

DateTime

- DateTime - Alguns Métodos
  - Add – Adiciona um período de tempo a data
  - AddDays, AddMonths, AddYears – Adiciona dias, meses e anos a uma data
  - AddHours, AddMinutes, AddSeconds – Adiciona hora, minutos e segundos a uma data
  - Parse – Converte uma string com informação de data e hora para seu equivalente de DateTime
  - ToString – Converte o DateTime em string
- DateTime dt = DateTime.Now;
- Console.WriteLine(dt.ToString());

TimeSpan

- TimeSpan – Estrutura que representa um intervalo de tempo, expresso em dia, hora, minuto, segundo, milissegundo
- Algumas Propriedades
  - Days, Hours, Minutes, Seconds, Milliseconds – Retorna dias, horas, minutos, segundos e milissegundos do intervalo (int)
- Exemplo
  - DateTime dt = new DateTime(2014, 10, 30);
  - TimeSpan ts = new TimeSpan(1, 9, 50, 0);
  - dt += ts;
  - Console.WriteLine(dt); // 31/10/2014 09:50:00

Membros e Classes Estáticas

- Campos de classe, definidos com a palavra reservada *static*, podem ser utilizados de forma que todos os objetos de uma classe compartilhem uma informação
- Métodos estáticos podem ser utilizados para prover funcionalidade em uma classe sem a necessidade de criação de um objeto
  - Os métodos de System.Math são estáticos
  - Métodos estáticos não acessam membros não estáticos
- Classes que possuam somente campos e métodos estáticos podem ser definidas como estáticas.
  - Não é possível criar nenhum objeto da classe

Exemplo de Membros Estáticos

- O atributo *count* armazena o nº de objetos instanciados
- O construtor incrementa o contador que é retornado por um método também estático
  - private static int count;
  - private bool estado;
  - public Lampada(bool aEstado) {
  - estado = aEstado;
  - count++;
  - }
  - public static int GetCount() {
  - return count;
  - }

Padrão de Projeto *Singleton*

- Garante a existência de apenas um objeto de uma classe
- Usa um construtor *private* acessado por um método *static*
- Retorna sempre o mesmo objeto da própria classe

```
class Singleton {
    private static Singleton instance;
    private Singleton() { }
    public static Singleton getInstance() {
        if (instance == null) instance = new Singleton();
        return instance;
    }
}
```

Exercícios

- Desenvolver uma agenda de contatos com as seguintes características:
- Classe Contato: atributos para nome, fone, email e data de nascimento.
- Classe Agenda: singleton, máximo de 50 contatos, métodos para adicionar, listar contatos, listar aniversariantes do mês.

Referencias Bibliográficas

- Introduction to C# Programming with Microsoft .Net
  - Microsoft Official Course 2609A
- Microsoft Visual C# 2005 - Passo a passo
  - John Sharp, Bookman, 2007
- Microsoft Asp.Net – Passo a passo
  - George Sheperd, Bookman, 2007
- Microsoft VS 2005 Express Edition Documentation
- POO – Programação Orientada a Objetos
  - Prof. Plácido Neto - IFRN