PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Instituto de Informática e Ciências Exatas - Curso de Sistemas de Informação
Laboratório de Programação Orientada por Objeto - Professor: Paulo Amaral

RELATÓRIO 1 (2/2015) - TRABALHO EM GRUPO

CLASSES E MÉTODOS

Introdução e ambientação ao C# (csc.exe) e Java (JDK).

1. ESPECIFICAÇÕES

Deve ser **entregue** um texto com o **relatório completo** do trabalho que conste na capa (1ª. página) os **nomes completos de todos os componentes do grupo**. O número de componentes do grupo será especificado pelo professor durante a aula, não sendo aceitos trabalhos anônimos.

No **relatório** deve constar tudo que foi especificado no exercício e os c**ódigos-fonte de todos os programas** envolvidos nos exercícios**.**

Todos programas devem **imprimir o nome completo de todos os componentes do grupo na tela** e os **arquivos fontes dos programas, além de serem incluídos no texto do relatório, devem ser entregues em arquivos separados no SGA**, devendo conter o **cabeçalho completo** com entrada, saída, descrição e nome completo de todos os componentes do grupo (não sendo aceitos trabalhos sem os nomes. **Cópias grosseiras** serão desconsideradas, ou seja, a **nota será igual a 0 (zero).**

A lista de exercícios tem partes que devem ser lidas (Leitura), outras (Prática) que devem ser exercitadas em casa e/ou nos laboratórios de informática e finalmente alguns **exercícios que devem ser entregues com o relatório.**

Haverá apresentação oral dos trabalhos

Obs: Baseado no material:
Dos professores João Caram, Hugo Bastos e Anna Tostes
Da Apostila de C# e Orientação a Objetos da K19 Treinamentos

2. INTRODUÇÃO

Em um computador as instruções e dados são armazenados na memória principal (RAM) durante o processamento. Cada posição de memória do computador possui um endereço numérico associado.

Basicamente, o que um programa faz é manipular dados. Em geral, esses dados são armazenadosem **posições da memória** localizadas na memória RAM do computador, as quais são referenciados por **nomes simbólicos** ao invés dos endereços numéricos correspondentes. Estes **nomes simból**icos são denominados de **variáveis** porque seus conteúdos podem variar durante a execução do programa.

Uma **variável** pode guardar dados de vários tipos: números, textos, booleanos (verdadeiro ou falso), referências de objetos, etc. Além disso, toda variável possui um nome que é utilizado quando a informação dentro da variável precisa ser manipulada pelo programa.

A cada **variável** está associado a um **tipo de dados**, o qual define como o dado é codificado, valores válidos e a quantidade de posições de memória necessárias para representar os dados que a variável armazena.

As linguagens de programação incluem alguns tipos de dados básicos, conhecidos como **tipos primitivos** ou **fundamentais** por serem suportados diretamente pelo compilador, sendo utilizados para a definição de variáveis, parâmetros, declarações, etc.

Os tipos de dados primitivos ou simples são grupos de valores indivisíveis como int, double, bool, char, etc.

Um **Tipo Abstrato de Dado** (TAD) é uma coleção bem definida de dados a serem armazenados e um grupo de operadores que podem ser aplicados para manipulação desses dados." (FERRARI, R.).

Um TAD também pode ser visto como um modelo matemático, acompanhado de operações definidas sobre o modelo.

Outras definições de TAD podem ser citadas:

É um conjunto de funções operando sobre uma estrutura de dados, sendo que a organização física da estrutura é encapsulada.

Figura 1 - Tipo Abstrato de Dado (TAD)

Exemplo: TAD Conta Bancária

Operadores, funções ou métodos:

- CriarConta
- Depositar
- Sacar
- MostrarSaldo

Tipos de dados usados como parâmetros das funções:

- numConta: número da Conta Bancária
- nomeTitular: nome do titular da conta
- SaldoConta

A **Programação Orientada para Objetos** (OOP) é um modo de abordar a tarefa de programação, onde os programas são organizados como coleções de objetos cooperativos que podem se comunicar.

Na Programação Orientada por Objetos uma **Classe implementa um TAD**.

Uma **Classe** é um agrupamento de objetos e consiste nos métodos e nos dados que um determinado objeto irá possuir. Ela contém as informações para criar, manipular e destruir o objeto. Deste modo, uma **classe** contém os **elementos** que representam o tipo e agrupa todas as **operações** realizadas sobre os elementos.

Uma classe é constituída por **atributos** (características) e **métodos** (comportamento).

Os **atributos** são semelhantes a campos de estruturas e os **métodos** são funções que operam sobre os atributos.

Exemplo: Classe Conta Bancária

Métodos:

CriarConta
Depositar
Sacar
MostrarSaldo

Atributos:

numConta: número da Conta Bancária nomeTitular:
nome do titular da conta
SaldoConta

Pode-se representar uma classe através de diagramas **UML**. O diagrama UML de uma classe é composto pelo nome da mesma e pelos atributos e métodos que ela define. Todos os objetos criados a partir da classe Conta terão os atributos e métodos mostrados no diagrama UML. Os valores dos atributos de dois objetos criados a partir da classe Conta podem ser diferentes.

Figura 2 - Diagrama UML da classe Conta.

Uma classe não é utilizada diretamente, sendo utilizada somente para criar objetos baseados nela.

Os Objetos são entidades (ou itens) unicamente identificados e representam uma instância concreta e dinâmica de uma classe.

Um **Objeto** é uma entidade lógica que contém dados e o código para manipular esses dados. **Objetos** são "**instanciados**" ou criados de uma classe.

Um objeto é uma instância (exemplar) concreta e dinâmica de uma classe, sendo criado quando uma mensagem solicitando a criação é recebida pela sua classe.

Os Objetos é que são utilizados nos programas.

Os **atributos** podem ser de **Classe** ou do **Objeto**.

Os **atributos de Classe** são **compartilhados** entre todos os objetos criados e são armazenados na **classe.** Exemplo: Um contador do número de objetos instanciados da classe.

Os **atributos do Objeto** (ou atributos de dados) são individuais e são armazenados no objeto. Exemplos: Nome, endereço do correntista.

Antes de um objeto ser criado, deve-se definir quais serão os seus atributos e métodos. Essa definição é realizada através de uma **classe** elaborada por um programador. A partir de uma classe, pode-se construir objetos na memória do computador que executa a aplicação.

3. Introdução ao C#

3.1 Microsoft .NET

Microsoft .NET (.NET Framework) é uma plataforma para desenvolvimento e execução de sistemas e aplicações. Todo e qualquer código gerado para **.NET**, pode ser executado em qualquer dispositivo que possua um framework de tal plataforma. Deste modo, pode ser considerada um ambiente para criação, instalação e execução de serviços e outros aplicativos com a finalidade de prover portabilidade de produtos desenvolvidos entre diversas plataformas, baseada em padrões de mercado como XML e SOAP.

Esta tecnologia possui diversas características em comum com a plataforma Java, onde o programador deixa de escrever código para um sistema ou dispositivo específico. Para obter portabilidade, usa a **máquina virtual CLI (COMMOM LANGUAGE INFRASTRUCTURE)** que contém: carregador de classes, **compilador JIT (Just-in-time**) que compila o código intermediário para o código nativo da CPU e um ambiente para coleta de lixo, que gerencia o uso da memória do computador. A CLI possui a mesma função que a JVM (Máquina Virtual Java) da tecnologia Java.

A **plataforma .NET** é executada sobre a **Common Language Runtime - CLR** (Ambiente de Execução Independente de Linguagem) interagindo com um Conjunto de Bibliotecas Unificadas (framework). Esta CLR é capaz de executar mais de vinte diferentes linguagens de programação, interagindo entre si como se fossem uma única linguagem, entre elas, estão C++, C#, Visual Basic, Delphi, Java, Fortran, etc. Além disto, permite a execução, construção e desenvolvimento de Web Services de forma integrada e unificada.

Um programa pode ser escrito em qualquer das linguagens de programação disponíveis para a plataforma e o código fonte gerado pelo programador é então compilado pela linguagem escolhida gerando um código intermediário em uma linguagem chamada **MSIL (Microsoft Intermediate Language)**. O novo código fonte gera um arquivo na linguagem de baixo nível, de acordo com o tipo de projeto:

- EXE - Arquivos Executáveis, Programas
- DLL - Biblioteca de Funções
- ASPX - Página Web
- ASMX - Web Service

No momento da execução do programa ele é novamente compilado, desta vez pelo compilador JIT, de acordo com a utilização do programa. Esta compilação é feita somente na primeira vez que o programa é utilizado, nas outras vezes que outro programa é executado, a compilação anterior é aproveitada. Através de ferramentas específicas é possivel "pré-compilar" o código para que não se tenha o custo da compilação JIT durante a execução.

O **FrameWork .Net** pode ser dividido em duas partes principais conforme exemplificado a seguir (Figura 3): a CLR (COMMON LANGUAGE RUNTIME), que pode ser definido como uma implementação comercial da CLI desenvolvida pela Microsoft e as Bibliotecas de Classe.

Estas duas partes são responsáveis por disponibilizar aos desenvolvedores .Net uma ampla variedade de funções abrangendo funcionalidades como sistemas de arquivo, acesso à banco de dados e acesso a redes em uma única estrutura logicamente organizada.

Um aplicativo desenvolvido em .Net, é formado por um conjunto de arquivos denominados Assembly (Figura 4), sendo que em sua parte principal, está escrita no Microsoft Intermediate language (MSIL).

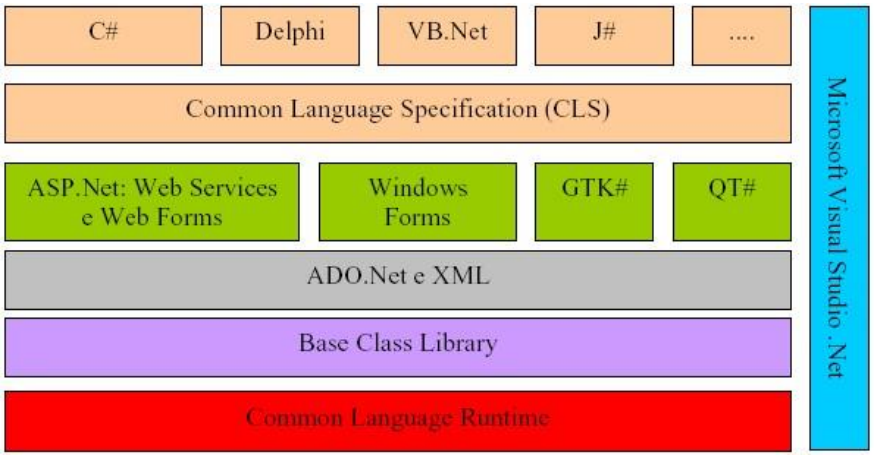


Figura 3: Estrutura do FrameWork .Net

No momento da execução de aplicativo .Net, o CLR traduz o código escrito no MISL em código nativo, o que torna possível à gerência de um aplicativo .Net pela CLR, dando origem ao termo código gerenciável.

Além do MSIL um Assembly também contém metadados, que descrevem em detalhes os tipos e componentes relevantes para que um código MSIL contido em um Assembly seja executado corretamente. Por último um Assembly inclui um manifesto, que se constitui em um documento que lista todos os arquivos e componentes de software requeridos por um Assembly.

A figura 5 mostra sinteticamente os passos executados pela CLR para a execução de um aplicativo .Net.

Finalmente a função considerada mais importante executada pela CLR é fazer a mediação entre o código gerenciado e o código não-gerenciado. Por código não gerenciado entende-se a parte do código que é executado fora do ambiente da CLR, que pode causar falha no aplicativo, vazamento de memória, e abrir brechas de segurança como buffer overflow.

As bibliotecas de classe do .Net Framework fornecem o código básico exigido por todos os aplicativos desenvolvidos em .net, permitindo que os desenvolvedores não se preocupem na escrita de códigos repetidos para funções corriqueiras como a leitura e escrita em arquivos.

Figura 4 - Estrutura de Um Assembly

Todo o código gerenciado na plataforma .Net é organizado em **grupos lógicos denominados classes**. Estas classes são agrupadas em hierarquias chamadas ***namespace***. As bibliotecas contidas no .Net framework estão contidas no namespace "System". Na figura 6 pode-se observar a organização de classes na tecnologia .Net.

Baseado nas informações apresentadas acima pode-se concluir que a principal vantagem da utilização de bibliotecas de classe na tecnologia .Net é o agrupamento das funções comumente usadas, descritas e agrupadas de uma forma lógica, facilitando a localização pelo desenvolvedor das funções necessárias para a conclusão de um determinado código.

A ECMA (European Computer Manufacturers Association), é uma associação industrial fundada em 1961 e dedica-se a padronização de sistemas de informação e comunicação. Alguns dos membros desta associação são a Microsoft, IBM, Sun, Intel e a Siemens.

Com o intuito de tornar o Framework .Net um padrão para o desenvolvimento de aplicativos, a Microsoft submeteu em outubro de 2000 para a ECMA as especificações da parte principal do .Net FrameWork (ECMA-335) e da linguagem C# (ECMA-334). Em dezembro de 2001, estes padrões foram aprovados e publicados tornando-se padrões internacionais.

O principal resultado obtido com estas publicações foi divulgar a tecnologia utilizada no .Net Framework possibilitando que qualquer pessoa ou empresa possa portar .Net Framework para uma plataforma desejada.

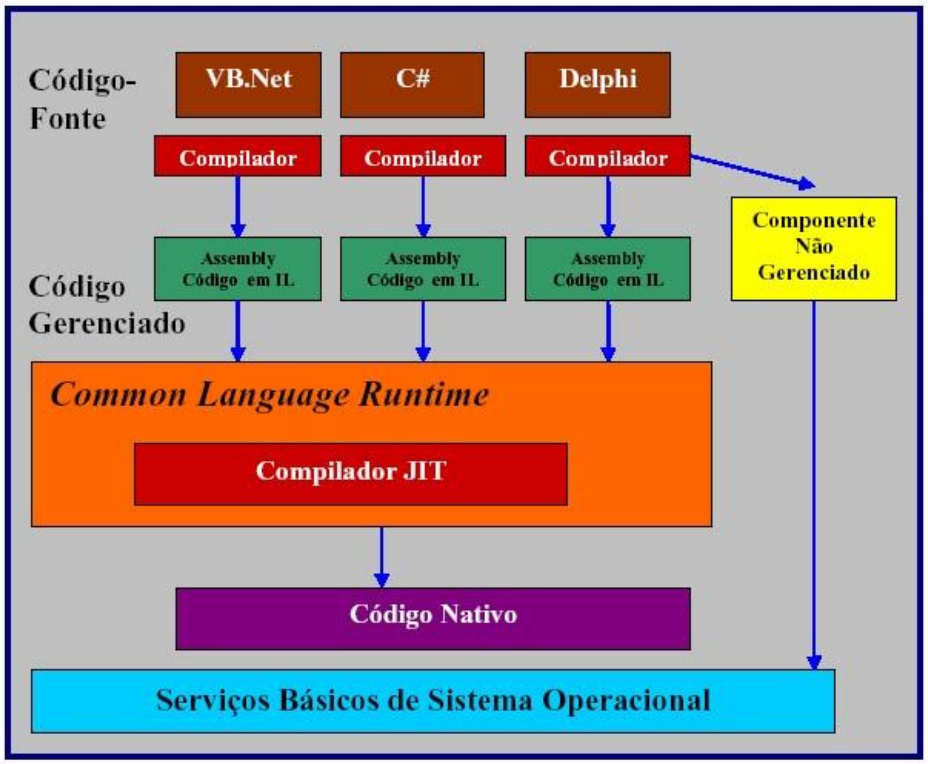


Figura 5 - Modelo de Execução da CLR

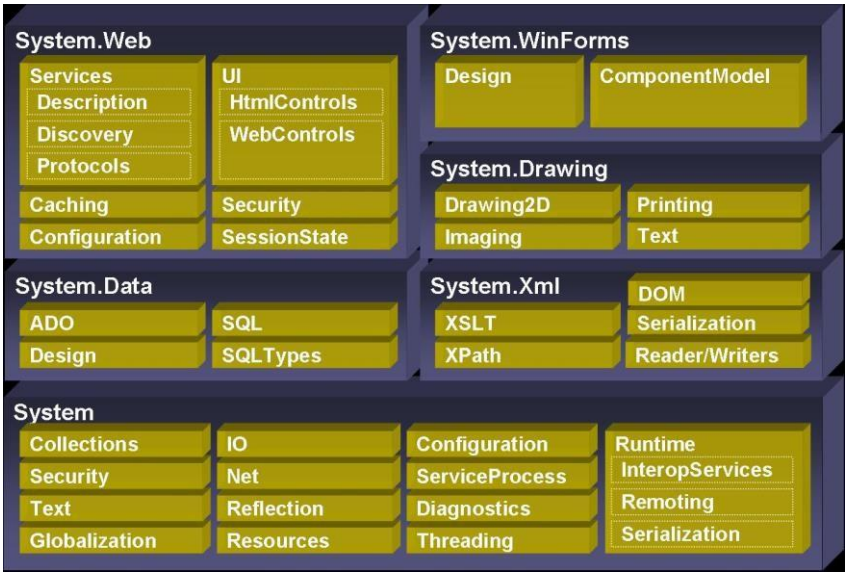


Figura 6 - NameSpace do .Net Framework

3.2 Introdução à linguagem C# (leitura)

C# é um linguagem de programação orientada a objetos desenvolvida pela Microsoft como parte da **plataforma .NET**, escalável para permitir que uma aplicação possa ser executada em plataformas (hardware/software) distintas como servidores, desktops, dispositivos móveis, etc.

Segue os padrões padrão ISO/IEC (*International Organization for Standardization/* International Engineering Consortium*)* e ECMA (*European Computer Manufactures Association)*.

Sua sintaxe orientada a objetos foi baseada no C++, mas foi fortemente influenciada por outras linguagens de programação como Object Pascal e Java.

As aplicações desenvolvidas em C# são baseadas em arquivos (com extensão .cs) contendo o código-fonte dos programas. Quando se compila um código em C#, é criado um *assembly*. Um *assembly* é uma coleção de arquivos que aparecem ao programador como uma única DLL, biblioteca de *link* dinâmico (*Dynamic Link Library*), ou executável (EXE). Em .NET, um *assembly* é a unidade básica de reutilização, versionamento, segurança e implantação. O CLR proporciona um número de classes para manipulação de *assemblies*.

A Framework classlibrary (FCL) do C# é uma biblioteca de tipos de classes, interfaces e valor que fornecem acesso à funcionalidade do sistema. É a base sobre a qual aplicações Framework, componentes e controles são construídos.

Todo o código gerenciado na plataforma .Net é organizado em grupos lógicos denominados classes.

Estas classes são agrupadas em hierarquias chamadas "namespace". As bibliotecas contidas no .Netframework estão contidas no namespace"System".

Na linguagem de programação C#, as variáveis devem ser declaradas para que possam ser utilizadas. A declaração de uma variável envolve definir um nome único (identificador) dentro de um escopo e um tipo de valor. As variáveis são acessadas pelos nomes e armazenam valores compatíveis com o seu tipo.

Em C# todo o tipo de dados possui um correspondente na CLR (Common Language Runtime), por exemplo: int em C# refere-se a System.Int32 na plataforma .NET.

Os **tipos de dados no C#** são divididos em 3 categorias:
• Tipos **valor (value types**): armazenam dados em memória.
• Tipos **referência (reference types**): armazenam uma referência, ou o endereço, para o valor atual.
• Tipos **ponteiro (pointer types)**: apenas apontam para um endereço de memória.

Tipos primitivos em C#

Tipo C#	Descrição	Faixa de valores
bool	Booleano	true ou false
byte	Inteiro de 8-bit com sinal	-127 a 128
Sbyte	Inteiro de 8-bit sem sinal	0 a 255
char	Caracter Unicode	16-bit U+0000 a U+ffff
int	Inteiro de 32-bit com sinal	-2.147.483.648 a 2.147.483.647
Uint	Inteiro de 32-bit sem sinal	0 a 4,294,967,295
long	Inteiro de 64-bit com sinal	–9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
Ulong	Inteiro de 64-bit sem sinal	0 a 18,446,744,073,709,551,615
Short	Inteiro de 16-bit com sinal	-32,768 a 32,767
Ushort	16-bit sem sinal	0 a 65,535
float	Ponto flutuante IEEE 32-bit	±1,5 × 10E-45 a ±3,4 × 10E38 (7 dígitos significativos)
double	Ponto flutuante IEEE 64-bit	±5,0 × 10E-324 a ±1,7 × 10E308 (15-16 dígitos significativos)
decimal	Ponto flutuante 128-bits	±1,0 × 10E-28 a ±7,9 × 1028 (28-29 dígitos)

O tipo **Array** em C# é uma coleção de elementos armazenados em sequência, acessíveis através de um índice numérico. Pode-se criar arrays com uma ou mais dimensões. Em C# o primeiro elemento de um array é o de índice zero (0).

Exemplos em C#: int [ ]
numeros = new int [100];
int [ ] numeros = {1, 2, 3, 4, 5};

int [,] numeros = new int [3, 2] {{1, 2}, {3, 4}, {5, 6} };
string [,] nomes = new string [2, 2] {{“Mara”, “Lotar”}, {“Mary”, “José”} };

Nenhum tipo primitivo da linguagem C# permite o armazenamento de texto. O tipo primitivo **char** armazena apenas um caractere. Quando é necessário armazenar um texto, deve-se utilizar o tipo **string**. Contudo, é importante salientar que o tipo string **não** é um tipo primitivo.

O tipo **struct (**estrutura), em C#, é um tipo de valor usado para encapsular pequenos grupos de variáveis relacionadas, como as coordenadas de um retângulo ou as características de um item em um inventário.

Structs compartilham grande parte da sintaxe de classes, embora sejam mais limitados do que as classes.

O exemplo a seguir mostra uma declaração simples de estrutura:

```
public struct Book{     public decimal price;  public string title;
        public string author;
}
```

Structs são tipos de valor e classes são tipos de referência. Diferentemente das classes, **structs** pode ser instanciada sem usar o operador new.

Uma **estrutura não pode herda**r uma outra estrutura ou classe, e também não pode ser base de uma classe.

Métodos são as ações que determinam o comportamento do objeto e compõem o conjunto de serviços fornecidos pelo objeto. Geralmente manipulam e alteram os atributos do objeto.

Métodos implementam uma ação executada por um objeto de uma classe e geralmente são estruturas que agrupam um conjunto de comandos.

Em POO as interações entre objetos são representadas por mensagens que são enviadas aos objetos.

Uma mensagem pode ser gerada pelo usuário, por exemplo, ao clicar o mouse.

Pode-se dividir um método em quatro partes:

Nome: É utilizado para chamar o método. Na linguagem C#, é uma boa prática definir os nomes dos métodos utilizando a convenção "Camel Case" com a primeira letra maiúscula.

Lista de Parâmetros: Define os valores que o método deve receber. Métodos que não devem receber nenhum valor possuem a lista de parâmetros vazia.

Corpo: Define o que acontecerá quando o método for chamado.

Retorno: A resposta que será devolvida ao final do processamento do método. Quando um método não devolve nenhuma resposta, ele deve ser marcado coma palavra reservada **void**.

Métodos, em C#, são empregados para implementar procedimentos e funções. As **funções** retornam um valor ao final da execução.

Os **parámetros** tornam o uso de funções e procedimentos mais genéricos, onde um mesmo código pode ser aplicado à vários dados.

Um parâmetro "*avisa*" a um procedimento ou função sobre quais dados ele deve operar, permitindo maior flexibilidade e o reaproveitamento de código.

Parâmetros podem ser passados em C# **por valor** e **por referencia.**

Quando os parâmetros são passados **por valor**, o *valor* da *variável* é **copiado** para uma **variável local.** A variável local é descartada após o fim do método. Exemplo:

```
int somar(int a, int b){
        a = a+b;
        return a;
}
int a, b;
a = Int.Parse(Console.Readline());
b = Int.Parse(Console.Readline());
int c = somar(a,b);
```

Quando os parâmetros são passados **por referência,** o programa cria um **apontador** para o local da variável original. Este **apontador** funciona como um atalho (ou uma "indicação") para o dado que deve ser utilizado pelo código. Na Passagem de parâmetros por referência pode causar a alteração do valor da variável ou estrutura original. Exemplo:

```
int somar(ref int a, ref int b){
        a = a+b;
        return a;
}

int a, b;
a = Int.Parse(Console.Readline());
b = Int.Parse(Console.Readline());
int c = somar(ref a, ref b);
```

Existe também a passagem de **parâmetro de saída:** ***out*.**

Este modo de passagem de parâmetro é semelhante a passagem por referência reference, exceto que o valor inicial do argumento que é passado pelo chamador não é importante.

Um parâmetro é declarado com o modificador **out**
O exemplo a seguir mostra o uso do **parâmetro de saída:** ***out.***

```
using System;
  class Test {
    static void Divide(int x, int y, out int result, out int remainder) {
          result = x / y; remainder = x % y;
    }


    static void Main() {
          int res, rem;
          Divide(10, 3, out res, out rem); Console.WriteLine("{0} {1}", res, rem);
         // Outputs "3 1"
    }
 }
```

Em C#, algumas estruturas são **sempre** passadas por referência. Exemplo: vetores, matrizes e objetos.

3.3 Compilação pela linha de comando com csc.exe

Pode-se invocar o compilador C# digitando o nome do arquivo executável (csc.exe) em um prompt de comando.

Para usar uma janela de prompt de comando padrão, deve-se ajustar o caminho para que você possa chamar csc.exe de qualquer subdiretório do computador.

O arquivo executável de csc.exe está localizado em geral em Microsoft.NET\framework\pasta de*Version* no diretório do Windows. O local pode variar dependendo da configuração precisa de um computador específico. Se mais de uma versão do.NET Framework é instalado no computador, você encontrará várias versões desse arquivo.

Para compilar C# na linha de comando é conveniente escrever um script **cslc** (.bat ou .ps1) com os seguintes comandos (Ex: usando cslc.bat).

C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe %1 %2 %3 %4

Exemplo de programa em C# (arquivo fonte **disponível no SGA**)

A estrutura um programa em C# é apresentada a seguir no exemplo a seguir:

```
//
// nome do programa: Programa prova0.cs (revisao - ambientacao para novos alunos)
//
// programadores: nome dos alunos (fulano, sicrano e beltrano)
// data: 26/01/2015
// descricao: imprime o tamanho de alguns tipos primitivos, usa metodos de E/S no console
// para obter dados dos alunos, chama métodos para mostrar a passagem por valor, por referencia e
//  o valor retornado.
// entrada: numero de matricula (6 dígitos)e nome do aluno
// saida: exibe na tela os valores intermediarios e finais dos parametros passados para
// os métodos e retornados por eles.
//
using System;
using System.Collections.Generic; using
System.Linq;
using System.Text;

namespace  Prova0{
   class Program  {
     static void Main (string[ ] args) {
          int A=0, B=0, C=0, i,j, k, l, m, n, MAT;
        int[ ] vetor = new int[10];
        double r = 0;
        string NOME1, NOME2;
       k = sizeof(float);
```

```
    l = sizeof(double);
    m = sizeof(int);
    n = sizeof(long);
    Console.WriteLine("Tamanho dos tipos primitivos: ");
    Console.Write("float = " + k + " bytes double = " + l);
    Console.WriteLine(" bytes int = " + m + " bytes long = " + n + " bytes");
    Console.Write ("Digite seu numero de matricula: ");
    NMAT = int.Parse (Console.ReadLine());
    j = NMAT;
    for (i = 0; j > 0; i++){
         vetor[i] = j % 10 ; j = j / 10;
     }
    Console.Write("Digite seu primeiro nome: ");
    NOME1 = Console.ReadLine();;
    Console.Write("Digite seu ultimo sobrenome: ");
    NOME2 = Console.ReadLine();
    A = vetor[1];  B = vetor[2]; C = vetor[3];
    Console.WriteLine("NOME1 = " + NOME1 + " NOME2 = " +  NOME2);
    Console.WriteLine("A = " + A + " B = " + B + " C = " + C );
    Faz1(A, B, C); // primeira chamada por valor
    Console.Write("A = " + A + " B = " + B + " C = " + C );
    Console.WriteLine(" r = " + r);
     r = Faz2(ref A, ref B, ref C); //segunda chamada por referencia
    Console.Write("A = " + A + " B = " + B + " C = " + C );
    Console.WriteLine(" r = " + r);
    Console.WriteLine("Digite enter para terminar");
    Console.Read();
 }

public static void Faz1(int x, int y,  int z) {
     Console.WriteLine("x = " + x + " y = " + y + " z = " + z);
     x = x + 10; y = y + 20; z = z + 30;
     z = x + y + z;
     Console.WriteLine("x = " + x + " y = " + y + " z = " + z);
}

public static  Double Faz2(ref int x, ref int y, ref int z){
    double w = 0;
    Console.WriteLine("x = " + x + " y = " + y + " z = " + z);
     x = x + 10; y = y + 20; z = z + 30;
     z = x + y + z;
     Console.WriteLine("x = " + x + " y = " + y + " z = " + z);
     w = x + y + z;
     Console.Write("x = " + x + " y = " + y + " z = " + z);
     Console.WriteLine(" w = " + w);
     return (w);
}
```

```
  }

}
```

Para compilar: cslc prova0.cs

Para executar: prova0.exe

A diretiva ***using*** especifica qual biblioteca o programa pode utilizar, como por exemplo, a diretiva *using System* especifica que o programa pode utilizar a biblioteca no namespace *System*, ou seja, evitam a necessidade de especificar o nome para cada classe

A palavra-chave de ***namespace*** é usada para declarar um escopo que contém um conjunto de objetos relacionados, sendo usados para organizar códigos de grandes projetos;

Há sempre um *namespace* global que faz referência ao *namespace System* do *.Netframework*

Uma declaração de namespace contém as bibliotecas de classe que podem ser usados no programa. Uma classe: contém as declarações de dados e métodos para uma aplicação.

O método **Main** é o ponto de entrada de um aplicativo de console C# ou de um aplicativo Windows. (Bibliotecas e serviços não requerem um método Main como um ponto de entrada.). Quando o aplicativo é iniciado, o método Main é o primeiro método invocado.

Pode haver apenas um ponto de entrada em um programa C#. Caso exista mais que uma classe que possua um método **Main**, deve-se compilar o programa com a opção **/main** do compilador, para especificar qual método **Main** usar como ponto de entrada.

A classe ***Console,*** do *namespace System*, fornece suporte básico para leitura e escrita de caracteres. Os principais métodos usados para essas funcionalidades são:

- *Write*: escreve uma informação específica em tela.
- *WriteLine*: escreve uma informação específica em tela e pula uma linha em seguida.
- *ReadLine*: captura uma linha de texto digitada pelo usuário e retorna seu valor na forma de *string*.
- *ReadKey*: captura um único caracter digitado pelo usuário. O retorno desse método é um objeto da classe *ConsoleKeyInfo*, que contém informações sobre a tecla digitada pelo usuário.
- *Clear*: limpa toda a tela do console. Útil para programas que exigem formatação dos resultados.

Ex 3.1 (PROVA 1 POO 2014): (para entregar)

Considere o programa **prova0.cs em C#** (arquivo fonte **disponível no SGA**) e a sequência de comandos executados no Prompt do Windows:

csc.exe prova0.cs
/prova0.exe

Observe que o programa solicita o número de matrícula e o nome parcial do aluno. Utilize no exercício somente os dados de um aluno do grupo.

Escreva o que será impresso pelo programa na tela, os **valores** assumidos por **todas as variáveis** (nos comandos nos quais elas são referenciadas) e todos os **cálculos** necessários (somente a resposta não será considerada).

a) TELA:

b) Valores assumidos pelas variáveis e cálculos (continuar no verso, caso necessário)

3.4 Argumentos de linha de comando em C#

Como já foi dito antes, o método **Main** é o ponto de entrada de um programa .exe; é onde o controle de programa inicia e termina.

Main é declarado dentro de uma classe ou estrutura, **deve ser static** e não deve ser público.

Main pode conter um tipo de retorno **void** ou **int**.

O método **Main** pode ser declarado com ou sem um parâmetro string[] que contém os argumentos da linha de comando.

Parâmetros são lidos como argumentos de linha de comando zero-indexados. Ao contrário do C e C++, o nome do programa de não é tratado como o primeiro argumento da linha de comando.

É possível enviar argumentos para o método **Main** definindo o método em uma das seguintes maneiras:

static int Main(string[ ] args) ou static void Main(string[ ] args)

O parâmetro do método **Main** é uma matriz(vetor) String que representa os argumentos de linha de comando.

Argumentos de linha de comando permitem ao usuário modificar a operação de uma aplicação a partir de sua execução. O usuário insere os argumentos na linha de comando no momento da execução da aplicação. Deve-se lembrar que os argumentos de linha de comando são especificados depois do nome da classe a ser e partir de sua execução. Estes argumentos são armazenados no vetor (array) de Strings. Cada elemento do array irá conter um dos argumentos passados. Os argumentos serão armazenados no array args.

Normalmente, pode-se determinar se os argumentos existem testando a propriedade de Length, por exemplo:

```
if (args.Length == 0){
   System.Console.WriteLine("Please enter a numeric argument.");
return 1; }
```

Pode-se converter os argumentos de cadeia de caracteres para tipos numéricos usando a classe **Convert** ou o método **Parse**. Por exemplo, a seguinte declaração converte string em um número long usando o método **Parse**: long num = Int64.Parse(args[0]);

Também é possível usar o tipo C# long, do qual os aliases Int64: long num = long.Parse(args[0]);

Pode-se usar o método ToInt64 de classe Convert para fazer a mesma coisa:

```
long num = Convert.ToInt64(s);
```

O exemplo a seguir mostra o programa arglc115.cs (arquivo fonte **disponível no SGA**) que mostra os argumentos de linha de comando em um aplicativo de console.

```
//
// nome do programa: arglc115.cs
//
// programadores: nome dos alunos (fulano, sicrano e beltrano)
// data: 26/01/2015
```

```
// descricao: exemplo de passagem de parametros na linha de comando (lc) //
entrada(s: parametros na linha de comandos (lc), sendo:
// o numero de matricula (6 dígitos) e o nome completo do aluno
// saida(s): exibe na tela os parametros na linha de comandom (lc)
// para executar  e testar digite:
// arglc115.exe 231650 joao manuel fragoso
// descricao de parametros opcionais na lc (variaveis pre-definidas do command prompt ou shell)
// args[0]:  primeiro parametro posicional na lc
// args[i]: i-esimo parametro posicionais na lc
//

using System;
using System.Text;

namespace         Arglc{
class Program {
   static void Main(string[] args){
          int i,j;             j
= args.Length;
 Console.WriteLine("\nVoce executou o comando arglc115.exe com " + j + "
parametros : ");
          for (i= 0; i < j ; i++)   Console.WriteLine("\nargs[" + i + "] = "  + args[i]);
                                  // ou System.Console.WriteLine("arg[{0}] = [{1}]", i, args[i]);
          Console.WriteLine("\n");

    }
  }
}
```

Para compilar: cslc arglc115.cs
Para executar: arglc115.cs.exe 231650 joao manuel fragoso

Ex 3.2 (para entregar):

Escreva um programa em C# que receba n números inteiros através da linha de comando, armazene em um vetor e que chame um método que receba estes números inteiros (armazenados no vetor) e retorne: o menor, o maior deles e a média de seus valores.

Ex3.3 (para entregar):

Escreva um programa em C# que receba através da linha de comando uma data composta por dia, mês e ano e chame um método "quantosDias" que receba como parâmetro esta data composta por dia, mês e ano e retorne o número de dias decorridos no ano até a data fornecida.

Ex3.4 (para entregar):

Escreva um programa em C# que leia os votos de 15 eleitores para presidente. Os candidatos são:

19 – Machado de Assis
21 – Guimarães Rosa

Ao final da votação, o algoritmo deve mostrar o resultado final em número de votos e em porcentagem.

Ex3.5 (para entregar):

Escreva um programa em C# que receba n números inteiros através da linha de comando, armazene em um vetor e que chame um método que receba estes números inteiros (armazenados no vetor) e ordene na ordem crescente de valores. Depois de acionar este método, deve-se exibir estes números ordenados na ordem crescente de valores.

Ex3.6 (Exercício para entregar)

Fazer um programa codificado em C#, usando obrigatoriamente o csc.exe, que implemente uma calculadora com as seguintes especificações:

- Deve efetuar as **quatro operações aritméticas básicas** (+, -, x e /) com números reais.
- Os operandos e o operador são **recebidos obrigatoriamente através de parâmetros passados na linha de comando (Args[ ])**.
- Deve ter obrigatoriamente uma classe que contenha **um método para cada operação** aritmética básica (+, -, x e /).
- Deve fazer o **tratamento de erro** nos casos de parâmetros inválidos ou inadequados.
- **Exemplo:**
  ENTRADA: ./calcs.exe 2.5 x 3
  SAÍDA: 2 x 3 = 7.5

Ex3.7 (Exercício para entregar)

Fazer um programa codificado em C#, usando obrigatoriamente o usando obrigatoriamente o csc.exe, que calcule o fatorial do número inteiro **recebido obrigatoriamente através de parâmetro passados na linha de comando (Args[ ])**. Deve ter obrigatoriamente uma classe que contenha **um método para calcular o fatorial.** Deve fazer o **tratamento de erro** nos casos de parâmetros inválidos, inadequados ou inexistente.

- **Exemplo:**
  ENTRADA: ./fat.exe 3 SAÍDA:
  3! = 6.

3.5 Classes em C#

O conceito de classe apresentado anteriormente é genérico e pode ser aplicado em diversas linguagens como o C#.

A **unidade de programação em C# é a classe**. Objetos também são instâncias dessas classes, e as funções são encapsuladas dentro dos “limites” das classes como métodos.

A classe Conta pode ser escrita utilizando a linguagem C#. Inicialmente, definem-se os seguintes atributos:

```
class Conta {
```

```
        public int numero;  // número da Conta Bancária
        public string nome; // nome do titular da conta public
        double Saldo;
}
```

A classe C# Conta é declarada utilizando a palavra reservada **class**. No corpo dessa classe, são declaradas três variáveis que são os atributos que os objetos possuirão. Como a linguagem C# é estaticamente tipada, os tipos dos atributos são definidos no código. O atributo saldo é do tipo double, que permite armazenar números com casas decimais, e o atributo número é do tipo int, que permite armazenar números inteiros. O modificador public é adicionado em cada atributo para que eles possam ser acessados a partir de qualquer ponto do código. Este e outros modificadores de visibilidade em outros relatórios.

Por convenção, os nomes das classes na linguagem C# devem seguir o padrão “pascal case” também conhecido como “upper camel case”.

Após definir a classe Conta, pode-se criar objetos a partir dela. Esses objetos devem ser alocados na memória RAM do computador. Felizmente, todo o processo de alocação do objeto na memória é gerenciado pela máquina virtual. O gerenciamento da memória é um dos recursos mais importantes oferecidos pela máquina virtual.

Do ponto de vista da aplicação, basta utilizar um comando especial para criar objetos e a máquina virtual se encarrega do resto. O comando para criar objetos é o **new**.

```
class TestaConta {
        static void Main () { new Conta ();  } }
```

Todo objeto em C# possui uma referência. A referência de um objeto é a única maneira de acessar seus atributos e métodos. Dessa forma, deve-se guardar as referências dos objetos que deseja-se utilizar.

A princípio, pode-se comparar a referência de um objeto com o endereço de memória desse objeto. De fato, essa comparação simplifica o aprendizado. Contudo, o conceito de referência é mais amplo. Uma referência é o elemento que permite que um determinado objeto seja acessado.

Uma referência está para um objeto assim como um controle remoto está para um aparelho de TV. Através do controle remoto de uma TV pode-se aumentar o volume ou trocar de canal.

Analogamente, pode-se controlar um objeto através da referência do mesmo.

Ao utilizar o comando new, um objeto é alocado em algum lugar da memória. Para acessar esse objeto, precisa-se de sua referência. O comando new devolve a referência do objeto que foi criado.

Para guardar as referências devolvidas pelo comando new, deve-se utilizar variáveis não primitivas.

```
Conta referencia = new Conta ();
```

A variável **referencia** receberá a referência do objeto criado pelo comando new. Essa variável é do tipo Conta. Isso significa que ela só pode armazenar referências de objetos do tipo Conta.

Pode-se alterar ou acessar os valores guardados nos atributos de um objeto com a referência a esse objeto. Os atributos são acessados pelo nome. No caso específico da linguagem C#, a sintaxe para acessar um atributo utiliza o operador "**.**".

```
Conta referencia = new Conta ();

referencia.saldo = 1000.00; referencia.nome
= “Joao Manuel “; referencia.numero = 1;

System.Console.WriteLine( referecia.saldo );
System.Console.WriteLine (referecia.limite );
System.Console.WriteLine (referecia.numero );
```

Pode-se instanciar um objeto e utilizar seus atributos sem inicializá-los explicitamente, pois os atributos são inicializados com valores padrão. Os atributos de tipos numéricos são inicializados com 0, os atributos do tipo boolean são inicializados com false e os demais atributos com null (referência vazia).

A inicialização dos atributos com os valores padrão ocorre na instanciação, ou seja, quando o comando new é utilizado. Dessa forma, todo objeto “nasce” com os valores padrão. Em alguns casos, é necessário trocar esses valores. Para trocar o valor padrão de um atributo, deve-se inicializá-lo na declaração. Por exemplo, suponha que o limite padrão das contas de um banco seja R$ 500. Nesse caso, seria interessante definir esse valor como padrão para o atributo limite.

```
class Conta{
 public double limite = 500;
}
```

Em um banco é possível realizar diversas operações em uma conta: depósito, saque, transferência, consultas,etc. Essas operações podem modificar ou apenas acessar os valores dos atributos dos objetos que representam as contas.

Essas operações são realizadas em **métodos** definidos na própria classe Conta. Por exemplo, para realizar a operação de depósito, podemos acrescentar o seguinte método na classe Conta.

```
void Deposita ( double valor ){ implementação
}
```

Para realizar um depósito, deve-se chamar o método Deposita() através da referência do objeto que representa a conta que terá o dinheiro creditado.

```
Conta c = new Conta ();// Referência de um objeto
c. Deposita (1000) ; // Chamando o método Deposita ()
```

Normalmente, os métodos acessam ou alteram os valores armazenados nos atributos dos objetos.

Por exemplo, na execução do método Deposita(), é necessário alterar o valor do atributo saldo do objeto que foi escolhido para realizar a operação.

Dentro de um método, para acessar os atributos do objeto que está processando o método, pode-se utilizar a palavra reservada **this**.

```
void Deposita ( double valor ){ this . saldo +=
        valor ; // implementação
}
```

O método Deposita() não possui nenhumretorno lógico. Por isso, foi marcado com void. Mas, para outros métodos, pode ser necessário definir um tipo de retorno específico.

EX 3.8 (Exercício para entregar)

Fazer um programa, codificado em C# que implemente a classe Conta com as seguintes especificações:

- Apresenta inicialmente na tela um menu com as seguintes opções:
    1. Depositar na conta
    2. Fazer retirada na conta
    3. Imprimir saldo da conta
    4. Sair do programa
- Obter a opção do usuário
- Chamar o método correspondente, apresentar o resultado e sempre voltar ao menu inicial, exceto quando for selecionada a opção 4 (Sair do programa).
- Deve ter obrigatoriamente uma classe que contenha **um método para cada operação** na conta: depositar, fazer retirada e imprimir o saldo.

Um esboço da classe **Conta** é apresentado a seguir:

```
class Conta {
      // atributos
         public int agencia ;
         public int numero;  // número da Conta Bancária
         public string nome; // nome do titular da conta public
         double Saldo;
     // metodos
    public void Deposita ( double valor )
    {
       saldo += valor ;
    }
```

```
    public void Retira ( double valor )
    {
      // IMPLEMENTAR
    }

    public void ImprimeSaldo ()
    {
      // IMPLEMENTAR
    }
.....
}
```

Ex 3.9 (Exercício para entregar)

Fazer um programa, codificado em C#, para conversão de temperaturas com as seguintes especificações:

- Apresentar inicialmente na tela um menu com as seguintes opções:
    1. Converter de Celsius para Fahrenheit
    2. Converter de Fahrenheit para Celsius
    3. Sair do programa
- Obter a opção do usuário
- Chamar o método correspondente, apresentar o resultado e sempre voltar ao menu inicial, exceto quando for selecionada a opção 3 (Sair do programa).

- Deve ter obrigatoriamente uma classe que contenha pelo menos os dois métodos para converter a temperatura em Celsius para Fahrenheit e Fahrenheit para Celsius.

4. CLASSES E MÉTODOS EM JAVA

4.1 Introdução à linguagem Java

Java é tanto uma linguagem de programação quanto uma plataforma. A linguagem Java foi desenvolvida pela Sun Microsystems, em 1991.

Nas linguagens tradicionais, o processo de criação de um arquivo executável passa por duas etapas. Na primeira etapa, cada módulo do programa é compilado para um código objeto, escrito em linguagem de máquina dependente da arquitetura de hardware em que o programa será executado. Em seguida, a segunda etapa consiste no processo de ligação e edição (link-edition). Neste processo, os códigos objetos são ligados uns aos outros, produzindo um único arquivo executável. O programa é então executado diretamente no sistema operacional de destino.

A linguagem de programação Java é diferente, pois ela tanto compila quanto interpreta um programa. O compilador Java traduz o programa para uma linguagem de máquina intermediária chamada bytecode, o qual é interpretado pela **Máquina Virtual Java (JVM)** mostrada na figura 7. O

interpretador traduz para a arquitetura específica do computador. A compilação acontece apenas uma vez, enquanto que a interpretação é repetida cada vez que o programa é executado.

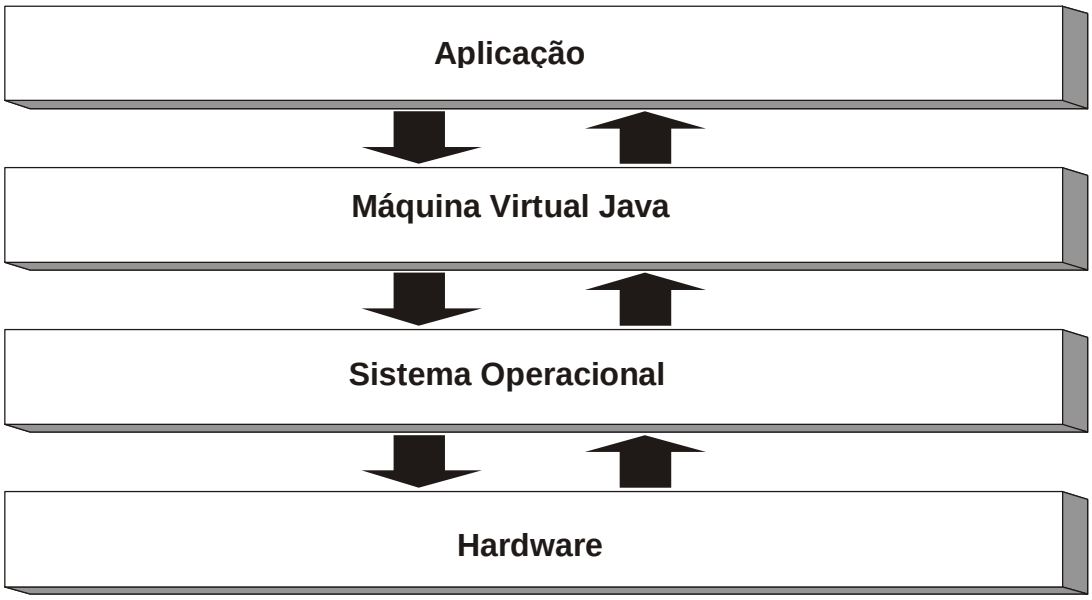


Figura 7 - Máquina Virtual Java (JVM)

O bytecode pode ser entendido como instruções em código de máquina para a JVM. Cada interpretador Java, seja uma ferramenta de desenvolvimento, uma parte de um sistema operacional ou um navegador da WWW, deve ter uma implementação da JVM. Deste modo, pode-se desenvolver o programa em Windows e rodar numa estação Solaris, Linux ou Mac OS X.

4.2 A plataforma Java

O ambiente Java é acompanhado de um conjunto de classes padrões que implementam o comportamento básico necessário para se construir aplicações baseadas em interfaces gráficas, redes e bancos de dados. Esta biblioteca de classes Java é chamada de API Java. O ambiente de desenvolvimento Java está disponível no site http://www.oracle.com.

Em resumo, uma plataforma Java tem dois componentes:
1. A JVM (Java Virtual Machine)
2. A API (Application Programming Interface)

A API Java é uma extensa coleção de componentes de softwares prontos para uso e que oferecem diversas capacidades como uma interface gráfica ou comunicação em rede. A API Java é composta por várias bibliotecas de classes e interfaces chamadas de pacotes. Cada pacote possui classes para capacidades específicas. O conjunto da JVM com a API Java formam o Ambiente de Execução Java, ou Java Runtime Environment (JRE). O Kit de Desenvolvimento Java, ou Java Developers Kit (JDK), inclui o JRE e o compilador, entre outras ferramentas de desenvolvimento.

Existem vários ambientes de desenvolvimento integrado, Integrated Development Environment (IDE), disponíveis para a linguagem Java. Entre as soluções livres disponíveis estão o Eclipse, que é a base para a implementação do Rational, e o NetBeans, que é da própria Sun MicroSystems.

4.3 Tipos em Java

Tipos primitivos em Java

Tipo Java	Descrição	Faixa de valores
boolean	Booleano de 1 bit	true ou false
byte	Inteiro de 8-bit com sinal	-127 a 128
char	Caractere Unicode	16-bit U+0000 a U+ffff
short	Inteiro de 16-bit com sinal	-32.768 e 32.767
int	Inteiro de 32-bit com sinal	-2.147.483.648 a 2.147.483.647
long	Inteiro de 64-bit com sinal	–9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
float	Ponto flutuante IEEE 32-bit	±1,40129846432481707 × 10E-45 a ±3,40282346638528860 × 10E38
double	Ponto flutuante IEEE 64-bit	±4,94065645841246544 × 10E-324 a ±1,79769313486231570 x × 10E308

Em Java, pode-se criar **arrays** com uma ou mais dimensões**.**

Arrays unidimensionais em Java são declarados de modo semelhante ao C#:

```
int [ ] numeros = new int [100];          // C# e Java int
[ ] numeros = {1, 2, 3, 4, 5};  // C# e Java
```

Arrays multiimensionais em Java são declarados de modo diferente ao C#:

```
int [ ][ ] numeros = new int [ ][ ] {{1, 2}, {3, 4}, {5, 6} }; // Java
```

4.4 Estrutura de uma aplicação Java

Quando se desenvolve um programa Java, são projetadas e construídas um conjunto de classes.

Quando o programa é executado, as classes são instanciadas, criadas e descartadas na medida em que for necessário.

Para se desenvolver um programa em Java, deve-se observar as seguintes normas:
1. Classes são escritas em arquivos com a extensão .java
2. Um arquivo .java pode conter diversas classes, mas apenas uma dessas classes poderá ser pública e estará visível ao resto da aplicação.

3. A classe pública de um arquivo .java deve ter exatamente o mesmo nome do arquivo .java (incluindo maiúsculas e minúsculas).
4. Para que um programa encontre as classes compiladas (arquivos .class), elas devem estar em diretórios conhecidos do Java, determinados pela variável de ambiente CLASSPATH. O diretório <diretorio_jdk>\jre\classes é o local padrão para localização de classes.

A classe é a unidade básica da programação de uma linguagem orientada a objetos. Todo código de programa em Java deve estar contido em uma classe.

Uma classe em Java tem dois componentes principais: a declaração da classe e o corpo da classe (definição). A declaração da classe é a primeira linha de código em uma classe. Ela deve, no mínimo, declarar o nome da classe.

A forma de definição de uma classe é:
class Conta {
. . .
}

A **declaração da classe** é feita pela palavra-chave class, seguida pelo identificador da classe. No exemplo apresentado, a classe declarada chama–se **Conta**. O código fonte que define as características da classe deve estar contido no bloco de comandos definido pelas chaves.

O corpo da classe segue a declaração da classe e aparece entre chaves. O corpo da classe possui a declaração de seus atributos, que fornecem seu estado, e dos métodos, que implementam seu comportamento. Os atributos e métodos da classe são chamados de membros da classe.

Membros públicos são os campos e funções membro que são visíveis externamente à classe. O conjunto de membros públicos de uma classe formam a interface pública da classe. Em Java, todos os membros são publicados para todas as classes de um mesmo pacote, a não ser que se defina o contrário.

4.5 Exemplo de programa em Java com argumentos de linha de comando:

Todo programa Java inicia sua execução no método **main,** o qual é um método de qualquer aplicativo Java.

Formato:

public static void **main** (String **args** [ ]) { }

Na declaração do método **main** são utilizados três **modificadores**:

1. **public** que indica que o método é acessível a partir de qualquer objeto da aplicação Java;

2. **static** que indica que o método é um atributo de classe, ou seja, não precisa que um objeto seja instanciado para que possa ser utilizado;
3. **void** que indica que o método não retorna valores.

Opcionalmente, o método **main,** pode receber um vetor de elementos do tipo String **args** como parâmetro. Esse vetor é o mecanismo pelo qual o sistema passa informações para a aplicação. Cada objeto String é chamado de argumento de linha de comando e pode afetar a operação da aplicação sem necessidade de recompilá-la.

Args[ ] é .é o parâmetro responsável por receber informações da linha de comando da aplicação., semelhante ao que foi mostrado na linguagem C#.

Qualquer número de argumentos pode ser passado para um programa java através da linha de comando. Ao executar o programa todas as palavras escritas após o nome da classe na linha de comando são argumentos. Os argumentos são limitados pelo espaço.

A função main não retorna um valor, pois sua assinatura é void. Entretanto, a JVM pode capturar códigos de saída através do comando System.exit(0); (onde 0 é o código de saída padrão para aplicação que foram executadas com sucesso).

Para ler dados em Java, usando o teclado, a partir do Java 1.5 ou J2SE 5, está disponível a classe **Scanner** do pacote **java.util**. Essa classe implementa as operações de entrada de dados pelo teclado no console.

Exemplo de programa em Java com argumentos de linha de comando (arquivo fonte **disponível no SGA**):

```
//
// nome do programa: Arglc115.java
//
// programadores: nome dos alunos (fulano, sicrano e beltrano)
// data: 28/01/2015
// descricao: exemplo de passagem de parametros na linha de comando (lc)
// entrada(s): parametros na linha de comandos (lc), sendo:
// o numero de matricula (6 dígitos)e o nome completo do aluno
// saida(s): exibe na tela os parametros na linha de comandom (lc)
// para executar e testar digite:
// java Arglc115 231650 joao manuel fragoso
// descricao de parametros opcionais na lc (variaveis pre-definidas do command prompt ou shell)
// args[0]:  primeiro parametro posicional na lc
// args[i]: i-esimo parametro posicionais na lc
//

class Arglc115 {
   public static void main(String[ ] args) {
     int i,j;
```

```
      j = args.length;
     System.out.println("\nVoce executou o comando java Arglc115 com " + j + " parametros : ");
     for (i= 0; i < j ; i++)   System.out.println("\nargs[" + i + "] = "  + args[i]);
     System.out.println();
     System.exit(0);
  }

}
```

A classe Arglc115 descrita no programa acima deve ser implementada em um arquivo chamado Arglc115.java.

O arquivo Arglc115.java pode ser digitado no Bloco de Notas, no Notepad++ ou editor no modo texto. No entanto, é comum que o Bloco de Notas acrescente a extensão *.txt* a cada arquivo salvo através do mesmo. Assim, após salvar o arquivo, certifique-se de que o mesmo possui a extensão correta. Caso ele tenha sido salvo com a extensão *.txt*, deve-se renomeá-lo.

Os arquivos podem ser compilados usando o Oracle JDK (*Java Development Kit*) da Java Platform, Standard Edition.

Para compilar e executar o programa:

javac Arglc115.java (compila e gera o executável Arglc115.class) **java** Arglc115 231650 JOAO MANUEL FRAGOSO (executa o programa)

4.1 (Exercício para entregar)

Fazer um programa, codificado em Java, que implemente uma calculadora com as mesmas especificações do exercício 3.8.

Ex 4.2 (PROVA 1 POO 2014): (para entregar)

Considere o programa **Prova0.java** arquivo fonte **disponível no SGA**) e a sequência de comandos executados no Prompt do Windows: **javac** Prova0.java (compila e gera o executável Prova0.class) **java** Prova0.java
Observe que o programa solicita o número de matrícula e o nome parcial do aluno. Utilize no exercício somente os dados de um aluno do grupo.

Escreva o que será impresso pelo programa na tela, os **valores** assumidos por **todas as variáveis** (nos comandos nos quais elas são referenciadas) e todos os **cálculos** necessários (somente a resposta não será considerada).

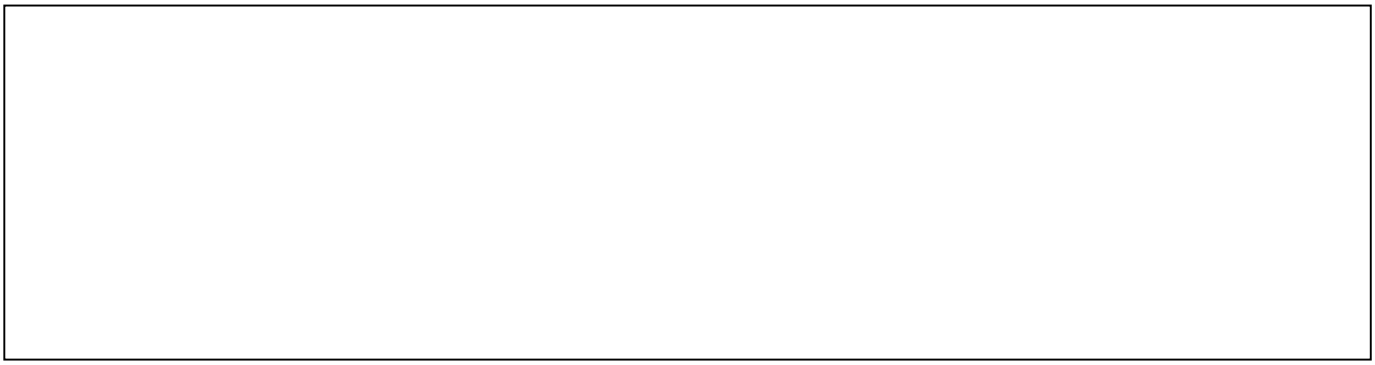
```
//
// nome do programa: Programa prova0.java (revisao de ATP - E/S em Java)
//
```

```
// programadores: nome dos alunos (fulano, sicrano e beltrano)
// data: 28/01/2015
// descricao: imprime o tamanho de alguns tipos primitivos, usa metodos de E/S no console para obter
// dados dos alunos, chama métodos para mostrar a passagem por valor, por referencia e o valor //
retornado
// entrada: numero de matricula (6 dígitos)e nome do aluno
// saida: exibe na tela os valores intermediarios e finais dos parametros passados para os metodos
//  e os retornados por eles
//

import java.io.IOException;
import java.io.*;
import  java.util.Scanner;  //  1.  importar  a  classe  Scanner
public class Prova0 {
  public static void main(String args[]) throws IOException{
    Scanner ler = new Scanner(System.in); // Instanciar e criar um objeto Scanner
     int A=0, B=0, C=0, i,j, k, l, m, n, NMAT;
     int[] vetor = new int[10];
    char t; double r = 0; String NOME1, NOME2;
     k = sizeof(float.class);
     l = sizeof(double.class);
    m = sizeof(int.class);  n = sizeof(long.class);
    System.out.println("Tamanho dos tipos primitivos: ");
    System.out.print("float = " + k + " bytes double = " + l);
    System.out.println(" bytes int = " + m + " bytes long = " + n + " bytes");
    System.out.print ("Digite seu numero de matricula: ");
    NMAT = ler.nextInt(); // entrada de dados (ler um valor inteiro)
     j = NMAT;
    System.out.println ("Resultado 1: ");
    for (i = 0; j > 0; i++){
       vetor[i] = j % 10 ;         j = j / 10;
        System.out.print (" " + i +  " " + vetor[i] + " " + j + " ");
    }
    j=i;
    System.out.print("\nDigite seu primeiro nome: ");
    NOME1 = readLine();
    System.out.printf("Digite seu ultimo sobrenome: ");
    NOME2 = readLine();
    A = vetor[1];  B = vetor[2]; C = vetor[3];
    System.out.println("NOME1 = " + NOME1 + " NOME2 = " +  NOME2);
    System.out.println ("Resultado 2: ");
    Sstem.out.println ("A = " + A + " B = " + B + " C = " + C );
    Faz1(A, B, C); // primeira chamada
     System.out.println ("A = " + A + " B = " + B + " C = " + C );
     System.out.println("r = " + r);
     System.out.println ("Resultado 3:" );
     for (i = 0; i < j ; i++){
         System.out.print (" " + i +  " " + vetor[i] + " ");
```

```
    }
    r = Faz2(A, B, C, vetor); //segunda chamada
    for (i = 0; i < j ; i++)System.out.print (" " + i +  " " + vetor[i]+ " ");
    System.out.println("\nr = " + r);
    System.out.println("Digite enter para terminar");
    t = (char)System.in.read(); // entrada de dados (ler um caractere)
  }
  public static void Faz1(int x, int y,  int z) {
     System.out.println("x = " + x + " y = " + y + " z = " + z);
     x = x + 10; y = y + 20; z = z + 30;
     z = x + y + z;
    System.out.println("x = " + x + " y = " + y + " z = " + z);
  }


public static  double Faz2( int x,  int y, int z, int[] vet){
          double w = 0;
          x = x + 10; y = y + 20; z = z + 30;
          vet[0]= x; vet[1]= y; vet[2]= z; vet[3]++;
          System.out.println("\nx = " + x + " y = " + y + " z = " + z);
          w = x + y + z;
          System.out.println("w = " + w);
          return (w);
}

public static int sizeof(Class dataType) {
 // usage int size = numDouble * sizeof(double.class) + numInt * sizeof(int.class);
          if (dataType == null) throw new NullPointerException();
          if  (dataType == int.class    || dataType == Integer.class)   return 4;
          if (dataType == short.class  || dataType == Short.class)     return 2;
          if (dataType == byte.class   || dataType == Byte.class)      return 1;
          if (dataType == char.class   || dataType == Character.class) return 2;
          if (dataType == long.class   || dataType == Long.class)      return 8;
          if (dataType == float.class  || dataType == Float.class)     return 4;
          if (dataType == double.class || dataType == Double.class)    return 8;
          return 4; // 32-bit memory pointer...
           // (I'm not sure how this works on a 64-bit OS)
}

public static String readLine() {
          int ch;
          String r= "";
          boolean     done=     false;
          while (!done) {
            try {
                 ch = System.in.read();
                  if (ch < 0 || (char)ch == '\n')
                     done = true;
```

```
                else if ((char)ch != '\r')
                    r = r + (char) ch;
            }
            catch(java.io.IOException       e){
                done = true;
            }
        }
return r;
 }

}
```

a) TELA:



b) Valores assumidos pelas variáveis e cálculos (continuar no verso, caso necessário)

4.6 Objetos em Java

Java utiliza semântica por referência para tratar objetos de uma classe. Isto significa que uma declaração de objeto na verdade é uma declaração de uma referência para um objeto. Uma referência é um ponteiro (apontador) constante. Referências são chamadas de alias (sinônimos), uma vez que elas simplesmente dão nome a um objeto existente na memória. A declaração de uma referência não implica na criação do objeto. O objeto será criado apenas quando for utilizada a cláusula new.

Declaração de um Objeto:

nomeClasse nomeObjeto; // Cria-se a referência mas não se cria o objeto.

Exemplo: Conta C1;
Cria-se uma referência para um objeto do tipo Conta, mas não se aloca a memória para armazenar o objeto. A Variável C1 aponta para NADA (null)

Criação do objeto através da cláusula new.
C1 = new Conta( );

Cria efetivamente um objeto Conva e faz com que a referência C1 aponte o objeto.

Ex 4.3 (Exercício para entregar)

Fazer um programa, codificado em Java, que implemente a classe Conta com as mesmas especificações do exercício da seção 3.8, ou seja, com as seguintes especificações:

- Apresenta inicialmente na tela um menu com as seguintes opções:
    1. Depositar na conta
    2. Fazer retirada na conta
    3. Imprimir saldo da conta
    4. Sair do programa
- Obter a opção do usuário
- Chamar o método correspondente, apresentar o resultado e sempre voltar ao menu inicial, exceto quando for selecionada a opção 4 (Sair do programa).
- Deve ter obrigatoriamente uma classe que contenha **um método para cada operação** na conta: depositar, fazer retirada e imprimir o saldo.

Um esboço da classe **Conta** é apresentado a seguir:

```
class Conta {
      // atributos
public int numero ;
public double saldo ;
     public int agencia ;

     // metodos
    public void Deposita ( double valor )
    {
       saldo += valor ;
    }

    public void Retira ( double valor )
    {
      // IMPLEMENTAR
    }

    public void ImprimeSaldo ()
    {
      // IMPLEMENTAR
    }
.....
}
```

4.4 (Exercício para entregar)

Complete o programa **ConversorP.java** (arquivo fonte **disponível no SGA**), para conversão de temperaturas com as seguintes especificações:

- Apresentar inicialmente na tela um menu com as seguintes opções:
    1. Converter de Celsius para Fahrenheit
    2. Converter de Fahrenheit para Celsius
    3. Sair do programa
- Obter a opção do usuário
- Chamar o método correspondente, apresentar o resultado e sempre voltar ao menu inicial, exceto quando for selecionada a opção 3 (Sair do programa).

- Deve ter obrigatoriamente uma classe que contenha pelo menos os dois métodos para converter a temperatura em Celsius para Fahrenheit e Fahrenheit para Celsius.