```
//conteúdo do arquivo pacote.h
#ifndef PACOTE_H
#define PACOTE_H
  #include <string>
  #include <iostream>
  class Pacote
  {
    friend istream &operator>> (istream &, Pacote &);
    friend ostream &operator<< (ostream &, Pacote &);
    public:
      Pacote(double = 1, double = getCustoPorQuilo());
      static void setCustoPorQuilo(double);
      static double getCustoPorQuilo();
      virtual double calculaCusto();
      …
  };
#endif
```

```
//conteúdo do arquivo principalpacote.cpp
#include "pacote.h"
#include "pacotedoisdias.h"
#include "pacotenoite.h"
#include <iostream>
using namespace std;
#define MAX 10
main()
{
  int opcao, numeroDeElementos=0;
  Pacote *vetor[MAX];
  do
  {
    cout << endl << "Digite:" << endl <<
    "1 - Inserir um pacote para entrega em dois dias;" <<
    endl << "2 - Inserir um pacote para entrega a noite;" << endl <<
    "3 - Imprimir enderecos para postagem e custos da postagem; " <<
    endl << "4 – Imprimir custos total das postagens." << endl <<
    "5 - Finaliza o programa." << endl << "Opcao? ";
    cin >> opcao;
```

```
switch(opcao)
{
  case 1:
    if (numeroDeElementos<MAX)
    {
      vetor[numeroDeElementos] = new PacoteDoisDias;
      cin >> (*vetor[numeroDeElementos]);
      numeroDeElementos++;
    }
    else
      cout << endl << "Nao ha espaco para mais pacotes" << endl;
    break;
  case 2:
    if (numeroDeElementos<MAX)
    {
      vetor[numeroDeElementos]= new PacoteNoite;
      cin >> (*vetor[numeroDeElementos]);
      numeroDeElementos++;
    }
    else
      cout << endl << "Nao ha espaco para mais pacotes" << endl;
    break;
```

```
        case 3:
          for (int i=0; i<numeroDeElementos; i++)
            cout << (*vetor[i]) << endl << endl << endl <<
            "Custo da postagem: " << vetor[i]->calculaCusto();
          break;
        case 4:
          {
            double auxiliar=0;
            for (int i=0; i<numeroDeElementos; i++)
              auxiliar += vetor[i]->calculaCusto();
            cout << endl << "Custo total das postagens: " <<
            auxiliar << endl;
          }
          break;
        case 5:
          cout << endl << "Obrigado por utilizar nosso software." << endl;
          break;
        default:
          cout << endl << "Opcao invalida!" << endl;
      }
    }while(opcao!=5);
    return 0;
}
```

Polimorfismo

Pode-se especificar uma classe abstrata em C++. Para tal, defini-se uma função-membro virtual como uma função virtual pura. Desta forma, a classe base que contém uma função virtual pura não pode ser instanciada.

Uma função virtual é definida como pura igualando seu protótipo a zero. Vejamos um exemplo:

```
#include <iostream>
using namespace std;
class Base {
  public:
     virtual void func()=0;
};
class Derivada: public Base {
  public:
    void func()
    {
      cout << "Esta eh func() de derivada" << endl;
    }
};
int main()
{
  Base *ptr;
  Derivada objeto1;
  ptr = &objeto1;
  ptr->func();
  Base objeto2; //Erro: pois a classe base é abstrata
  return 0;
}
```

Polimorfismo

Um aluno atento deve ter se perguntado:

Existe uma maneira de determinar em tempo de execução para qual das classes derivadas um ponteiro para a classe base está apontando?

A resposta é sim. A linguagem C++ disponibiliza para esta finalidade o *dynamic_cast*.

O exemplo a seguir o utiliza.

UNIVASF

```
#include <iostream>
using namespace std;
class Base {
  public:
     virtual void func()=0; };
class Derivada1: public Base {
  public:
    void func() { cout << "Esta eh func() de derivada1" << endl; } };
class Derivada2: public Base {
  public:
    void func() { cout << "Esta eh func() de derivada2" << endl; } };
int main()
{
  Base *ptr;
  Derivada1 objeto1, *ptr2;
  Derivada2 objeto2;
  ptr = &objeto1;
  ptr2 = dynamic_cast <Derivada1 *> (ptr);
  if (ptr2)
    ptr->func();
  ptr = &objeto2;
  ptr->func();
  return 0;
}
```

Exercício:

Utilizando este conceito implemente uma nova opção no menu do exercício do slide 415 a qual possibilita determinar o custo total apenas dos pacotes que serão entregues à noite.

```
//conteúdo do arquivo principalpacote.cpp
...
    "5 - Imprimir o custo total apenas dos pacotes que serao
entregues a noite;" << endl <<
        ...
        case 5:
          {
            double auxiliar=0;
            PacoteNoite *ptrAux;
            for (int i=0; i<numeroDeElementos; i++)
            {
              ptrAux = dynamic_cast <PacoteNoite *> (vetor[i]);
              if (ptrAux)
                auxiliar += vetor[i]->calculaCusto();
            }
            cout << endl << "Custo total das postagens para " <<
            "entrega a noite: " << auxiliar << endl;
          }
          break;
          ...
    }while(opcao!=6);
    return 0;
}
```

Linguagem de Programação Java

UNIVASF

Linguagem de Programação Java

Breve histórico:

- Green (1991) – projeto de pesquisa coorporativa da Sun Microsystems;
- Desenvolveu uma linguagem baseada em C++ denominada Oak (árvore de carvalho);
- Desenvolvida por James Gosling;
- Renomeada para Java;
- Apresentada ao público em 1995.

UNIVASF

Linguagem de Programação Java

Características básicas:

- Desenvolvida inicialmente para dispositivos eletrônicos inteligentes;
- Utilizada para adicionar conteúdo dinâmico na World Wide Web;
- Atualmente, também utilizada para desenvolver aplicativos coorporativos de grande porte.

UNIVASF

Linguagem de Programação Java

Java é uma linguagem de programação orientada a objetos projetada para ser portável a todas as plataformas. Esta portabilidade baseia-se no fato da linguagem ser interpretada.

O processo de compilação gera um código independente de máquina denominado *bytecode*.

Durante a execução o bytecode é interpretado por uma máquina virtual (JVM) instalada na máquina real, em outras palavras, a JVM lê o bytecode e o traduz para uma linguagem que o computador possa entender.

Sendo assim, para portar Java para uma arquitetura especifica, basta instalar a máquina virtual especifica para a arquitetura em questão.

UNIVASF

Linguagem de Programação Java

Antes de partirmos para a construção de nosso primeiro programa Java vamos analisar algumas características da linguagem, como os tipos primitivos, identificadores, operadores, e etc.

Iniciaremos pelos **Tipos Primitivos.**

Java não trata-se de uma linguagem de programação orientada a objetos pura, ou seja, ela dispõe de um conjunto de tipos primitivos que não constituem objetos.

Estes tipos primitivos possibilitam a representação de valores: booleanos, caracteres, numéricos inteiros e numéricos em ponto flutuante.

UNIVASF

TIPOS PRIMITIVOS

- Palavra-reservada: boolean - define variáveis lógicas
  - Podem assumir valores true ou false
  - O valor padrão é false
  - Ocupa 1 bit
  - Exemplo de declaração:

    boolean sinalizador, sinalizador2 = true;

    sinalizador = true;

Operador de atribuição

TIPOS PRIMITIVOS

- Palavra-reservada: char - define variáveis caractere
  - Capazes de armazenar valores de caracteres Unicode
  - Ocupa 16 bits
  - O valor padrão é o caracter NULL
  - Exemplo de declaração:

```
char caractere1, caractere2 = 'a';
caractere1 = 'b';
```

TIPOS PRIMITIVOS

- Palavra-reservadas: byte, short, int e long - definem variáveis do tipo inteiro

Tipo	Tamanho	Faixa de valores
byte	8 bits	-128 a 127
short	16 bits	-32.768 a 32,767
int	32 bits	-2.147.483.648 a 2.147.483.647
long	64 bits	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

O valor padrão destes tipos é 0

Exemplo de declaração:

```
int v1=3, v2;
long v3;
```

TIPOS PRIMITIVOS

- Palavra-reservadas: float e double - definem variáveis do tipo real

Tipo	Tamanho	Faixa de valores
float	32 bits	$\pm 1.40239846 \times 10^{-45}$ a $\pm 3.40282347 \times 10^{+38}$ com nove dígitos de precisão
double	64 bits	$\pm 4.94065645841246544 \times 10^{-324}$ a $\pm 1.79769313486231570 \times 10^{+308}$ com dezoito dígitos de precisão

O valor padrão destes tipos é 0.0

Exemplo de declaração:

```
float v4=3.1, v5;
double v6=3.1415921234;
```

IDENTIFICADORES

Compostos por seqüências de caracteres que devem obedecer as seguintes regras:

- podem ser compostos por letras, dígitos e pelos símbolos _ e $;
- não podem ser inicializados por um dígito;
- não podem ser iguais a palavras reservadas.

Observação: Java é sensível ao caso.

Para melhorar a legibilidade dos programas em Java, vamos adotar a grafia camelo.

UNIVASF

Operadores Aritméticos

- Unários: -, --, ++

Exemplos:

```
-1

-5.9

...
short a;
...
a++;
```

- **Operadores Aritméticos**
  - Binários: +, -, *, /, %

Associação

Símbolo	*Operação*
+	Soma
-	Subtração
*	Multiplicação
/	Divisão real e Quociente da divisão inteira
%	Resto da divisão inteira

Operadores Aritméticos

Precedência (Hierarquia nas operações)

Hierarquia	Operação
1	- (unário)
2	*, /, %
3	+, -

Operadores Relacionais

Operador	Ação
>	maior que
>=	maior ou igual a
<	menor que
<=	menor ou igual a
==	igual a
!=	diferente de

Linguagem de Programação Java

- Operadores Lógicos

Operador	Ação
&&	e
||	ou
!	negação

Linguagem de Programação Java

- Operadores Lógicos bit-a-bit

Operador	Ação
&	e
|	ou
^	xou
<<	deslocamento a esquerda
>>	deslocamento a direita
>>>	deslocamento a direita com preenchimento de zeros
~	complemento

- Operadores de Atribuição

  =, +=, -=, *=, /=, %=, <<=, >>=, >>>=, &=, |=, ^=

  Exemplos: a=5;

  a+=5; ⇔ a=a+5;

  a-=5; ⇔ a=a-5;

- Operador Condicional

  ?:

  Exemplo: E ? V1 : V2

COMENTÁRIOS

```
/* Determina comentários em
      múltiplas linhas */

// comentário até o final da linha

/** Comentário específico de Java. Os textos destes
comentários podem ser utilizados pela ferramenta
javadoc para gerar a documentação do código em
formato hipertexto */
```

CARACTERES ESPECIAIS OU SEQÜÊNCIAS DE ESCAPE

Caractere	Significado
\n	nova linha
\t	tab
\b	backspace
\r	retorno de carro
\f	avança página da impressora (form feed)
\\	barra invertida
\’	apóstrofe
\”	aspas

ESTRUTURAS DE CONTROLE DE FLUXO

```
if (condicao)
   comando

if (condicao)
{
   sequenciaDeComandos
}
```


UNIVASF

ESTRUTURAS DE CONTROLE DE FLUXO

```
if (condicao)
   comando1
else
   comando2
```


UNIVASF

ESTRUTURAS DE CONTROLE DE FLUXO

```
if (condicao)
{
   sequenciaDeComandos1
}
else
{
   sequenciaDeComandos2
}
```

ESTRUTURAS DE CONTROLE DE FLUXO

```
switch (variavel) // variavel deve ser inteira ou caractere
{
   case valor1:
      blocoDeComandos
      break;
   case valor2:
      blocoDeComandos
      break;
   ...
   default:
      blocoDeComandos
}
```


UNIVASF

ESTRUTURAS DE CONTROLE DE FLUXO

```
while (condicao)
{
   blocoDeComandos
}

do
{
   blocoDeComandos
} while (condicao);
```

ESTRUTURAS DE CONTROLE DE FLUXO

```
for (inicializacao; condicao; incremento)
{
   blocoDeComandos
}

break

continue
```


UNIVASF

ESTRUTURAS DE CONTROLE DE FLUXO

Em situações onde há diversos laços de repetição (comandos de iteração) aninhados, os comandos *break* e *continue* transferem a execução para o ponto correspondente no bloco onde ocorrem.

Se for necessário especificar a transferência para outro laço de repetição, os comandos *break* e *continue* rotulados podem ser utilizados.

Veremos um exemplo:

UNIVASF

Linguagem de Programação Java

```
...
label: for( ... ; ... ; ... )
{
      ...
      while ( ... )
      {
         ...
         if ( ... )
            break label;
        ...
      }
      ...
}
...
```

ESTRUTURA GERAL DE UM PROGRAMA EM JAVA

```
public class NomeDaClassePrincipal
{
   public static void main (String args[])
   {
      ...
   }
}
```

ESTRUTURA GERAL DE UM PROGRAMA EM JAVA

```
public class ClassePrincipal
{
   public static void main (String args[])
   {
      System.out.print(“Primeiro programa Java.”);
   }
}
```

ESTRUTURA GERAL DE UM PROGRAMA EM JAVA

```
public class ClassePrincipal
{
   public static void main (String args[])
   {
      System.out.println(“Primeiro programa Java.”);
   }
}
```

ESTRUTURA GERAL DE UM PROGRAMA EM JAVA

```
public class ClassePrincipal
{
   public static void main (String args[])
   {
      System.out.printf("%c%s%c", '\n',
      "Primeiro programa Java.",'\n');
   }
}
```

Linguagem de Programação Java

```
//exemplo de leitura de dados inteiros do teclado
import java.util.Scanner;
public class Soma
{
  public static void main(String args[])
  {
    Scanner input = new Scanner(System.in);
    int numero1;
    int numero2;
    int soma;
    System.out.print("Forneca um valor inteiro: ");
    numero1 = input.nextInt();
    System.out.print("Forneca outro valor inteiro: ");
    numero2 = input.nextInt();
    soma = numero1 + numero2;
    System.out.printf("A soma de eh %d\n", soma);
  }
}
```

COMPILAÇÃO

Para compilar um programa em Java utilizamos o comando:

javac nomeDoArquivo.java

O comando anterior compilará o código fonte contido no arquivo e gerará, caso o mesmo não apresente erros de compilação, o bytecode armazenando-o no arquivo:

nomeDoArquivo.class

O qual pode ser executado pela JVM com o seguinte comando:

java nomeDoArquivo

CLASSES (tipos por referência)

Como os conceitos da OO já foram trabalhados em UML e implementados na linguagem C++ otimizaremos nosso estudo de Java, apenas demonstrando como os conceitos OO são implementados em Java e tratando de algumas particularidades desta linguagem.

Vamos analisar o exemplo da declaração da classe Ponto2D:

```
import java.util.Scanner;
public class Ponto2D /** classe que dará nome ao arquivo*/
{
  private float x; /** variável de instância*/
  private float y; /** variável de instância*/
  public void setX (float novoX) /** método de instância*/
  {
    x = novoX;
  }
  public float getX () /** método de instância*/
  {
    return x;
  }
  public void setY (float novoY) /** método de instância*/
  {
    y = novoY;
  }
  public float getY () /** método de instância*/
  {
    return y;
  }
  public void apresentaCoordenadas() /** método de instância*/
  {
    System.out.printf("\n(%f, %f)\n", getX(), getY());
  }
```

```
    public static void main(String args[])
    {
      Scanner input = new Scanner(System.in);
      float x, y;
      Ponto2D p = new Ponto2D(); //instanciação de um objeto
      System.out.println("Coordenadas iniciais do ponto:");
      p.apresentaCoordenadas(); //invocação de método
      System.out.print("Forneca a coordenada x do ponto: ");
      x = input.nextFloat();
      p.setX(x); //invocação de método
      System.out.print("Forneca a coordenada y do ponto: ");
      y = input.nextFloat();
      p.setY(y); //invocação de método
      p.apresentaCoordenadas(); //invocação de método
    }
}
```

```
public class Ponto2D /** classe Ponto2D com construtor */
{
  private float x;
  private float y;
  public Ponto2D() /** método construtor */
  {
    x = 1;
    y = 1;
  }
  public void setX (float novoX)
  {
    x = novoX;
  }
  public float getX ()
  {
    return x;
  }
  public void setY (float novoY)
  {
    y = novoY;
  }
```

```
  public float getY ()
  {
    return y;
  }
  public void apresentaCoordenadas() /** utilização do print para
imprimir valores não literais */
  {
    System.out.print ("\n(");
    System.out.print (getX());
    System.out.print (", ");
    System.out.print (getY());
    System.out.println (")");
  }
  public void apresentaCoordenadas2() /** utilização do print para
imprimir valores não literais */
  {
    System.out.println ("\n(" + getX() + ", " + getY() + ")");
  }
}
```

```
import java.util.Scanner;
public class Ponto2D
{
   private float x;
   private float y;
   public Ponto2D(float novoX, float novoY) /** método construtor */
   {
      x = novoX;
      y = novoY;
   }
   public void setX (float novoX)
   {/*...*/}
   public float getX ()
   {/*...*/}
   public void setY (float novoY)
   {/*...*/}
   public float getY ()
   {/*...*/}
   public void apresentaCoordenadas()
   {
      System.out.printf("\n(%f, %f)\n", getX(), getY());
   }
```

```
  public static void main(String args[])
  {
    Scanner input = new Scanner(System.in);
    float x, y;
    Ponto2D p = new Ponto2D(); /* ERRO! Instanciação de um objeto inadequada */
    Ponto2D p = new Ponto2D(5.2f, 7.3f); //instanciação de um objeto
    System.out.println("Coordenadas iniciais do ponto:");
    p.apresentaCoordenadas(); //invocação de método
    System.out.print("Forneca a coordenada x do ponto: ");
    x = input.nextFloat();
    p.setX(x); //invocação de método
    System.out.print("Forneca a coordenada y do ponto: ");
    y = input.nextFloat();
    p.setY(y); //invocação de método
    p.apresentaCoordenadas(); //invocação de método
  }
}
```

Exercício

Declare uma classe Data que possua como variáveis de instância três inteiros representado o dia, o mês e o ano. Implemente métodos de instância que manipulem adequadamente as variáveis de instância existentes.

```
public class Data
{
   private int dia;
   private int mes;
   private int ano;
   public Data(int novoDia, int novoMes, int novoAno)
   {
            setAno(novoAno);
            setMes(novoMes);
            setDia(novoDia);
   }
   public Data() // sobrecarga de método
   {
   }
   public int getDia ()
   {
     return dia;
   }
   public int getMes ()
   {
     return mes;
   }
}
```

```
public int getAno ()
{
   return ano;
}

public void setDia(int d)
{
   dia = verificaDia(d);
}
public void setMes(int m)
{
   if (m>0 && m<=12)
      mes = m;
   else
   {
      mes = 1;
      System.out.println ("Mes invalido (" + m + ") setado para 1.");
   }
}
```

```
public void setAno(int a)
{
  if (a>=1900 && a<2011)
    ano = a;
  else{
    ano = 1900;
    System.out.println ("\nAno invalido (" + a + ") setado para 1900.");
  }
}
int verificaDia(int diaTeste)
{
  final int diasPorMes[] = {0,31,28,31,30,31,30,31,31,30,31,30,31}; //vetor
  if (diaTeste>0 && diaTeste <= diasPorMes[getMes()])
    return diaTeste;
  if (getMes()==2 && diaTeste==29 && (getAno()%400==0 ||
(getAno()%4==0 && getAno()%100!=0)))
    return diaTeste;
  System.out.println ("\nDia invalido (" + diaTeste + ")setado para 1.");
  return 1;
}
```

```
  public static void main(String args[])
  {
    Data d = new Data(11, 06, 2010);
    System.out.printf("\n%d/%d/%d\n",d.getDia(),d.getMes(),d.getAno());
    Data d2 = new Data(25, 12, 2000);
   System.out.println("\n"+d2.getDia()+"/"+d2.getMes()+"/"+d2.getAno());
  }
}
```

VETORES

Como vimos um exemplo de sintaxe para declaração de um vetor e observamos que um vetor pode ser inicializado na declaração.

Porém, também podemos declarar um vetor da seguinte forma:

```
int v[];
float v2[] = new float[10];
v = new int[2];
char[] v3;
```

Um vetor é um objeto e possui uma referência para um elemento de vetor (*array*). Cada vetor possui uma variável de instância pública denominada length que contém o comprimento do vetor.

VETORES

Um detalhe sutil com relação à declaração de vetores é que em

```
int v[], v2, v3;
```

temos apenas o vetor v, e em

```
char[] v4, v5, v6;
```

temos três vetores.

Em Java existe uma forma peculiar da estrutura for, o exemplo a seguir demonstra sua sintaxe e utilização.

```
public class Teste
{
   static public void main (String args[])
   {
      char []v = {'M','a','r','c','e','l','o'};
      System.out.println(v.length);
      for (char aux:v)
         System.out.print(aux);
   }
}
```

Exercício

Declare uma classe Compromisso que possua como variáveis de instância um inteiro representando a hora do compromisso, uma string representando a descrição do compromisso e um objeto da classe Data. Implemente métodos de instância que manipulem adequadamente as variáveis de instância existentes.