UNIVERSIDADE DO ESTADO DE SANTA CATARINA
DCC – Departamento de Ciência da Computação
Joinville – SC

CURSO: Bacharelado em Ciência da Computação
DISCIPLINA: POO0001 – Programação Orientada a Objetos
PROFESSOR: Diego Buchinger
AULA 01 – Introdução a Orientação a Objetos e a linguagem de programação Java

1. PARADIGMAS DE PROGRAMAÇÃO

Paradigma de programação é a filosofia adotada na maneira de construir um software. Um mesmo software pode ser construído usando diferentes paradigmas, mas o uso de um determinado paradigma pode facilitar e tornar mais rápida a sua produção. Assim, a escolha de um paradigma apropriado para a construção de um software é aconselhada. Alguns paradigmas usuais são:

- Imperativo ou Procedural (ex: C e Fortran)
- Lógico (ex: Prolog)
- Funcional (ex: Lisp, Haskell)
- Orientado a Objetos (ex: Java, C++, C#, SmallTalk)

Linguagens de programação que utilizam diferentes paradigmas costumam ter diferenças significativas.

Nas nossas aulas vamos estudar o paradigma orientado a objetos, partindo do pressuposto que o aluno já tenha familiaridade com o paradigma imperativo (i.e. sabe escrever programas na linguagem de programação C). Na verdade, o paradigma orientado a objetos pode inclusive ser entendido como uma extensão do paradigma imperativo com a introdução de novos conceitos. Este paradigma surgiu das tentativas de solucionar problemas complexos usando um desenvolvimento confiável, menos complexo e de baixo custo de desenvolvimento e manutenção. É uma proposta na qual o sistema é modelado como um conjunto de objetos (coisas) que interagem entre si através de troca de mensagens.

O surgimento do Paradigma orientado a objetos tem relação com uma mudança na percepção do agrupamento e separação dos componentes e funcionalidades importantes de um sistema. Em um sistema acadêmico, por exemplo, era comum a separação das funções pelas suas ações, por exemplo: *inserirAluno*, *inserirNota*, *inserirFrequencia*; *alterarAluno*, *alterarNota*, *alterarFrequencia*; e *removerAluno*, *removerNota*, *removerFrequencia*. Apesar de ser um agrupamento coerente, uma outra possibilidade que passou a ser mais explorada foi a separação dessas funções pelos **objetos** que estavam sendo afetados, por exemplo: *alunoInserir*, *alunoAlterar*, *alunoRemover*; *notaInserir*, *notaAlterar*, *notaRemover; frequenciaInserir*, *frequenciaAlterar*, *frequenciaRemover*. Este foi o princípio da percepção de que associar dados e ações a seus respectivos objetos poderia ser vantajoso para o desenvolvimento de software complexo, diminuindo a distância entre a modelagem computacional e o mundo real.

A ideia de orientação a objetos surgiu na década de 1960, com a criação da linguagem Simula (*Simulation Language*). Em 1967 esta linguagem introduziu conceitos de classe e herança que se tornaram duas características marcantes da orientação a objetos (mais detalhes sobre essas características serão explicados nas próximas aulas). O termo Programação Orientada a Objetos foi

introduzido apenas em 1983, com a linguagem de programação Smalltalk-80, mas logo outras linguagens híbridas (imperativas e orientadas a objetos) como o C++ e Object Pascal surgiram.

Mas enfim, por que usar o paradigma orientado a objetos? Existem diversas vantagens associadas ao seu uso: alta reutilização de código, redução no tempo de manutenção, possibilita maior abstração do sistema, pode aumentar a qualidade e a produtividade, e possui boa aceitação comercial – até porque muitos sistemas comerciais tem um grande foco em dados de objetos do mundo real, o que é justamente um ponto forte da orientação a objetos. Entretanto, deve-se ter cuidado para não pensar em POO como uma solução para tudo ou então não seguir boas práticas sugeridas para este paradigma.

Hoje já existem diversas linguagens de programação que permitem a implementação de projetos utilizando o paradigma orientado a objetos, entre elas: Java, C++, C#, Ada, Smalltalk, Object Pascal, Python, entre outras. Para esta disciplina vamos utilizar e estudar a linguagem Java e como ela permite o uso de programação orientada a objetos.

2. JAVA

Em 1991 a Sun Microsystems financiou um projeto de pesquisa interna com o objetivo de criar uma linguagem para desenvolver programas para dispositivos eletrônicos inteligentes de pequeno porte como eletrodomésticos e eletroeletrônicos. A pesquisa resultou na criação de uma linguagem de programação baseada em C++ que foi denominada inicialmente de OAK, mas foi renomeada para Java e que apresentava a capacidade de executar um mesmo programa em vários aparelhos diferentes (*write once, run anywhere*), utilizando o conceito de máquina virtual. Esta característica fez com que o Java se adequasse bem as necessidades da época.

2.1 Características da Linguagem

A seguir são apresentadas algumas das principais características da linguagem Java:

- Orientada a Objetos: implementa quase todas as características da orientação a objetos, exceto herança múltipla e sobrecarga de operadores;
- Portabilidade: é independente de plataforma, mas necessita que a Máquina Virtual Java (JVM) esteja instalada;
- Híbrida: é uma linguagem compilada e interpretada (ver Figura 1);
- Segurança: não há ponteiros, utiliza mecanismos de tratamento de exceções e realiza coleta automática de lixo (porção de memória que deixou de ser utilizada);
- Suporta execuções concorrentes/paralelas: permite o uso de threads;
- Recursos de Rede e Suporte a programação de sistemas distribuídos: possui bibliotecas que permitem ou facilitam a utilização de protocolos de rede, de sockets, *Remote Method Invocation* (RMI) e *Service Oriented Architecture* (SOA);

A Máquina Virtual Java (JVM) desempenha um papel fundamental para a execução de programas implementados nesta linguagem. É a JVM que transforma o código fonte em um código intermediário (arquivos ".class"), também chamado de bytecode, o qual posteriormente é lido e interpretado pela própria JVM no dispositivo onde o programa será executado. Assim, cada especificação da JVM é direcionada para um determinado hardware.

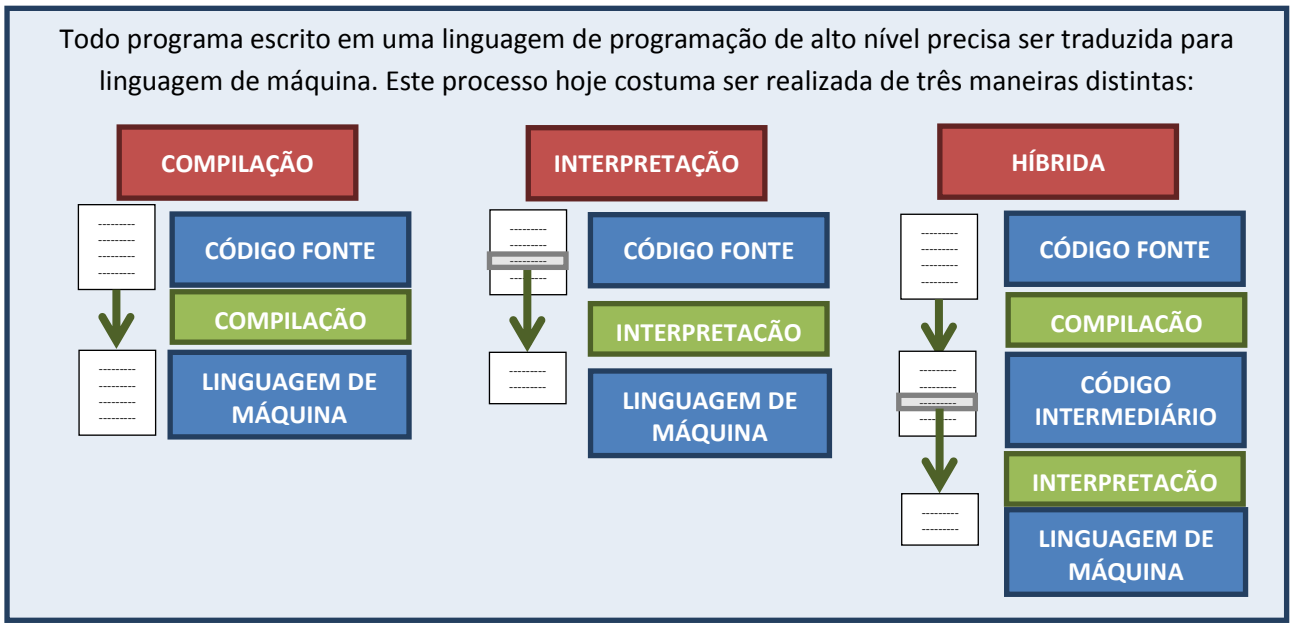

FIGURA 1 – Métodos de tradução de código fonte para linguagem de máquina

2.2 Outros Detalhes do Java

A tecnologia Java é uma boa opção de linguagem de programação orientada a objetos pois sua plataforma recebe constantes atualizações (está na versão 9 atualmente), possui uma comunidade ativa e é bastante utilizada no mercado. Atualmente a tecnologia Java é de propriedade da Oracle, que realizou a compra da Sun Microsystems em 2009. A linguagem de programação Java possui uma estrutura composta por diversos componentes que são corriqueiramente denominados pelas suas siglas. Assim é importante conhecer as principais nomenclaturas envolvidas:

- API – Interface de Programação de Aplicativos *(Application Programming Interface)*: conjunto de bibliotecas que podem ser reutilizadas no desenvolvimento para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da implementação do software, mas apenas usar seus serviços.
- JDK – *Java Development Kit*: Conjunto de ferramentas de desenvolvimento Java (compilador + depurador + API + documentação). Requisito para programar na linguagem.
- JRE – *Java Runtime Environment*: Máquina Virtual Java, bibliotecas e demais componentes necessários para executar aplicações escritas em Java.
- Java SE – *Java Standard Edition*: plataforma padrão do Java. Serve para o desenvolvimento de aplicações desktop comum.
- Java ME – *Java Micro Edition*: plataforma Java para móveis e sem fio (pequeno porte).
- Java EE – *Java Enterprise Edition*: baseado no Java SE com a adição de serviços web, componentes, gerenciamento e bibliotecas de comunicação. Utilizada para o desenvolvimento de aplicações corporativas, distribuídas e orientadas a serviços.

3 HANDS-ON

Vamos iniciar escrevendo um programa básico sem auxílio de uma Interface de Desenvolvimento (IDE) neste momento. A primeira coisa a ser feita é preparar o ambiente para que seja possível compilar e executar a aplicação.

3.1 Configurando o Ambiente

Os programas de compilação e interpretação de programas Java se encontram no diretório "bin" dentro da pasta de instalação, e são, respectivamente: javac e java. Para compilar um programa precisamos utilizar o programa javac que poderia ser invocado em linha de comando (no prompt de comando do Windows ou terminal do Linux) semelhante a seguinte maneira:

```
C:\Program Files\Java\jdk1.8.0_131\bin\javac   meuprograma.java
```

Não é muito prático certo? Para simplificar este processo, devemos adicionar uma variável de ambiente que indica um caminho em que um programa requisitado deve ser procurado. No Windows podemos adicionar tal variável da seguinte maneira:

1. Clicar com o botão direito sobre computador e escolher a opção "Propriedades"
2. Escolher a opção do painel da esquerda: "Opções avançadas do sistema"
3. Na guia Avançado (já vem ativa por padrão), clicar no botão "Variáveis de Ambiente..."
4. Na nova janela procure pelo registro de nome "Path" na região de "Variáveis do sistema", selecione este item e clique sobre o botão "Editar..."
5. Adicione ao final do valor da variável um caractere ponto-e-vírgula (;) seguido do diretório da instalação do java, por exemplo: "C:\Program Files\Java\jdk1.8.0_131\bin" e pressione o botão OK, e novamente o botão OK na janela de Variáveis de Ambiente.

Pronto! Agora quando você executar qualquer comando no prompt de comando do Windows, ele irá procurar se o programa a ser executado se encontra neste novo diretório. Agora a compilação de um código pode ser executada da seguinte maneira:

```
javac   meuprograma.java
```

O segundo passo da configuração do ambiente tem relação com a execução do programa e o conceito de variável CLASSPATH. Esta variável indica aos programas, incluindo os programas do JDK, quais diretórios devem ser vasculhados para encontrar as classes. O CLASSPATH pode ser definido de maneira fixa por uma variável de sistema, ou então provisoriamente no momento da execução do programa. Para definir o CLASSPATH de maneira fixa deve-se criar ou editar uma variável de sistema com este mesmo nome (CLASSPATH) e adicionar o caractere ponto (.), separado por ponto-e-vírgula caso já exista algum valor. Fazendo isso, para executar um programa Java basta executar o programa java e informar qual é o arquivo de *bytecodes* a ser executado:

```
java   meuprograma
```

De maneira alternativa, o CLASSPATH pode ser definido provisoriamente utilizando uma simples opção do programa Java (-cp) que indica qual o CLASSPATH a ser utilizado:

```
java   -cp   .   meuprograma
```

Uma observação final importante é que se um prompt de comando já estiver aberto antes de realizar a configuração das variáveis de ambiente, será necessário fechar e abrir novamente este programa para que as alterações tenham efeito.

3.2 Primeiro Programa Java

Para criar nosso primeiro programa Java devemos criar um arquivo com extensão ".java", mas cuidado, pois alguns sistemas operacionais ocultam a verdadeira extensão do arquivo por padrão e você pode acabar criando um arquivo do tipo: nome.java.txt. Como primeiro passo, crie um arquivo com o nome HelloWorld.java e copie o seguinte código-fonte:

```
1.  public class HelloWorld {
2.     public static void main( String[] args ){
3.        System.out.println("Hello World");
4.     }
5.  }
```

O código acima declara uma classe chamada HelloWorld, que é um elemento da orientação a objetos que veremos mais adiante, então declara a função main, que é o ponto de partida do programa, e dentro desta função (na terceira linha) escreve na tela a string “Hello World”. Neste ponto você já pode compreender uma crítica usual ao Java em relação à sua característica de ser muito verboso (algumas operações exigem muita escrita). Entretanto, veremos um pouco mais adiante que esta característica na verdade atende muito bem aos conceitos de orientação a objetos que estaremos utilizando.

Enfim, chegou o momento de compilarmos e executarmos o nosso programa que diz “Hello World”! O primeiro passo é compilar o código, transformando-o em *bytecodes* (lembre-se que o Java é uma linguagem híbrida e não gera um executável!), um arquivo com a extensão “.class” e então usar a JVM para interpretar este arquivo de *bytecodes*. Para isto vamos executar os comandos:

```
javac   HelloWorld.java
java  -cp  .  HelloWorld
```

Se você fez tudo certo deverá ver a mensagem “Hello World” mostrada na sua tela. Caso algum erro de digitação do programa tenho ocorrido algum erro deverá ser exibido já na execução do primeiro comando. Caso algum erro de configuração do ambiente tenha ocorrido, uma mensagem poderá ser exibida no primeiro ou no segundo comando, dependendo do tipo de erro. Dois erros comuns são: não adicionar a variável de Path corretamente o que resultará em uma mensagem dizendo que o comando javac não foi reconhecido, ou não configurar o classpath corretamente e não utilizar a opção –cp, o que pode resultar em uma mensagem dizendo que não foi possível localizar nem carregar a classe principal do arquivo que tentou-se executar.

3.3 Criando Programas Mais Elaborados

Chegou o momento de criarmos programas mais elaborados, mas por enquanto não vamos focar nos conceitos de orientação a objetos ainda. Vamos programar primeiramente de forma imperativa (sim, mesmo sendo uma linguagem com orientação a objetos é possível programar de forma imperativa, até mesmo porque a orientação a objetos é um tipo de extensão deste outro paradigma, como já foi mencionado anteriormente) e para fazer isso vamos escrever todo o código do programa dentro da função main, que é o ponto de início do programa. A seguir serão descritos alguns aspectos da sintaxe da linguagem, dos principais comandos e também alguma informação sobre como realizar leitura e escrita.

➢ Tipos simples e declaração de variáveis: os tipos primitivos do Java são parecidos com os tipos primitivos do C++ e podem ser vistos como uma extensão dos tipos da linguagem C. Um fato importante sobre os tipos é que, diferentemente da linguagem C e C++, na qual os limites dos valores inteiros, ou de ponto flutuante são determinados dependendo da arquitetura e do compilador, os tipos primitivos do Java possuem valores mínimos e máximos bem definidos. Segue uma tabela com os tipos primitivos:

TABELA 1 – tipos primitivos da linguagem de programação Java

Tipo	Tamanho	Mínimo	Máximo	Default
boolean	-	-	-	false
char	16-bit	Unicode 0	Unicode 2^{16} - 1	\u0000
byte	8-bit	-128	+127	0
short	16-bit	-2^{15}	$+2^{15}$ - 1	0
int	32-bit	-2^{31}	$+2^{31}$ - 1	0
long	64-bit	-2^{63}	$+2^{63}$ - 1	0
float	32-bit	IEEE 754	IEEE 754	0.0
double	64-bit	IEEE 754	IEEE 754	0.0
void	-	-	-	-

A declaração e uso de variáveis é similar as definições da linguagem de programação C, tanto no que tange às restrições de nomenclatura de variáveis e a forma de declará-las:

```
Tipo  identificador  [= valor];
```

Note que strings não são um tipo primitivo no Java e são inclusive diferentes do que um vetor de caracteres usual. Entraremos em mais detalhes sobre strings mais adiante.

➢ Escrever no console: para escrever no console (em tela) é utilizado o comando System.out.print, System.out.println e System.out.printf. O segundo comando é bastante utilizado na prática, pois ele pula automaticamente uma linha após imprimir o texto na tela, já o terceiro é utilizado quando se deseja definir a formatação da saída a ser exibida, com uma sintaxe-se parecida com a função printf da linguagem C. Um aspecto importante a ser mencionado é a possibilidade de concatenar textos e variáveis a serem mostradas usando o operador + sem a necessidade de usar as expressões de formatação do printf. Por exemplo:

```
int valor = 25;
System.out.println("Valor total: " + valor + " reais");
```

➢ Leitura de valores pelo teclado: para ler dados do teclado precisamos de uma "variável" mais sofisticada e que exige a inclusão/importação de uma biblioteca. Vamos utilizar uma variável de um tipo não primitivo chamado de Scanner (sim, com S maiúsculo). Neste momento, imagine este tipo não primitivo como uma estrutura de dados que implementa algumas operações que facilitam a leitura de dados. Entretanto, para poder utilizar um Scanner, precisamos primeiro importar a biblioteca que possui a sua definição e uma biblioteca para definir o padrão de números (é possível que sua máquina esteja com o padrão português por padrão, obrigando com que os números reais lidos tenham vírgula ao invés de ponto). Para tanto, adicionamos duas linhas no início do código fonte:

```
import java.util.Scanner;
import java.util.Locale;
```

Para declarar uma variável Scanner e usá-la para ler um valor inteiro podemos utilizar algo semelhante ao `Programa1A.java` mostrado abaixo. Neste programa temos a declaração da variável do tipo Scanner e sua inicialização na linha 6, a definição do local como US na linha 7, para ler valores reais separados por ponto ao invés de vírgula. Fazemos então a leitura de um valor inteiro na linha 8, guardando o resultado na variável valor e mostramos este resultado na linha 9. Assim como foi utilizado teclado.nextInt(); para ler um inteiro, poderíamos ler um valor de ponto flutuante float usando teclado.nextFloat(); ou então teclado.nextDouble() para ler um valor de ponto flutuante de precisão dupla (double), ou ainda teclado.nextChar() para ler um único caractere de entrada.

Arquivo: Programa1A.java

```
1.  import java.util.Scanner;
2.  import java.util.Locale;
3.  public class HelloWorld {
4.     public static void main( String[] args ){
5.        int valor = 0;
6.        System.out.print("Digite um valor inteiro: ");
7.        Scanner teclado = new Scanner( System.in );
8.        teclado.useLocale( Locale.US );
9.        valor = teclado.nextInt();
10.       System.out.println("Valor digitado foi: " + valor);
11.    }
12. }
```

Cuidado ao ler caracteres! Assim como na linguagem C e C++, depois de lermos um valor numérico, se o usuário pular uma linha ou digitar um espaço o próximo caractere a ser lido será este pulo de linha (\n) ou o espaço.

Exercício de Fixação:

1) Escreva um programa em Java que peça ao usuário: um valor real indicando o preço de um produto e um valor inteiro indicando quantos itens daquele produto estão sendo comprados. Na sequência, calcule o valor total da compra e mostre o resultado na tela.

- Estruturas condicionais: as estruturas condicionais do Java tem uma sintaxe quase idêntica à linguagem C. Existe a condicional *if* que pode ser utilizada isoladamente ou em conjunto com um else:

```
if( condição ){
    bloco de código;
}
else{
    bloco de código;
}
```

Existe uma diferença na comparação de tipos não primitivos na qual não podemos usar simplesmente o operador de igualdade (==) para comparar se os valores de dois elementos são iguais – comparar duas strings, por exemplo. Além disso, a condição dentro dos parênteses do *if* não aceita conversão indireta de tipos como ocorre no C, onde um inteiro é convertido para um valor booleano, sendo o valor zero equivalente a *false* e qualquer outro valor equivalente a *true*. Em Java, a comparação deve ter como resultado um valor booleano.

Outra estrutura condicional que é semelhante ao C é a condicional *switch case* que é exemplificada abaixo. Em Java, a variável de escolha do *switch* pode ser um tipo primitivo, como no exemplo, mas também aceita tipos não primitivos sobre determinadas condições, sendo possível usar uma string, por exemplo.

```
int escolha = 5;
switch( escolha ){
    case 1:
        ...
        break;
    case 2:
        ...
        break;
    ...
    default:
        ...
        break;
}
```

- Estruturas de repetição: as estruturas de repetição em Java também são muito parecidas com as estruturas da linguagem C, sendo passíveis de uso as instruções *while*, *do while* e *for*. As suas sintaxes são ilustradas abaixo, sendo importante ressaltar que a sintaxe do comando *for* aceita declaração de variável dentro do seu bloco inicializador (como ocorre em diversas outras linguagens, mas não é aceito nas normas de compilação mais antigas do C), sendo que esta variável se torna válida apenas dentro do for e não pode ser usada fora deste.

```
int i = 0;
while( i < 5 ){
    ...
    i++;
}
```

```
int i = 0;
do{
    ...
    i++;
} while(i < 5);
```

```
for(int i=0; i<5; i++){
    ...
}
```

Cuidado para não sobrescrever nomes de variáveis, pois isto não é permitido!

```
int x=5;
for( int x=1; x<=3; x++ ){
    System.out.println("x=" + x);
}
```

Exercícios

1) Escreva um programa em Java que receba 3 valores que representam os lados de um triângulo (assuma que são valores válidos para formar um triângulo) e classifique-o:
- equilátero (3 lados iguais);
- isósceles (2 lados iguais);
- escaleno (nenhum lado igual);

2) Escreva um programa em Java que calcula os 10 primeiros múltiplos de um número inteiro positivo fornecido pelo usuário. Mostre o resultado no formato de tabuada.

3) Construa um programa em Java que leia um número x, calcule e escreva o valor da função f(x), dado por:

i. $0 \leq x < 5$, $f(x) = x$
ii. $5 \leq x < 10$, $f(x) = 2x+1$
iii. $x \geq 10$, $f(x) = x-3$

4) Faça um programa em Java que leia os valores do peso e da altura de pessoas, enquanto não for digitado o número -1, e conte e escreva quantas pessoas estão acima do peso. A condição [peso/(altura²) > 25] indica que a pessoa está acima do peso.

5) Construa um programa em Java que simule um caixa eletrônico. O programa deverá perguntar ao usuário o valor do saque e como resultado informar quantas notas de cada valor serão fornecidas. As notas disponíveis são as de 1, 5, 10, 20 e 50 reais. O valor mínimo de saque em um dia é de 10 reais e o máximo de 600 reais. O programa deve informar o menor número de notas possível para cada saque.

Exemplo: para sacar a quantia de 256 reais, o programa fornece cinco notas de 50, uma nota de 5 reais e uma nota de 1 real.

6) Sabendo que um número primo é aquele que é divisível somente por ele mesmo e por um, faça um programa em Java que mostre todos os números primos entre 1 e 100.