Hortolândia, 25 de setembro de 2013.

Aulas 33 e 34 – Revisão para a Prova 1

1. Introdução

A presente aula tem como objetivo revisar os principais conceitos apresentados na disciplina até o momento.

Com isso, almeja-se auxiliar os alunos na preparação para a avaliação da semana que vem (Prova 1).

Os conceitos que serão revisados são: **tipos de dados**, **paradigmas de programação**, **classes e objetos**, **abstração**, **encapsulamento**, **herança**.

2. Tipos de Dados (aulas 5 e 6)

2.1. Tipos Primitivos

Tipos primitivos são classificações que as linguagens de programação utilizam para tratar as informações. Eles facilitam a criação de algoritmos, ao mesmo tempo que possibilitam um melhor dimensionamento e utilização dos recursos de hardware/software disponíveis.

Em **Java**, existem os seguintes tipos primitivos:

- Tipos numéricos
  - ✓ Tipos inteiros
    - **byte** - Representam números inteiros de 8 bits (1 byte). Podem assumir valores entre -128 a 127.
    - **short** - Representam números inteiros de 16 bits (2 bytes). Podem assumir valores entre -32.768 até 32.767.
    - **int** - Representam números inteiros de 32 bits (4 bytes). Podem assumir valores entre -2.147.483.648 até 2.147.483.647.
    - **long** - Representam números inteiros de 64 bits (8 bytes). Podem assumir valores entre -9.223.372.036.854.775.808 até 9.223.372.036.854.775.807.
  - ✓ Tipos reais
    - **float** - Representam números reais de 32 bits com precisão simples. Podem assumir valores de ponto flutuante com formato definido pela especificação IEEE 754.
    - **double** - Representam números reais de 64 bits com precisão dupla. Assim como o float. Podem assumir valores de ponto flutuante com formato definido pela especificação IEEE 754.
- Tipos Textuais
  - ✓ **char** - Representam notação de caracteres de 16 bits (2 bytes) para formato Unicode UTF-16. Podem assumir caracteres entre '\u0000' a '\uffff' e valores numéricos entre 0 a 65535.
  - ✓ **String** – Não é um tipo primitivo de fato, e sim uma classe, mas é a forma padrão de utilização de texto em Java. O tipo String utiliza vetores do tipo char

para formar o texto, mas provê métodos diversos de controle para cada um deles

- Tipo lógico

✓ **boolean** - Representam apenas 1 bit de informação (0 ou 1). Podem assumir apenas os valores *true* e *false*.

Em Java, cada tipo primitivo possui também uma classe associada, que é responsável por prover métodos úteis para o tipo, tais como converter um texto para um valor numérico, o contrário, dentre outras funcionalidades.

As classes de cada tipo primitivo são:

- ✓ **byte** – **Byte**
- ✓ **short** – **Short**
- ✓ **int** – **Integer**
- ✓ **long** – **Long**
- ✓ **float** – **Float**
- ✓ **double** – **Double**
- ✓ **boolean** – **Boolean**
- ✓ **char** – **Character**
- ✓ **String** – **String**

Exemplos de utilização de tipos primitivos, na forma de exercícios

1) Um programa deverá possuir um contador, que será responsável pelo armazenamento da quantidade de pessoas que se inscrevem em um concurso.

Indique:

• o tipo de dados mais apropriado para o contador.

• A instrução de declaração do contador.

• A instrução de atribuição do valor “5” ao contador.

Resposta

```
public class Questao1{
public static voi main(String[] args){
int cont;
cont = 5;
System.out.println(“Valor de cont: ” + cont);
}
}
```

2) Um programa deverá possuir uma variável que armazene o salário de um funcionário. Esta variável salario deverá ser inicializada com o valor 2500.00. Ao término do programa, o mesmo deverá exibir no prompt de comandos o valor do salário do funcionário.

Resposta

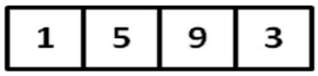
```
public class Questao2{
public static voi main(String[] args){
float salario;
```

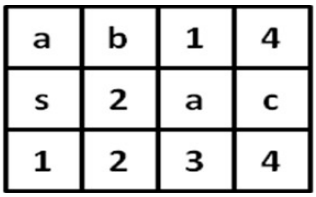
```
salario = 2500.00f;
System.out.println("Valor do salário: " + salario);
}
}
```

2.2. Vetores e Matrizes

Vetores e **matrizes** tem uma característica em comum, conseguem armazenar vários elementos de um mesmo tipo de informação. Assim, um vetor nunca terá mistura de vários tipos de dados, daí o nome "**homogêneo**".

Abaixo uma representação visual de um **vetor** e uma **matriz** de 2 dimensões:





Em Java, os vetores e matrizes são declarados e inicializados de forma parecida:

- **Vetor**
  - tipo nome[ ] = new tipo[qtdeElementos];
  - ex: int qtdesPecas[ ] = new int[50];
- **Matriz**
  - tipo nome[ ][ ] = new tipo[qtdeElementos][qtdeElementos];
  - ex: char Velha[ ][ ] = new char[3][3];
  - Matrizes podem ter 2 ou mais dimensões. Para cada dimensão adicional é colocado um par suplementar de colchetes na declaração;

Um vetor/matriz, é acessado por um único nome, seguido de um endereço numérico entre colchetes indicando a posição desejada a se acessar.

Exemplos:

int vetor[] = new int[5];

vetor[4]=5; // atribui o valor 5 à posição 4 da variável vetor

char velha[][] = new char[3][3];

velha[0][1] = 'x'; // atribui o caractere x à linha 0 e coluna 1 da matriz chamada velha

- Exemplo de visualização de todos os elementos de um vetor com laço for:

```
int vetor[] = {1,2,3,4,5};
for(int i=0;i<5;i++){
    System.out.print(vetor[i]+" ");
}
```

3. Paradigmas de Programação (aulas 9 e 10)

Na área de Linguagens de Programação, existem diversos paradigmas que tem sido utilizados pela comunidade de computação. Cada um desses paradigmas tem induzido à

construção de linguagens específicas, que exploram e evidenciam de forma mais direta e natural os conceitos correspondentes a cada paradigma (MELO et al, 2003).

Segundo Melo et al (2003), os principais paradigmas de programação são: funcional, imperativo, orientado a objetos, baseado em lógica e baseado em satisfação de restrições.

Algumas linguagens foram desenvolvidas para suportar um paradigma específico (Smalltalk e Java suportam o paradigma de orientação a objetos enquanto Haskell suportam o paradigma funcional), enquanto outras linguagens suportam múltiplos paradigmas (como o LISP, Perl, Python, C++ e Oz).

Os principais paradigmas utilizados no desenvolvimento de sistemas são: o paradigma **imperativo** e o **orientado a objetos**.

3.1. Paradigma Imperativo

A essência da programação imperativa (e também da máquina de Turing) se resume a três conceitos:

1. A descrição de estados de uma máquina abstrata por valores de um conjunto de variáveis, sendo que uma variável é um identificador de um local – um endereço físico de memória, por exemplo – que atua como repositório para determinado conjunto de valores.

2. Reconhecedores desses estados, que são expressões compostas por relações entre valores e/ou resultados de operações utilizando valores. Alguns desses valores podem ser substituídos por variáveis e nesse caso o valor presente na variável será o valor utilizado na expressão.

3. Comandos, que podem ser de dois tipos:

   a) Comandos de atribuição, que constroem valores efetuando operações a partir de valores preexistentes e atualizam os conteúdos de variáveis;

   b) Comandos de controle, que determinam qual o próximo comando a ser executado.

A execução de um programa imperativo se assemelha à simulação da operação de uma máquina física. Cada estado da máquina, uma vez reconhecido, leva a uma sequência de ações.

As ações alteram o estado da máquina, suscitando novas ações e assim por diante até que seja reconhecido um “estado final”, que indica a conlusão de uma tarefa.

Exemplos de linguagens que seguem o paradigma imperativo: **Algol**, **Pascal**, **C**.

Um exemplo de programa escrito em Linguagem C está representado na Listagem 1 a seguir.

```
// Exemplo de programa em C

#include <stdio.h>

// Arquivo de cabeçalho (header)

void main() {

int contador;

// declarações simples

float PrecoDoQuilo;
```

```
double TaxaDeCambio;

char LetraDigitada;

int IdadeManoel, IdadeJoao, IdadeMaria;

// Pode-se colocar mais de uma

// variável na mesma linha

double TaxaDoDolar,

TaxaDoMarco,

TaxaDoPeso,

TaxaDoFranco;

.......

}
```

Listagem 1: Exemplo de programa escrito em Linguagem C (trecho de código).

3.2. Paradigma Orientado a Objetos

O **paradigma da orientação a objetos** tem por princípio a solução de problemas pela cooperação de vários elementos, da mesma forma que usamos a prestação de serviço de outras pessoas para resolver vários de nossos problemas.

As linguagens de programação orientadas a objetos permitem a criação de objetos, os quais são entidades computacionalmente ativas que guardam um conjunto de dados (os atributos), e os serviços (os métodos) que ele pode prover.

Exemplos de linguagens que seguem o paradigma orientado a objetos: **Smalltalk**, **Java**, **C++**, **C#**.

3.2.1. Classes e Objetos

O **processo de abstração**, quando aplicado na programação orientada a objetos, leva o desenvolvedor primeiramente a representar um determinado problema computacional a ser resolvido na forma de um modelo de **classes de objetos**.

Este modelo de classes obtido é então transformado, por meio da atividade de programação, em estruturas de **classes**, escritas de acordo com as regras de uma determinada linguagem de programação, escolhida pelo desenvolvedor.

Tais classes são então compiladas (ou seja, transformadas em código executável) e então executadas pelo computador.

As classes, uma vez instanciadas pelo computador (ou seja, espaço de memória foi alocado para as mesmas), são chamadas de **objetos**.

A representação dos objetos é semelhante à das classes, sendo que a única diferença é que os objetos possuem valores para seus atributos e podem ter seus métodos executados por outros objetos.

A Figura 2 ilustra as representações possíveis de uma classe em UML.

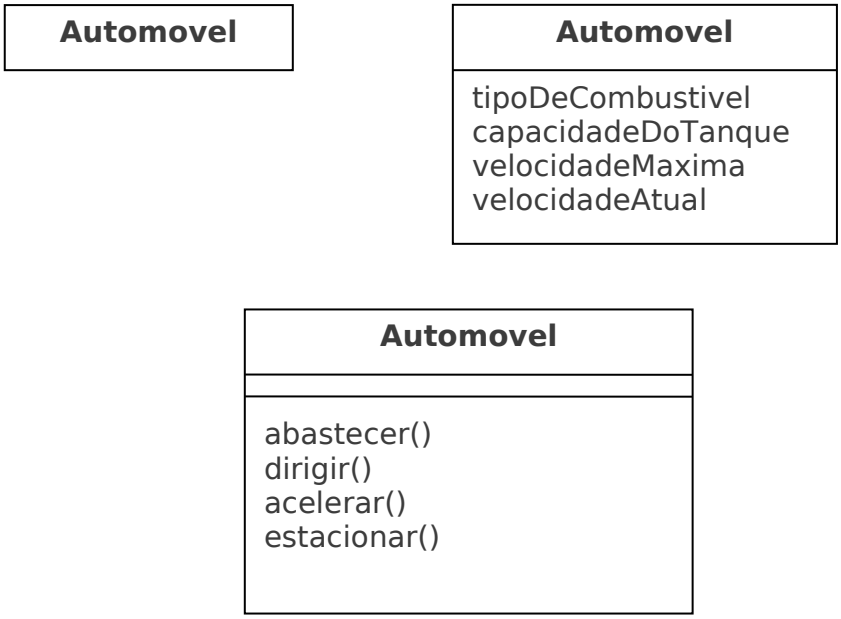


Figura 2: Representações de uma Classe em UML (Linguagem Unificada de Modelagem).

4. Abstração (aulas 17 e 18)

A **abstração** é o processo utilizado na análise de uma situação real, através do qual se observa uma realidade, tendo-se por objetivo a determinação dos aspectos e fenômenos considerados essenciais, excluindo-se todos os aspectos considerados irrelevantes ou secundários.

O processo de abstração consiste basicamente em identificar, para um determinado objeto ou problema do mundo real, os seus principais aspectos e características, bem como quais são os seus principais comportamentos.

No caso particular da **programação orientada a objetos**, tais abstrações do mundo real são representadas na forma de **classes de objetos**. Uma **classe** de objetos é, portanto, um modelo de determinados objetos do mundo real, os quais possuem características em comum.

No modelo de classes, cada classe é representada por um nome de classe, um conjunto de atributos e um conjunto de métodos (operações) que um objeto desta classe pode executar.

Como exemplos de classes, podem-se citar: a classe Lampada (apresentada na seção anterior), uma classe Automovel, uma classe Estudante, uma classe Matricula, uma classe Funcionario, e assim por diante.

5. Encapsulamento (aulas 25 e 26)

A orientação a objetos **encapsula** dados (os **atributos**) e funções (**métodos**, comportamentos) em objetos. Os dados e as funções de um objeto estão intimamente vilculados (DEITEL et al, 2001).

Os objetos têm a propriedade de **ocultação de informações**. Isso significa que, embora os objetos possam saber como se comunicar uns com os outros através de interfaces bem definidas - os **métodos públicos** - normalmente não é permitido aos objetos saber como outros objetos são implementados.

Ou seja, os detalhes de implementação ficam **escondidos** dentro dos próprios objetos.

Isso é semelhante à situação em que se está dirigindo um veículo. Não é obrigatoriamente necessário saber todos os detalhes da construção mecânica e funcionamento interno do mesmo para se conduzí-lo.

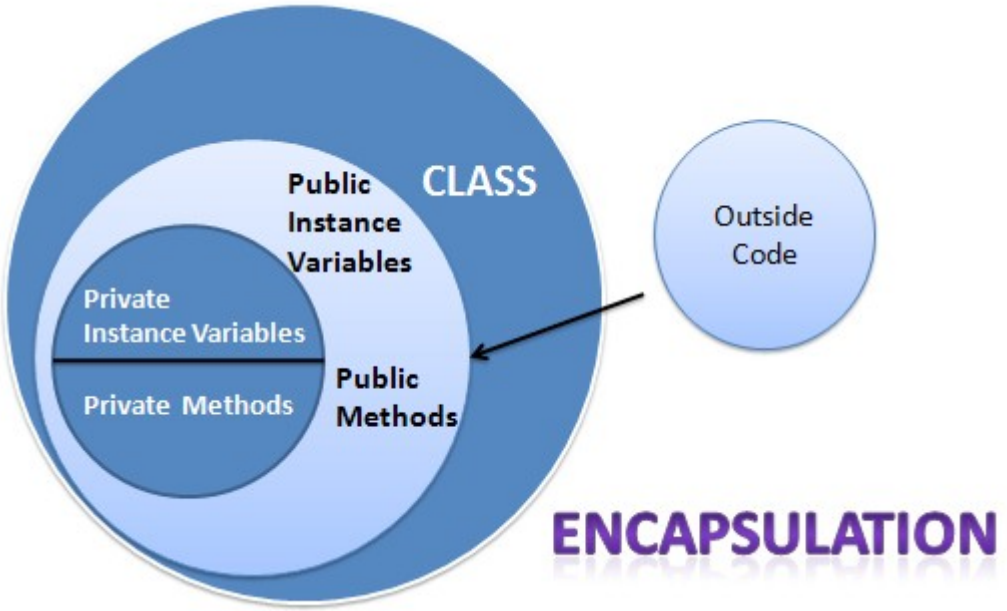


Figura 3: Encapsulamento dos atributos e métodos de uma Classe.

A Figura 3 ilustra a característica de encapsulamento de uma classe. A **classe** oferece ao "mundo externo" (código externo) **métodos e variáveis de instância públicos**. Porém ela **oculta métodos e variáveis de instância**, atribuindo-lhes o modificador de acesso "***private***".

6. Herança (aulas 29 e 30)

A **herança** é uma forma de reutilização de software em que novas classes são criadas a partir das classes existentes, absorvendo seus atributos e comportamentos e adicionando novos recursos que as classes exigem.

A herança tira proveito dos relacionamentos entre classes, nos quais os objetos de uma certa classe - como uma classe de veículos - têm as mesmas características (DEITEL et al., 2001).

As classes de objetos recém criadas derivam-se ao absorver as características das classes existentes e ao adicionar suas próprias características.

Como exemplo, um objeto de uma classe "Conversível" tem as mesmas características de um "Automóvel", porém o teto sobe e abaixa. Mesmo dentro de um determinado modelo de conversível, pode-se ter outros modelos mais específicos.

A Figura 4 ilustra o relacionamento de herança existente entre um veículo conversível "Roadster" e um conversível "comum".

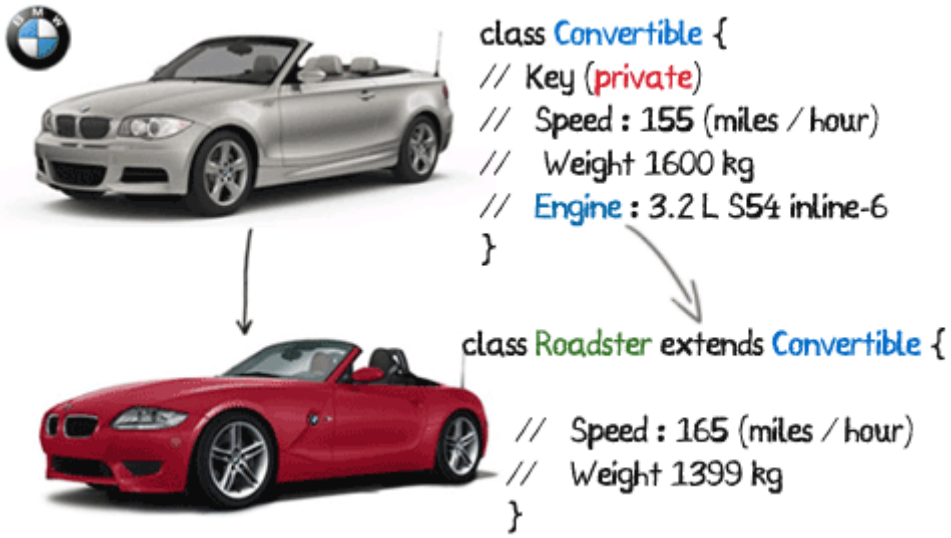


Figura 4: Relacionamento de herança entre classes de automóveis conversíveis.
(www.9lessons.info)

Outro exemplo é a hierarquia existente entre as classes **Animal**, **Cachorro** e **Gato**. Tanto o cachorro quanto o gato são animais, portanto, podem ser modelados como classes que estendem a classe Animal, herdam seus atributos e métodos.

A Figura 5 ilustra esse relacionamento.

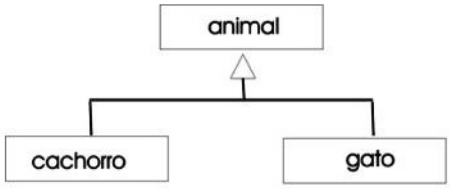


Figura 5: Relacionamento de herança entre as classes Animal, Cachorro e Gato.

7. Referências Bibliográficas

BORATTI, Isaias Camilo. **Programação Orientada a Objetos usando Delphi.** Quarta Edição. Editora Visual Books. Florianópolis, 2007.

DEITEL, H.M., DEITEL, P.J. **Java – Como programar**. Terceira edição. Porto Alegre: Bookman Editora, 2001.

PALMEIRA, T.V.V. **Abstração, Encapsulamento e Herança: Pilares da POO em Java**. Disponível em: http://www.devmedia.com.br/abstracao-encapsulamento-e-heranca-pilares-da-poo-em-java/26366. Última consulta: 11/09/2013.

SANTOS, Rafael. **Introdução à Programação orientada a objetos usando Java**. 8ª Reimpressão. Rio de Janeiro: Campus - Elsevier, 2003.