



Disciplina Laboratório de Computação II	Curso Engenharia de Software	Turno Noite	Período 2º
Professor Felipe Cunha (felipebh@gmail.com)			

Aula 02 – Introdução à sintaxe Java

A Programação Orientada para Objetos (POO) é o método de implementação onde os programas são organizados como coleções de objetos cooperativos, cada objeto representando uma instância de uma classe.

A classe é um mecanismo para definição de um Tipo Abstrato de Dados (TAD), e descreve características genéricas e comportamento de uma série de objetos semelhantes. Classes consistem na descrição de atributos e métodos para um conjunto de objetos.

Todos os objetos são “instanciados” ou criados de uma classe. Os Objetos são entidades (ou itens) unicamente identificados. Eles representam uma instância concreta e dinâmica de uma classe; possuem identidade, dados e comportamento próprios da classe. Pode-se dizer, então, que um objeto é “uma variável do tipo de dados definido pela classe”. A relação entre objeto e classe é a mesma entre variável e tipo (alocação de memória, atribuição etc). Por exemplo:

`int i` - `int` é o tipo e `i` é a variável

`Classe Objeto` - `Classe` é o tipo e `Objeto` é a variável

Os atributos de uma classe são também chamados de campos ou membros. Os métodos de uma classe também são chamados de funções membro. Os membros públicos formam a interface da classe (visão externa). Membros públicos são os campos e funções membro que são visíveis externamente à classe.

Exemplo de classe e objeto:

Uma classe `Conta` pode ser usada para instanciar contas bancárias de indivíduos específicos. Cada conta bancária contém diferentes saldos e saldos mínimos, mas as ações executadas sobre qualquer conta bancária são as mesmas.

Deste modo, a classe **`ContaBancaria`** seria composta pelos atributos:

- Saldo : do tipo `float`
- SaldoMinimo : do tipo `float`

e pelos métodos (ou funções membro):

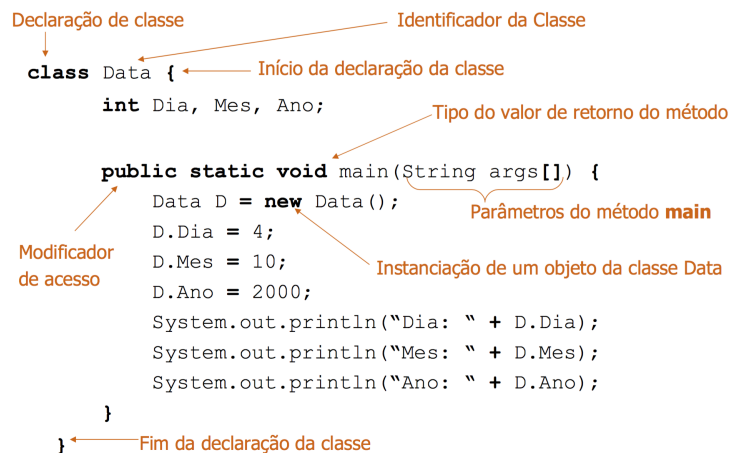
- `Sacar(float quantia)`
- `Depositar(float quantia)`
- `AlterarSaldoMinimo(float novoSaldo)`
- `ConsultarSaldo()`

Duas instâncias dessa classe seriam os objetos `CJoao` e `CPedro`, que representariam as contas bancárias de João e de Pedro. Cada objeto possui seu próprio estado, que constitui um saldo e um saldo mínimo. As mensagens que podem ser enviadas para as instâncias `CJoao` e `CPedro` da classe **`ContaBancaria`** são:

- Sacar dinheiro desta conta
- Depositar dinheiro nesta conta

- Consultar o saldo da conta
- Alterar o saldo mínimo da conta

A figura abaixo exibe a estrutura de uma classe java;



Características do Java

O Java é uma Linguagem orientada para objetos, desenvolvida pela Sun Microsystems, em 1991. O Código fonte e objeto (código compilado) são portáteis para diversas arquiteturas e sistemas operacionais. Ou seja, um aplicativo desenvolvido em Java deverá rodar em qualquer sistema operacional sem precisar ser recompilado. Esta portabilidade é alcançada porque o programa Java é compilado para uma linguagem intermediária, chamada Bytecode. Este código precisa de uma Máquina Virtual Java (JVM) para ser executado. A JVM fará, então, a tradução do bytecode para o sistema operacional em que estiver instalada. O Java está disponível no site <http://java.sun.com>.

Estrutura de uma aplicação Java

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.

Lançando programas Java

Em aplicações, tudo começa pelo método `main`.

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

`args[]` é correspondente ao `argv[]` do C, exceto que `args[0]` é equivalente ao `argv[1]`. `main` não retorna um valor, apesar de que a JVM pode capturar códigos de saída: `System.exit(0);` `main` é método da classe principal de um Aplicativo Java.

```
class Exemplo1 {

    public static void main (String args[ ]) {
        System.out.println("Alo_Mundo!");
        System.exit(0);
    }
}
```

Introdução à sintaxe Java

Comentários:

```
/* Comentário padrão
 * Usado para comentar um bloco de linhas. Semelhante ao C.
 */

// Comentário se estende até o final da linha. Semelhante ao C.

/** Comentário da ferramenta javadoc.
 * Possui tags especiais para geração automática de documentação.
 * Permite comentar de maneira formal os métodos, atributos e
 * parâmetros de métodos de classe, além de autor, versão etc...
 */
```

DICA: Leia a documentação do javadoc e defina um padrão de comentário para você ou sua empresa. Ele pode ajudar sobremaneira a criar documentações de software para trabalho em equipe (como help).

As únicas linhas de programação que podem estar fora de uma classe são a diretiva **package** e as diretivas **import**, além de comentários. Todo o programa deve estar dentro de uma classe.

Declaração de Variáveis

Em Java é possível se declarar variáveis de tipos primitivos ou objetos de classes. Declaração de variáveis ou objetos:

```
tipo nomeVar1, nomeVar2, ..., nomeVarN; // semelhante ao C
```

Execute o exemplo abaixo para entender declaração de variáveis:

```
class Exemplo2 {

    public static void main (String args[ ]) {

        // Declaração de variáveis
        String name = "Aluno";

        int matric = 220044;

        System.out.print("Alo_");
        System.out.println(name);
        System.out.println("Matricula:_ " + matric);
        System.exit(0);
    }
}
```

Tipos Primitivos

Tipo	Tamanho	Precisão
byte	8 bits	-128 a 127
short	16 bits	-32.768 a 32.768
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
float	32 bits	IEEE 754
double	64 bits	IEEE 754
char	16 bits	UNICODE
boolean	–	true, false

Valores de Constantes

- Atribuição através do símbolo =
- Constantes inteiras: 4, 4L, -4, 0777, 0xFF
- Constantes reais: 1.999, 2.5F, 10e45, .36E-2
- Constantes booleanas: true, false
- Constantes caractere: 'a', '\n', '\'', '\xFF', '\uAF00
- Constantes string: "casa", "", "fim\n", "Java\u2122"

Java é fortemente tipada e não permite coerção implícita como no C, quando há perda de dados.

```
double X1 = 0;           // Correto
double X2 = 0.0;         // Correto
float Y1 = 0.0;          // Errado
float Y2 = 0.0F;         // Correto
int A = (Y1 == Y2);      // Errado
boolean A = (Y1 == Y2);  // Correto

// Para fazer conversão explícita (type casting)

int A = (int) X1;        // converte double para int
```

Execute os códigos abaixo para entender a sintaxe do Java:

```
// MostraVar.java
class MostraVar
{
    public static void main(String args[])
    {
        // Declara uma variável chamada var.
        int var;
        // Atribui o valor 25 à variável var.
        var = 25;
        // Exibe o valor de var.
        System.out.println("Valor_de_var: " + var);

        // Multiplica var por 2.
        var = var * 2;
```

```
        // Mostra o novo valor de var.
        System.out.print("var_multiplicada_por_2:");
        System.out.println(var);
    } // Fim de main()
```

Este código exibe a sintaxe do condicional **if**:

```
// MostraIf.java
class MostraIf {
    public static void main(String args[]) {
        int i, j;
        i = 50;
        j = 100;

        if (i < j)
            System.out.println("i é 'menor' que j.");

        i = i * 2;

        if (i == j)
            System.out.println("Agora i é 'igual' a j.");

        i = i * 2;

        if (i > j)
            System.out.println("Agora i é 'maior' que j.");

        // Esta linha não será exibida.
        if (i == j)
            System.out.println("Isso não será exibido.");
    } // Fim de main()
} // Fim da classe MostraIf.
```

Este código exibe a sintaxe do condicional **switch - case**:

```
public class MostraSwitch {
    public static void main(String[] args) {
        int mes = 8;
        switch (mes) {
            case 1: System.out.println("Jan"); break;
            case 2: System.out.println("Fev"); break;
            case 3: System.out.println("Mar"); break;
            case 4: System.out.println("Abr"); break;
            case 5: System.out.println("Maio"); break;
            case 6: System.out.println("Jun"); break;
            case 7: System.out.println("Jul"); break;
            case 8: System.out.println("Ago"); break;
            case 9: System.out.println("Set"); break;
            case 10: System.out.println("Out"); break;
            case 11: System.out.println("Nov"); break;
            case 12: System.out.println("Dez"); break;
            default: break;
        }
    }
}
```

```

    }
}

```

Este código exibe a sintaxe da estrutura de repetição **for**:

```

// TesteFor
class TesteFor {
    public static void main(String args[]) {
        int i;
        for (i = 0; i < 10; i = i + 1)
            System.out.println("Valor_de_i:_ " + i);
    } // Fim de main()
} // Fim da classe TesteFor.

```

Desvio de fluxo

- As estruturas “break” e “continue” alteram o fluxo de controle.
 - break ocasiona saída imediata da estrutura de repetição ou seleção e execução continua na primeira linha após a estrutura.
 - continue pula o corpo restante de uma estrutura de repetição e execução continua na próxima iteração da estrutura de repetição.
 - goto é uma palavra reservada mas não é utilizada (causa problemas de controle de escopo e encapsulamento de dados).
 - “break” e “continue” podem ser rotuladas para sair de mais de uma estrutura aninhada e execução continua na primeira linha após a estrutura composta rotulada, ou seja, estrutura entre chaves precedida por um rótulo.
-

```

// Desvio de fluxo – break
import javax.swing.JOptionPane;

public class TesteBreak {
    public static void main(String args[]) {
        String output = "";
        int count;

        for (count = 1; count <= 10; count++) {
            if (count == 5)
                break; // interrompe loop se count = 5
            output += count + "_";
        }

        output += "\nInterrompeu_loop_com_count=_ " + count;
        JOptionPane.showMessageDialog(null, output);
        System.exit(0);
    }
}

```

```
// Desvio de fluxo - continue
import javax.swing.JOptionPane;

public class TesteContinue {
    public static void main(String args[]) {
        String output = "";
        int count;

        for (count = 1; count <= 10; count++) {
            if (count == 5)
                continue; // salta o resto do cod do loop se count = 5
            output += count + "_";
        }

        output += "\Saltou_a_impressao_do_5.";

        JOptionPane.showMessageDialog(null, output);
        System.exit(0);
    }
}
```

As estruturas **while** e **do – while** seguem exatamente a mesma sintaxe do C.

Exercicio para entregar

1 - Implemente a classe conta descrita anteriormente. Crie os atributos citados e mais um identificador para o número da conta corrente. Crie um método **main** que constroi dois objetos da classe conta utilizando a cláusula **new** como descrito na classe Data. Teste as funções sacar e depositar, imprimindo os resultados na tela.

2 - Crie um vetor de objetos conta com 10 posições. Cada conta terá um saldo inicial, um identificador (número) e o nome do cliente. Crie um menu de opções no qual o usuário pode alterar os dados da conta, fazer depósito e saques.

O aluno deverá ser entregue no SGA apenas o arquivo com a extensão .java. Não deverá ser entregue nenhum outro arquivo. Lembre-se de colocar o cabeçalho no arquivo como mostrado na aula anterior.