LINGUAGEM DE PROGRAMAÇÃO JAVA

Cefet – Maracanã -RJ

BCC/TSI

Prof. Gustavo Guedes

E-mail: gustavo.guedes@cefet-rj.br

Lendo a entrada do teclado

```java
import java.util.Scanner;

public class LeituraTeclado {
    public static void main(String[] args) {
        System.out.println("Insira uma linha: ");
        Scanner scanner = new Scanner(System.in);
        String lendoLinha = scanner.nextLine();
        System.out.println("A linha entrada foi: "+lendoLinha);

        System.out.println("Insira uma idade: ");
        int lendoInteiro = scanner.nextInt();
        System.out.println("A idade é: "+lendoInteiro);

        scanner.close();
    }
}
```

Arrays

- Os arrays geralmente são usados para agrupar objetos do mesmo tipo. Os arrays permitem que você se refira ao grupo de objetos usando um nome comum.
- Você pode declarar arrays de qualquer tipo, primitivo ou de classe:

- char s [ ];
- Point p [ ];

- No java, um array é um objeto, mesmo quando formado por tipos primitivos, e tal como acontece com outros tipos de classe, a declaração não cria o objeto propriamente dito.

Arrays

- Em vezdisso, a declaração de um array cria uma referência que você pode usar para se referir a um array. A memória real usada pelos elementos do array é alocada dinamicamente por uma declaração new ou pelo inicializador de um array.
- Você também pode declarar arrays como:

- char [ ] p;
- Point [ ] p;

- A diferença é que quando utilizamos o colchete à esquerda, os mesmos se aplicam a todas as variáveis que estão à direita dos colchetes. No outro caso, não.

Arrays

- Você pode criar arrays, assim como todos os objetos, usando a palavra reservada new. Por exemplo, para criar um array de um tipo primitivo (int):

- int[] s = new int[26];

- Cria um array de 26 posições de int. Uma vez criados, os elementos do array são inicializados com o valor default (0, nesse caso) para int. Um array pode ser preenchido, como por exemplo:

- s[0] = 15;
- s[1] = 3;

Arrays

- Em Java, o array sempre começa com 0 (zero) e deve ser mantido dentro do intervalo legal - maior ou igual a 0, e menor que a extensão do array.
- Qualquer tentativa de acessar um elemento do array que está fora desses limites causa uma exceção de runtime.
- Para criar array de objetos, usa-se a mesma sintaxe:

- String [] s = new String[10];

- para criar o objeto String, fazemos da seguinte forma:

- s[0] = new String("Cabana");

Arrays

- O Java permite um atalho para criar arrays com valores iniciais:

- String names [] = {“Bruna”, “Renata”, “Fernanda”};

- Esse código é equivalente a:

- String names[];
- names = new String[3];
- names[0] = “Bruna”;
- names[1] = “Renata”;
- names[2] = “Fernanda”;

Arrays

- Todos os índices de arrays começam com 0 (zero). O número de elementos de um array é armazenado como parte do objeto array no atributo length. Se acontecer um acesso fora dos limites, isso provocará uma exceção runtime.
- Use o atributo length para fazer a iteração em um array da seguinte maneira:

- int array[ ] = new int [10];
- for (int i=0; i < array.length; i++){

  System.out.println(array[i]);
- }
- O uso do atributo length do array facilita a manutenção do programa, porque você não precisa saber o número de elementos presentes no array no momento da compilação.

Redimensionamento de arrays

- Depois de criado, um array não pode ser redimensionado. Entretanto, você pode usar a mesma variável de referência para se referir a um array inteiramente novo:

- int myArray[ ] = new int[6];
- myArray = new int[10];

- Nesse caso, o primeiro array estará efetivamente perdido, a menos que uma outra referência a ele seja mantida em algum lugar.

Redimensionamento de arrays

- O Java fornece um método especial na classe System.arraycopy(), para copiar arrays. Por exemplo, você pode usar o método arraycopy da seguinte maneira:

```
1   //original array
2   int[] myArray = { 1, 2, 3, 4, 5, 6 };
3
4   // new larger array
5   int[] hold = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
6
7   // copy all of the myArray array to the hold
8   // array, starting with the 0th index
9   System.arraycopy(myArray, 0, hold, 0, myArray.length);
```

- Ao lidar com arrays de objetos, o método System.arraycopy() copia referências, não objetos. Os objetos propriamente ditos não são alterados.

FOREACH

- for (int i:idades) {
- System.out.println(i);
- }

- //resolve o problema do .lenght

Exercício

- Crie uma classe chamada Mochila
  - Atributos: cor, anoFabricacao

- Crie uma classe com o “main”. Crie um array de Mochilas e popule esse array com 4 mochilas cujos dados são provenientes do console.

- Percorra esse array e calcule a média dos anos de fabricação.

MODIFICADORES

```
public class Conta {
    String nomeCliente;
    double saldo;
    public void sacar(double valor) {
        if ((saldo - valor ) > 0) {
            saldo = saldo - valor;
            System.out.println("Saque realizado. Saldo = "+saldo);
        } else {
            System.out.println("Saque NAO realizado. Saldo ("+saldo+") menor que valor de saque: "+valor);
        }
    }
    public void depositar(double valor) {
        saldo = saldo + valor;
        System.out.println("Saldo = "+saldo);
    }
    public void transferir(Conta destino, double valor) {
        sacar(valor);
        destino.depositar(valor);
    }
}
```

```
public class CaixaAutomatico {
    public static void main(String[] args) {
        Conta x = new Conta();
        x.nomeCliente = "Ana";
        x.saldo = 15;
        System.out.println("Quanto quer sacar?");
        Scanner s = new Scanner(System.in);
        double valor = s.nextDouble();
        x.saldo = x.saldo - valor;
        System.out.println("Liberando "+valor+ " reais...");
        System.out.println("Saldo: " + x.saldo);
        s.close();
    }
}
```

MODIFICADORES

```
public class Conta {
        private double saldo;
        private double limite;
```

- É muito comum, e faz todo sentido, que os atributos da classe Conta sejam private e que quase todos seus métodos sejam public. Desta forma, toda conversa de um objeto com outro é feita por troca de mensagens, isto é, acessando seus métodos. Algo muito mais educado que mexer diretamente em um atributo que não é seu!

MODIFICADORES

- Melhor ainda! O dia em que precisarmos mudar como é realizado um saque na nossa classe Conta, adivinhe onde precisaríamos modificar? Apenas no método saca, o que faz pleno sentido. Como exemplo, imagine cobrar CPMF de cada saque: basta você modificar ali, e nenhum outro código, fora a classe Conta, precisará ser modificado. Mais: as classes que usam esse método nem precisam ficar sabendo de tal modificação! Ganhamos muito em esconder o funcionamento do nosso método na hora de dar manutenção e fazer modificações.

- Isso é chamado de encapsular: esconder todos os membros da classe, além de esconder como funcionam os métodos. Encapsular é fundamental para que seu sistema seja suscetível a mudanças: não precisaremos mudar uma regra em vários lugares. Precisaremos mudar, apenas, em um lugar, já que essa regra está encapsulada.

Encapsulamento

- É sempre bom programar pensando na interface da sua classe, como seus usuários a estarão utilizando, e não somente em como ela irá funcionar.

- A implementação em si, o conteúdo dos métodos, não têm tanta importância para o usuário dessa classe, uma vez que ele só precisa saber o que cada método pretende fazer, e não como ele faz, pois isto pode mudar com o tempo.

- O conjunto de métodos públicos de uma classe é também chamado de interface da classe, pois esta é a única maneira a qual você se comunica com objetos dessa classe.

Encapsulamento

- Oculta detalhes de implementação de uma classe.
- Força o usuário a usar uma interface para acessar os dados.
- Facilita a manutenção do código.

Encapsulamento

- Oculta detalhes de implementação de uma classe.
- Força o usuário a usar uma interface para acessar os dados.
- Facilita a manutenção do código.

Métodos get e set

```
public int getNumero() {
        return numero;
}
```

Java Beans

- Quando criamos uma classe com todos os atributos privados, métodos get e set para seus atributos e um construtor sem argumetos, na verdade estamos criando um Java Bean.

Construtores

- Até agora, as nossas classes não possuíam nenhum construtor.
- Então como é que era possível dar new, se todo new chama um construtor obrigatoriamente?
- Quando você não declara nenhum construtor na sua classe, o Java cria um para você. Esse construtor é o construtor default, ele não recebe nenhum argumento e o corpo dele é vazio.
- A partir do momento que você declara um construtor, o
- construtor default não é mais fornecido.

Construtores

- Quando usamos a palavra chave new, estamos construindo um objeto. Sempre quando o new é chamado, ele executa o construtor da classe. O construtor da classe é um bloco declarado com o mesmo nome que a classe:

```
class Conta {
    int numero;
    Cliente titular;
    double saldo;
    double limite;

    // construtor
    Conta() {
        System.out.println("Construindo uma conta.");
    }

    // ..
}
```

- O que aparecerá quando fizermos o que está abaixo?
- A mensagem “Construindo uma conta.” aparecerá. Sempre que o objeto é criado, o construtor é chamado.

Declaração de Construtores

- Um construtor é um conjunto de instruções criadas para inicializar uma instância. Podem ser passados parâmetros para o construtor, da mesma maneira que se faz para um método. A declaração básica tem o seguinte formato:
- [modificadores] <nome_da_classe> ([<lista_de_argumentos>]){
- [instruções (corpo)]
- }
- 1 public class Dog {
- 2 private String nome;
- 3 public Dog() {
- 5 }
- 6 public Dog(String varNome) {
- 7 nome = varNome;
- 8 }
- 9 public String getNome(){
- 10 return nome;
- 11 }
- 12 }

O CONSTRUTOR DEFAULT (PADRÃO)

- Existe pelo menos um construtor em cada classe.
- Se o programador não definir um construtor, o construtor default será incluído automaticamente.
  - O construtor default não leva nenhum argumento
  - O corpo do construtor default é vazio.
- Permite que você crie instâncias de um objeto com new XXX() sem a necessidade de escrever um construtor.

- **Notação: Se você incluir uma declaração de construtor em uma classe que anteriormente não tinha nenhum construtor explícito, você perde o construtor default. A partir desse ponto, a menos que o construtor que você escreveu não leve argumentos, as chamadas para new XXX() causarão erros de compilação.**

Métodos static e campos static

- Conforme sabemos,cada classe fornece métodos que realizam tarefas comuns sobre objetos da classe. Embora a maioria dos métodos seja executada em resposta a chamadas de métodos em objetos específicos, esse nem sempre é o caso. Às vezes um método realiza uma tarefa que não depende do conteúdo de um objeto. Esse método se aplica a classe em que é declarado como um todo e é conhecido como método static ou método da classe. Você pode chamar qualquer método static especificando o nome da classe em que o método é declarado, seguindo por um ponto (.) e pelo nome do método, como em:
- NomeDaClasse.nomeDoMetodo(argumentos)

- Por exemplo, vamos utilizar o método sqrt (raiz quadrada) da classe Math:
- Math.sqrt(900.0);

- Para gerar a saída:
- System.out.println(Math.sqrt(900.0));

Métodos static e campos static

- O método main é declarado static porque ao executar a JVM, ela tenta invocar o método main da classe que você especifica, sendo que nenhum objeto dessa classe foi instanciado.

Exercício

- Crie a classe Utils
- Crie 3 métodos estáticos chamados calculaMedia. O primeiro irá calcular a média entre dois argumentos do tipo double, o segundo entre três argumentos e o terceiro entre quatro argumentos do tipo double.
- Crie uma classe principal que utilize os três métodos criados.

Operadores lógicos

- Os operadores lógicos retornam um resultado booleano. O valor 0 não é interpretado automaticamente como falso e os valores diferentes de zero não são automaticamente interpretados como verdadeiros.
- Os operadores booleanos são:

```
! - NOT      & - AND
| - OR       ^ - XOR
```

- Os operadores booleanos de short-circuit são:

```
&& - AND    || - OR
```

- Podemos utilizá-los da seguinte forma:

- if (a || b) {}

Concatenação com Strings

- O operador + executa uma concatenação de objetos String, produzindo uma nova String.
- String x = “casa”;
- String y = “ bonita”;
- String z = x + y + “ legal”;
- Z -> “casa bonita legal”
- Se algum argumento do operador + for um objeto String, o outro
- argumento é convertido em um objeto String. Todos os objetos podem ser convertidos automaticamente para um objeto String, embora o resultado possa ficar um pouco inteligível. O objeto que não é um objeto String é convertido em uma String equivalente usando a função toString().
- Experimente criar: Animal a = new Animal();
- System.out.println(a); //será chamado implicitamente o toString() de a.