

Programação Orientada a Objetos

Prof. Paulo André Castro
pauloac@ita.br
www.comp.ita.br/~pauloac
ITA – Stefanini

Paulo André Castro

POO

ITA - Stefanini

1

Planejamento

- Aula 1
 - Introdução
 - Conceitos Básicos
 - Classe, Objeto, Método, Herança, interfaces, polimorfismo, Encapsulamento
 - Introdução a linguagem Java
- Aula 2
 - Modelagem de Programas Orientada a Objetos
 - Introdução a Padrões de Projeto (Design Patterns)
 - Introdução a linguagem Java
- Aula 3
 - Introdução a Ambientes Integrados de Desenvolvimento
 - Desenvolvimento de Programas Básicos
 - Manipulação de E/S em Java
 - Tipos genéricos

Paulo André Castro

POO

ITA - Stefanini

2

Planejamento

- Aula 4
 - Programando Interfaces Gráficas com Java I
- Aula 5
 - Programando Interfaces Gráficas com Java - II
- Aula 6
 - Programação concorrente (Threads)
 - Conexão com outros programas em Rede
- Aula 7
 - Conectividade com Banco de Dados (JDBC)
 - Padrão de projeto para acesso a Dados: DAO Design Pattern
- Aula 8
 - XML
 - Introdução a Web Services

Paulo André Castro

POO

ITA - Stefanini

3

Referências

- "Conceitos Essenciais de Computação com Java". C. Horstmann. Ed. Bookman.
- "Core Java – Volume I", Cay Horstmann, Gary Cornell. Ed. SunSoft Press.
- "Core Java – Volume II – Advanced Features", Cay Horstmann, Gary Cornell. Ed. SunSoft Press.
- "Core web Programming", Marty Hall, Larry Brown, 2003.
- "Java – Como Programar", Deitel & Deitel, 6a. Edição, Ed. Prentice Hall. 2005.
- E Internet
 - java.sun.com, www.theserverside.com, etc.

Paulo André Castro

POO

ITA - Stefanini

4

Sumário de Hoje

- **Introdução**
- **Conceitos Básicos**
 - Nomenclatura básica em OO
 - Variáveis e Instâncias
 - Métodos
 - Construtores
 - Herança e Polimorfismo
- **Introdução a linguagem Java**

Paulo André Castro

POO

ITA - Stefanini

5

Introdução

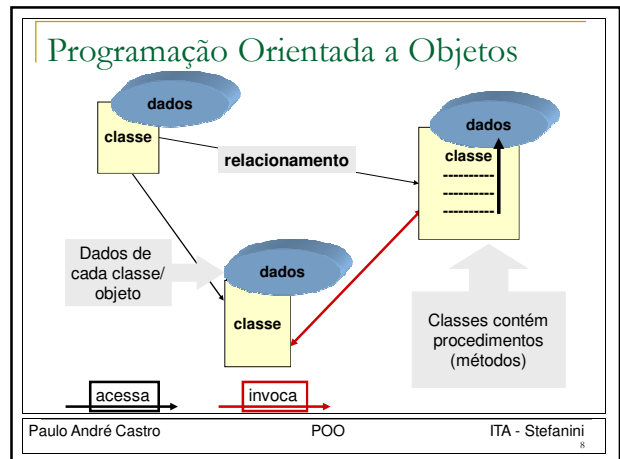
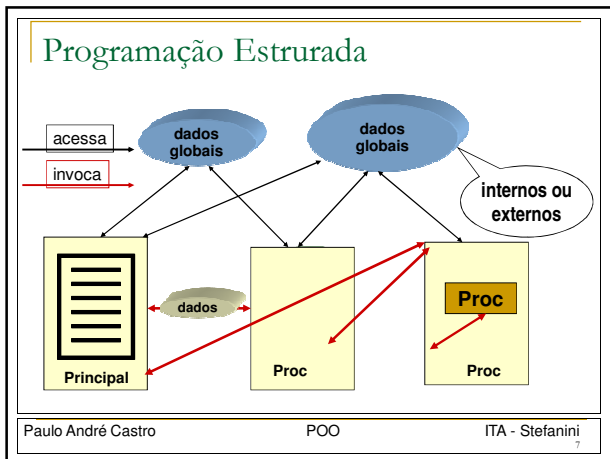
- **Programação Estruturada x Programação Orientada a Objetos**
 - Modelagem com base no conceito de módulo ou sub-programa
 - Modelagem com base no conceito de classe e seus relacionamentos
- **Linguagens Orientadas a Objetos**
 - Simula, SmallTalk
 - C++, C#, (VB?), etc.
 - Java

Paulo André Castro

POO

ITA - Stefanini

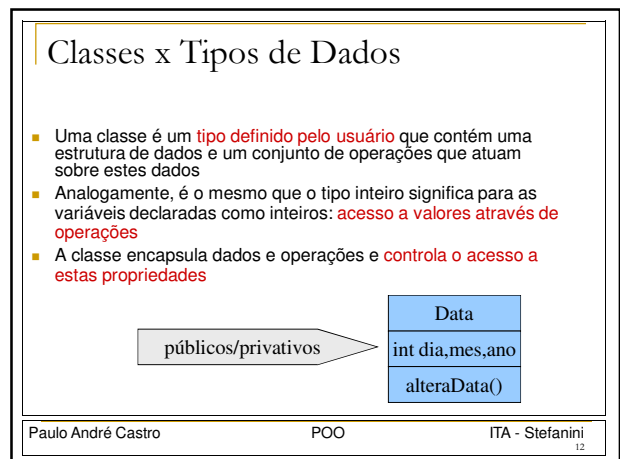
6



- ### Sumário de Hoje
- Introdução
 - **Conceitos Básicos**
 - Nomenclatura básica em OO
 - Variáveis e Instâncias
 - Métodos
 - Encapsulamento
 - Herança e Polimorfismo
 - Introdução a linguagem Java
- Paulo André Castro POO ITA - Stefanini 9

- ### Conceitos básicos de OO
- **Classe:** um categoria de entidades (“coisas”)
 - Corresponde a um tipo, ou coleção, ou conjunto de entidades afins
 - **Objeto:** Uma entidade com existência física que pertence a um determinado conjunto de entidades afins (classe)
- Paulo André Castro POO ITA - Stefanini 10

- ### Exemplos de Classe e Objeto
- **Classes:**
 - Carro, Avião, Pessoa
 - **Objetos:**
 - Carro: Porsche 910 Placa XXXX
 - Avião: Boeing 737-300 Prefixo: PY-XXX
 - Pessoa: José da Silva CPF: XXXXXXXX
- Paulo André Castro POO ITA - Stefanini 11



UML

- Unified Modeling Language
 - Linguagem Unificada de Modelagem
 - Linguagem Visual de Modelagem Orientada a Objetos
 - Referência: The Unified Modeling Language User Guide; G. Booch, J. Rumbaugh, I. Jacobson. ACM Press. 2000

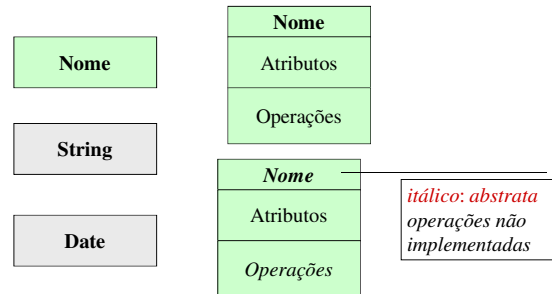
Paulo André Castro

POO

ITA - Stefanini

13

UML: notações de classes



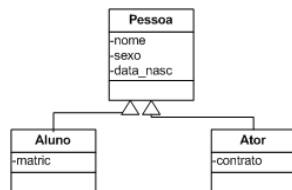
Paulo André Castro

POO

ITA - Stefanini

14

Exemplo em UML - Herança



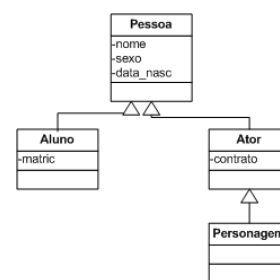
Paulo André Castro

POO

ITA - Stefanini

15

Exemplo em UML – Está correto ?



Paulo André Castro

POO

ITA - Stefanini

16

Exemplo de Orientação a Objetos - Java

- Classe

```
public class Pessoa {
    private int idade;
    private boolean sexo; // Verdadeiro para mulheres
    private boolean ehResponsavel() {
        if(idade>21 )
            return true;
        if(idade>18 && !sexo)
            return true;
        else
            return false;
    }
    public Pessoa(String nome, int id, boolean sex) {Nome=nome; idade=id; sexo=sex; }
}
.....
```

Paulo André Castro

POO

ITA - Stefanini

Mais em Orientação a Objetos

- Herança
 - Uma classe pode utilizar métodos e atributos de outras classes sem a necessidade de re-escrever ou copiar nada através do mecanismo de Herança
 - `public class Funcionario extends Pessoa {`

```
    double salario;
    int id;
    public double getSalario() {
        return salario;
    }
}
```

Paulo André Castro

POO

ITA - Stefanini

Classes e sub-classes

- Classe, classe-pai, super-classe, classe base:
 - Carro
 - Motor
 - Avião
- Sub-classe, classe-filha, classe derivada
 - Carro: Porsche 910
 - Motor: Ford 16V
 - Avião: Boeing 737

Paulo André Castro

POO

ITA - Stefanini

19

Relação de Herança

- O filho herda todas as características do pai
 - Comportamento: funções
 - Atributos: valores
- Em linguagens OO, geralmente há meios de restringir o que será ou não herdado

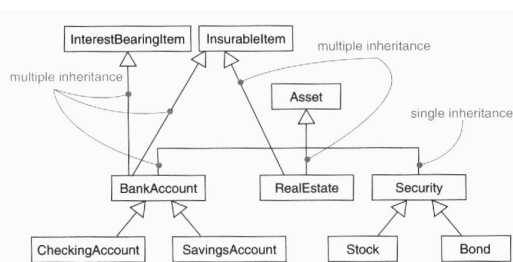
Paulo André Castro

POO

ITA - Stefanini

20

Exemplo de Herança Múltipla



Paulo André Castro

POO

ITA - Stefanini

21

Relação Agregação (Todo-Parte)

- Uma classe é composta por uma ou mais classes:
- Exemplos:
 - Carro e Motor
 - Avião e Motor
 - Pessoa e Braço

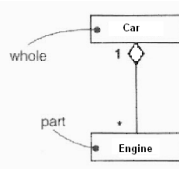
Paulo André Castro

POO

ITA - Stefanini

22

Agregação em UML



Paulo André Castro

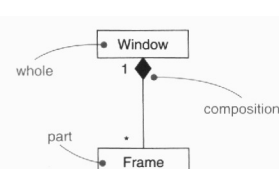
POO

ITA - Stefanini

23

Composição: Tipo especial de Agregação

Parte pertence a apenas um todo e tem tempo de vida igual ao todo



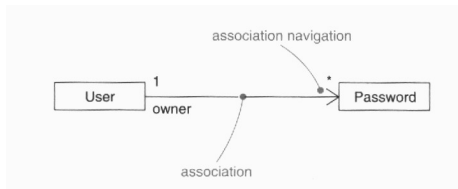
Paulo André Castro

POO

ITA - Stefanini

24

Outras Relações: Associação



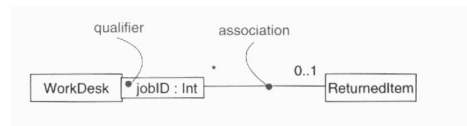
Paulo André Castro

POO

ITA - Stefanini

25

Outras Relações: Associação com Qualificação



Paulo André Castro

POO

ITA - Stefanini

26

Métodos e Atributos

- Classes podem ter vários métodos e atributos
 - Método: define um comportamento de uma classe
 - Atributo: define uma informação a ser mantida por cada instância de uma classe
- Escopo
 - Escopo de Classe:
 - Ex. Boeing 737 atributo: número de motores
 - Escopo de Objeto ("Instância")
 - Ex. Boeing 737 atributo: número de assentos

Paulo André Castro

POO

ITA - Stefanini

27

Encapsulamento

- Encapsulamento: É a capacidade de "esconder" parte do código e dos dados do restante do programa
- Pode-se definir um grau de visibilidade aos métodos e atributos de cada Classe.
- Há vários graus de visibilidade mas todas as linguagens implementam pelo menos os seguintes:
 - Público: Todos podem acessar (ler e escrever)
 - Privado: Apenas a própria classe pode acessar.

Paulo André Castro

POO

ITA - Stefanini

28

Polimorfismo

- Um mesmo comando enviado para objetos diferentes gera (ou pode gerar) ações diferentes.
- Exemplo:
 - Comando: Mover
 - Carro
 - Avião
 - Pessoa

Paulo André Castro

POO

ITA - Stefanini

29

Exemplo de Orientação a Objetos – classe Pessoa

```
Arquivo: Pessoa.java
public class Pessoa {
    private int idade;
    private boolean sexo; // Verdadeiro para mulheres
    private boolean ehResponsavel() {
        if(idade>21 )
            return true;
        if(idade>18 && !sexo)
            return true;
        else
            return false;
    }
    public Pessoa(String nome, int id, boolean sex) {Nome=nome; idade=id;
        sexo=sex;}
}
```

Paulo André Castro

POO

ITA - Stefanini

30

Exemplo de Orientação a Objetos – subclasse Casado

```
Arquivo: Casado.java
public class Casado extends Pessoa {
    public boolean ehResponsavel() {
        return true;
    }
    public Casado(String nome,int id,boolean sexo) {
        super(nome,idade,sexo);
    }
}
```

Paulo André Castro

POO

ITA - Stefanini

31

Polimorfismo – Código Java

```
public class ExemploPessoa {
    public static void main(String[] args) {
        Casado casado=new Casado("Zé",17,false);
        Pessoa cidadao=new Pessoa("Maria",18,true);
        Pessoa[] trabalhador= new Pessoa[2];
        trabalhador[0]=casado;
        trabalhador[1]=cidadao;
        for(int i=0; i<2; i++) {
            String aux;
            if(trabalhador[i].ehResponsavel())
                aux.strcpy(trabalhador[i].Nome+"é responsável");
            else
                aux.strcpy(trabalhador[i].Nome+ "não é responsável.");
            System.out.println(aux);
        }
    }
}
Qual o resultado da execução?
```

Paulo André Castro

POO

ITA - Stefanini

Mais em Orientação a Objetos

- Alterando comportamento nas classes filhas. Sobrescrever método.

```
public class Casado extends Pessoa {
    public boolean ehResponsavel() {
        return true;
    }
    public Casado(String nome,int id,boolean sexo) {
        super(nome,idade,sexo);
    }
}
```

Paulo André Castro

POO

ITA - Stefanini

33

Mais em Orientação a Objetos – Sobrecarga de Métodos

- Sobrecarga permite a existência de vários métodos de mesmo nome, porém com assinaturas levemente diferentes ou seja variando no número e tipo de argumentos e no valor de retorno

```
public Logaritmo {
    .....
    public double log(double x) {
        return Math.log(x);
    }
    public double log(double x, double b) {
        return (Math.log(x)/Math.log(b));
    }
}
```

Paulo André Castro

POO

ITA - Stefanini

34

Métodos Abstratos

- Métodos abstratos não tem implementação, porém obrigam as classes filhas a realizarem esta implementação. Útil para criar padronizações para as classes derivadas
- Se uma classe tem um (ou mais) métodos abstratos torna-se uma classe abstrata e não pode ser instanciada
- Exemplo:
 - Avião e o método Mover

Paulo André Castro

POO

ITA - Stefanini

35

Herança e Interfaces

- Herança Múltipla: quando uma classe pode herdar métodos e atributos de várias classes
- Não existe herança múltipla em Java, para evitar erros e diminuir a complexidade da programação
- Uma classe pode herdar apenas de uma outra classe
- Todas as classes herdam da classe Object
- Classe totalmente abstrata: Interface

Paulo André Castro

POO

ITA - Stefanini

Exemplo - Java

```
class Pessoa{
    public String nome;
    public char sexo;
    public Date dataNasc;
    ----
}
```

```
public class Ator extends Pessoa{
    public String contrato;
    /* campos herdados
    public String nome;
    public char sexo;
    public Date dataNasc; */
    ----
}
```

```
public class Aluno extends Pessoa{
    public long matric;
    /* campos herdados
    public String nome;
    public char sexo;
    public Date dataNasc; */
    ----
}
```

Paulo André Castro

POO

ITA - Stefanini

37

Interface

- Interface: Contrato(s) que uma classe deve respeitar. Isto é, um conjunto de métodos que a classe não pode deixar de implementar.

```
interface Veiculo {
    void mover();
    double velocidade;
}
```

Paulo André Castro

POO

ITA - Stefanini

38

Herança e Interfaces

- Uma classe em Java pode implementar várias interfaces

```
public class Class1 extends Class2 implements
Interface1,Interface2,Interface3 {
```

```
.....
....
}
```

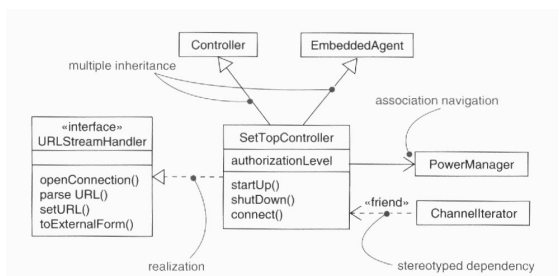
Paulo André Castro

POO

ITA - Stefanini

39

Outras Relações: Realização



Paulo André Castro

POO

ITA - Stefanini

40

Resumo até agora

- Componentes de uma classe
 - Métodos
 - Atributos(variáveis)
 - Relações
- Relações entre Classes
 - Herança
 - Agregação
 - Associação
 - Realização(Class/Interface)

Paulo André Castro

POO

ITA - Stefanini

41

Sumário

- Introdução
- Conceitos Básicos
 - Nomenclatura básica em OO
 - Variáveis e Instâncias
 - Métodos
 - Construtores
 - Herança e Polimorfismo
- Introdução a linguagem Java
 - Primeiros Programas
 - Fundamentos
 - Tratamento de Erros
 - Coleções

Paulo André Castro

POO

ITA - Stefanini

42

Java x C++

- **Similarities with C++**
 - User-defined classes can be used the same way as built-in types.
 - Basic syntax
- **Differences from C++**
 - Methods (member functions) are the only function type
 - Object is the topmost ancestor for all classes
 - All methods use the run-time, not compile-time, types (i.e. all Java methods are like C++ virtual functions)
 - The types of all objects are known at run-time
 - All objects are allocated on the heap (always safe to return objects from methods)
 - Single inheritance only

Paulo André Castro

POO

ITA - Stefanini

43

Criando Programas java

- **Name of file must match name of class**
 - It is case sensitive, even on Windows
- **Processing starts in main**
 - `public static void main(String[] args)`
 - Routines usually called "methods," not "functions."
- **Printing is done with System.out**
 - `System.out.println`, `System.out.print`
- **Compile with "javac"**
 - Open DOS window; work from there
 - Supply full case-sensitive file name (with file extension)
- **Execute with "java"**
 - Supply base class name (no file extension)

Paulo André Castro

POO

ITA - Stefanini

44

Exemplo

- **File: HelloWorld.java**

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world.");
    }
}
```
- **Compiling**

```
DOS> javac HelloWorld.java
```
- **Executing**

```
DOS> java HelloWorld
Hello, world.
```

Paulo André Castro

POO

ITA - Stefanini

45

Exemplo de Orientação a Objetos - Java

- **Classe**

```
public class Pessoa {
    private int idade; private String nome;
    private boolean sexo; // Verdadeiro para mulheres
    private boolean ehResponsavel() {
        if(idade>21 )
            return true;
        if(idade>18 && !sexo)
            return true;
        else
            return false;
    }
    public Pessoa(String n, int id, boolean sex) {
        nome=n; idade=id; sexo=sex;
    }
    .....
}
```

Paulo André Castro

POO

ITA - Stefanini

Criando Classes em Java

- **Convenção de Nomes em Java**
 - Classes iniciam com letras maiúsculas
 - Métodos, atributos e variáveis locais iniciam com minúsculas
- **Declaração de Classes**

```
public class MyClasse {
    .....
}
```
- **Criando Objetos a partir de Classes**
 - Uso do "new"

Paulo André Castro

POO

ITA - Stefanini

47

Orientação a Objetos em Java

- **Instâncias**
 - `String teste=new String(60);`
 - `String teste2="Isto é um teste";`
 - `ServerSocket servidor= new ServerSocket(25);`
- **Construtores e destrutores**
 - Os construtores são similares ao C++
 - Não há destrutores em Java.

Paulo André Castro

POO

ITA - Stefanini

Exemplo de Orientação a Objetos - Java

- **Objetos**
 - Pessoa a=new Pessoa("Joao",25);
 - Pessoa b= new Pessoa("Ana",23);
 - Pessoa c=new Pessoa("Carla",28);
 - d=b;
- **Atributos de uma classe**
 - a.idade == ?
 - d.nome = ?

Paulo André Castro

POO

ITA - Stefanini

49

Formato de uma Definição de Classe em Java

```
modifier class Classname {  
  
    modifier data-type field1;  
    modifier data-type field2;  
    ...  
    modifier data-type fieldN;  
  
    modifier Return-Type methodName(parameters) {  
        //statements  
    }  
  
    ...  
  
    modifier Return-Type methodName2(parameters) {  
        //statements  
    }  
}
```

Paulo André Castro

POO

ITA - Stefanini

50

Acessando variáveis de instância

- Use um ponto entre o nome da variável e o campo
 - objectName.fieldName;
- Por exemplo, usando a classe Point da biblioteca Java
 - Point p=new Point(2,3); //criação de objeto p
 - int x2= p.x*p.x; // x2 é 4
 - int xPlusY=p.x+p.y; // xPlusY é 5
 - p.x=3;
 - x2=p.x* p.x; // x2 agora é 9
- Dentro de um objeto, seus métodos podem acessar as variáveis de instância (e de Classe) sem utilizar o ponto

Paulo André Castro

POO

ITA - Stefanini

51

Exemplo – Orientado a Objetos?

```
class Ship1 {  
    public double x, y, speed, direction;  
    public String name;  
}  
  
public class Test1 {  
    public static void main(String[] args) {  
        Ship1 s1 = new Ship1();  
        s1.x = 0.0;  
        s1.y = 0.0;  
        s1.speed = 1.0;  
        s1.direction = 0.0; // East  
        s1.name = "Ship1";  
        Ship1 s2 = new Ship1();  
        s2.x = 0.0;  
        s2.y = 0.0;  
        s2.speed = 2.0;  
        s2.direction = 135.0; // Northwest  
        s2.name = "Ship2";  
    }  
}
```

Paulo André Castro

POO

ITA - Stefanini

52

Exemplos

```
...  
s1.x = s1.x + s1.speed  
    * Math.cos(s1.direction * Math.PI / 180.0);  
s1.y = s1.y + s1.speed  
    * Math.sin(s1.direction * Math.PI / 180.0);  
s2.x = s2.x + s2.speed  
    * Math.cos(s2.direction * Math.PI / 180.0);  
s2.y = s2.y + s2.speed  
    * Math.sin(s2.direction * Math.PI / 180.0);  
System.out.println(s1.name + " is at ("  
    + s1.x + "," + s1.y + ")");  
System.out.println(s2.name + " is at ("  
    + s2.x + "," + s2.y + ")");  
}  
}
```

Paulo André Castro

POO

ITA - Stefanini

53

Resultado

• Compiling and Running:

```
javac Test1.java  
java Test1
```

Output:

```
Ship1 is at (1,0).  
Ship2 is at (-1.41421,1.41421).
```

Paulo André Castro

POO

ITA - Stefanini

54

Problemas na Modelagem?

- Código específico a uma classe está escrito apenas nesta classe?

Paulo André Castro

POO

ITA - Stefanini

55

Métodos: Exemplo

```
class Ship2 {
    public double x=0.0, y=0.0, speed=1.0, direction=0.0;
    public String name = "UnnamedShip";

    private double degreesToRadians(double degrees) {
        return(degrees * Math.PI / 180.0);
    }

    public void move() {
        double angle = degreesToRadians(direction);
        x = x + speed * Math.cos(angle);
        y = y + speed * Math.sin(angle);
    }

    public void printLocation() {
        System.out.println(name + " is at ("
            + x + "," + y + ").");
    }
}
```

Paulo André Castro

POO

ITA - Stefanini

56

Exemplo (cont.)

```
public class Test2 {
    public static void main(String[] args) {
        Ship2 s1 = new Ship2();
        s1.name = "Ship1";
        Ship2 s2 = new Ship2();
        s2.direction = 135.0; // Northwest
        s2.speed = 2.0;
        s2.name = "Ship2";
        s1.move();
        s2.move();
        s1.printLocation();
        s2.printLocation();
    }
}

• Compiling and Running:
  javac Test2.java
  java Test2

• Output:
  Ship1 is at (1,0).
  Ship2 is at (-1.41421,1.41421).
```

Paulo André Castro

POO

ITA - Stefanini

57

Métodos Especiais: Construtores

- Construtores são métodos de uma classe que tem o mesmo nome desta classe e são chamados quando da criação de um objeto desta classe.
 - São usados para "inicializar" um objeto
 - São chamados apenas através de um comando "new"
 - Java provê automaticamente um construtor sem parâmetros, caso nenhum construtor seja declarado explicitamente
 - Por isso, podemos utilizar: Ship s1=new Ship(); Mesmo sem criar um construtor Ship()

Paulo André Castro

POO

ITA - Stefanini

58

Métodos Especiais: Construtores (2)

```
class Ship3 {
    public double x, y, speed, direction;
    public String name;

    public Ship3(double x, double y,
        double speed, double direction,
        String name) {
        this.x = x; // "this" differentiates instance vars
        this.y = y; // from local vars.
        this.speed = speed;
        this.direction = direction;
        this.name = name;
    }

    private double degreesToRadians(double degrees) {
        return(degrees * Math.PI / 180.0);
    }
}
```

Paulo André Castro

POO

ITA - Stefanini

59

Métodos Especiais: Construtores (3)

```
public void move() {
    double angle = degreesToRadians(direction);
    x = x + speed * Math.cos(angle);
    y = y + speed * Math.sin(angle);
}

public void printLocation() {
    System.out.println(name + " is at ("
        + x + "," + y + ").");
}

}

public class Test3 {
    public static void main(String[] args) {
        Ship3 s1 = new Ship3(0.0, 0.0, 1.0, 0.0, "Ship1");
        Ship3 s2 = new Ship3(0.0, 0.0, 2.0, 135.0, "Ship2");
        s1.move();
        s2.move();
        s1.printLocation();
        s2.printLocation();
    }
}
```

Paulo André Castro

POO

ITA - Stefanini

60

Métodos Especiais: Construtores (4)

- **Compiling and Running:**

```
javac Test3.java  
java Test3
```
- **Output:**

```
Ship1 is at (1,0).  
Ship2 is at (-1.41421,1.41421).
```

Paulo André Castro

POO

ITA - Stefanini

61

A variável especial: this

- The **this** object reference can be used inside any non-static method to refer to the current object
- The common uses of the **this** reference are:
 1. To pass a reference to the current object as a parameter to other methods

```
someMethod(this);
```
 2. To resolve name conflicts
 - Using **this** permits the use of instance variables in methods that have local variables with the same name
- Note that it is only necessary to say **this.fieldName** when you have a local variable and a class field with the same name; otherwise just use **fieldName** with no **this**

Paulo André Castro

POO

ITA - Stefanini

62

Destrutores

- Não há destrutores em Java
 - Ao contrário de C++
- Garbage Collector
 - Coletor de lixo: Faz a limpeza (remoção da memória) de variáveis que não serão mais utilizadas pelo programa.
 - Pode ser chamado através de :
 - `System.gc();`

Paulo André Castro

POO

ITA - Stefanini

63

Convenções e Boas Práticas em OO

Resumo

- Código referente exclusivamente a uma classe deve ficar dentro desta classe
- Uma classe deve ter o menor número possível de métodos públicos, mas deve ter pelo menos um método público
- É uma boa prática evitar variáveis públicas. Prefira utilizar métodos para acessar as variáveis. Os métodos `get` e `set`.
- Use métodos construtores para inicializar objetos

Paulo André Castro

POO

ITA - Stefanini

64

Convenções e Boas Práticas em OO

Resumo

- Classes devem iniciar com letras maiúsculas, métodos, atributos e variáveis com letras minúsculas
- Métodos devem ter um tipo de retorno ou "void"
- Acesse atributos através de `objectName.fieldName`
- Acesse métodos através de `objectName.methodName()`
- Métodos estáticos não precisam de instâncias da classe
- Construtores são métodos especiais sem tipo de retorno

Paulo André Castro

POO

ITA - Stefanini

65

Convenções e características em Java

Resumo

- A referência `this` aponta para o objeto atual
- Java faz seu próprio gerenciamento de memória e portanto não requer destrutores
- Java permite herança simples e o uso de interfaces
- As classes abstratas e interfaces não podem ser instanciadas
- Uma classe Java pode implementar várias interfaces

Paulo André Castro

POO

ITA - Stefanini

66

Fundamentos de Programação Java

- Comandos
- Sintaxe
- Estrutura
- Exemplos

Paulo André Castro

POO

ITA - Stefanini

67

Fundamentos de Programação Java

- Início e Fim de Blocos de Comandos
 - “{” e “}”
- comandos if, if-else, while, do-while, for e switch-case são idênticos ao C/ C++
- Comando break <label>;

Paulo André Castro

POO

ITA - Stefanini

Tipos básicos de Variáveis

Tipo	Tamanho
byte	1 bytes
short	2 bytes
int	4 bytes
long	8 bytes
float	4 bytes
double	8 bytes
char	2 bytes
boolean	1 bit

Paulo André Castro

POO

ITA - Stefanini

Variáveis

- Declaração
 - int inteiro; char letra;
 - byte apenasUmByte;
- Declaração e Inicialização
 - float saldoConta=0.0;
 - boolean serOuNaoSer=false;

Paulo André Castro

POO

ITA - Stefanini

Tipos Complexos

- int[] arrayInteiro= new int[40];
- char sTexto[]= new char[60]; // use String's
- Classe String
 - String s=new String("isto e uma string");
 - String s2="isto e outra string";
 - String c=s+s2;
 - System.out.println("S="+s+"S2="+s2+"c="+c);

Paulo André Castro

POO

ITA - Stefanini

Tipos Complexos

- **Arrays are accessed with []**
 - Array indices are zero-based
 - The argument to `main` is an array of strings that correspond to the command line arguments
 - `args[0]` returns first command-line argument
 - `args[1]` returns second command-line argument
 - Etc.
- **The length field gives the number of elements in an array**
 - Thus, `args.length` gives the number of command-line arguments
 - Unlike in C/C++, the name of the program is not inserted into the command-line arguments

Paulo André Castro

POO

ITA - Stefanini

72

Exemplo

- File: ShowTwoArgs.java

```
public class ShowTwoArgs {
    public static void main(String[] args) {
        System.out.println("First arg: " +
            args[0]);
        System.out.println("Second arg: " +
            args[1]);
    }
}
```

Paulo André Castro

POO

ITA - Stefanini

73

Exemplo

- **Compiling**
DOS> javac ShowTwoArgs.java
- **Executing**
DOS> java ShowTwoArgs Hello World
First args Hello
Second arg: Class

DOS> java ShowTwoArgs
[Error message]

Paulo André Castro

POO

ITA - Stefanini

74

Comandos...

- **Comando** := *Comando_simples* | *Comando_composto* | *Comando_condicional* | *Comando_iterativo* | *Comando_de_seleção*
- **Comando_simples** := *Comando_de_atribuição* | *Comando_de_entrada* | *Comando_de_saida* | *Chamada_de_subprograma* | *Comando_vazio*
- **Comando_vazio** := ;
- **Comando_composto** := { *Comando Comando*
Comando }

Paulo André Castro

POO

ITA - Stefanini

75

Comandos...

- **Comando_condicional** := *if* (*Expressão*)
Comando_1 | *if* (*Expressão*)
Comando_1 else Comando_2
- **Comando_iterativo** := *Comando_while* |
Comando_do | *Comando_for*

Paulo André Castro

POO

ITA - Stefanini

76

Comando de Seleção em Java

```
switch ( expressão ) {
    case V11:
    case V12:
        .
        .
    case V1n: lista de comandos; break;
    case V21:
    case V22:
        .
        .
    case V2n: lista de comandos; break;
        .
        .
    case Vpx: lista de comandos; break;
    default: lista de comandos;
}
```

Paulo André Castro

POO

ITA - Stefanini

77

Comandos...

- **Comando atribuição condicional**
 - Forma: *Expr1 ? Expr2 : Expr3*
 - Calcula-se *Expr1*;
 - Se o valor for **Verdadeiro**, calcula-se o valor de *Expr2*, que será o valor da expressão condicional;
 - Se o valor for **Falso**, calcula-se o valor de *Expr3*, que será o valor da expressão condicional.

Paulo André Castro

POO

ITA - Stefanini

78

Comandos de repetição

- **while**

```
while (continueTest) {
    body;
}
```
- **do**

```
do {
    body;
} while (continueTest);
```
- **for**

```
for(init; continueTest; updateOp) {
    body;
}
```

Paulo André Castro

POO

ITA - Stefanini

79

Comandos Iterativos...

- **Comandos for e while versus Comando do-while**
- **Comando break:** Saída anormal de um comando iterativo ou comando de seleção mais interno.

```
while (.....){
    .....
    if (.....) break;
    if (.....) continue;
    .....
}
..... /* Proximo comando a ser executado depois do break */
.....
```

- **Comando continue:** encerra a iteração corrente e inicia a iteração seguinte.

Paulo André Castro

POO

ITA - Stefanini

80

Exemplo - while

```
public static void listNums1(int max) {
    int i = 0;
    while (i <= max) {
        System.out.println("Number: " + i);
        i++; // "++" means "add one"
    }
}
```

Paulo André Castro

POO

ITA - Stefanini

81

Exemplo - do/while

```
public static void listNums2(int max) {
    int i = 0;
    do {
        System.out.println("Number: " + i);
        i++;
    } while (i <= max);
    // ^ Don't forget semicolon
}
```

Paulo André Castro

POO

ITA - Stefanini

82

Exemplo - for

```
public static void listNums3(int max) {
    for(int i=0; i<max; i++) {
        System.out.println("Number: " + i);
    }
}
```

Paulo André Castro

POO

ITA - Stefanini

83

Percorrendo todos os elementos de um array

```
public class LoopTest {
    public static void main(String[] args) {
        listNums1(5);
        listNums2(6);
        listNums3(7);
    }

    public static void listNums1(int max) {...}
    public static void listNums2(int max) {...}
    public static void listNums3(int max) {...}
}
```

Paulo André Castro

POO

ITA - Stefanini

84

Percorrendo todos os elementos de um array

```
public class ShowArgs {
    public static void main(String[] args) {
        for(int i=0; i<args.length; i++) {
            System.out.println("Arg " + i +
                " is " +
                args[i]);
        }
    }
}
```

Paulo André Castro

POO

ITA - Stefanini

85

Comando condicional - if

- **Single Option**

```
if (boolean-expression) {
    statement;
}
```
- **Multiple Options**

```
if (boolean-expression) {
    statement1;
} else {
    statement2;
}
```

Paulo André Castro

POO

ITA - Stefanini

86

Expressões Condicionais

- **==, !=**
 - Equality, inequality. In addition to comparing primitive types, == tests if two objects are identical (the same object), not just if they appear equal (have the same fields). More details when we introduce objects.
- **<, <=, >, >=**
 - Numeric less than, less than or equal to, greater than, greater than or equal to.
- **&&, ||**
 - Logical AND, OR. Both use short-circuit evaluation to more efficiently compute the results of complicated expressions.
- **!**
 - Logical negation.

Paulo André Castro

POO

ITA - Stefanini

87

Exemplos

```
public static int max2(int n1, int n2) {
    if (n1 >= n2)
        return (n1);
    else
        return (n2);
}
```

Paulo André Castro

POO

ITA - Stefanini

88

Comparação de Strings

```
public static void main(String[] args) {
    String match = "Test";
    if (args.length == 0) {
        System.out.println("No args");
    } else if (args[0] == match) {
        System.out.println("Match");
    } else {
        System.out.println("No match");
    }
}
```

- **Prints "No match" for all inputs**
 - Fix:

```
if (args[0].equals(match))
```

Paulo André Castro

POO

ITA - Stefanini

89

Criando Arrays

- **Step 1: allocate an array of references:**

```
type[] var = new type[size];
```
- **Eg:**

```
int[] values = new int[7];
Point[] points = new Point[someArray.length];
```
- **Step 2: populate the array**

```
points[0] = new Point(...);
points[1] = new Point(...);
...
Points[6] = new Point(...);
```
- **If you fail to populate an entry**
 - Default value is 0 for numeric arrays
 - Default value is null for object arrays

Paulo André Castro

POO

ITA - Stefanini

90

Arrays de várias dimensões

```
int[][] twoD = new int[64][32];

String[][] cats = { { "Caesar", "blue-point" },
                    { "Heather", "seal-point" },
                    { "Ted", "red-point" } };

int[][] irregular = { { 1 },
                      { 2, 3, 4 },
                      { 5 },
                      { 6, 7 } };
```

Paulo André Castro

POO

ITA - Stefanini

91

Exemplo

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

        int[][] triangle = new int[10][];

        for(int i=0; i<triangle.length; i++) {
            triangle[i] = new int[i+1];
        }

        for (int i=0; i<triangle.length; i++) {
            for(int j=0; j<triangle[i].length; j++) {
                System.out.print(triangle[i][j]);
            }
            System.out.println();
        }
    }
}
```

Paulo André Castro

POO

ITA - Stefanini

92

Resultado

```
> java TriangleArray
0
00
000
0000
00000
000000
0000000
00000000
000000000
0000000000
00000000000
```

Paulo André Castro

POO

ITA - Stefanini

93

Tratamento de Erros: Tradicional

- O tratamento de erros em linguagens sem Exceções, gera um código "sujo" com código tratamento de erro:
 - `ret=funcao1();`
 - `if(ret==ERRO)`
 - `//Trata erro`
 - `ret=funcao2();`
 - `if(ret==ERRO)`
 - `//Trata Erro 2`

Paulo André Castro

POO

ITA - Stefanini

94

Tratamento de Erros: Exceções

- Em Java, o sistema de tratamento de erros é baseado exceções
 - Exceções devem ser tratados em blocos `try/catch`
 - Quando ocorre uma exceção esta é direcionada para o correspondente `catch`
- Formato:

```
try {
    statement1;
    statement2;
    ...
} catch(SomeException someVar) {
    handleTheException(someVar);
}
```

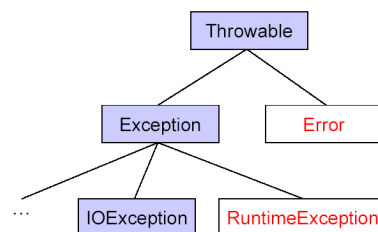
Paulo André Castro

POO

ITA - Stefanini

95

Diagrama Simplificado de Exceções



Paulo André Castro

POO

ITA - Stefanini

96

Try-catch

- Um bloco try pode ter associados vários blocos catch

```
try {
    ...
} catch (ExceptionType1 var1) {
    // Do something
} catch (ExceptionType2 var2) {
    // Do something else
}
```

- A exceção será tratado pelo bloco catch mais específico
- Caso não seja encontrado algum apropriado, a exceção será direcionada para blocos try mais externos
 - Caso não seja encontrado nenhum try apropriado dentro do método, este irá jogar a exceção

Paulo André Castro

POO

ITA - Stefanini

97

Um exemplo de Try-catch

```
....
BufferedReader in = null;
String lineIn;
try {
    in = new BufferedReader(new FileReader("book.txt"));
    while((lineIn = in.readLine()) != null) {
        System.out.println(lineIn);
    }
    in.close();
} catch (FileNotFoundException fnfe) {
    System.out.println("File not found.");
} catch (EOFException eofe) {
    System.out.println("Unexpected End of File.");
} catch (IOException ioe) {
    System.out.println("IOError reading input: " + ioe);
    ioe.printStackTrace(); // Show stack dump
}
```

Paulo André Castro

POO

ITA - Stefanini

98

A cláusula finally

- Ao final de um conjunto de blocos catch pode-se, opcionalmente, incluir uma cláusula **finally**. Caso nenhum bloco catch, seja executado o **finally** será sempre executado

```
try {
    ...
} catch (SomeException someVar) {
    // Do something
} finally {
    // Always executed
}
```

Paulo André Castro

POO

ITA - Stefanini

99

Jogando exceções

- If a potential exception is not handled in the method, then the method must declare that the exception can be thrown

```
public SomeType someMethod(...) throws SomeException {
    // Unhandled potential exception
    ...
}
```

- Note: Multiple exception types (comma separated) can be declared in the throws clause

- Explicitly generating an exception

```
throw new IOException("Blocked by firewall.");
throw new MalformedURLException("Invalid protocol");
```

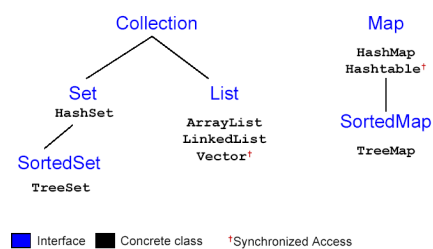
Paulo André Castro

POO

ITA - Stefanini

100

Estruturas de Dados no Java 2



Paulo André Castro

POO

ITA - Stefanini

101

Collection Interfaces

- Collection**
 - Abstract class for holding groups of objects
- Set**
 - Group of objects containing **no duplicates**
- SortedSet**
 - Set of objects (**no duplicates**) stored in **ascending order**
 - Order is determined by a **Comparator**
- List**
 - Physically** (versus logically) ordered sequence of objects
- Map**
 - Stores objects (unordered) identified by **unique keys**
- SortedMap**
 - Objects stored in **ascending order** based on their key value
 - Neither duplicate or null keys are permitted

Paulo André Castro

POO

ITA - Stefanini

102

Duas Estruturas de Dados Muito Úteis

- **Vector**
 - Um array de *Object* de tamanho variável
 - Tempo para acessar um objeto é independente da sua posição na lista
 - No jdk 1.2 ou superior, pode-se utilizar ArrayList
 - ArrayList não é sincronizado (thread-safe), por isso tende a ser mais rápido
- **Hashtable**
 - Armazena pares: nome-valor como Object
 - Valores não podem ser nulos
 - No jdk 1.2 ou superior, pode-se utilizar HashMap
 - HashMap não é sincronizado (thread-safe), por isso tende a ser mais rápido

Paulo André Castro

POO

ITA - Stefanini

103

Métodos úteis em Vector

- **addElement/insertElementAt/setElementAt**
 - Add elements to the vector
- **removeElement/removeElementAt**
 - Removes an element from the vector
- **firstElement/lastElement**
 - Returns a reference to the first and last element, respectively (without removing)
- **elementAt**
 - Returns the element at the specified index
- **indexOf**
 - Returns the index of an element that equals the object specified
- **contains**
 - Determines if the vector contains an object

Paulo André Castro

POO

ITA - Stefanini

104

Utilizando Vector

- **elements**
 - Returns an Enumeration of objects in the vector

```
Enumeration elements = vector.elements();
while(elements.hasMoreElements()) {
    System.out.println(elements.nextElement());
}
```
- **size**
 - The number of elements in the vector
- **capacity**
 - The number of elements the vector can hold before becoming resized

Paulo André Castro

POO

ITA - Stefanini

105

Métodos úteis em Hashtable

- **put/get**
 - Stores or retrieves a value in the hashtable
- **remove/clear**
 - Removes a particular entry or all entries from the hashtable
- **containsKey/contains**
 - Determines if the hashtable contains a particular key or element
- **keys/elements**
 - Returns an enumeration of all keys or elements, respectively
- **size**
 - Returns the number of elements in the hashtable
- **rehash**
 - Increases the capacity of the hashtable and reorganizes it

Paulo André Castro

POO

ITA - Stefanini

106

Exemplo de Uso de um Hashtable

```
import java.util.Hashtable;

public class ExemploHashtable {
    public static void main(String[] args) {
        Hashtable numbers = new Hashtable();
        numbers.put("one", new Integer(1));
        numbers.put("two", new Integer(2));
        numbers.put("three", new Integer(3));
        String key="three";
        Integer n = (Integer)numbers.get(key);
        if (n != null) {
            System.out.println(key+" = " + n);
        }
    }
}
```

Paulo André Castro

POO

ITA - Stefanini

107

Resultado

- >three = 3

Paulo André Castro

POO

ITA - Stefanini

108

Exemplo de Uso de uma Coleção Vector

```
import java.util.Iterator;
import java.util.Vector;

public class Colecoes {

    public static void main(String[] args) {
        Vector vetStrings=new Vector();
        for(int i=1;i<=5;i++)
            vetStrings.add("Linha "+i);
        //Laços de Iteração
        for (Iterator iter = vetStrings.iterator(); iter.hasNext();) {
            String element = (String) iter.next();
            System.out.println(element);
        }
    }
}
```

Paulo André Castro

POO

ITA - Stefanini

109

Resultado

```
>Linha 1
Linha 2
Linha 3
Linha 4
Linha 5
```

Paulo André Castro

POO

ITA - Stefanini

110

Classes Genéricas no JDK 5.0

- Classe genéricas: classes que podem ser parametrizadas para trabalharem sobre classes específicas
 - Tipos parametrizáveis: (Design Patterns, GoF)
 - Templates: C++
 - Classes genéricas: Java, C#

Paulo André Castro

POO

ITA - Stefanini

111

Avanços em Collections no JDK 5.0

- Coleções genéricas:
`Vector<String> vetStrings=new Vector<String>();`
- Laços de Iteração Aprimorados
`for(String element: vetStrings) {
 System.out.println(element);
}`

Paulo André Castro

POO

ITA - Stefanini

112

Exemplo de Uso de uma Coleção Genérica

```
import java.util.Vector;

public class ColecoesGenericas {

    public static void main(String[] args) {

        Vector<String > vetStrings=new Vector<String>();
        for(int i=1;i<=5;i++)
            vetStrings.add("Linha "+i);
        //Laços de Iteração Aprimorados
        for(String element: vetStrings) {
            System.out.println(element);
        }
    }
}
```

Paulo André Castro

POO

ITA - Stefanini

113

Resultado

```
>Linha 1
Linha 2
Linha 3
Linha 4
Linha 5
```

Paulo André Castro

POO

ITA - Stefanini

114

HashTable Genérico

```
import java.util.Hashtable;

public class ExemploHashtableGenerico {
    public static void main(String[] args) {
        Hashtable<String,Integer> numbers = new Hashtable<String,Integer>();
        numbers.put("one", new Integer(1));
        numbers.put("two", new Integer(2));
        numbers.put("three", new Integer(3));
        String key="three";
        Integer n = numbers.get(key);
        if (n != null) {
            System.out.println(key+" = " + n);
        }
    }
}
```

Paulo André Castro

POO

ITA - Stefanini

115

Resultado

- >three = 3

Paulo André Castro

POO

ITA - Stefanini

116

Classes Wrapper

- Todo tipo primitivo em Java, tem uma classe correspondente que pode encapsula-lo

Primitive Data Type	Corresponding Object Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Paulo André Castro

POO

ITA - Stefanini

117

Uso de Classes Wrapper

- Define constantes úteis, por exemplo:

```
Integer.MAX_VALUE
Float.NEGATIVE_INFINITY
```

- Conversão entre tipos de dados
 - Utilize parseXXX para fazer conversões:

```
try {
    String value = "3.14e6";
    Double d = Double.parseDouble(value);
} catch (NumberFormatException nfe) {
    System.out.println("Can't convert: " + value);
}
```

Paulo André Castro

POO

ITA - Stefanini

118

Outras conversões...

Data Type	Convert String using either ...
byte	Byte.parseByte(<i>string</i>)
	new Byte(<i>string</i>).byteValue()
short	Short.parseShort(<i>string</i>)
	new Short(<i>string</i>).shortValue()
int	Integer.parseInt(<i>string</i>)
	new Integer(<i>string</i>).intValue()
long	Long.parseLong(<i>string</i>)
	new Long(<i>string</i>).longValue()
float	Float.parseFloat(<i>string</i>)
	new Float(<i>string</i>).floatValue()
double	Double.parseDouble(<i>string</i>)
	new Double(<i>string</i>).doubleValue()

Paulo André Castro

POO

ITA - Stefanini

119

Resumindo...

- Laços de repetição, comandos condicionais e o acesso a arrays é feito em Java da mesma forma que em C++
- String é uma classe em Java, não um array de caracteres
- Nunca compare Strings usando ==
- O tratamento de erros é feito através de exceções (blocos try-catch-finally)
- Vector, ArrayList e HashMap são estruturas de dados muito úteis disponíveis em Java
 - Podem manter um número arbitrário de elementos

Paulo André Castro

POO

ITA - Stefanini

120

Sumário de Hoje

- Introdução
- Conceitos Básicos
 - Nomenclatura básica em OO
 - Variáveis e Instâncias
 - Métodos
 - Construtores
 - Herança e Polimorfismo
- Introdução a linguagem Java