Java First-Tier: Aplicações

Orientação a Objetos em Java (I)

Grupo de Linguagens de Programação

Departamento de Informática
PUC-Rio

Paradigmas de Programação

- Programação Funcional
- Programação Procedural
- Programação Orientada por Objetos

2

Paradigma OO

3

Nomenclatura

- A unidade básica de programação em linguagens orientadas a objetos é a classe
- Classes definem atributos e métodos comuns a vários objetos
- Todo objeto é uma instância de uma classe

4

Nomenclatura

- Objetos possuem um estado representado pelos valores dos atributos definidos em sua classe
- O conjunto de métodos que um objeto pode executar é definido pela sua classe

5

Tipos Abstratos de Dados

6

TAD

- Modela uma estrutura de dados através de sua funcionalidade.
- Define a interface de acesso à estrutura.
- Não faz qualquer consideração com relação à implementação.

7

Exemplo de TAD: Pilha

- Funcionalidade: armazenagem LIFO
- Interface:

`boolean isEmpty()`
verifica se a pilha está vazia

`push(int n)`
empilha o número fornecido

`int pop()`
desempilha o número do topo e o retorna

`int top()`
retorna o número do topo

8

TAD × Classes

- Uma determinada implementação de um TAD pode ser realizada por meio de uma classe.
- A classe deve prover todos os métodos definidos na interface do TAD.
- Um objeto dessa classe implementa uma instância do TAD.

9

Classes & Objetos

10

Classes em Java

- Em Java, a declaração de novas classes é feita através da construção **class**.
- Podemos criar uma classe **Point** para representar um ponto (omitindo sua implementação) da seguinte forma:

```
class Point {
  ...
}
```

11

Atributos

- Como dito, classes definem dados que suas instâncias conterão.
- A classe **Point** precisa armazenar as coordenadas do ponto sendo representado de alguma forma.

```
class Point {
  int x, y;
}
```

12

Instanciação

- Uma vez definida uma classe, uma nova instância (objeto) pode ser criada através do comando **new**.
- Podemos criar uma instância da classe **Point** da seguinte forma:

```
Point p = new Point();
```

13

Uso de Atributos

- Os atributos de uma instância de **Point** podem ser manipulados diretamente.

```
Point p1 = new Point();
p1.x = 1;
p1.y = 2;
// p1 representa o ponto (1,2)
Point p2 = new Point();
p2.x = 0;
p2.y = 0;
// e p2 o ponto (0,0)
```

14

Referências para Objetos

- Em Java, nós sempre fazemos referência ao objeto. Dessa forma, duas variáveis podem se referenciar ao mesmo ponto.

```
Point p1 = new Point();
Point p2 = p1;
p2.x = 2;
p2.y = 3;
// p1 e p2 representam o ponto (2,3)
```

15

Métodos

- Além de atributos, uma classe deve definir os métodos que irá disponibilizar, isto é, a sua interface.
- A classe **Point** pode, por exemplo, prover um método para mover o ponto de um dado deslocamento.

16

Declaração de Método

- Para mover um ponto, precisamos saber quanto deslocar em x e em y. Esse método não tem um valor de retorno pois seu efeito é mudar o *estado* do objeto.

```
class Point {
  int x, y;
  void move(int dx, int dy) {
    x += dx;
    y += dy;
  }
}
```

17

Envio de Mensagens: Chamadas de Método

- Em Java, o envio de uma mensagem é feito através de uma chamada de método com passagem de parâmetros.
- Por exemplo, a mensagem que dispara a ação de deslocar um ponto é a chamada de seu método **move**.

```
p1.move(2,2);
// agora p1 está deslocado de duas unidades,
// no sentido positivo, nos dois eixos.
```

18

this

- Dentro de um método, o objeto pode precisar de sua própria referência.
- Em Java, a palavra reservada **this** significa essa referência ao próprio objeto.

```
class Point {
  int x, y;
  void move(int dx, int dy) {
    this.x += dx;
    this.y += dy;
  }
}
```

19

Inicializações

- Em várias circunstâncias, é interessante inicializar um objeto.
- Por exemplo, poderíamos querer que todo ponto recém criado estivesse em (0,0).
- Esse tipo de inicialização se resume a determinar valores iniciais para os atributos.

20

Inicialização de Atributos

- Por exemplo, a classe **Point** poderia declarar:

```
class Point {
  int x = 0;
  int y = 0;
  void move(int dx, int dy) {
    this.x += dx;
    this.y += dy;
  }
}
```

21

Construtores

- Ao invés de criar pontos sempre em (0,0), poderíamos querer especificar a posição do ponto no momento de sua criação.
- O uso de *construtores* permite isso.
- Construtores são mais genéricos do que simples atribuições de valores iniciais aos atributos: podem receber parâmetros e fazer um processamento qualquer.

22

Declaração de Construtores

- O construtor citado para a classe **Point** pode ser definido da seguinte forma:

```
class Point {
  int x, y;
  Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  ...
}
```

23

Usando Construtores

- Como o construtor é um método de inicialização do objeto, devemos utilizá-lo no momento da instanciação.

```
Point p1 = new Point(1,2); // p1 é o ponto (1,2)
Point p2 = new Point(0,0); // p2 é o ponto (0,0)
```

24

Construtor Padrão

- Quando não especificamos nenhum construtor, a linguagem Java declara, implicitamente, um construtor padrão, vazio, que não recebe parâmetros.
- Se declararmos algum construtor, esse construtor padrão *não* será mais declarado.

25

Finalizações

- Pode ser necessário executar alguma ação antes que um objeto deixe de existir.
- Para isso são utilizados os *destrutores*.
- Destrutores são métodos que são chamados automaticamente quando um objeto deixa de existir.
- Em Java, destrutores são chamados de *finalizadores*.

26

Gerência de Memória

- Java possui uma gerência automática de memória, isto é, quando um objeto não é mais referenciado pelo programa, ele é automaticamente coletado (destruído).
- A esse processo chamamos “coleta de lixo”.
- Nem todas as linguagens OO fazem coleta de lixo e, nesse caso, o programador deve destruir o objeto explicitamente.

27

Finalizadores em Java

- Quando um objeto Java vai ser coletado, ele tem seu método **finalize** chamado.
- Esse método deve efetuar qualquer procedimento de finalização que seja necessário antes da coleta do objeto.

28

Membros de Classe

- Classes podem declarar membros (atributos e métodos) que sejam comuns a todas as instâncias, ou seja, membros compartilhados por todos os objetos da classe.
- Tais membros são comumente chamados de ‘membros de classe’ (versus ‘de objetos’).
- Em Java, declaramos um membro de classe usando o qualificador **static**. Daí, o nome ‘membros estáticos’ usado em Java.

29

Membros de Classe: Motivação

- Considere uma classe que precise atribuir identificadores unívocos para cada objeto.
- Cada objeto, ao ser criado, recebe o seu identificador.
- O identificador pode ser um número gerado seqüencialmente, de tal forma que cada objeto guarde o seu mas o próximo número a ser usado deve ser armazenado na *classe*.

30

Membros de Classe: Um Exemplo

- Podemos criar uma classe que modele produtos que são produzidos em uma fábrica.
- Cada produto deve ter um código único de identificação.

31

Membros de Classe: Codificação do Exemplo

```
class Produto {
  static int próximo_id = 0;
  int id;
  Produto() {
    id = próximo_id;
    próximo_id++;
  }
  ...
}
```

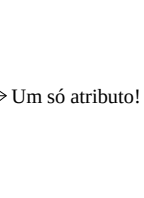
32

Membros de Classe: Análise do Exemplo

```
// Considere que ainda não há nenhum produto.
// Produto.próximo_id = 0

Produto lápis = new Produto();
  // lápis.id = 0
  // lápis.próximo_id = 1

Produto caneta = new Produto();
  // caneta.id = 1
  // caneta.próximo_id = 2
```



33

Membros de Classe: Acesso Direto

- Como os membros estáticos são da classe, não precisamos de um objeto para acessá-los: podemos fazê-lo diretamente sobre a classe.

```
Produto.próximo_id = 200;
// O próximo produto criado terá id = 200.
```

34

Membros de Classe: Outras Considerações

- Java possui apenas declarações de classes: a única forma de escrevermos uma função é como um método em uma classe.

35

this revisitado

Nós vimos que um método estático pode ser chamado diretamente sobre a classe. Ou seja, não é necessário que haja uma instância para chamarmos um método estático. Dessa forma, não faz sentido que o **this** exista dentro de um método estático.

36

Noção de Programa

- Uma vez que tudo o que se escreve em Java são declarações de classes, o conceito de programa também está relacionado a classes: a execução de um programa é, na verdade, a execução de uma classe.
- Executar uma classe significa executar seu método estático **main**. Para ser executado, o método **main** deve possuir uma assinatura bem determinada.

37

Executando uma Classe

- Para que a classe possa ser executada, seu método **main** deve possuir a seguinte assinatura:

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

38

Olá Mundo!

- Usando o método **main** e um atributo estático da classe que modela o sistema, podemos escrever nosso primeiro programa:

```
class Mundo {
  public static void main(String[] args) {
    System.out.println("Olá Mundo!");
  }
}
```

39

Idiossincrasias de Java

- Uma classe deve ser declarada em um arquivo homônimo (case-sensitive) com extensão **.java**.

40

(Tipos Expressões Comandos)

41

Tipos Básicos de Java

- **boolean** **true** ou **false**
- **char** caracter UNICODE (16 bits)
- **byte** número inteiro com sinal (8 bits)
- **short** número inteiro com sinal (16 bits)
- **int** número inteiro com sinal (32 bits)
- **long** número inteiro com sinal (64 bits)
- **float** número em ponto-flutuante (32 bits)
- **double** número em ponto-flutuante (64 bits)

42

Classes Pré-definidas

- Textos
- Vetores

```
String texto = "Exemplo";
int[] lista = {1, 2, 3, 4, 5};
String[] nomes = {"João", "Maria"};

System.out.println(nomes[0]); // Imprime "João".
```

43

Alguns Operadores

```
x = 5;
z = x + 1;  /* z = 6 */
y = x – 1;  /* y = 4 */
y = x++;    /* y = 5 e x = 6 */
y = x--;    /* y = 6 e x = 5 */
y = ++x;    /* y = 6 e x = 6 */
y = --x;    /* y = 5 e x = 5 */
z = x * y;  /* z = 25 */
w = 30 / 4; /* w = 7.5f */
z = 10 % x;  /* z = 0 */
```

44

Mais Operadores

```
k = x == y;   /* k = true porque x=y=5 */
k = x != y;             /* k = false */
k = ((z == 0)&&(y == 5)); /* k = true  */
k = ((z == 0)||(y != x)); /* k = true  */
k = (x >= y);           /* k = true  */
k = (x > y);            /* k = false */
w = x > z ? x : z;      /* w = 5.0f  */
z += 3;               /* z = z + 3 = 3 */
x -= 3;               /* x = x - 3 = 2 */
z /= 3;               /* z = z / 3 = 1 */
x *= 4;               /* x = x * 4 = 8 */
```

45

Expressões

- Tipos de expressões
  - conversões automáticas
  - conversões explícitas

```
byte b = 10;
float f = 0.0F;
```

46

Comandos

- Comando
  - expressão de atribuição
  - formas pré-fixadas ou pós-fixadas de ++ e --
  - chamada de métodos
  - criação de objetos
  - comandos de controle de fluxo
  - bloco
- Bloco = { <lista de comandos> }

47

Controle de Fluxo

- **if-else**
- **switch-case-default**
- **while**
- **do-while**
- **for**
- **break**
- **return**

48

if-else

```
if (a>0 && b>0)
  m = média(a, b);
else
{
  errno = -1;
  m = 0;
}
```

49

switch-case-default

```
int i = f();
switch (i)
{
  case -1:
    ...
    break;
  case 0:
    ...
    break;
  default:
    ...
}
```

50

while

```
int i = 0;
while (i<10)
{
  i++;
  System.out.println(i);
}
```

51

do-while

```
int i = 0;
do
{
  i++;
  System.out.println(i);
}
while (i<10);
```

52

for

```
for (int i=1; i<=10; i++)
  System.out.println(i);
```

53

break

```
int i = 0;
while (true)
{
  if (i==10) break;
  i++;
  System.out.println(i);
}
```

54

label

```
início:
for (int i=0; i<10; i++)
  for (int j=0; j<10; j++)
  {
    if (v[i][j] < 0) break início;
    ...
  }
...
```

55

return

```
int média(int a, int b)
{
  return (a+b)/2;
}
```

56

Próxima Aula de Laboratório

Serão realizados exercícios para fixar os conceitos dados nesta aula.

O primeiro deles será a implementação de uma calculadora que possuirá, apenas, as quatro operações matemáticas básicas.

O segundo será a implementação de um sistema bancário que conterá, apenas, duas classes: banco e conta corrente.

57