Programação Orientada a Objetos

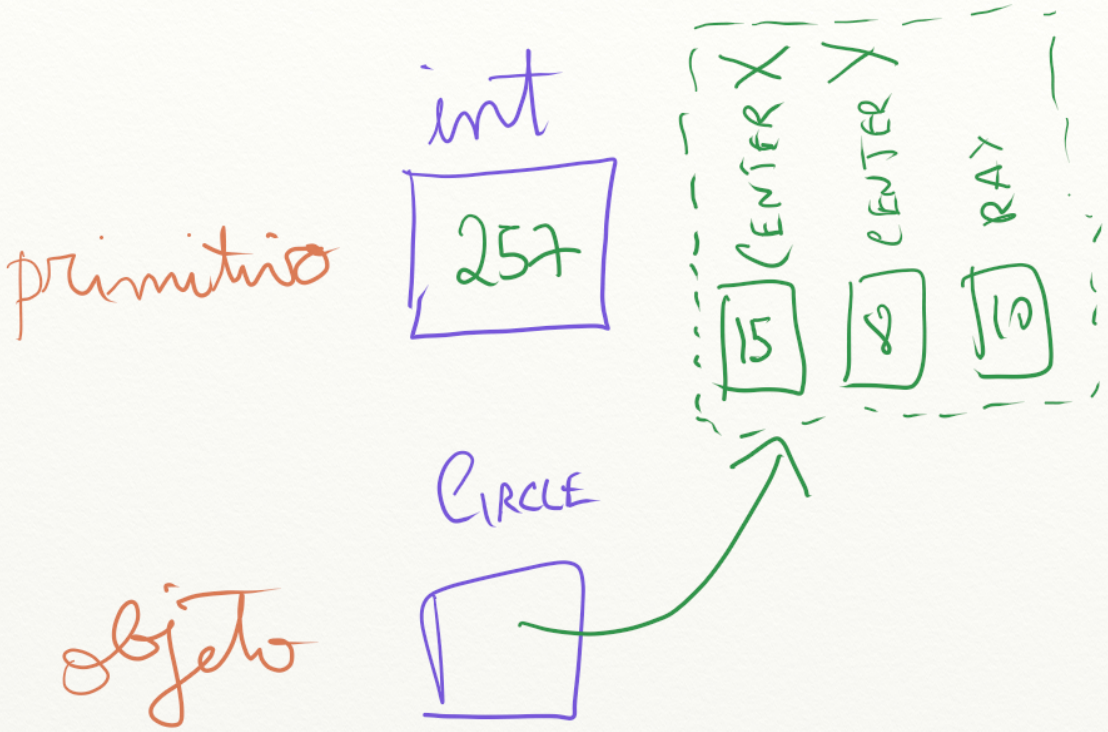
Tipos, Ponteiros e Datatype-Generic Programming

André Santanchè e Oscar Rojas
Instituto de Computação – UNICAMP
Maio 2019

Tipos de Dados Primitivos
x
Classes

Duplo Papel das Variáveis

- Variáveis que contêm dados de tipos **primitivos**
- Variáveis que são ponteiros para **objetos**



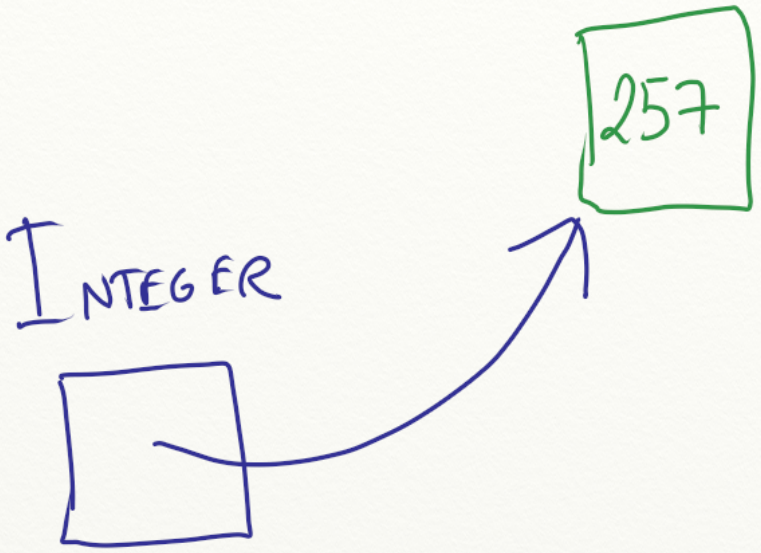
Ponteiros Implícitos

- Java os apresenta de maneira indistinta, sem ponteiros explícitos

```
int valor;
Circle c;
```

Wrapper

- Tipos primitivos tratados como objetos:



Wrapper Classes

Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
long	Integer
float	Float
double	Double
boolean	Boolean

Exercício

- Liste quais as vantagens e desvantagens de representar tipos primitivos na forma de objetos.

Vantagens e desvantagens

- Vantagens
  - Passagem de parametro por referencia
  - Tratamento homogeneo
- Desvantagens
  - Memória
  - Tempo de processamento

Vetor Estático e Dinâmico

Vetor Estático

- Declaração

  `<tipo>[] <declaração`$_1$`>, ..., <declaração`$_n$`>;`

  `<tipo> <declaração`$_1$`>[], ..., <declaração`$_n$`>[];`

  - <declaração>
    - Sintaxe: <nome> = <inicialização>
    - Chaves são usadas para inicializar cada dimensão
    - Ex.: `int primos[] = {1, 2, 3, 5, 7};`

- Quando a inicialização não é inline o vetor ou matriz precisa ser instanciado

  `<nome> = new <tipo>[<tamanho>]`

  - Ex.:

```
int primos[];
primos = new int[5];
```

Vetor Dinâmico Baseado em Classes

- Crescem conforme a demanda
- ArrayList
  - não sincronizada
  - múltiplas rotinas paralelas podem atualizá-lo simultaneamente
- Vector
  - sincronizada
  - Somente uma rotina atualiza de cada vez

Vetor Dinâmico Baseado em Classes

- Crescem conforme a demanda
- ArrayList
  - não sincronizada
  - múltiplas rotinas paralelas podem atualizá-lo simultaneamente
  - **velocidade**
- Vector
  - sincronizada
  - somente uma rotina atualiza de cada vez
  - **consistencia**

Vetor Dinâmico Baseado em Classes

- Crescem conforme a demanda
- ArrayList
  - não sincronizada
  - múltiplas rotinas paralelas podem atualizá-lo simultaneamente
- Vector
  - sincronizada
  - Somente uma rotina atualiza de cada vez

Exercício

- Quais as vantagens e desvantagens das abordagens de `Vector` e `ArrayList`?

Generalidade (Genericity)

Genericity x Inheritance

- **Genericity** – “[…] defining elements that have more than one interpretation. depending on parameters representing types”
- **Inheritance** – “[…] to define elements as extensions or restrictions of previously defined ones.”
- “Both methods apply some form of polymorphism.”

(Meyer, 1986)

Datatype-Generic Programming

- Termo Programação Genérica tem diferentes interpretações de acordo com o contexto:
  - Polimorfismo paramétrico
  - Abstração de dados
  - Meta-programação
  - etc.

(Gibbons, 2007)

Generalidade por Valor

```
System.out.println("===");
System.out.println("==========");
```

```
static void travessao(int tamanho) {
   for (int t = 1; t <= tamanho; t++)
      System.out.print("=");
   System.out.println();
}
```

```
travessao(3);
travessao(10);
```

Generalidade por Tipo

- Forma de usar depende da linguagem
  - ML (1973) – pioneira (Wikipedia, 2015)
  - Ada
  - C++ - templates
  - Java - generics

Java <Generics>

<Generics>

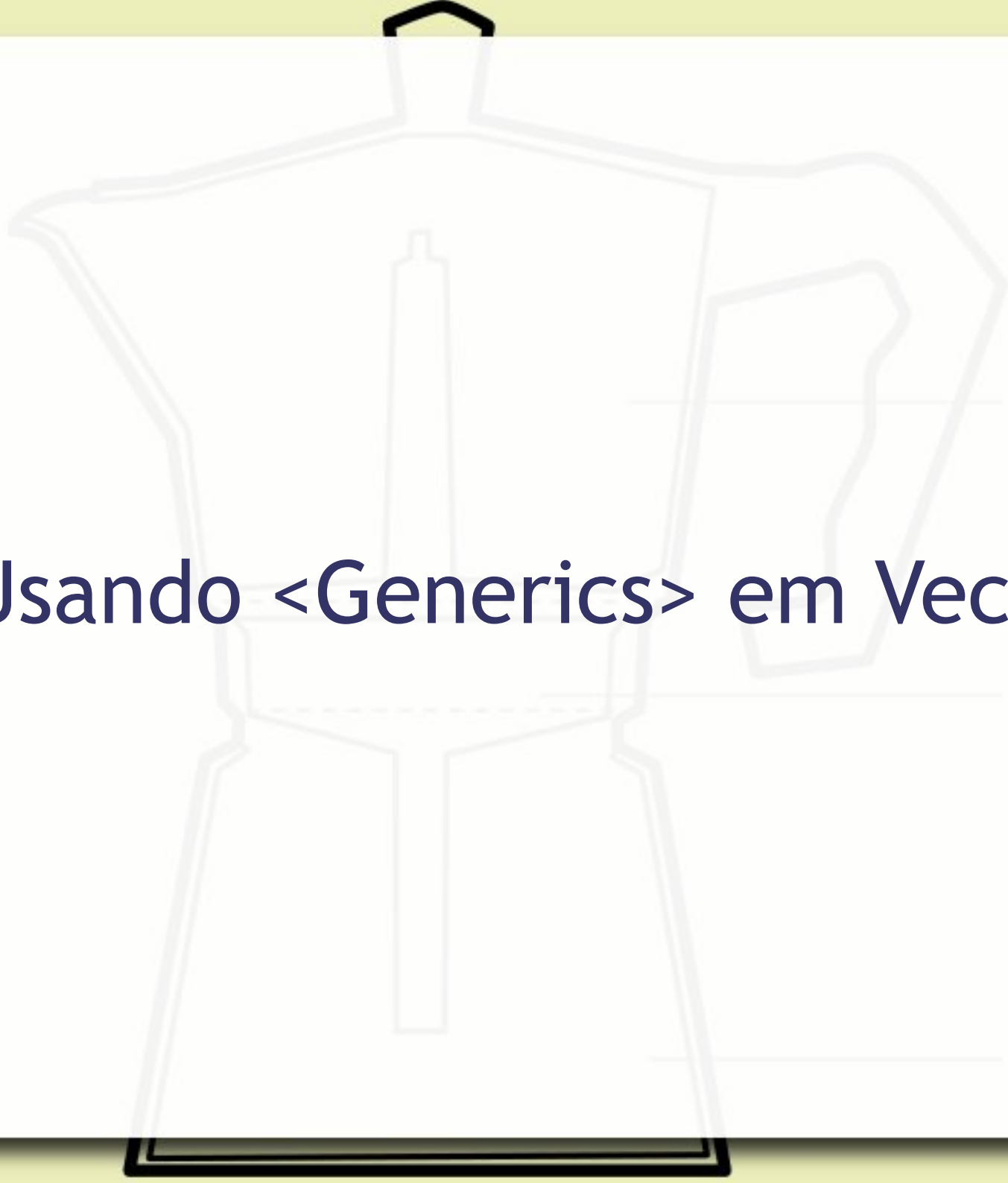
- Introduzido no JDK 1.5
- Permite que programadores declarem sua intenção de tipo
- Possibilita mais verificações em tempo de compilação
  - ajuda a redução de erros no código

(Bracha, 2004) (Sun, 2011)

<Generics>

- Tipos declarados entre < >
- Usado em funções habilitadas para generics
  - Exemplo: Collections

(Bracha, 2004) (Sun, 2011)



Usando <Generics> em Vector

Criando <Generics>

Unconstrained Genericity

- Unconstrained Genericity
  - Sem restrições de tipo recebido (Meyer, 1986)
- Java
  - <Tipo> → somente

Constrained Genericity

- Constrained Genericity
  - Com restrições de tipo recebido (Meyer, 1986)
- Java
  - <Tipo extends Tipo_superior> → somente

Referências

- Bracha, G. (2004). **Generics in the Java Programming Language**. http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf.
- Meyer, B. (1986). **Genericity Versus Inheritance**. SIGPLAN Not., 21(11), 391–405.
- Gibbons, J. (2007). **Datatype-Generic Programming**. In R. Backhouse, J. Gibbons, R. Hinze, & J. Jeuring (Eds.), Datatype-Generic Programming (Vol. 4719, pp. 1–71). Springer Berlin Heidelberg.
- Sun (2011) **The Java Tutorials – Generics**. http://download.oracle.com/javase/tutorial/java/generics/index.html

André Santanchè
http://www.ic.unicamp.br/~santanche

License