Genericidade

Genericidade

- A genericidade pode ser vista como uma forma de introduzir um nível adicional de abstracção nos programas.
- O seu objectivo é permitir que um mesmo pedaço de código possa ser usado em contextos diferentes, evitando-se com isso a repetição.
- Escrever um pedaço de código genérico é perceber o que há de comum a uma família de algoritmos ou estruturas de dados, e abstrair aquilo que é comum daquilo que é variável.

Genericidade

- Precisa-se de classes semelhantes
  - Com operações semelhantes
  - Com atributos semelhantes
  - Variando apenas em alguns tipos (e.g., o tipo dos elementos a guardar em diferentes listas)
- É necessário definir essas classes separadamente?

Decisões de desenho

```
public class Ponto {
    private int x;
    private int y;
    …
}
```

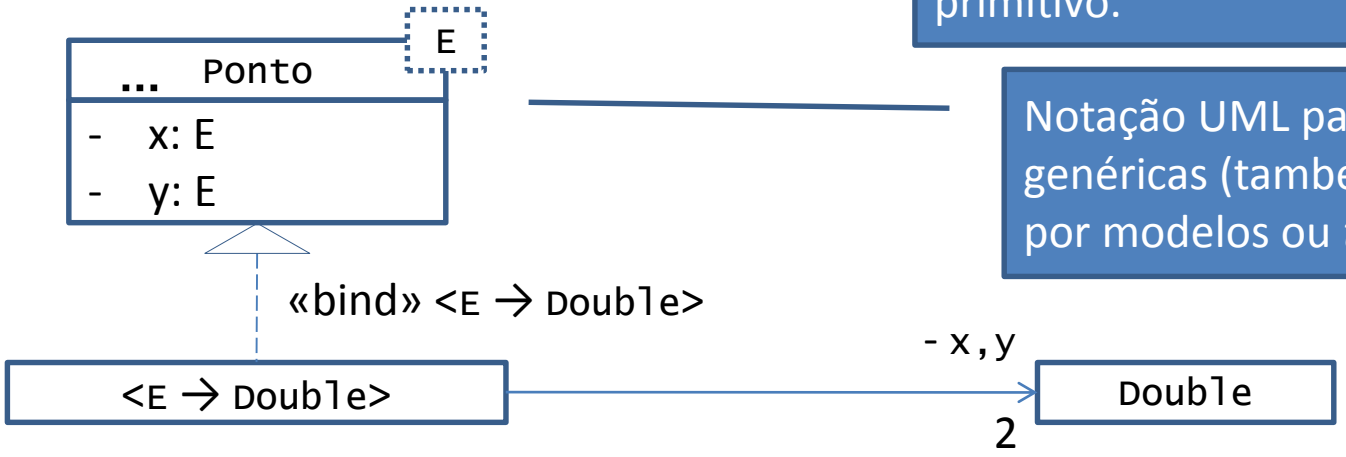
```
public class Ponto {
    private double x;
    private double y;
    …
}
```

Qual a melhor?

Ponto genérico: implementação

```
public class Ponto<E> {
    private E x;
    private E y;
}
```

Classe genérica. E é um parâmetro.
O correspondente argumento tem de ser um tipo.
O parametro não pode ser um tipo primitivo.



Notação UML para classes genéricas (também conhecidas por modelos ou *templates*).

Ponto genérico: utilização

```
Ponto<Integer> a = new Ponto<Integer>(1, 2);

Ponto<Double> b = new Ponto<Double>(2.0, 1.0);
```

Ponto: implementação

```
public class Ponto<E> {

    private E x;
    private E y;

    public Ponto(E x, E y) {

      this.x = x;
      this.y = y;



    }
    …

}
```

Lista genérica?

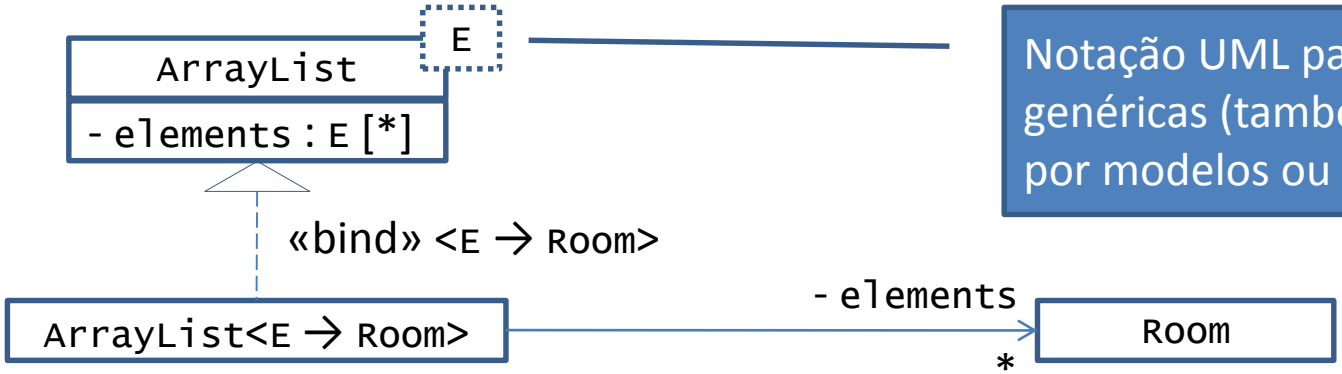
```
public class ArrayList {
    private Object[] elements;

    …
}
```

Lista genérica: implementação

```
public class ArrayList<E> {
    private E[] elements;
    …
}
```

Classe genérica. `E` é um parâmetro. O correspondente argumento tem de ser um tipo. Atenção: não é possível em Java esta implementação, embora seja equivalente à real.



Notação UML para classes genéricas (também conhecidas por modelos ou *templates*).

Lista genérica: utilização

```
public class Level {
    private ArrayList<Room> rooms =
        new ArrayList<Room>();
    public Level(final int numberOfRooms) {
        for (int roomNumber = 1;
             roomNumber != numberOfRooms + 1;
             roomNumber++)
            rooms.addLast(new Room(roomNumber));
    }
    public void show() {
        while (rooms.hasNext())
            out.println(rooms.next());
    }

}
```

Para fazer isto o que é preciso?

Lista ligada genérica: implementação

```
public class ArrayList<E> {
    private E[] elements;
    private numberOfElements;
    public void addLast(E element) {
       if (full()) {
          duplicateSize();
       }
       elements[numberOfElements] = element;
       numberOfElements++;
    }
    …
}
```

Mais informação / Referências

- Y. Daniel Liang, *Introduction to Java Programming*, 7.ª edição, Prentice-Hall, 2008.

Sumário

- Genericidade