Generics

POO

Prof. Marcio Delamaro

O que são

- Tipos genéricos
- São uma forma de definir e utilizar classes de forma genérica
- Dá flexibilidade ao código
- Evita o uso exagerado de casting

Exemplo

- Vamos supor que você queira implementar a estrutura de uma pilha (`push`, `pop`, `isEmpty`) usando um array.

(nem vou pedir para fazerem, trivial)

Pilha implementada

```
public class PilhaInt{
int v[];
int topo;


public PilhaInt(){
  v = new int[100];
  topo = -1;
}


public void push(int b){
  v[topo+1] = b;
  topo++;
}
```

```
public int pop(){
   int b = v[topo];
   topo--;
   return b;
}
```

Exemplo

- Vamos supor que você queira implementar a estrutura de uma pilha (`push`, `pop`, `isEmpty`) usando um array.

- Queremos ter pilhas para diferentes tipos de dados.
- Por exemplo:
  - uma pilha para Integer
  - uma pilha para String
  - uma pilha para Dado

Exemplo

- Vamos supor que você queira implementar a estrutura de uma pilha (`push`, `pop`, `isEmpty`) usando um array.

- Queremos ter pilhas para diferentes tipos de dados.
- Por exemplo:
  - uma pilha para Integer
  - uma pilha para String
  - uma pilha para Dado

Solução:
Implementar uma pilha que se possa colocar qualquer tipo de objeto

Pilha genérica

```
public class Pilha {
   // qualquer objeto pode ser colocado
   Object v[];
   int topo;

   //construtor
   public Pilha() {
       v = new Object[100];
       topo = -1;
   }
```

Pilha genérica

```
public void push(Object b) {
   v[topo+1] = b;
   topo++;
}


public Object pop() {
   Object b = v[topo];
   topo--;
   return b;
}
```

Usando a pilha genérica

```
Pilha p = new Pilha();
Dado d1 = new Dado(),
     d2 = new Dado();
p.push(d1);
p.push(d2);
d1 = (Dado) p.pop();
d2 = (Dado) p.pop();
```

Usando a pilha genérica

```
p.push(d1);
p.push(new Integer(10));
p.push(d2);
d1 = (Dado) p.pop();
d2 = (Dado) p.pop();
```

Usando a pilha genérica

```
p.push(d1);
p.push(new Integer(10));
p.push(d2);
d1 = (Dado) p.pop();
d2 = (Dado) p.pop();
```

Usando a pilha genérica

```
p.push(d1);
p.push(new Integer(10));
p.push(d2);
d1 = (Dado) p.pop();
d2 = (Dado) p.pop();
```

```
Exception in thread "main" java.lang.ClassCastException:
java.lang.Integer cannot be cast to ssc103.bozo.dados.Dado
    at semgeneric.Pilha.main(Pilha.java:43)
```

Usando a pilha genérica

```
p.push(10);
int k = p.pop();
int j = (Integer) p.pop();
```

Usando a pilha genérica

```
p.push(10);
int k = p.pop();
int j = (Integer) p.pop();
```

Conversão automática int → Integer

Usando a pilha genérica

```
p.push(10);
```

Conversão automática int → Integer

```
int k = p.pop();
```

Erro. Conversão Object → int

```
int j = (Integer) p.pop();
```

Usando a pilha genérica

```
p.push(10);
```

Conversão automática int → Integer

```
int k = p.pop();
```

Erro. Conversão Object → int

```
int j = (Integer) p.pop();
```

Conversão automática Integer → int

Resumindo

- A implementação da pilha genérica
  - permite utilizar qualquer tipo de objeto;
  - pode levar a erros pela má utilização, por exemplo, empilhando tipos errados
  - requer um grande número de castings, o que pode poluir oo programa.

Usando *generics*

- O uso de generics permite que se defina uma pilha genérica.
- Qualquer tipo de objeto pode ser colocado nessa pilha.
- Ao se **instanciar** a pilha, pode-se dizer qual o tipo de objeto vai ser empilhado.

Pilha genérica II

```
public class Pilha<T> {
  T v[];
  int topo;


  public Pilha() {
    v = (T[]) new Object[100];
    topo = -1;
  }
```

Pilha genérica II

```
public class Pilha<T> {
   T v[];
   int topo;


   public Pilha() {
      v = (T[]) new Object[100];
      topo = -1;
   }
```

Indica que um tipo vai ser usado como “parâmetro” na criação do objeto.

Pilha genérica II

```
public class Pilha<T> {
  T v[];
  int topo;

  public Pilha() {
    v = (T[]) new Object[100];
    topo = -1;
  }
```

O array a ser utilizado tem todos os elementos desse tipo.

Pilha genérica II

```
public class Pilha<T> {
  T v[];
  int topo;

  public Pilha() {
    v = (T[]) new Object[100];
    topo = -1;
  }
```

O array a ser utilizado tem todos os elementos desse tipo.

Mas é necessário fazer um casting ao alocar o array.

push e pop?

Como ficam ?

push e pop?

```
public void push(T b) {
   v[topo+1] = b;
   topo++;
}
```

push e pop?

```
public void push(T b) {
   v[topo+1] = b;
   topo++;
}
```

push e pop?

```
public void push(T b) {
   v[topo+1] = b;
   topo++;
}

public T pop() {
  T b = v[topo];
  topo--;
  return b;
}
```

Usando a pilha genérica II

```
Pilha<Dado> p = new Pilha<Dado>();

Dado d1 = new Dado(),
     d2 = new Dado();

p.push(d1);

p.push(d2);

d1 = p.pop();

d2 = p.pop();
```

Usando a pilha genérica II

```
Pilha<Integer> p2 = new Pilha<Integer>();
p2.push(10);
p2.push(15);
int k = p2.pop();
```

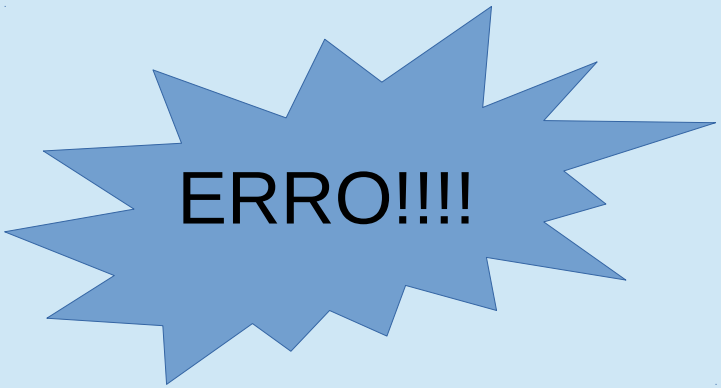
Usando a pilha genérica II

```
Pilha<Integer> p2 = new Pilha<Integer>();
p2.push(10);
p2.push(15);
int k = p2.pop();
```

Conversão int ↔ Integer não requer casting;

Usando a pilha genérica II

```
Pilha<Integer> p2 = new Pilha<Integer>();
p2.push(10);
p2.push(15);
int k = p2.pop();
d1 = new Dado();
p2.push(d1);
```



Totalmente genérica

- Mesmo definindo a pilha usando tipo genérico, é possível criar uma pilha totslmente genérica
- Ou seja, uma pilha que aceite qualquer tipo de objetos
- Todos objetos misturados
- Basta não definir o tipo a ser usado

Totalmente genérica

```
Pilha p3 = new Pilha();
p3.push(10);
p3.push(d1);
p3.push("Genérico");
int j = (Integer) p3.pop();
```

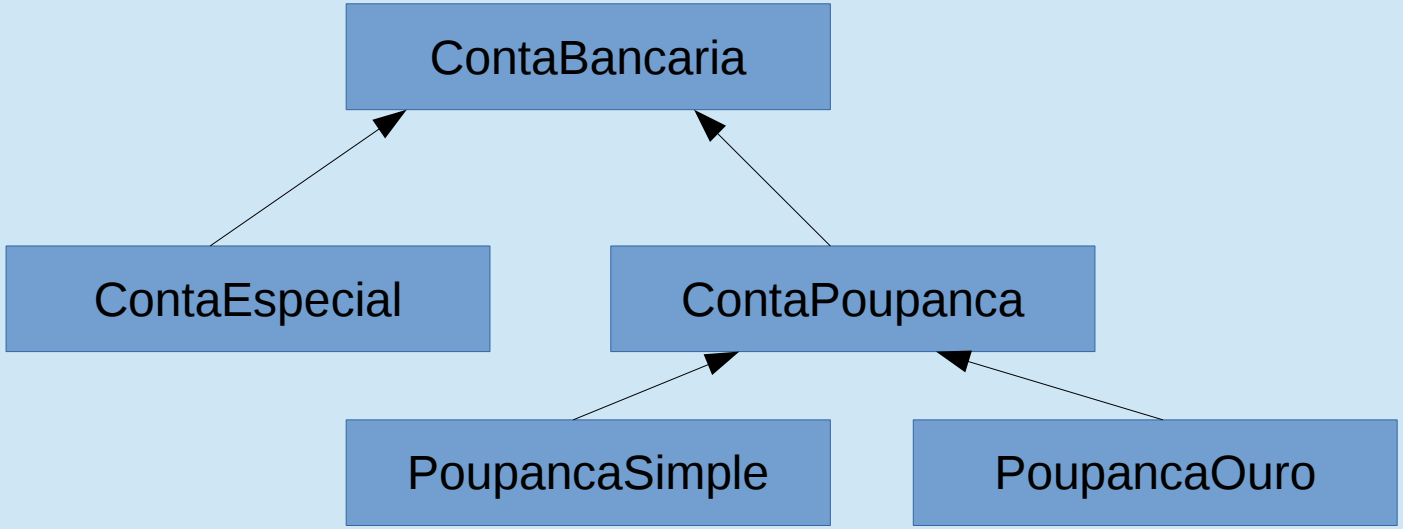
Totalmente genérica

```
Pilha p3 = new Pilha();
p3.push(10);
p3.push(d1);
p3.push("Genérico");
int j = (Integer) p3.pop();
```

O compilador emite advertências mas ainda assim permite o uso.

Restrições de tipo

- Ao declarar a classe genérica é possível restringir o tipo a ser usado
- Por exemplo, queremos criar uma pilha em que os elementos tem que ser uma conta bancária.



Restrições de tipo

```
public class PilhaConta<T extends ContaBancaria> {

  T v[];
  int topo;

  public PilhaConta() {
      v = (T[]) new ContaBancaria[100];
      topo = -1;
  }
```

Restrições de tipo

```
public class PilhaConta<T extends ContaBancaria> {

  T v[];
  int topo;

  public PilhaConta() {
      v = (T[]) new ContaBancaria[100];
      topo = -1;
  }
```

Restrições de tipo – usando

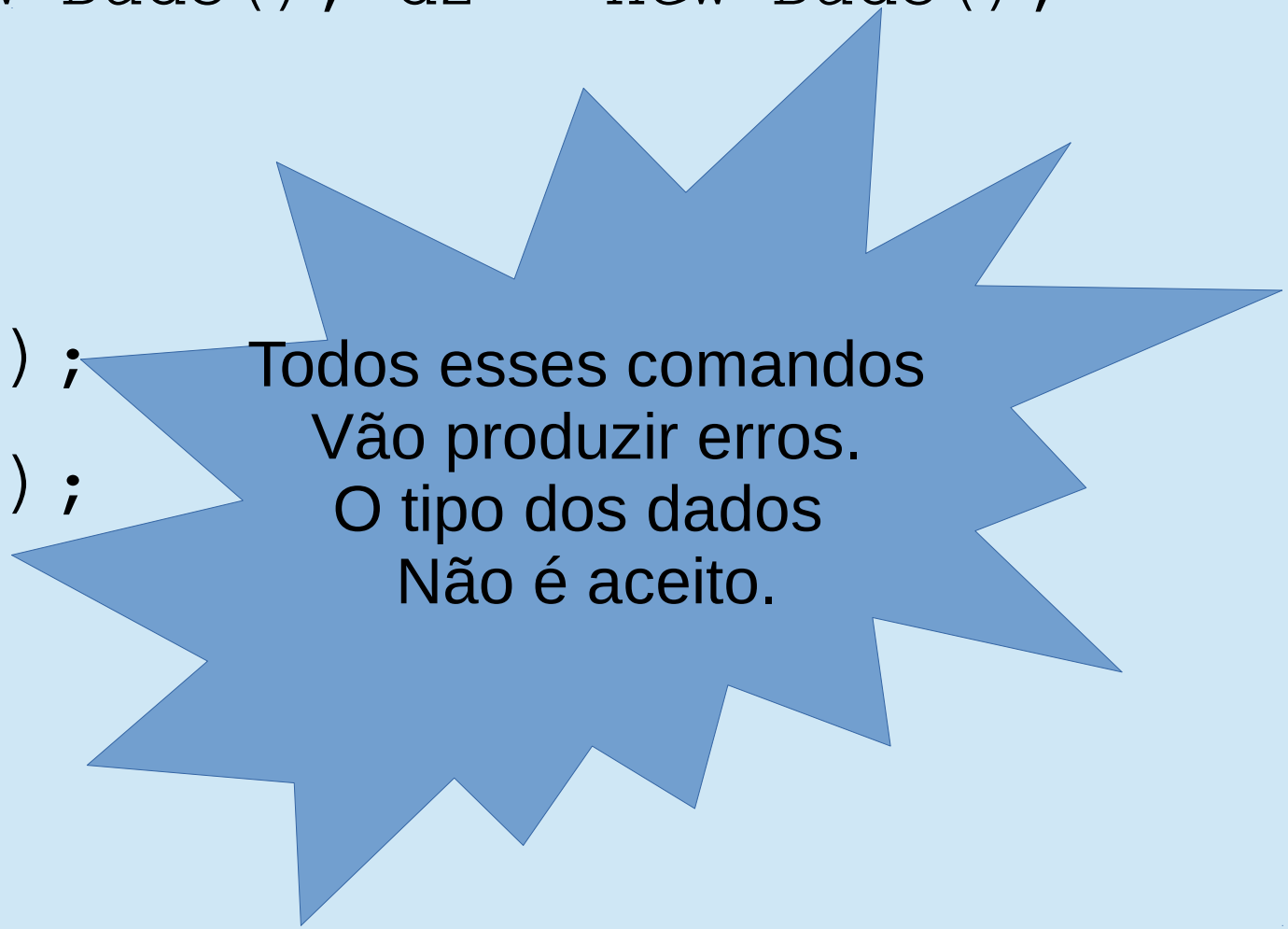
```
PilhaConta<ContaBancaria> p =
     new PilhaConta<ContaBancaria>();


p.push(new PoupancaOuro("Ze da Silva",
15));

p.push(new ContaEspecial("Ze da Silva",
15000));
```

Restrições – usando

```
Dado d1 = new Dado(), d2 = new Dado();
p.push(d1);
p.push(d2);
d1 = p.pop();
d2 = p.pop();
```



Restrições – usando

```
PilhaConta p3 = new PilhaConta();
p3.push(10);
p3.push(d1);
p3.push("Genérico");
```

Mesmo que p3 seja instanciado sem tipo definido

Todos esses comandos Vão produzir erros. O tipo dos dados Não é aceito.