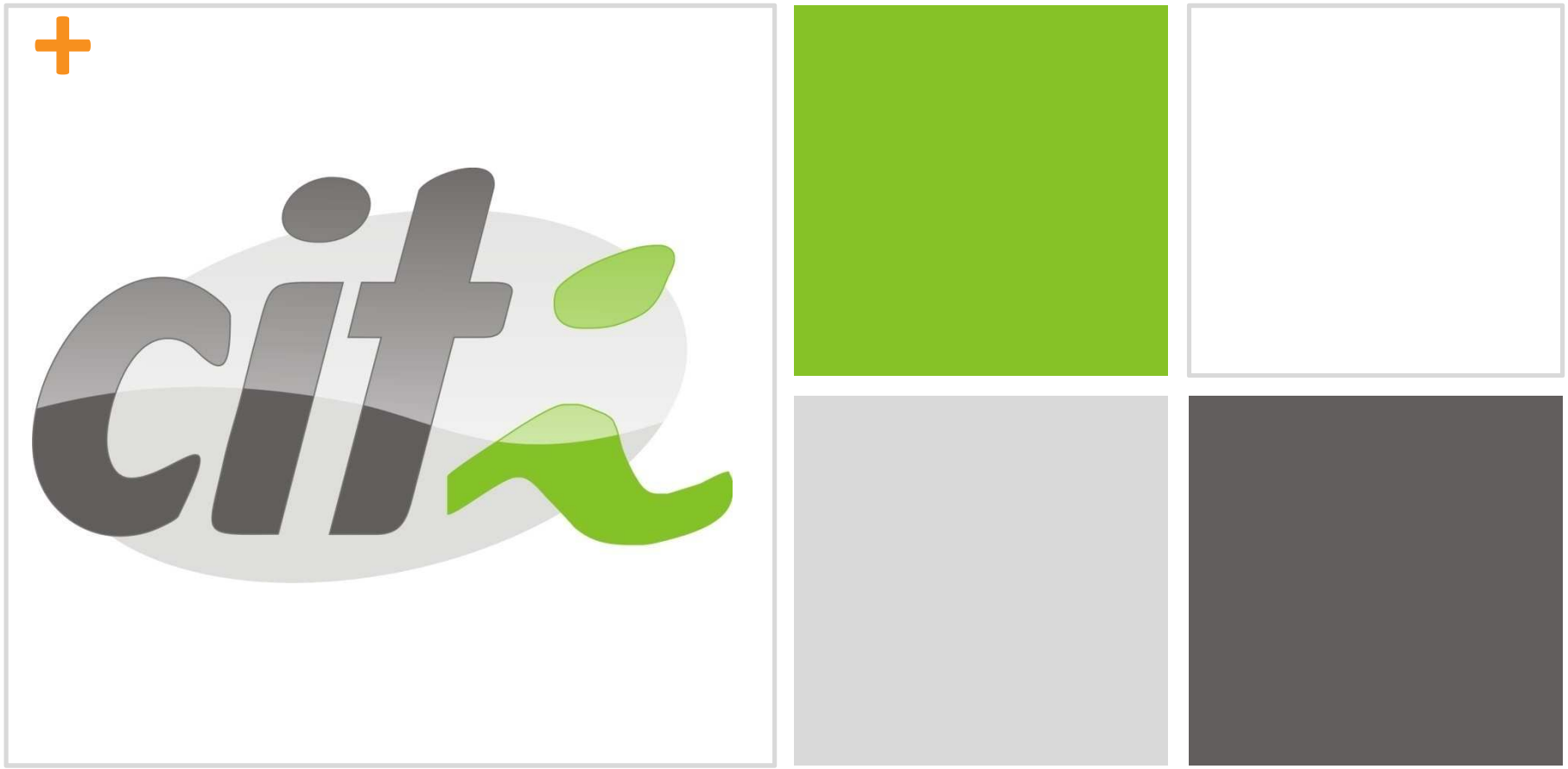


Java Básico

Igor Ebrahim (ies@cin.ufpe.br)

Módulo 9

Pacotes, Tipos Genéricos e Classes Wrappers

Pacotes

- Agrupam classes e interfaces relacionadas
- Facilitam a localização das classes e interfaces
- Associado a um diretório (“pasta”)
  - Estrutura física

Pacotes

Nomes de Pacotes

- Um pacote com o seguinte nome:

org.citi.banco.conta

corresponde ao diretório (relativo):

org/citi/banco/conta

Pacotes

Pacotes e Subdiretórios

- Não existe o conceito de “ subpacote”
- Ou seja: exemplos e exemplos.exemplo11 são pacotes distintos, apesar de estarem “relacionados” fisicamente (como diretórios)

Pacotes

Modificadores de Acesso

- Só pra relembrar...
  - public
    - Visíveis em pacotes diferentes
  - protected
    - Visíveis apenas no pacote onde são declarados
  - “default”
    - Visíveis apenas no pacote onde são declarados
  - private
    - Visíveis apenas dentro da classe onde são declarados

Pacotes

Importação de Pacotes

- Importação de definição de tipo específico:

```
package outro.pacote;
import um.pacote.NomeDaClasse;
/* ... */
```

- Importação de todas as definições de tipo público

```
package outro.pacote;
import um.pacote.*;
/* ... */
```

Pacotes

Estruturando Aplicações

- Agrupar classes relacionadas
  - Implementação
  - Conceitual

- Evite dependência mútua!

```
package a;
import b.*;
/* ... */
```

```
package b;
import a.*;
/* ... */
```

Pacotes

Estruturando Aplicações

- Estruturação típica:
  - Vários pacotes para as classes da interface gráfica (GUI), um para cada conjunto de telas associadas
  - Um pacote para a classe fachada e exceções associadas
  - Um pacote para cada coleção de negócio, incluindo as classes básicas, coleções de dados (repositórios), interfaces e exceções associadas
  - Um pacote util contendo classes auxiliares de propósito geral

Prática

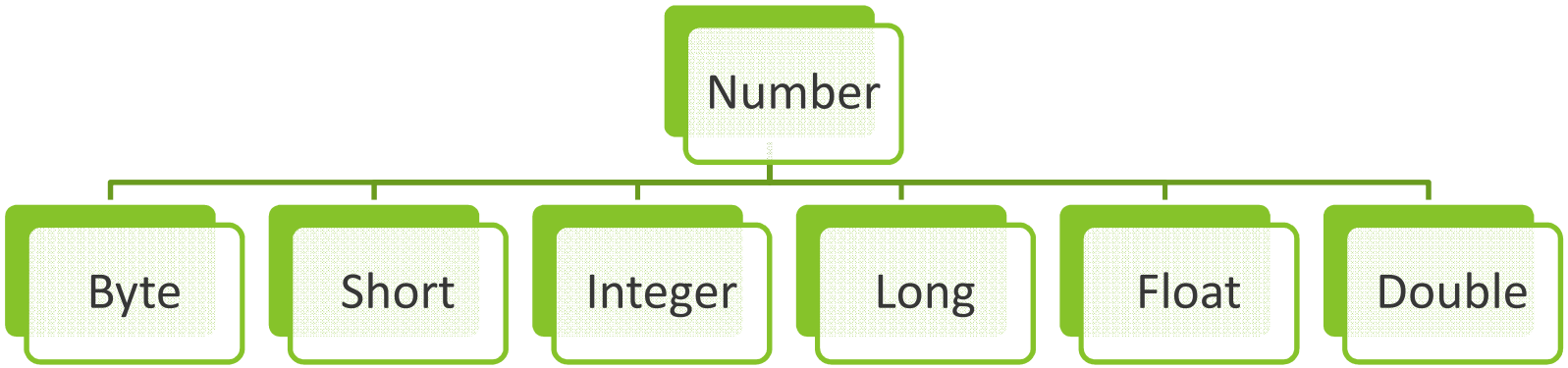
1. Organize os pacotes do seu projeto.

Classes Wrappers

- Classes que representam tipos primitivos de Java
- Utilizadas quando precisamos manipular tipos primitivos como objetos

Classes Wrappers

Hierarquia



Classes Wrappers

Utilizando

- As classes wrappers podem ser utilizadas como tipos primitivos
- Inbox
  - Armazenar implicitamente um primitivo em wrapper
- Outbox
  - Recuperar um primitivo de um wrapper

Classes Wrappers

Utilizando

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

   // Utilização normal
   Integer x = new Integer(10);

   // Inbox
   Integer y = 30;

   // Outbox
   int z = x + y;

}
```

Tipos Genéricos

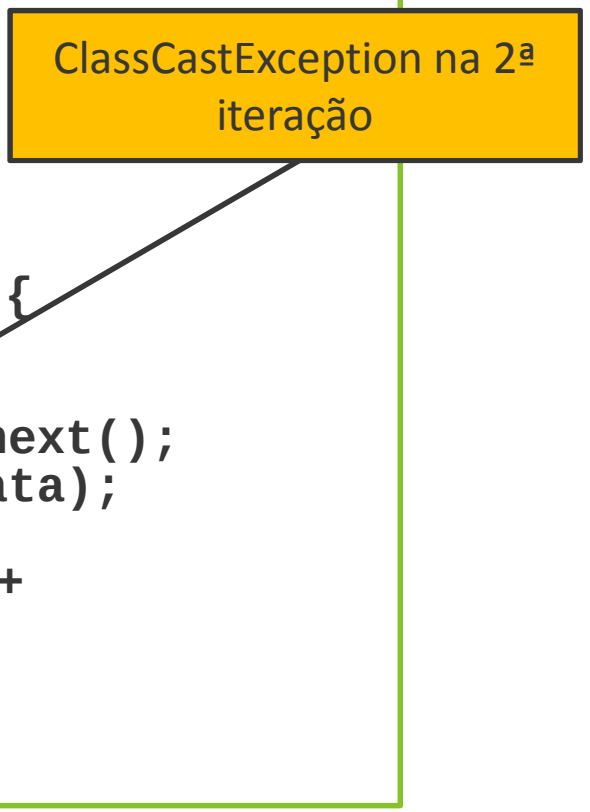
- Métodos e classes genéricas estão entre as capacidades mais poderosas de Java para reutilização de software com segurança de tipo em tempo de compilação;
- Ou seja:
  - Maior facilidade de desenvolvimento;
  - Maior robustez:
    - Cria classes type-safe;
    - Evita o uso extensivo de type casts e instanceof.
- Métodos/classes genéricas permitem que, com uma única declaração de método/classe, o programador especifique um conjunto de métodos/classes;

Tipos Genéricos

Sem Generics

```
Date hoje =  new Date();
List lista = new ArrayList();
lista.add(hoje);
lista.add("10/12/2005");

public void imprimirListaData(List datas){
       Iterator i = datas.iterator();
       while(i.hasNext()){
                     Date data = (Date)i.next();
                     System.out.println(data);
       }
       System.out.println("A lista tem " +
datas.size() + " datas.");
}
```



Tipos Genéricos

Sem Generics

Sem Exception, mas com erro de lógica

```
Date hoje =  new Date();
List lista = new ArrayList();
lista.add(hoje);
lista.add("10/12/2005");

public void imprimirListaData(List datas){
        Iterator i = datas.iterator();
        while(i.hasNext()){
                        Object o = i.next();
                        if(o instanceof Date){
                                Date data = (Date)o;
                                System.out.println(data);
                        }
        }
        System.out.println("A lista tem " + datas.size() +
" datas.");
}
```

Tipos Genéricos

Com Generics

```
Date hoje =  new Date();
List<Date> lista = new ArrayList<Date>();
lista.add(hoje);
lista.add("10/12/2005");

public void imprimirListaData(List<Date> datas){
        Iterator<Date> i = datas.iterator();
        while(i.hasNext()){
                        Date data = i.next();
                        System.out.println(data);
        }
        System.out.println("A lista tem " + datas.size() + "
datas.");
}
```

Erro de compilação

Métodos Genéricos

- Considere três métodos printArray sobrecarregados → esses métodos imprimem as representações de string dos elementos de um array de Integers, um array de Doubles e um array de Characters, respectivamente:
- Apenas Tipos Referências podem ser usados com Generics

Métodos Genéricos

Sem Generics

```
public void printArray(Integer[] inputArray) {
        for (Integer element : inputArray)
                System.out.print(element + " ");
}
```

```
public void printArray(Double[] inputArray) {
        for (Double element : inputArray)
                System.out.print(element + " ");
}
```

```
public void printArray(Character[] inputArray) {
        for (Character element : inputArray)
                System.out.print(element + " ");
}
```

Métodos Genéricos

Com Generics

Seção de parâmetro de tipo

```
public <E> void printArray( E[] inputArray ) {
        for ( E element : inputArray )
                System.out.print(element + " ");
}
```

- Muito menos código
- Cada seção de parâmetro de tipo pode conter um ou mais parâmetros de tipo separados por vírgulas
- É recomendado que parâmetros de tipos sejam especificados como letras maiúsculas individuais

Classe Genérica

ou Parametrizada

```
public class Registro<K,V>{

        private K chave;
        private V valor;

        Registro(K chave,V valor){
                this.chave = chave;
                this.valor = valor;
        }

}
```

Argumentos de tipo

```
...
Registro<String, Integer> registro =
        new Registro<String, Integer>(“um”, new Integer(1));
...
```

Classe Genérica

Limite

```
public class Registro<K,V extends Number>{
        private K chave;
        private V valor;

        Registro(K chave,V valor){
                this.chave = chave;
                this.valor = valor;
        }
}
```

Com classe ou interface usamos a palavra reservada extends

```
//OK
Registro<String, Double> registro =
        new Registro<String, Double >("um",new Double(1.0));
//Erro de compilação
Registro<String, Endereco> registro =
        new Registro<String, Endereco >("um",new Endereco());
```

Genéricos

Curinga

- Suponha que você queira implementar um método genérico sum que some os número em uma coleção, como o ArrayList
- Queremos ser capazes de somar todos os números na ArrayList independente dos seus tipos

```
public double sum( ArrayList< Number > list ) {
    double total = 0;
    for ( Number element : list )
          total = total + element.doubleValue();
    return total;
}
```

Será que esse método funcionaria se passássemos como parâmetro um ArrayList< Integer > ?

Genéricos

Curinga

- Ao compilar o programa, o compilador emitiria a seguinte mensagem de erro:
  - sum(java.util.ArrayList<java.lang.Number>) in NomeDaClasse cannot be applied to (java.util.ArrayList<java.lang.Integer>)
- Embora Number seja superclasse de Integer, ArrayList<Number> não é superclasse de ArrayList<Integer>
- Como criar uma versão mais flexível?
  - Usando os argumentos de tipo curinga

Genéricos

Curinga

- Os curingas permitem especificar:
  - parâmetros de método,
  - valores de retorno,
  - variáveis,
  - campos,
  - etc.
- Atuam como supertipos de tipos parametrizados

Genéricos

Curinga

```
public double sum ( ArrayList< ? extends Number > list ) {
     double total = 0;

     for ( Number element : list )
         total = total + element.doubleValue();

     return total;
}
```

- Utilizar um curinga na seção de parâmetros de tipo de um método ou utilizar um curinga como um tipo explícito de uma variável no corpo do método é um erro de sintaxe

Prática

1. Refatore seu sistema bancário de forma que seja minimizada a repetição de código entre classes de repositório utilizando o conhecimento sobre classes e métodos genéricos.