Programação Orientada a Objetos

Programação com Tipos Genéricos

www.comp.ita.br/~pauloac/ces22/

Tipos Genéricos

Sumário

- **A necessidade dos Genéricos**
- **Métodos Genéricos**
- **Classes Genéricas**
- **Utilização de wildcards**
- **Métodos com número variável de parâmetros**

Introdução – Métodos Sobrecarregados

```
// Utilizando métodos sobrecarregados para imprimir um array de
    diferentes tipos.
public class OverloadedMethods
{
   // método printArray para imprimir um array de Integer
   public static void printArray( Integer[] inputArray )
   {
      // exibe elementos do array
      for ( Integer element : inputArray )
         System.out.printf( "%s ", element );

      System.out.println();
   } // fim do método printArray
```

Introdução – Métodos Sobrecarregados 2

```
// método printArray para imprimir um array de Double
   public static void printArray( Double[] inputArray )
   {
      // exibe elementos do array
      for ( Double element : inputArray )
         System.out.printf( "%s ", element );
      System.out.println();
   } // fim do método printArray

   // método printArray para imprimir um array de Character
   public static void printArray( Character[] inputArray )
   {
      // exibe elementos do array
      for ( Character element : inputArray )
         System.out.printf( "%s ", element );

      System.out.println();
   } // fim do método printArray
```

Introdução – Métodos Sobrecarregados - 3

```
public static void main( String args[] )
  {
     // cria arrays de Integer, Double e Character
     Integer[] integerArray = { 1, 2, 3, 4, 5, 6 };
     Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
     Character[] characterArray = { 'H', 'E', 'L', 'L', 'O' };

     System.out.println( "Array integerArray contains:" );
     printArray( integerArray ); // passa um array de Integers
     System.out.println( "\nArray doubleArray contains:" );
     printArray( doubleArray ); // passa um array Doubles
     System.out.println( "\nArray characterArray contains:" );
     printArray( characterArray ); // passa um array de Characters
  } // fim de main
} // fim da classe OverloadedMethods
```

Resultado – Métodos Sobrecarregados

Array integerArray contains:

1 2 3 4 5 6

Array doubleArray contains:

1.1 2.2 3.3 4.4 5.5 6.6 7.7

Array characterArray contains:

H E L L O

Problema

- Implementação da mesma idéia sobre diferentes tipos de dados, leva a código repetitivo. Mais difícil de manter e propenso a erros (copy and paste)

Método Genérico

```
// Utilizando métodos genéricos para imprimir diferentes tipos de
    arrays.
public class GenericMethodTest
{
   // método genérico printArray
   public static < E > void printArray( E[] inputArray )
   {
      // exibe elementos do array
      for ( E element : inputArray )
         System.out.printf( "%s ", element );

      System.out.println();
   } // fim do método printArray
```

Método Genérico - 2

```
public static void main( String args[] )
   {
      // cria arrays de Integer, Double e Character
      Integer[] integerArray = { 1, 2, 3, 4, 5, 6 };
      Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
      Character[] characterArray = { 'H', 'E', 'L', 'L', 'O' };
      System.out.println( "Array integerArray contains:" );
      printArray( integerArray ); // passa um array de Integers
      System.out.println( "\nArray doubleArray contains:" );
      printArray( doubleArray ); // passa um array Doubles
      System.out.println( "\nArray characterArray contains:" );
      printArray( characterArray ); // passa um array de Characters
   } // fim de main
} // fim da classe GenericMethodTest
```

Resultado – Método Genérico

Array integerArray contains:

1 2 3 4 5 6

Array doubleArray contains:

1.1 2.2 3.3 4.4 5.5 6.6 7.7

Array characterArray contains:

H E L L O

Método Genérico

- É possível restringir quais classes podem ser utilizadas nos métodos genéricos

- Por exemplo, a expressão abaixo restringe a classes que herdam de Number
  - **< T extends Number>**

Exemplo 2 de Método Genérico

```
public class MaximumTest{
  // determina o maior dos três objetos Comparable
  public static < T extends Comparable< T > > T maximum( T x, T y, T z )
    {
     T max = x; // supõe que x é inicialmente o maior

     if ( y.compareTo( max ) > 0 )
        max = y; // y é o maior até agora

     if ( z.compareTo( max ) > 0 )
        max = z; // z é o maior

     return max; // retorna o maior objeto
  } // fim do método Maximum
```

Exemplo 2 de Método Genérico

```
public static void main( String args[] )
{
   System.out.printf( "Maximum of %d, %d and %d is %d\n\n", 3, 4, 5,
      maximum( 3, 4, 5 ));
   System.out.printf( "Maximum of %.1f, %.1f and %.1f is %.1f\n\n",
      6.6, 8.8, 7.7, maximum( 6.6, 8.8, 7.7 ));
   System.out.printf( "Maximum of %s, %s and %s is %s\n", "pear",
      "apple", "orange", maximum( "pear", "apple", "orange" ));
} // fim de main
} // fim da classe MaximumTest
```

Resultado – Exemplo 2 de Método Genérico

Maximum of 3, 4 and 5 is 5

Maximum of 6,6, 8,8 and 7,7 is 8,8

Maximum of pear, apple and orange is pear

Classe Genérica

- Classe Genérica
  - Criando uma classe que trabalha com vários tipos de classes

Exemplo Classe Genérica – Pilha

```
public class Stack< E >    {
   private final int size; // número de elementos na pilha
   private int top; // localização do elemento superior
   private E[] elements; // array que armazena elementos na pilha

   // construtor sem argumento cria uma pilha do tamanho padrão
   public Stack()   {
      this( 10 ); // tamanho padrão da pilha
   } // fim do construtor sem argumentos da classe Stack

   // construtor cria uma pilha com o número especificado de elementos
   public Stack( int s )    {
      size = s > 0 ? s : 10; // configura o tamanho da Stack
      top = -1; // Stack inicialmente vazia

      elements = ( E[] ) new Object[ size ]; // cria o array
   } // fim do construtor de Stack
```

Exemplo Classe Genérica – Pilha - 2

```
// insere o elemento na pilha; se bem-sucedido retorna true;
   // caso contrário, lança uma FullStackException
   public void push( E pushValue )    {
      if ( top == size - 1 ) // se a pilha estiver cheia
         throw new FullStackException( String.format(
            "Stack is full, cannot push %s", pushValue ) );

      elements[ ++top ] = pushValue; // insere pushValue na Stack
   } // fim do método push
   // retorna o elemento superior se não estiver vazia; do contrário lança uma
      EmptyStackException
   public E pop()    {
      if ( top == -1 ) // se pilha estiver vazia
         throw new EmptyStackException( "Stack is empty, cannot pop" );

      return elements[ top-- ]; // remove e retorna o elemento superior da Stack
   } // fim do método pop
} // fim da classe Stack <E>
```

Exemplo de uso da Classe Genérica – Pilha – StackTest -1/6

```
public class StackTest   {
   private double[] doubleElements = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6 };
   private int[] integerElements = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };

   private Stack< Double > doubleStack; // a pilha armazena objetos Double
   private Stack< Integer > integerStack; // a pilha armazena objetos Integer

   // testa objetos Stack
   public void testStacks()
   {
      doubleStack = new Stack< Double >( 5 ); // Stack de Doubles
      integerStack = new Stack< Integer >( 10 ); // Stack de Integers

      testPushDouble(); // insere doubles em doubleStack
      testPopDouble(); // remove de doubleStack
      testPushInteger(); // insere ints em intStack
      testPopInteger(); // remove de intStack
   } // fim do método testStacks
```

Exemplo Classe Genérica – Pilha – StackTest – 2/6

```
// testa o método push com a pilha de doubles
   public void testPushDouble()    {
      // insere elementos na pilha
      try      {
         System.out.println( "\nPushing elements onto doubleStack" );
         // insere elementos na Stack
         for ( double element : doubleElements )        {
            System.out.printf( "%.1f ", element );
            doubleStack.push( element ); // insere em doubleStack
         } // fim do for
      } // fim do try
      catch ( FullStackException fullStackException ) {
         System.err.println();
         fullStackException.printStackTrace();
      } // fim da captura de FullStackException
   } // fim do método testPushDouble
```

Exemplo Classe Genérica – Pilha – StackTest – 3/6

```
// testa o método pop com a pilha de doubles
   public void testPopDouble()    {
      // Retira elementos da pilha
      try       {
         System.out.println( "\nPopping elements from doubleStack" );
         double popValue; // armazena o elemento removido da pilha
         // remove todos os elementos da Stack
         while ( true ) {
            popValue = doubleStack.pop(); // remove de doubleStack
            System.out.printf( "%.1f ", popValue );
         } // fim do while
      } // fim do try
      catch( EmptyStackException emptyStackException )       {
         System.err.println();         emptyStackException.printStackTrace();
      } // fim da captura de EmptyStackException
   } // fim do método testPopDouble
```

Exemplo Classe Genérica – Pilha - StackTest – 4/6

```
// testa o método pop com a pilha de integers
   public void testPopInteger()  {
      // remove elementos da pilha
      try      {
         System.out.println( "\nPopping elements from integerStack" );
         int popValue; // armazena o elemento removido da pilha
         // remove todos os elementos da Stack
         while ( true )          {
            popValue = integerStack.pop(); // remove de integerStack
            System.out.printf( "%d ", popValue );
         } // fim do while
      } // fim do try
      catch( EmptyStackException emptyStackException )      {
         System.err.println();         emptyStackException.printStackTrace();
      } // fim da captura de EmptyStackException
   } // fim do método testPopInteger
```

Exemplo Classe Genérica – Pilha - StackTest – 5/6

```
// testa o método push com a pilha de integers
  public void testPushInteger()   {
     // insere elementos na pilha
     try      {
        System.out.println( "\nPushing elements onto integerStack" );
        // insere elementos na Stack
        for ( int element : integerElements )        {
           System.out.printf( "%d ", element );
           integerStack.push( element ); // insere em integerStack
        } // fim do for
     } // fim do try
     catch ( FullStackException fullStackException )      {
        System.err.println();        fullStackException.printStackTrace();
     } // fim da captura de FullStackException
  } // fim do método testPushInteger
```

Exemplo Classe Genérica – Pilha - StackTest – 6/6

```
public static void main( String args[] )
  {
     StackTest application = new StackTest();
     application.testStacks();
  } // fim de main
} // fim da classe StackTest
```

Exemplo Classe Genérica – Pilha - FullStackException

```
public class FullStackException extends RuntimeException {
   // construtor sem argumento
   public FullStackException()   {
      this( "Stack is full" );
   } // fim do construtor sem argumentos de FullStackException

   // construtor de um argumento
   public FullStackException( String exception )   {
      super( exception );
   } // fim do construtor de FullStackException de um argumento
} // fim da classe FullStackException
```

Exemplo Classe Genérica – Pilha - EmptyStackException

```
public class EmptyStackException extends RuntimeException
{
   // construtor sem argumento
   public EmptyStackException()
   {
      this( "Stack is empty" );
   } // fim do construtor sem argumentos de EmptyStackException

   // construtor de um argumento
   public EmptyStackException( String exception )
   {
      super( exception );
   } // fim do construtor de um argumento de EmptyStackException
} // fim da classe EmptyStackException
```

Resultado – Classe genérica Stack – 1/2

```
Pushing elements onto doubleStack
1,1 2,2
StackExample.FullStackException: Stack is full, cannot push 6.6
at StackExample.Stack.push(Stack.java:31)
at StackExample.StackTest.testPushDouble(StackTest.java:37)
at StackExample.StackTest.testStacks(StackTest.java:19)
at StackExample.StackTest.main(StackTest.java:118)

StackExample.EmptyStackException: Stack is empty, cannot pop
at StackExample.Stack.pop(Stack.java:41)
at StackExample.StackTest.testPopDouble(StackTest.java:59)
at StackExample.StackTest.testStacks(StackTest.java:20)
at StackExample.StackTest.main(StackTest.java:118)
```

Resultado – Classe genérica Stack – 2/2

StackExample.FullStackException: Stack is full, cannot push 11
at StackExample.Stack.push(Stack.java:31)
at StackExample.StackTest.testPushInteger(StackTest.java:82)
at StackExample.StackTest.testStacks(StackTest.java:21)3,3 4,4 5,5 6,6

Popping elements from doubleStack
5,5 4,4 3,3 2,2 1,1
Pushing elements onto integerStack
1 2 3 4 5 6 7 8 9 10 11
at StackExample.StackTest.main(StackTest.java:118)
StackExample.EmptyStackException: Stack is empty, cannot pop
at StackExample.Stack.pop(Stack.java:41)
at StackExample.StackTest.testPopInteger(StackTest.java:104)
at StackExample.StackTest.testStacks(StackTest.java:22)
at StackExample.StackTest.main(StackTest.java:118)
Popping elements from integerStack
10 9 8 7 6 5 4 3 2 1

Métodos Genéricos e Classes Genéricas

- StackTest tem métodos praticamente idênticos: testPushInteger, testPushDouble, testPopInteger,testPopDouble

- Porque não usar Métodos Genéricos para testar uma classe genérica

Métodos e Classes Genéricas – StackTest2

```
public class StackTest2  {
   private Double [] doubleElements = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6 };
   private Integer [] integerElements =  { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };
   private Stack< Double > doubleStack; // a pilha armazena objetos Double
   private Stack< Integer > integerStack; // a pilha armazena objetos Integer

   // testa objetos Stack
   public void testStacks()    {
      doubleStack = new Stack< Double >( 5 ); // Stack de Doubles
      integerStack = new Stack< Integer >( 10 ); // Stack de Integers
      testPush( "doubleStack", doubleStack, doubleElements );
      testPop( "doubleStack", doubleStack );
      testPush( "integerStack", integerStack, integerElements );
      testPop( "integerStack", integerStack );
   } // fim do método testStacks
```

Métodos e Classes Genéricas – StackTest2- testPush

```
// método genérico testPush insere elementos em uma Stack
   public < T > void testPush( String name, Stack< T > stack,    T[]
     elements )  {
      // insere elementos na pilha
      try       {
         System.out.printf( "\nPushing elements onto %s\n", name );
         // insere elementos na Stack
         for (T element : elements)       {
            System.out.printf( "%s ", element );
            stack.push( element ); // insere o elemento na pilha
         }
      } // fim do try
      catch ( FullStackException fullStackException )      {
         System.out.println();         fullStackException.printStackTrace();
      } // fim da captura de FullStackException
   } // fim do método testPush
```

Métodos e Classes Genéricas – StackTest2 - testPop

```
// método genérico testPop remove elementos de uma Stack
  public < T > void testPop( String name, Stack< T > stack )   {
    // remove elementos da pilha
    try     {
      System.out.printf( "\nPopping elements from %s\n", name );
      T popValue; // armazena o elemento removido da pilha
      // remove elementos da Stack
      while ( true )        {
        popValue = stack.pop(); // remove da pilha
        System.out.printf( "%s ", popValue );
      } // fim do while
    } // fim do try
    catch( EmptyStackException emptyStackException )     {
      System.out.println();        emptyStackException.printStackTrace();
    } // fim da captura de EmptyStackException
  } // fim do método testPop
```

Métodos e Classes Genéricas – StackTest2 - main

```
public static void main( String args[] )
   {
      StackTest2 application = new
   StackTest2();
      application.testStacks();
   } // fim de main
} // fim da classe StackTest2
```

Resultado – StackTest2 – 1/2

Stack2Example.FullStackException: Stack is full, cannot push 6.6
at Stack2Example.Stack.push(Stack.java:31)
at Stack2Example.StackTest2.testPush(StackTest2.java:39)
at Stack2Example.StackTest2.testStacks(StackTest2.java:20)
at Stack2Example.StackTest2.main(StackTest2.java:75)
Stack2Example.EmptyStackException: Stack is empty, cannot pop
at Stack2Example.Stack.pop(Stack.java:41)
at Stack2Example.StackTest2.testPop(StackTest2.java:61)
at Stack2Example.StackTest2.testStacks(StackTest2.java:21)
at Stack2Example.StackTest2.main(StackTest2.java:75)
Stack2Example.FullStackException: Stack is full, cannot push 11
at Stack2Example.Stack.push(Stack.java:31)
at Stack2Example.StackTest2.testPush(StackTest2.java:39)
at Stack2Example.StackTest2.testStacks(StackTest2.java:22)
at Stack2Example.StackTest2.main(StackTest2.java:75)

Resultado – StackTest2 – 2/2

Pushing elements onto doubleStack
1.1 2.2 3.3 4.4 5.5 6.6

Popping elements from doubleStack
5.5 4.4 3.3 2.2 1.1

Pushing elements onto integerStack
1 2 3 4 5 6 7 8 9 10 11

Popping elements from integerStack
10 9 8 7 6 5 4 3 2 1
Stack2Example.EmptyStackException: Stack is empty, cannot pop
at Stack2Example.Stack.pop(Stack.java:41)
at Stack2Example.StackTest2.testPop(StackTest2.java:61)
at Stack2Example.StackTest2.testStacks(StackTest2.java:23)
at Stack2Example.StackTest2.main(StackTest2.java:75)

Tipos Brutos “Raw Types”

- Tipos brutos (“raw types”) são classes parametrizadas que são instanciadas sem parâmetros. Ex.:
  - Stack s=new Stack(10);

- Nesse caso, o parâmetro omitido é assumido como Object.

- É possível fazer referências de tipos não brutos para brutos e vice-versa, porém não é recomendável e geram avisos do compilador
  - Stack tipoBruto=new Stack<String>(10);
  - Stack<Integer> tipoEspecifico=new Stack(20);

Exemplo - Tipos Raw – 1/5

```
public class RawTypeTest  {
   private Double[] doubleElements = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6 };
   private Integer[] integerElements =
      { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 };
   // método para testar classes Stack com tipos brutos
   public void testStacks()    {
      // Pilha de tipos brutos atribuídos à classe Stack da variável de tipos brutos
      Stack rawTypeStack1 = new Stack( 5 );
      // Stack< Double > atribuído a Stack da variável de tipos brutos
      Stack rawTypeStack2 = new Stack< Double >( 5 );
      // Pilha de tipos crus atribuídos à variável Stack< Integer >
      Stack< Integer > integerStack = new Stack( 10 );
```

Exemplo - Tipos Raw – 2/5

```
//continuação método testStacks
     testPush( "rawTypeStack1", rawTypeStack1, doubleElements );
      testPop( "rawTypeStack1", rawTypeStack1 );
      testPush( "rawTypeStack2", rawTypeStack2, doubleElements );
      testPop( "rawTypeStack2", rawTypeStack2 );
      testPush( "integerStack", integerStack, integerElements );
      testPop( "integerStack", integerStack );
   } // fim do método testStacks
```

Exemplo - Tipos Raw – 3/5

```
// método genérico insere elementos na pilha
   public < T > void testPush( String name, Stack< T > stack,
      T[] elements )    {
      // insere elementos na pilha
      try      {
         System.out.printf( "\nPushing elements onto %s\n", name );
         // insere elementos na Stack
         for ( T element : elements )         {
            System.out.printf( "%s ", element );
            stack.push( element ); // insere o elemento na pilha
         } // fim do for
      } // fim do try
      catch ( FullStackException fullStackException )     {
         System.out.println();        fullStackException.printStackTrace();
      } // fim da captura de FullStackException
   } // fim do método testPush
```

Exemplo - Tipos Raw – 4/5

```
// método genérico testPop remove elementos da pilha
  public < T > void testPop( String name, Stack< T > stack )   {
     // remove elementos da pilha
     try      {
        System.out.printf( "\nPopping elements from %s\n", name );
        T popValue; // armazena o elemento removido da pilha
        // remove elementos da Stack
        while ( true )        {
           popValue = stack.pop(); // remove da pilha
          System.out.printf( "%s ", popValue );
        } // fim do while
     } // fim do try
     catch( EmptyStackException emptyStackException ) {
        System.out.println();        emptyStackException.printStackTrace();
     } // fim da captura de EmptyStackException
  } // fim do método testPop
```

Exemplo - Tipos Raw – 5/5

```
public static void main( String args[] )
   {
      RawTypeTest application = new RawTypeTest();
      application.testStacks();
   } // fim de main
} // fim da classe RawTypeTest
```

Resultado – Tipos Raw – 1/3

Pushing elements onto rawTypeStack1

1.1 2.2 3.3 4.4 5.5 6.6

Popping elements from rawTypeStack1

5.5 4.4 3.3 2.2 1.1 RawTypeExample.FullStackException: Stack is full, cannot push 6.6

at RawTypeExample.Stack.push(Stack.java:31)

at RawTypeExample.RawTypeTest.testPush(RawTypeTest.java:44)

at RawTypeExample.RawTypeTest.testStacks(RawTypeTest.java:23)

at RawTypeExample.RawTypeTest.main(RawTypeTest.java:80)

Pushing elements onto rawTypeStack2

1.1 2.2 3.3 4.4 5.5 6.6

Resultado – Tipos Raw – 2/3

Popping elements from rawTypeStack2
5.5 4.4 3.3 2.2 1.1

RawTypeExample.EmptyStackException: Stack is empty, cannot pop
at RawTypeExample.Stack.pop(Stack.java:41)
at RawTypeExample.RawTypeTest.testPop(RawTypeTest.java:66)
at RawTypeExample.RawTypeTest.testStacks(RawTypeTest.java:24)
at RawTypeExample.RawTypeTest.main(RawTypeTest.java:80)
RawTypeExample.FullStackException: Stack is full, cannot push 6.6
at RawTypeExample.Stack.push(Stack.java:31)
at RawTypeExample.RawTypeTest.testPush(RawTypeTest.java:44)
at RawTypeExample.RawTypeTest.testStacks(RawTypeTest.java:25)
at RawTypeExample.RawTypeTest.main(RawTypeTest.java:80)
RawTypeExample.EmptyStackException: Stack is empty, cannot pop
at RawTypeExample.Stack.pop(Stack.java:41)
at RawTypeExample.RawTypeTest.testPop(RawTypeTest.java:66)
at RawTypeExample.RawTypeTest.testStacks(RawTypeTest.java:26)
at RawTypeExample.RawTypeTest.main(RawTypeTest.java:80)

Resultado – Tipos Raw – 3/3

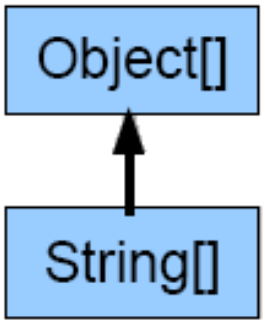
Pushing elements onto integerStack
1 2 3 4 5 6 7 8 9 10 11

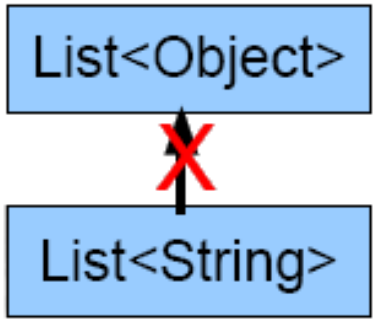
Popping elements from integerStack
10 9 8 7 6 5 4 3 2 1

RawTypeExample.FullStackException: Stack is full, cannot push 11
at RawTypeExample.Stack.push(Stack.java:31)
at RawTypeExample.RawTypeTest.testPush(RawTypeTest.java:44)
at RawTypeExample.RawTypeTest.testStacks(RawTypeTest.java:27)
at RawTypeExample.RawTypeTest.main(RawTypeTest.java:80)
RawTypeExample.EmptyStackException: Stack is empty, cannot pop
at RawTypeExample.Stack.pop(Stack.java:41)

Superclasses e subclasses genéricas

- Classes genéricas são invariantes, isto é: Dadas duas classes A e B, List<A> e List<B> nunca serão subtipo ou supertipo um do outro, mesmo que A e B sejam superclasse e subclasse.
- Em arrays, observa-se covariância. Veja o exemplo abaixo:





Superclasses e subclasses genéricas - 2

- A solução para o problema é utilizar Wildcards (caracteres coringas)
- Vejamos em seguida dois exemplos:
  - Um uso correto de método de classe sem wildcard e
  - Um exemplo de método que usa classe genérica sem uso de Wildcard com Erros devido ao subtipo não reconhecido

Exemplo de método que usa classe genérica sem uso de Wildcard

```
public class TotalNumbers  {
   public static void main( String[] args )     {
      // create, initialize and output ArrayList of Integers
      // then display total of the elements
      Number[] numbers = { 1, 2.4, 3, 4.1 };
      ArrayList< Number> numberList = new ArrayList<Number >();
      for (Number element: numbers )
         numberList.add( element ); // place each number in integerList
      System.out.printf( “numberList contains: %s\n", integerList );
      System.out.printf( "Total of the elements in numberList: %.1f\n",
         sum( numberList ) );
   } // end main
```

Exemplo de método que usa classe genérica sem uso de Wildcard

```
// calculate total of ArrayList elements
  public static double sum( ArrayList< Number > list )
  {
     double total = 0; // initialize total

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

     return total;
  } // end method sum
} // end class TotalNumbersErrors
```

Resultado – Uso de método que usa classe genérica sem uso de Wildcard

numberList contains: [1, 2.4, 3, 4.1]

Total of the elements in numberList: 10,5

Exemplo de método que usa classe genérica sem uso de Wildcard com Erros devido ao subtipo não reconhecido

```
public class TotalNumbersErrors  {
   public static void main( String[] args )    {
      // create, initialize and output ArrayList of Integers
      // then display total of the elements
      Integer[] integers = { 1, 2, 3, 4 };
      ArrayList< Integer > integerList = new ArrayList< Integer >();
      for ( Integer element : integers )
         integerList.add( element ); // place each number in integerList

      System.out.printf( "integerList contains: %s\n", integerList );
      System.out.printf( "Total of the elements in integerList: %.1f\n",
         sum( integerList ) ); // Erro de compilação
   } // end main
```

Exemplo de método que usa classe genérica sem uso de Wildcard com Erros devido ao subtipo não reconhecido

```
// calculate total of ArrayList elements
public static double sum( ArrayList< Number > list )
{
   double total = 0; // initialize total

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

   return total;
  } // end method sum
} // end class TotalNumbersErrors
```

Erro de Compilação

- Erro de compilação devido ao fato do argumento recebido ArrayList<Integer> não ser subclasse de ArrayList<Number>

- Uso de wildcards pode evitar o problema….

Classe que utiliza Wildcard para evitar erros de reconhecimento de subtipo

```
public class WildcardTest  {
   public static void main( String args[] )     {
      // cria, inicializa e gera saída de ArrayList de Integers, então
      // exibe o total dos elementos
      Integer[] integers = { 1, 2, 3, 4, 5 };
      ArrayList< Integer > integerList = new ArrayList< Integer >();

      // insere elementos na integerList
      for ( Integer element : integers )
         integerList.add( element );

      System.out.printf( "integerList contains: %s\n", integerList );
      System.out.printf( "Total of the elements in integerList: %.0f\n\n",
         sum( integerList ));
```

Classe que utiliza Wildcard para evitar erros de reconhecimento de subtipo - 2

```
// cria, inicializa e gera saída do ArrayList de Doubles, então
// exibe o total dos elementos
Double[] doubles = { 1.1, 3.3, 5.5 };
ArrayList< Double > doubleList = new ArrayList< Double >();
// insere elementos na doubleList
for ( Double element : doubles )
   doubleList.add( element );

System.out.printf( "doubleList contains: %s\n", doubleList );
System.out.printf( "Total of the elements in doubleList: %.1f\n\n",
  sum( doubleList ));

// cria, inicializa e gera saída de ArrayList de números contendo
// Integers e Doubles e então exibe o total dos elementos
Number[] numbers = { 1, 2.4, 3, 4.1 }; // Integers and Doubles
ArrayList< Number > numberList = new ArrayList< Number >();
```

Classe que utiliza Wildcard para evitar erros de reconhecimento de subtipo - 3

```
    // insere elementos na numberList
    for ( Number element : numbers )
       numberList.add( element );
       System.out.printf( "numberList contains: %s\n", numberList );
       System.out.printf( "Total of the elements in numberList: %.1f\n",
       sum( numberList ));
  } // fim de main
  // calcula o total de elementos na pilha
  public static double sum(ArrayList< ? extends Number >list )   {
    double total = 0; // inicializa o total
    // calcula a soma
    for ( Number element : list )
       total += element.doubleValue();
    return total;
  } // fim do método sum
} // fim da classe WildcardTest
```

Resultado - Classe que utiliza Wildcard

integerList contains: [1, 2, 3, 4, 5]

Total of the elements in integerList: 15

doubleList contains: [1.1, 3.3, 5.5]

Total of the elements in doubleList: 9,9

numberList contains: [1, 2.4, 3, 4.1]

Total of the elements in numberList: 10,5

Métodos com Múltiplos Parâmetros

- Métodos com número variável de parâmetros já estava disponível em C/C++ há muito tempo, como o printf por exemplo

- A partir da versão 5 tal característica foi adicionada a Java. Método com múltiplos parâmetros:
- public int maximo(int… x) { }

Classe com método de múltiplos parâmetros

```
public class MultParametersTest {
  public static double media(double... x) {
   double sum=0;
     for (int i = 0; i < x.length; i++)
              sum+=x[i];

      if(x.length>0)     return sum/x.length;
  return -1;
  }
 public static void main(String[] args) {
  System.out.printf("A média de (%4.2f,%4.2f) é: %4.2f\n",3.0,4.0, media(3.0,4.0));
  System.out.printf("A média de (%4.2f,%4.2f,%4.2f) é: %4.2f\n",3.0,4.0,5.0,
     media(3,4,5));
  System.out.printf("A média de (,) é: %4.2f\n",media());
}
}
```

Classe com método de múltiplos parâmetros – Ex. 2

```
public class MultParametersTest2 {
   public static double media(Number... x) {
    double sum=0;
   for (int i = 0; i < x.length; i++)
  sum=sum+x[i].doubleValue();

   if(x.length>0)    return sum/x.length;
   return -1;
   }
      public static void main(String[] args) {
       System.out.printf("A média de ("+3+","+4+") é: %4.2f\n", media(3.0,4.0));
      System.out.printf("A média de ("+3.0+","+4.0+","+5.0+") é: %4.2f\n", media(new
      Integer(3),new Double(4),5));
      System.out.printf("A média de (,) é: %4.2f\n",media());
}
}
```

Informações Adicionais sobre Genéricos

- Core Java, Horstmann, C., Cornel. G.

- How to program, Deitel &Deitel,

Distribuição de Aplicativos

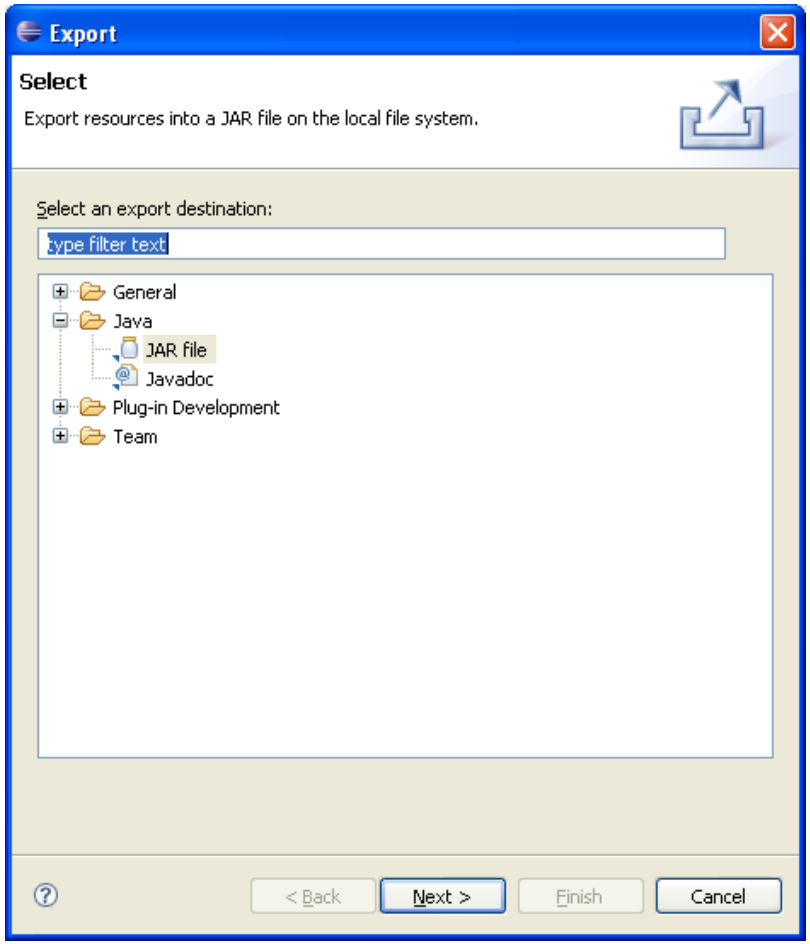
- Java ARchive: Arquivos JAR são um meio simples de empacotar e distribuir aplicativos Java. Arquivos jar são arquivos formato zip com arquivos de metadados, código binário (bytecode) e também arquivos de recursos (imagens, arquivos texto, etc).

- Arquivos podem ser executáveis com um arquivo de metadados opcional (META-INF/MANIFEST.MF) que informa qual a classe principal do aplicativo

- Um arquivo jar executável pode ser executado diretamente pela máquina virtual java, com o comando: java –jar arquivo.jar. É possível associar a extensão jar a maquina virtual e assim tornar o aplicativo executável a partir de um click duplo

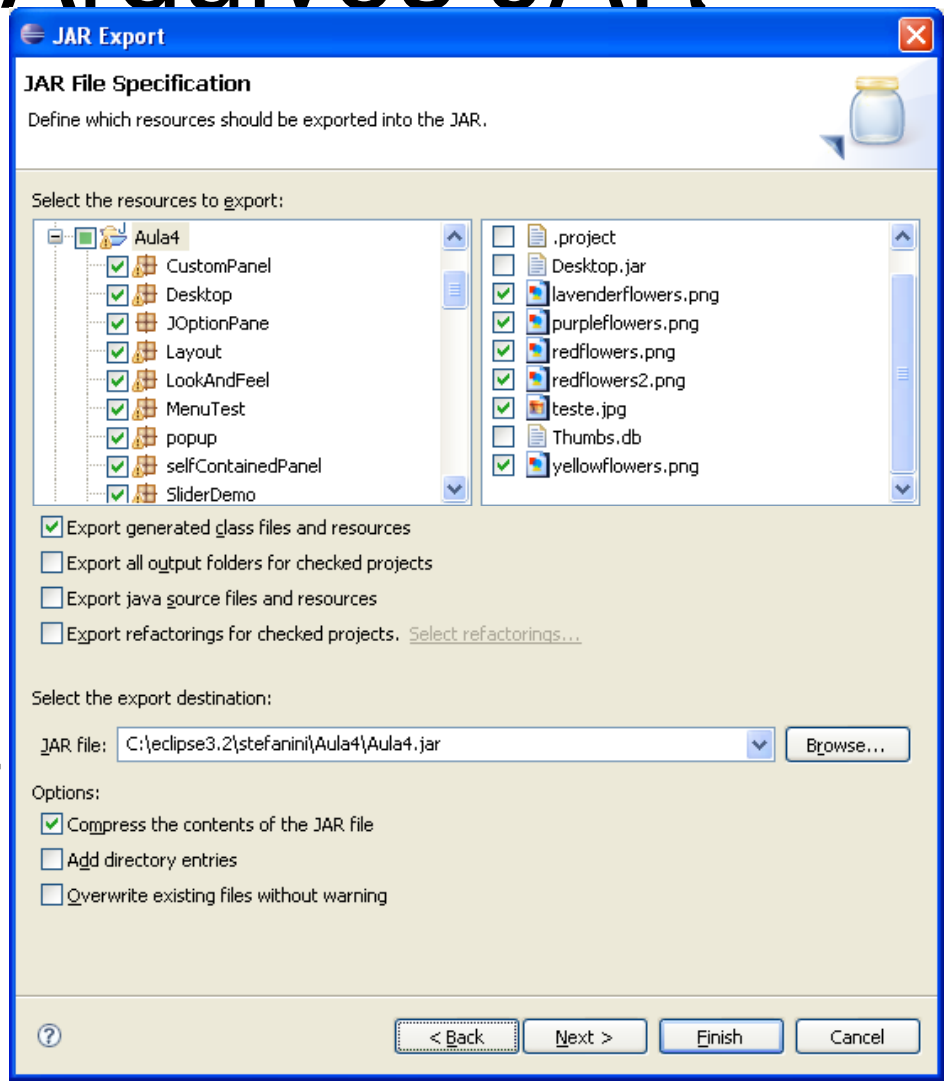
Arquivos JAR

- Criação
  - Opção Export no Popup Menu do Projeto
  - Selecionar Java | JAR File
  - Next



Criação de Arquivos JAR

- Selecione os recursos (imagens, arquivos Java, etc.) que devem estar no projeto
  - Atenção para selecionar também bibliotecas utilizadas
- Escolher nome e caminho do arquivo a ser gerado



Criação de Arquivos JAR

- Pacotes Selados (Sealed Packages): Pacotes que devem obrigatoriamente estar no mesmo arquivo Jar, evita que se utilize erroneamente outras implementações que estiverem no classpath
- Arquivo Manifest gerado automaticamente ou incluído
- Definição de classe principal (classe com método main a ser executado inicialmente)