LABORATÓRIOS DE JAVA5

PROF. F. MÁRIO MARTINS

DI/UM

VERSÃO 1.0

2007

FICHA PRÁTICA 1

LABORATÓRIO DE JAVA BASE

TIPOS PRIMITIVOS &
ESTRUTURAS DE CONTROLO

FICHA PRÁTICA 1

LABORATÓRIO BASE DE JAVA5

SÍNTESE TEÓRICA

JAVA5 é uma linguagem de programação por objectos. Porém, a tecnologia JAVA é muito mais do que a linguagem de programação em que se baseia. A figura seguinte mostra a arquitectura de software correspondente ao ambiente J2SE5 que é necessário instalar nas nossas máquinas para executarmos e criarmos programas escritos em JAVA5.

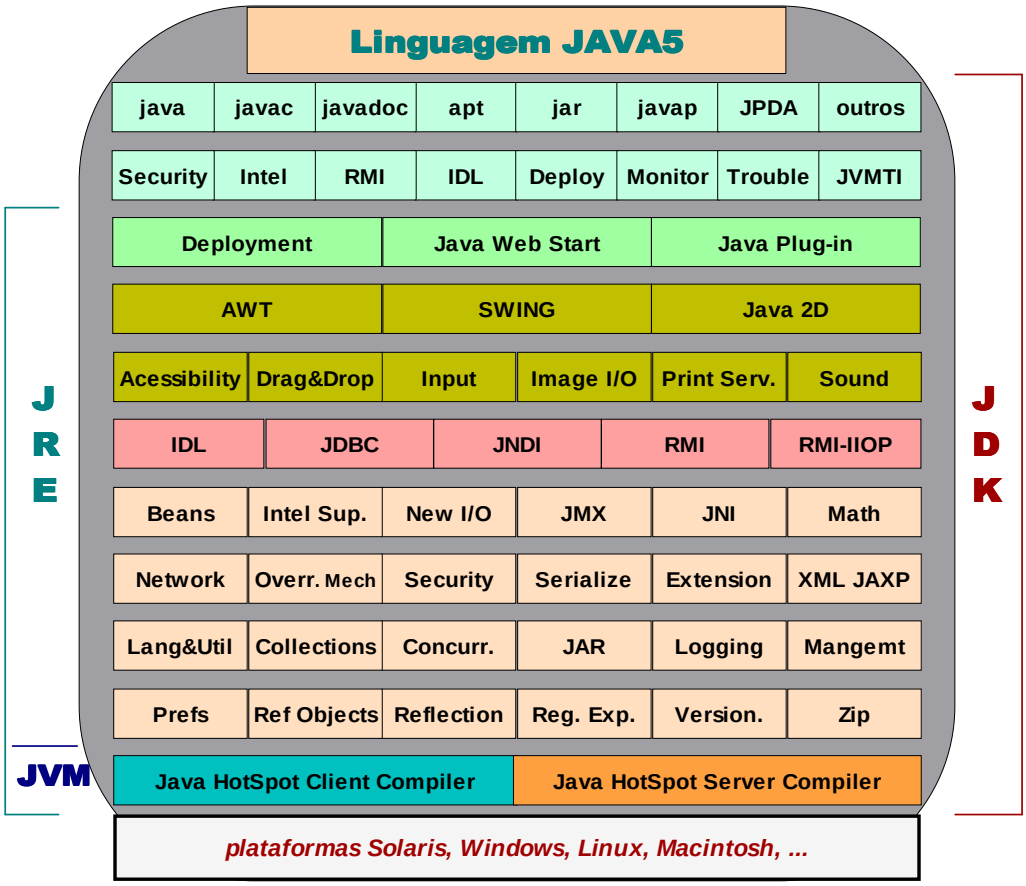


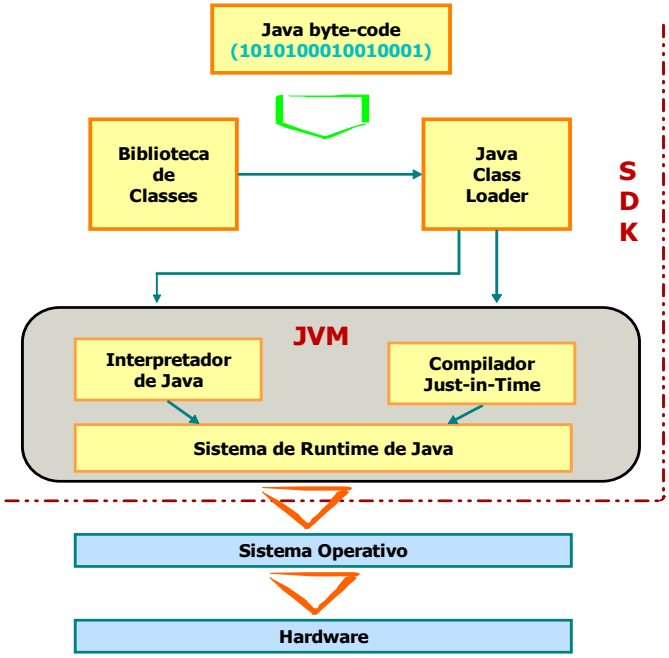
Figura 1 - Arquitectura J2SE5

Quando programamos em JAVA5 temos à nossa disposição todas estas bibliotecas predefinidas, que possuem disponíveis classes para quase todas as mais diferentes funcionalidades necessárias às nossas aplicações.

Porém, o nosso objectivo neste contexto é conhecermos o núcleo fundamental da linguagem, e as suas construções básicas para realizarmos **programação sequencial**, mas seguindo princípios rigorosos da Engenharia de Software que são mais facilmente respeitados se utilizarmos correctamente características e propriedades disponíveis no paradigma da Programação por Objectos e suas linguagens (cf. C++, C# e JAVA).

A execução de um programa JAVA passa fundamentalmente pela compilação do seu código fonte para um código intermédio, designado **byte-code**. Este *byte-code*, que é o resultado da compilação, é um código standard que poderá ser em seguida executado (interpretado) por uma

qualquer Java Virtual Machine (JVM). Naturalmente que, para cada sistema operativo e arquitectura, existirá uma JVM específica que interpreta correctamente o *byte-code* em tal contexto (cf. Windows, Linux, PDA, Java Card, etc.). Neste facto reside a grande portabilidade e flexibilidade da linguagem JAVA.



SINTAXE ESSENCIAL

1.- ESTRUTURA BASE DE UM PROGRAMA JAVA

Em JAVA tudo são classes. Um programa JAVA é uma classe especial que, entre outros, possui obrigatoriamente um método `main()` pelo qual se inicia a execução do código do programa. O nome do programa (classe) deverá ser igual ao do ficheiro fonte que a contém. Exemplo: `public class Teste1` deverá ser guardada no ficheiro `Teste1.java`.

```
public class Teste1 {
    public static void main(String args[]) {

        // declarações e código
        // .............
    }
}
```

Porém, e por razões de estruturação do código, nada impede que se criem métodos externos ao método `main()` que pertencem igualmente ao programa e que podem ser invocados a partir do método `main()` e, até, invocarem-se entre si.
A figura seguinte mostra este tipo um pouco mais complexo de estruturação do programa, mas que é apenas um caso particular do primeiro.
Finalmente, e por razões a ver posteriormente, todos estes métodos devem possuir o atributo `static`, podendo ser `public` ou não (usaremos de momento sempre `public`).

```
public class Teste2 {

   public static <tipoRes> metodo_Aux1 (argumentos opcionais) {
      // .....
   }

   public static <tipoRes> metodo_Aux2 (argumentos opcionais) {
      // .....
   }

   public static void main(String args[]) {
      // ............

      // declarações e código
      // ............

   }
}
```

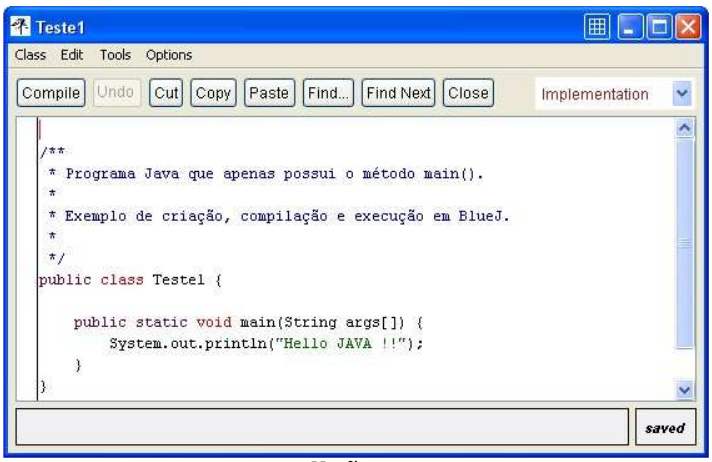
2.- Compilação e Execução a partir de linha de comandos

Ficheiro fonte: Teste1.java
Compilação: > javac Teste1.java ⇨ cria ficheiro Teste1.class
Execução: > java Teste1

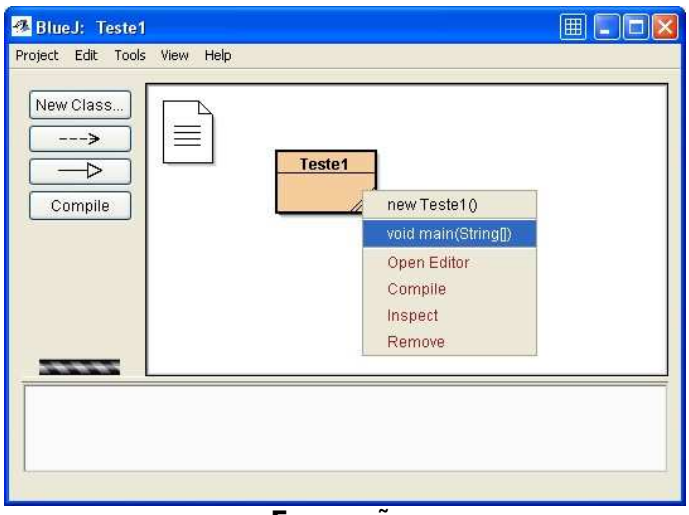
3.- Edição, Compilação e Execução usando BlueJ

- Invocar o BlueJ (eventualmente usar opção **New Project ...**)
- Cria/Editar o ficheiro fonte usando a opção **New Class ...**
- Se se fizer apenas **Save** é criado um ícone sombreado com o nome do ficheiro criado, que se apresentará um tracejado indicando que não se encontra compilado ainda;
- Pode no entanto usar-se a opção **Compile** imediatamente após o fim da edição;
- Se não existirem erros, aparece o mesmo ícone mas sem qualquer tracejado;
- Para ficheiros não compilados, clicar no botão direito do rato e executar a opção **Compile**
- Para executar o programa, sobre o ícone clicar no botão direito do rato e seleccionar o método **main()**
- Os resultados são apresentados na **Terminal Window** do BlueJ

EXEMPLO:



Edição



Execução

Resultados

4.- Tipos Primitivos e Operadores

Os tipos primitivos de JAVA são os usuais tipos simples que se associam a variáveis nas quais pretendemos guardar **valores** (cf. Pascal, C, etc.). A tabela seguinte mostra os tipos disponíveis, gama de valores aceites e número de bits de representação.

Tipo	Valores	Omissão	Bits	Gama de valores
`boolean`	false, true	false	1	false a true
`char`	caracter unicode	\u0000	16	\u0000 a \uFFFF
`byte`	inteiro c/ sinal	0	8	-128 a +127
`short`	inteiro c/ sinal	0	16	-32768 a +32767
`int`	inteiro c/ sinal	0	32	-2147483648 a 2147483647
`long`	inteiro c/ sinal	0L	64	≈ -1E+20 a 1E+20
`float`	IEEE 754 FP	0.0F	32	± 3.4E+38 a ± 1.4E-45 (7 d)
`double`	IEEE 754 FP	0.0	64	± 1.8E+308 a ± 5E-324 (15d)

Tabela 1 – Tipos primitivos de Java

5.- Declaração e Inicialização de variáveis.

Associadas a um tipo de dados, poderemos ter uma sequência de declarações de variáveis separadas por vírgulas, eventualmente com atribuições dos valores de expressões para a inicialização de algumas delas, cf. a forma geral e os exemplos seguintes:

id_tipo *id_variável [= valor] [, id_variável [= valor] …] ;*

```
int dim = 20, lado, delta = 30;
char um = '1'; char newline = '\n';
byte b1 = 0x49;        // hexadecimal, cf. 0x como prefixo
```

```
long diametro = 34999L; double raio = -1.7E+5;
double j = .000000123;     // parte inteira igual a 0
int altura = dim + delta;     // inicialização usando cálculo de expressão
```

Associados a estes tipos simples existe um conjunto de operadores necessários à sua utilização, comparação, etc. Vejamos os mais importantes na tabela seguinte.

Precedência	Operador	Tipos dos Operandos	Associação	Operação
◎	+, -	número	D	sinal; unário
◎	!	boleano	D	negação
12	*, /	número, número	E	multipl, divisão
◎	/,%	inteiro, inteiro	E	quoc. int e módulo
11	+, -	número, número	E	soma e subtracção
9	<, <=	números	E	comparação
◎	>, >=	aritméticos	E	comparação
8	==	primitivos	E	igual valor
◎	!=	primitivos	E	valor diferente
◎	^	boleanos	E	OUEXC lógico
◎	|	boleanos	E	OU lógico
4	&&	boleano	E	E condicional
3	||	boleano	E	OU condicional
1	=	variável, qualquer	D	atribuição
◎	*= /= %=	variável, qualquer	D	atribuição após oper.
◎	+= -=	variável, qualquer	D	atribuição após oper.
◎	<<= >>=	variável, qualquer	D	atribuição após oper.
◎	>>>= &=	variável, qualquer	D	atribuição após oper.
◎	^= |=	variável, qualquer	D	atribuição após oper.

Tabela 2 – Operadores de JAVA para tipos simples

6.- Declaração de Constantes.

As constantes JAVA são também, por razões de estilo e de legibilidade, identificadas usando apenas **letras maiúsculas**, em certos casos usando-se também o símbolo _. As seguintes declarações,

```
final double PI = 3.14159273269;
final double R_CLAP = 8.314E+7;
```

definem, num dado contexto, **constantes identificadas**, cujos valores não poderão ser alterados por nenhuma instrução. O atributo **final** garante que um erro de compilação será gerado caso haja a tentativa de modificar tal valor.

7.- Comentários.

```
// este é um comentário monolinha; termina no fim desta linha
/* este é multilinha; só termina quando aparecer o delimitador final */
/** este é um comentário de documentação para a ferramenta javadoc */
```

8.- Estruturas de Controlo

```
if (condição) instrução;
if (condição) { instrução1; instrução2; ... }
if (condição) instrução1; else instrução2; ou { instruções }
if (condição) { instrução1; instrução2; ... } else { instrução1; instrução2; ...}

switch (expressão_simples) {
   case valor_1 : instruções; [break;]
   ...
   case valor_n : instruções; [break;]
   default: instruções;
}

for (inicialização; condição de saída; incremento) instruções;

for (Tipo variável : array de elementos de tipo Tipo) instruções;

while (condição de execução) { instruções; }

do { instruções; } while(condição de repetição);
```

9.- Importação Normal e Estática – Reutilização de Classes

Importação por omissão: **import java.lang.*;**
Importação global de um package: **import java.util.*;**
Importação selectiva de classes: **import java.lang.Math;**
Importação estática (elimina prefixos): **import static java.lang.Math.abs;**

10.- Output Básico e Formatado

10.1- As instruções básicas de escrita (*output*) de JAVA são dirigidas ao dispositivo básico de saída, o monitor, que a partir do programa é visto como um "ficheiro" de caracteres do sistema designado por **System.out**. Assim, são mensagens básicas todas as *strings* (cf. "abc" mais operador de concatenação **+** para *strings* e valores de tipos simples) enviadas para tal ficheiro usando a instrução **println()**, o que se escreve:

```
System.out.println("Bom dia e ...\tbom trabalho!\n\n\n");
System.out.println("Linguagem : " + lp + 5);
System.out.println(nome + ", eu queria um " + carro + "!");
```

Se nos programas em que tivermos que realizar muitas operações de saída escrevermos no seu início a cláusula de importação **import static java.lang.System.out;**, então, em vez de se escrever **System.out.println(.)** bastará escrever-se **out.println(.)**, cf. em:

```
out.println("Hello Java!"); out.println(nome + "tem " + idade + " anos.");
```

10.2.- JAVA possui também uma forma formatada de escrita de dados, baseada em especial no método **printf()**.
O método **printf(formatString, lista_valores)** permitirá a saída de uma lista de valores (sejam variáveis, constantes ou expressões), que serão formatados segundo as directivas dadas na ***string* de formatação,** que é o primeiro parâmetro, e que pode incluir **texto livre**. A forma geral **de formatação** de valores de tipos primitivos é a seguinte (para cada valor a formatar):

%[índice_arg$] [flags] [dimensão][.decimais] conversão

Os **caracteres de conversão** são os que indicam o tipo de valor resultado da conversão do parâmetro, sendo: **c** (carácter), **b** (boleano), **o** (octal), **h** (hexadecimal), **d** (inteiro), **f** (real, vírgula fixa), **s** (*string*), **e** (real, vírgula flutuante), **t** (data) e **n** (*newline* independente da plataforma). Um valor de dado tipo se formatado para outro tipo compatível é automaticamente convertido.
As *flags* podem permitir alinhar os resultados à esquerda (**-**), obrigar a incluir sempre o sinal (**+**), colocar zeros no início (0), colocar espaços no início (**' '**) ou colocar parêntesis (**(**) se o número for negativo.

```
float f1 = 123.45f; double d2 = 234.678; double d3 = 12.45E-10;
out.printf("R1 %5.2f R2 %3$-12.7f Exp1 %2$8.4e%n",f1, d2, d3);
```

Por exemplo, usando apenas caracteres de conversão podemos automaticamente fazer a conversão de um número inteiro na base 10 para a base 8 e para a base 16.

```
int x = 1261;
out.printf("Inteiro %d = Octal %o = Hexa %h%n", x,x,x);
out.printf("Inteiro %d = Octal %1$o = Hexa %1$h%n", x);
```

11.- INPUT COM CLASSE SCANNER

A classe **java.util.Scanner** possui métodos para realizar leituras de diversos tipos a partir de diversos ficheiros, e não só. Interessa-nos aqui ver como podemos usar esta classe para realizar leituras de valores de tipos simples a partir do teclado. O teclado está, por omissão, associado a uma variável designada **System.in**. Assim, teremos que associar um **Scanner** ao teclado para se poder ler os valores primitivos necessários aos programas. Vejamos os passos:

1.- Escrever no início do ficheiro a cláusula de importação **import java.util.Scanner**;
2.- Criar um **Scanner** que se vai associar ao teclado, e que vamos designar por **input**:

```
Scanner input = new Scanner(System.in);
```

3.- Tendo o scanner **input** associado ao teclado, usar métodos de **Scanner** para ler os valores, cf.:

```
String nome = input.next();          // lê uma string
String linha = input.nextLine();     // lê uma linha de texto terminada por \n
int x = input.nextInt();             // lê um inteiro válido
double pi = input.nextDouble();      // lê um real válido, sendo , o separador
input.nextTipoSimples();             // lê um valor de qualquer tipo simples
input.close();                       // fecha o scanner
```

Nota: Posteriormente usaremos uma classe Input especialmente desenvolvida

12.- CLASSES PREDEFINIDAS IMPORTANTES

java.lang.Math

```
Math.PI; Math.E:        // valores de PI e da base E com grande precisão
tipo_numérico abs(tipo_numérico val);   // valor absoluto
double sqrt(double val);                // raiz quadrada
double pow(double base, double exp);    // potenciação
double random();                        // resultado no intervalo [0.0 1.0[
tipo_numérico max (tipo_numérico val1, tipo_numérico val2);
tipo_numérico min (tipo_numérico val1, tipo_numérico val2);
int round(float val);    float round(double val);
double sin(double val);   double cos(double val);      // seno e coseno
```

```
java.lang.Integer java.lang.Double java.lang.Float …
   MAX_VALUE; MIN_VALUE     // máximo e mínimo de um tipo numérico simples

java.util.Calendar   java.util.GregorianCalendar    // classes úteis para datas
   Calendar now = Calendar.getInstance(); // instante actual no sistema
   GregorianCalendar agora = new GregorianCalendar();  // idem
   GregorianCalendar hoje = new GregorianCalendar(2007, Calendar.MARCH,
                                                  10, 23, 15);  // define data
   // tempo decorrido desde o início da ERA até à data e hora actuais em ms
   long milis = agora.getTimeInMillis();
   // diferença entre duas datas (agora e inicio) em número de dias
   long dif_milis = agora.getTimeInMillis() - inicio.getTimeInMillis();
   int dias = dif_milis/(24*60*60*1000);
   // escrever a hora actual e escrever a data actual
   out.printf("%tT%n", agora)                          // 12:23:35
   out.printf("%1$tY/%1$tm/%1$td%n", agora);    // 2005/03/21

java.lang.String
   Constantes: "" "abcd" "Uma linha\n" "Exemplo\t\tFinal\n\n"
   Concatenação: "abc" + "25" "Luís " + "Carlos" "Linha1\n" + "Linha2\n"
   String valueOf(tipo_simples val); // converte para string um valor simples
   char charAt(int índex);       // devolve o carácter na posição índex da string
   int length();                          //devolve o comprimento da string
   String substring(int inic, int fim);   // devolve uma substring
   boolean equals(String str)             // igualdade de strings
```

EXERCÍCIOS RESOLVIDOS:

Ex1: Ler um nome e uma idade e imprimir um texto com os resultados.

```
import java.util.Scanner;
public class Leitura1 {
  public static void main(String[] args) {
    String nome; int idade;
    // Scanner: classe para leitura
    Scanner input = new Scanner(System.in); // lê via teclado
    System.out.print("Nome: "); nome = input.next();
    System.out.print("Idade: "); idade = input.nextInt();
    System.out.println(nome + " tem " + idade + " anos.");
  }
}
```

Ex2: Ler 10 inteiros e determinar o maior inteiro introduzido.

```
import java.util.Scanner;
public class MaxInt {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in); // lê via teclado
    int valor; int maior = Integer.MIN_VALUE;
    for(int i = 1; i <= 10; i++) {
        System.out.print("Inteiro " + i + " : ");
        valor = input.nextInt();
        if (valor > maior) maior = valor;
    }
    System.out.println("Maior inteiro = " + maior);
  }
}
```

Ex3: Sendo N dado pelo utilizador, ler N reais e dar os resultados das suas potências de expoente Exp, também introduzido pelo utilizador.

```
import static java.lang.System.out;
import java.util.Scanner;
public class Expoentes {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in); // lê via teclado
    int total; double valor; int exp;
    out.print("Total de valores a ler: "); total = input.nextInt();
    out.print("Expoente a que os quer elevar: "); exp = input.nextInt();
    for(int i = 1; i <= total; i++) {
        out.print("Introduza valor real: "); valor = input.nextDouble();
        out.printf("Valor %5.2e elevado a %2d = %8.4e%n", valor, exp,
                    Math.pow(valor, exp));
    }
  }
}
```

Ex4: Ler uma sequência de inteiros positivos (terminada pelo valor -1) e determinar a diferença entre o maior e o menor inteiro lidos. Imprimir esse valor, bem como o maior e o menor.

```
import java.util.Scanner;
import static java.lang.System.out;
public class DiferencaInt {
  public static void main(String[] args) {
    int valor; int maior = Integer.MIN_VALUE; int menor = Integer.MAX_VALUE;
    int diferenca;
    Scanner input = new Scanner(System.in); // lê via teclado
    valor = input.nextInt();
    while(valor != -1) {
        if (valor > maior) maior = valor;
        else
            if (valor < menor) menor = valor;
    }
    diferenca = maior – menor;
    out.println("Maior = " + maior + " Menor = " + menor + "\n");

    out.println("A diferença entre eles é : " + diferenca);
  }
}
```

Ex5: Escrever um programa que calcule o factorial de um valor inteiro, dado como argumento do método **main()** através dos argumentos deste método.

```
import static java.lang.System.out;
public class FactorialInt{
  public static long factorial(long n) {
    if (n==1) return 1; else return n*factorial(n-1);
  }
  public static void main(String[] args) {
    long i = Integer.parseInt(args[0]);
    out.println( i + "! = " + factorial(i) );
  }
}
```

Execução: **> FactorialInt 4**

Nota: Para outro tipo teríamos **parseDouble()**, **parseFloat()**, **parseByte()**, etc.

Ex6: Escrever um programa que determine a data e hora do sistema, realize um ciclo com 10 milhões de incrementos unitários de uma dada variável, determine a hora após tal ciclo, e calcule o total de milissegundos que tal ciclo demorou a executar.

```
import java.util.Scanner;
import java.util.Calendar;
import static java.lang.System.out;
public class Ex6 {
    public static void main(String[] args) {
        int x = 0;
        Calendar inicio = Calendar.getInstance();
        for(int i = 0; i < 10000000; i++) x = x +1;
        Calendar fim = Calendar.getInstance();
        long milisegs = fim.getTimeInMillis() - inicio.getTimeInMillis();
        out.println("O ciclo demorou " + milisegs + " milisegundos a executar.");
        out.println("A variável x tem o valor: " + x);
    }
}
```

Ex7: Escrever um programa que use um método auxiliar que aceite duas datas e determine a sua diferença em dias, horas, minutos e segundos. O resultado do método deverá ser uma *string*.

```
import java.util.Scanner;
import java.util.GregorianCalendar;
import static java.lang.System.out;
public class Ex7 {

    public static String difDatas(GregorianCalendar fim,
                                  GregorianCalendar inicio) {
        long totalmilis = fim.getTimeInMillis() - inicio.getTimeInMillis();
        long milis = totalmilis%1000;
        long totalseg = totalmilis/1000; long seg = totalseg%60;
        int totalmin = (int) totalseg/60; int min = totalmin%60;
        int totalhoras = totalmin/60; int horas = totalhoras%60;
        int totaldias = totalhoras/24;

      return totaldias + " dias, " + horas +" horas, "+ min+ " minutos e "
              + seg + "," + milis + " segundos.";
    }

    public static void main(String[] args) {
        GregorianCalendar inicio = new GregorianCalendar();
        inicio.set(GregorianCalendar.YEAR, 2007); inicio.set(GregorianCalendar.MONTH, 3);
        inicio.set(GregorianCalendar.DAY_OF_MONTH, 8);
        inicio.set(GregorianCalendar.HOUR, 12); inicio.set(GregorianCalendar.MINUTE, 20);
        inicio.set(GregorianCalendar.SECOND, 33);
        inicio.set(GregorianCalendar.MILLISECOND, 111);
        // ou new GregorianCalendar(2007, Calendar.MARCH, 8, 23, 15);
        GregorianCalendar fim = new GregorianCalendar();
        fim.set(GregorianCalendar.YEAR, 2007); fim.set(GregorianCalendar.MONTH, 3);
        fim.set(GregorianCalendar.DAY_OF_MONTH, 10);
        fim.set(GregorianCalendar.HOUR, 22); fim.set(GregorianCalendar.MINUTE, 12);
        fim.set(GregorianCalendar.SECOND, 15);
        fim.set(GregorianCalendar.MILLISECOND, 444);
        // ou new GregorianCalendar(2007, Calendar.MARCH, 10, 12, 15);
        String difTempo = difDatas(fim, inicio);
        out.println("Diferença: " + difTempo);
    }
}
```

Ex8: Escrever um programa aceite N classificações (números reais) entre 0 e 20 e determine a sua média (usar printf() para os resultados).

```
import static java.lang.System.out;
public class Ex8 {
   public static void main(String[] args) {
      double nota, soma = 0.0;
      int total;
      out.print("Total de notas a ler: ");
      total = Input.lerInt();
      for(int i = 1; i <= total; i++) {
         out.println("Nota N. " + i);
         nota = Input.lerDouble(); soma += nota;
      }
      out.printf("A média das %2d notas é %4.2f%n", total, soma/total);
   }
}
```

Ex9: Escrever um programa aceite N temperaturas inteiras (pelo menos duas) e determine a média das temperaturas, o dia (2,3, …) em que se registou a maior variação em valor absoluto relativamente ao dia anterior e qual o valor efectivo (positivo ou negativo) dessa variação.
Os resultados devem ser apresentados sob a forma:

```
A média das N temperaturas foi de ____ graus.
A maior variação de temperatura registou-se entre os dias __ e __ e foi de ___ graus.
A temperatura entre o dia __ e o dia __ subiu/desceu ___ graus.
```

```
import java.util.Scanner;
import static java.lang.System.out;
import static java.lang.Math.abs;
public class Temperaturas {
   public static void main(String[] args) {
      int temp; int anterior; int dia = 0; int total;
      int maiorDif = 0; int soma = 0;
      int difReal = 0, difAbs = 0;
      Scanner input = new Scanner(System.in); // lê via teclado
      out.print("Total de temperaturas a ler: ");
      total = Input.lerInt();
      out.print("Temperatura " + 1 + " : ");
      temp = Input.lerInt(); soma = temp;
      for(int i = 2; i <= total; i++) {
         anterior = temp;
         out.print("Temperatura " + i + " : ");
         temp = Input.lerInt();
         soma += temp; difReal = temp - anterior;
         difAbs = abs(temp - anterior);
         if (difAbs > maiorDif) {
            dia = i; maiorDif = difAbs;
         }
      }
      // resultados
      out.printf("A média das %2d temperaturas é %4.2f%n", total, ((double) soma)/total);
      out.println("Maior variação de temperaturas entre os dias " + (dia-1) + " e " + dia);
      String txt = difReal > 0 ? "subiu " : "desceu ";
      out.println("A temperatura entre esses dias " + txt + difAbs + " graus.");
   }
}
```

Ex10: Escrever um programa que leia sucessivas vezes o raio (real) de um círculo e calcule a área e o perímetro respectivos com grande precisão (5 decimais). Usar printf() para os resultados. O programa apenas deverá terminar com a leitura de um raio = 0.0

```
import static java.lang.Math.PI;
import static java.lang.System.out;
public class AreaCirculo {
   public static void main(String[] args) {
      double raio;
      out.print("Valor do raio: ");
      raio = Input.lerDouble();
      while(raio != 0.0) {
         out.printf("Raio = %7.3f => Área = %9.5f e Perímetro = %9.5f%n", raio,
                2*PI*raio*raio, 2*PI*raio);
         out.print("Valor do raio: "); raio = Input.lerDouble();
      }
   }
}
```

Ex11: Escrever um programa que faça a leitura de uma sequência não vazia de números reais terminada por 0.0 e calcule o seu somatório (∑) e o seu produtório (∏) com precisão de 4 casas decimais no resultado.

```
import static java.lang.System.out;
public class Ex11 {
   public static void main(String[] args) {
      double soma = 0; double prod = 1; double num;
      int conta = 0;
      out.print("Número Real: "); num = Input.lerDouble();
      while(num != 0.0) {
         soma += num; prod *= num; conta++;
         out.print("Número Real: "); num = Input.lerDouble();
      }
      out.printf("Soma dos %2d números reais = %8.4f ; Produto = %12.4f%n%n",
             conta, soma, prod);
   }
}
```

Ex12: Escrever um programa leia um inteiro N e imprima todos os números ímpares inferiores a N.

```
import static java.lang.System.out;
public class Impares {
   public static void main(String[] args) {
      int limite;
      out.print("Ler número limite: ");
      limite = Input.lerInt();
      out.println("Números ímpares menores ou iguais a " + limite);
      for(int i = 1; i <= limite; i = i + 2) out.println(i);
   }
}
```

Ex13: Escrever um programa que apresente ao utilizador um menu vertical com as opções:

1.- Inserir 2.- Remover 3.- Consultar 4.- Gravar 5.- Sair

Em seguida, o programa deverá ler um **int**, que apenas será válido se entre 1 e 5, e deverá apresentar ao utilizador, textualmente, a opção escolhida (Inserir, Remover, etc.) ou a mensagem "Opção Inválida!". O programa deverá repetir a apresentação do menu até que o utilizador seleccione a opção 5.- Sair.

```
import static java.lang.System.out;
public class Ex13 {
   public static void menu() {
      out.println("-----------------------------------------");
      out.println("--     OPCOES DISPONÍVEIS     ---");
      out.println("-----------------------------------------");
      out.println(" 1. INSERIR ");
      out.println(" 2. REMOVER ");
      out.println(" 3. CONSULTAR ");
      out.println(" 4. GRAVAR ");
      out.println(" 5. SAIR ");
      out.println("-----------------------------------------");
   }

   public static int opcao() {
      int opcao;
      boolean valida = false;
      do {
         out.print("OPCAO: "); opcao = Input.lerInt();
         valida = (opcao >= 1) && (opcao <= 5);
         if(!valida) out.println("OPCAO INVÁLIDA !!");
        }
      while(!valida);
      return opcao;
   }
   public static void texto_opcao(int op) {
      switch(op) {
         case 1 : out.println("INSERIR"); break;
         case 2 : out.println("REMOVER"); break;
         case 3 : out.println("CONSULTAR"); break;
         case 4 : out.println("GRAVAR"); break;
         case 5 : out.println("SAIR"); break;
      }
   }

   public static void main(String[] args) {
      int escolha;
      do {
         menu();
         escolha = opcao();
         texto_opcao(escolha);
      }
      while(escolha != 5);
      out.println("FIM DE PROGRAMA !!");
   }
}
```

Ex14: Escrever um programa que gere um número aleatório entre 1 e 100. O programa dará 5 tentativas ao utilizador para acertar no número gerado. A cada tentativa do utilizador, o programa indicará se o número gerado é maior ou menor que o número dado pelo utilizador. À terceira tentativa falhada o utilizador perde. Quer perca quer acerte, o programa deve perguntar ao utilizador se quer continuar a jogar ou não. Se sim, novo número será gerado e o jogo retomado.

```
import static java.lang.System.out;
import static java.lang.Math.random;
public class Ex14 {
   public static int geraNumero() {
      int numAleat = 1 + (int) (random() * 100);
      return numAleat;
   }
```

```
   public static void main(String[] args) {
      int numero; int opcao; int tentativa; int palpite;
      int resposta;
      boolean acertou;
      do {
        numero = geraNumero(); tentativa = 1;
        out.print("Tentativa " + tentativa + " - Qual o numero gerado? ");
        palpite = Input.lerInt();
        acertou = false;
        while( tentativa < 5 && !acertou) {
           if(palpite == numero) acertou = true;
           else {
                 if (palpite > numero) out.println("PARA BAIXO !!");
                 else out.println("PARA CIMA !!");
                 tentativa++;
                 out.print("Tentativa " + tentativa + " - Qual o numero gerado? ");
                 palpite = Input.lerInt();
                }
         }
         // verifica se acertou ou não
         if (acertou) out.println("PARABÉNS ACERTOU !!");
         else out.println("FALHOU ! O número era o " + numero);

         out.println("Continuar Jogo (1) ou Sair(outro): ");
         resposta = Input.lerInt();
      }
      while(resposta == 1);
      out.println("FIM DO JOGO ........");
   }
}
```

Ex15: Escrever um programa que leia o ano, mês e dia de nascimento de uma pessoa e calcule a sua idade actual, indicando ao utilizador a data de nascimento lida, o dia de hoje e a idade que foi calculada.

```
/**
 * Cálculo da Idade actual de uma pessoa
 *
 * @author FMM
 * @version 1.0/2007
 */
import java.util.GregorianCalendar;
import static java.lang.System.out;
public class Ex15 {
   public static void main(String[] args) {
     int ano, mes, dia; String resp = "";
     do {
      out.print("Ano de nascimento: ");
      ano = Input.lerInt();
      do {
         out.print("Mês de nascimento: ");
         mes = Input.lerInt();
      }
      while(mes <= 0 || mes > 12);
      do {
         out.print("Dia de nascimento: ");
         dia = Input.lerInt();
      }
      while(dia <= 0 || dia > 31);
```

```
        // Nota: Janeiro = 0 para um Calendar !!
        GregorianCalendar dataNascimento =
          new GregorianCalendar(ano, mes-1, dia);
        out.printf("NAscido a %1$td/%1$tm/%1$tY%n", dataNascimento);
        GregorianCalendar hoje = new GregorianCalendar();
        // diferença de anos
        int anos = hoje.get(GregorianCalendar.YEAR) -
                          dataNascimento.get(GregorianCalendar.YEAR);
        out.printf("DATA ACTUAL : %1$td-%1$tm-%1$tY%n", hoje);
        out.println("IDADE NÃO CORRIGIDA: " + anos + " anos.");
        GregorianCalendar diaDeAnosEsteAno = dataNascimento;
        // dia de anos este ano
        diaDeAnosEsteAno.add(GregorianCalendar.YEAR, anos);
        out.printf("ANOS A %1$td-%1$tm-%1$tY%n", diaDeAnosEsteAno);
        // já fez anos ou não ??
        if (hoje.before(diaDeAnosEsteAno))
            { out.println("Ainda lá não chegamos !!"); anos--; }
        out.println("A pessoa tem " + anos + " anos !");
        out.println("----------------------------------------");
        out.print("Quer calcular mais (S/*) ? "); resp = Input.lerString();
        out.println();
       }
       while(resp.equals("S") || resp.equals("s"));
       out.println("---- FIM DO CALCULADOR DE IDADES .... ");
    }
}
```

FICHA PRÁTICA 2

LABORATÓRIO DE ARRAYS

FICHA PRÁTICA 2

LABORATÓRIO DE ARRAYS

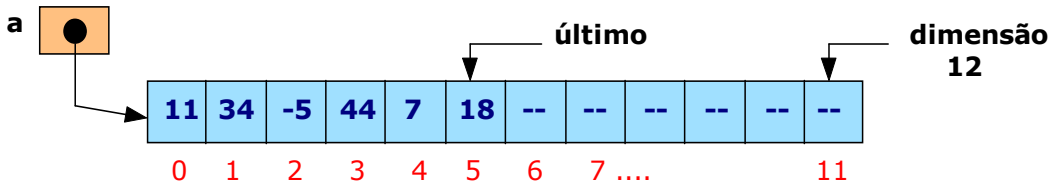
SÍNTESE TEÓRICA

Os ***arrays*** de JAVA são estruturas lineares indexadas, ou seja, cada posição do *array* possui um endereço inteiro para acesso ao elemento nele contido (1º elemento no índice 0). Os *arrays* podem conter valores de tipos primitivos ou objectos. Os *arrays* de JAVA não são objectos, ou seja, não são criados por nenhuma classe nem respondem a mensagens. São no entanto de tipo referenciado, ou seja, a variável que identifica o *array* contém o endereço de memória do *array* (é uma referência).

Sendo estruturas lineares indexadas, os **elementos** de um *array* ocupam **posições** referenciáveis por um **índice** inteiro com valores a partir de 0.

A dimensão física de um *array*, também designada a sua **capacidade**, pode ser definida aquando da sua declaração ou posteriormente, mas é diferente do seu **comprimento**, que se associa ao número efectivo de elementos que, num dado momento, estão armazenados no *array*.

Para um *array* de dimensão d, o seu comprimento actual c será sempre c <= d e o índice do último elemento será sempre c-1. Para um *array* **a**, a instrução **a.length** devolve um inteiro que corresponde à sua dimensão actual, não o actual número de elementos. Para arrays numéricos, inicializados a 0 ou 0.0 há quer ter cuidado com **length** pois os elementos a zero iniciais são contados também, e não correspondem a dados introduzidos. Assim, sempre que o número de elementos não coincida garantidamente com a dimensão, uma variável que conte os elementos efectivos do array deverá ser usada.



A dimensão física de um *array*, bem como o tipo dos seus elementos, são em geral definidos aquando da sua declaração, como em:

```
int[] vector = new int[100];
```

A dimensão pode, no entanto, ser definida posteriormente, usando a construção **new**, não podendo o *array* ser usado enquanto tal dimensão não for especificada.

```
String[] nomes;
nomes = new String[50];
```

A capacidade/dimensão definida para um *array* é fixa, ou seja, é imutável ao longo da execução do programa. A capacidade pode ser também definida de forma implícita e automática através da sua inicialização, sendo, neste caso, a capacidade do *array* igual ao número de elementos introduzidos na sua inicialização, cf. o exemplo:

```
int[] valores = { 12, 56, -6, 45, 56, 8 };  // dim = 6
double[] notas = { 12.5, 15.6, 10.9, 15.2, 6.6, 8.7, 9.0, 11.1 };  // dim = 8
```

Os *arrays* podem ser multidimensionais (linhas, colunas, etc.) e não apenas a uma só dimensão (linha). Os *arrays* monodimensionais são muitas vezes referidos como **vectores**.

Os *arrays* multidimensionais são em geral referidos como **matrizes**. O número de dimensões de um *array* é clarificado na sua definição, pois cada dimensão corresponde sintacticamente a mais um [].

```
int[][] matriz_valores = new int[20][50];          // matriz de 20 linhas por 50 colunas
double[][] notas = new double[3][300];             // matriz 3 testes por 300 alunos
double [][][] temps = new double[15][12][31];      // cidades x meses x dias e temperaturas
```

SINTAXE ESSENCIAL

1.- DECLARAÇÕES, INICIALIZAÇÕES E DIMENSIONAMENTO

```
int lista[];   // estilo C
int[] lista;   // estilo JAVA

int[] turma = new int[100];
double[] medias = new double[50];
byte[] mem = new byte[800*600];

short matriz[][] = new short[10][50];

short matx[][] = new short[10][]; // 2ª dimensão é variável
matx[0] = new short[15]; matx[1] = new short[40];

String[] nomes = new String[20];
String[] alunos = { “Pedro”, “Rita”, “Ana” };
String[][] linhas = { {“A”, “minha”}, {“casa”, “tem”, “um”}, {“rio”} };
String[][] galo = { {“O”, “O”, “X”},
                    {“X”, “X”, “O”},
                    {“O”, “X”, “O”} }
Ponto[] plano = new Ponto[200];
Object obj[] = new Object[100];
```

2.- COMPRIMENTO E ACESSO AOS ELEMENTOS

```
// comprimento
int comp = lista.length;
int numAlunos = alunos.length;

// acesso
int val = lista[0]; int num = lista[val*2];
short snum = matx[5][3];
String nome = nomes[index];
String pal = linhas[l][c];
out.println(lista[i]); out.println(nomes[i]);
out.printf(“Val = %d%n”, lista[i]);
```

3.- VARRIMENTO = ACESSO A TODOS OS ELEMENTOS

```
for(int i = 0; i <= a.length-1; i++) { ...a[i].....}   // por índice

for(IdTipo elem : IdArray) { …elem … }                 // for(each)
```

```
// Imprimir todos os elementos de um array
for(int i=0; i< lista.length; i++) out.println(lista[i]);

for(int elem : lista) out.println(elem);

// Exemplos de somatórios
int soma = 0;
for(int i=0; i< lista.length; i++) soma = soma + lista[i];

int soma1 = 0;
for(int elem : lista) soma1 += elem;

// Exemplos de concatenação de strings
String total = “”;
for(int i=0; I < nomes.length; i++) { total = total + nomes[i]; }

String total = “”;
for(String nome : nomes) { total += nome; }

// Contagem de pares e ímpares num array de inteiros
int par = 0, impar = 0;
for(int i = 0; i < a.lenght; i++)
  if (a[i]%2 == 0) par++; else impar++;
out.printf(“Pares = %d - Impares = %d%n”, par, impar);

// Total de inteiros > MAX de um array de arrays de inteiros
int maiores = 0; int MAX = Integer.MIN_VALUE;
for(int l = 0; l < nums.length; l++) {
   for(int c = 0; c < nums[l].length; c++)
      if (nums[l][c] > MAX) maiores++;
}

// Concatenação de strings de um array bidimensional
String[][] nomes = { {"Rita", "Pedro"}, ..... };
String sfinal = "";
for(int l = 0; l < nomes.length; l++) {
     for(int c = 0; c < nomes[l].length; c++) sfinal += nomes[l][c];
}

// usando for(each)
sfinal = "";
for(String[] lnomes : nomes)
     for(String nome : lnomes) sfinal += nome;
```

4.- Leitura de Valores para um array (usando a classe Input)

```
// Ler um número n, dado pelo utilizador, de valores de dado tipo, e guardá-los
// sequencialmente num array
int valor = 0;
out.print(“Quantos números inteiros quer introduzir ? “);
int n = Input.lerInt();
for(int i = 0; i <= n-1; i++) {
   valor = Input.lerInt(); lista[i] = valor;
}

// ou ainda, de forma mais simples mas equivalente:
int n = Input.lerInt(); int valor = 0;
for(int i = 0; i <= lista.length-1; i++) lista[i] = Input.lerInt();
```

```
// Ler um número não previamente fixado de valores de dado tipo e guardá-los
// num array pela sua ordem de leitura; Terá sempre que existir uma condição
// de paragem da leitura, seja porque foi lido um valor definido como valor de
// paragem (flag), seja porque o array já não tem mais capacidade.

int VALOR_STOP = -9999;          // valor que serve de sentinela/flag para parar a leitura
int[] lista = new int[MAXDIM];   // MAXDIM é uma constante predefinida no programa
boolean stop = false; int conta = 0; int valor;
while(!stop && conta<=MAXDIM-1) {
   valor = Input.lerInt();
   if(valor == VALOR_STOP)
         stop = true;
   else
         { lista[conta] = valor; conta++ }
}
```

5.- Algoritmo de Procura

```
// Procura de um valor lido (chave) num array, dando como resultado a sua
// posição, ou -1 caso não seja encontrado.

int[] lista = new int[MAXDIM];  // MAXDIM é uma constante predefinida no programa
..... // leitura ou inicialização
int chave; boolean encontrada = false;
int indice =0; int pos = -1;
out.print("Qual o valor a procurar no array? : ");
chave = Input.lerInt();
while(!encontrada && indice<=MAXDIM-1) {
   if(lista[indice] == chave) {
         encontrada = true; pos = indice;
   }
}
out.println("Valor " + chave + " encontrado na posição " + pos");
```

6.- Cópia Entre Arrays

```
System.arraycopy(array_fonte, índice_inicial_f, array_destino,
                 índice_inicial_d, quantos);

System.arraycopy(a, 0, b, 0, a.length);  // a.length elementos de a[0] para b desde b[0]
System.arraycopy(lista, 5, lista1, 1, 4);  // 4 elems a partir de lista[5] para lista1 desde 1
```

7.- Métodos da Classe java.util.Arrays (tipo => tipo simples)

```
int binarySearch(tipo[] a, tipo chave);   // devolve índice da chave, se existir, ou < 0;
boolean equals(tipo[] a, tipo[] b);       // igualdade de arrays do mesmo tipo;
void fill(tipo[] a, tipo val);            // inicializa o array com o valor parâmetro;
void sort(tipo[] a);                      // ordenação por ordem crescente;
String toString(tipo[] a);                // representação textual dos elementos;

String deepToString(array_multidim);      // repres. textual para multidimensionais;
boolean deepEquals(array_multi1, array_multi2);   // igualdade de arrays multidim;
```

Exercícios:

Ex1: Declarar, inicializar e imprimir os elementos de um *array* de inteiros.

```
// declarar, inicializar e imprimir os elementos de um array //
int[] lista = {12, 2, 45, 66, 7, 23, 99};
System.out.println("-----  ELEMENTOS DO ARRAY  ----------------------");
for(int i = 0; i < lista.length; i++) System.out.println("Elemento "+ i + "= " + lista[i]);
System.out.println("-------------------------------------------------");

// solução alternativa usando método da classe Arrays
int[] lista = {12, 2, 45, 66, 7, 23, 99};
out.println(Arrays.toString(lista));
```

Ex2: Escrever um programa que faça a leitura de N valores inteiros para um *array* e determine qual o maior valor introduzido e qual a sua posição no *array*.

```
import static java.lang.System.out;
public class ExArrays2 {
   public static void main(String[] args) {
      int n; // total de valores a serem lidos
      int[] lista = new int[100];
      int valor; int MAX = Integer.MIN_VALUE;
      int pos = -1;
      out.print("Numero de valores a ler: "); n = Input.lerInt();
      // leitura dos N valores para o array
      for(int i = 0; i <= n-1; i++) {
        out.print("Valor " + (i+1) + " : "); valor = Input.lerInt();
        lista[i] = valor; // lista[i] = Input.lerInt()
      }
      // determinação do MÁXIMO e da sua posição - SOLUÇÃO 1
      for(int i = 0; i < lista.length; i++)
         if(lista[i] > MAX) { MAX = lista[i]; pos = i; }
      out.println("Máximo1 = " + MAX + " - na posição: " + (pos + 1));
      // solução 2
      int ct = 0; MAX = Integer.MIN_VALUE;
      for(int elem : lista) {
         if(elem > MAX) { MAX = lista[ct]; pos = ct; }
         ct++;
      }
      out.println("Máximo2 = " + MAX + " - na posição: " + (pos + 1));
   }
}
```

Ex3: Modificar o programa anterior de modo a que a leitura dos N elementos para um *array* de inteiros seja realizada usando um método auxiliar que recebe o valor de N como parâmetro.

```
import static java.lang.System.out;
public class ExArrays3 {

   public static int[] leArrayInt(int n, int DIM) { // n = nº de elementos a serem lidos
      int[] a = new int[DIM];
      // leitura dos valores para o array
      for(int i = 0; i <= n-1; i++) {
         out.print(" Valor " + (i+1) + " : "); a[i] = Input.lerInt();
      }
      return a;
   }
```

```
    public static void main(String[] args) {
        int DIM = 100; int[] lista = new int[DIM]; int n, pos = 0;
        do {
            out.print("Número de elementos a ler: < " + DIM + " : ");
            n = Input.lerInt();
        }
        while(n > DIM);
        lista = leArrayInt(n, DIM);
        // determinação do MÁXIMO e da sua POSIÇÃO - 2 soluções
        int MAX = Integer.MIN_VALUE;
        for(int i =0; i<= lista.length-1; i++)
            if( lista[i] > MAX ) { MAX = lista[i]; pos = i; }
        out.println("Máximo1 = " + MAX + " na posição: " + (pos +1));
        int i = 0;
        // solução 2 usando foreach
        for(int elem : lista) {
         i++;
         if( elem> MAX) { MAX = lista[i]; pos = i; }
        }
        out.println("Máximo2 = " + MAX + " na posição: " + (pos + 1));
    }
}
```

Ex4: Modificar o programa anterior de modo a que quer a leitura dos N elementos quer a determinação do máximo elemento do *array* sejam realizados em métodos auxiliares do método main().

```
/**
 * Programa que usa um método auxiliar para ler inteiros válidos e inseri-los num array
* de inteiros que é devolvido como resultado. Um outro método auxiliar determina o
* maior elemento de um array com número de elementos dado como parâmetro.
* O programa apresenta-se, deste modo, devidamente estruturado.
 *
 * @author F. Mário Martins
 * @version 1.0/2005
 */

import static java.lang.System.out;
public class ExArrays4 {

  public static final int DIM = 100;

  public static int[] leArray(int num) {
     int[] nums = new int[num];
     int n = 0;
     for(int i = 0; i< num; i++) {
         out.print("Valor " + (i+1) + " : ");
         nums[i] = Input.lerInt();
     }
     return nums;
  }

  public static int max(int[] nums, int total) {
     int max = Integer.MIN_VALUE;
     for(int i = 0; i < total; i++)
       if (nums[i] > max) max = nums[i];
     return max;
  }
```

```
  public static void main(String[] args) {
    int[] arrayNum = new int[DIM];
    out.print("Total de números a ler: "); int dim = Input.lerInt();
    arrayNum = leArray(dim);
    int maximo = max(arrayNum, dim);
    out.println("Máximo = " + maximo);
    Arrays.sort(arrayNum);
    out.println("--  Array Ordenado  --");
    for(int i = 0; i<dim; i++) out.println(arrayNum[i]);
  }
}
```

Ex5: Escrever um programa que faça a leitura de N elementos inteiros para um *array,* mas que os insira de forma a que o *array* se mantenha sempre ordenado por ordem crescente.

```
/**
 * Programa que usa um método auxiliar para ler inteiros válidos e inseri-los num array
 * de inteiros que é mantido permanentemente ordenado.
 * Assim, a inserção de cada elemento é uma inserção ordenada. Quanso o elemento
 * a inserir é menor que algum dos elementos do array, tem que se fazer o "shift" de
todas
 * as posições a partir deste elemento maior, 1 posição para a frente.
 *
 * @author F. Mário Martins
 * @version 1.0/2005
 */

import static java.lang.System.out;
public class ExArrays5 {

  public static final int DIM = 100;

  public static int[] leArray(int num) {
     int[] nums = new int[num+1];
     int valor; int index;
     // i será sempre o número de elementos já inseridos no array
     for(int i = 0; i < num; i++) {
        out.print("Valor " + (i+1) + " : ");
        valor = Input.lerInt();
        // determina a posição para inserir => encontrar o 1º valor maior
        index = 0;
        while(nums[index] <= valor  && index < i ) { index++; }
        //
        if( i == 0) nums[0] = valor;
        else {
                // a partir do último elemento do array, faz shift de todos para a
                // posição seguinte no array
               for(int p = i; p > index; p--) nums[p] = nums[p-1];
               // insere o novo valor mantendo o array ordenado
               nums[index] = valor;
              }
        for(int x = 0; x <= i; x++) out.println(nums[x]);
      }
     return nums;
  }

  public static void main(String[] args) {
    int[] arrayNum = new int[DIM];
    out.print("Total de números a ler: "); int dim = Input.lerInt();
    arrayNum = leArray(dim);
    out.println("--  Array Ordenado  --");
```

```
    for(int i = 0; i < dim; i++) out.println(arrayNum[i]);
  }
}
```

Ex6: Escrever um programa que faça a leitura de N elementos inteiros para um *array,* receba dois índices válidos do *array* lido e crie um *array* apenas com os elementos entre esses dois índices. Usar um método auxiliar.

```
import static java.lang.System.out;
public class ExArrays6 {

  public static final int DIM = 100;

  public static int[] leArray(int num) {
     int[] nums = new int[num];
     for(int i = 0; i< num; i++) {
        out.print("Valor Indice " + i + " : ");
        nums[i] = Input.lerInt();
     }
     return nums;
  }

  public static int[] selecciona(int[] nums, int start, int stop) {
     int[] res = new int[stop-start+1];
     for(int i = 0; i <= stop-start; i++) res[i] = nums[start + i];
     return res;
  }

  public static void main(String[] args) {
    int[] arrayNum = new int[DIM]; int inicio, fim;
    out.print("Total de números a ler: "); int dim = Input.lerInt();
    arrayNum = leArray(dim);

    do {
      out.print("Indice inicial para selecção (0 ..): ");
      inicio = Input.lerInt();
    }
    while(inicio < 0 || inicio > dim -1);

    do {
       out.print("Indice final ( > inicial): ");
       fim = Input.lerInt();
    }
    while(fim < inicio || fim > dim - 1);

    int[] subarray = selecciona(arrayNum, inicio, fim);

    out.println("--  Array Resultado --");
    for(int i = 0; i <= subarray.length - 1; i++) out.println(subarray[i]);
  }
}
```

Ex7: Escrever um programa que leia uma série de palavras terminada por "zzz" para um array, aceite repetidamente uma palavra até que seja introduzida a palavra "xxx" e verifique se tal palavra existe no array. Caso exista o programa deverá removê-la do array.

```
import static java.lang.System.out;
public class Ex7Arrays {

   public static int MAXDIM = 100;
```

```
   public static int lePalavras(String[] palavras) {
      // preenche o array parâmetro e devolve o número de palavras lidas
      String palavra; int conta = 0;
      out.print("Palavra 1 : "); palavra = Input.lerString();
      while(!palavra.equals("zzz") && !palavra.equals("ZZZ") && conta < MAXDIM) {
         palavras[conta] = palavra; conta++;
         out.print("Palavra " + (conta + 1) + " : "); palavra = Input.lerString();
      }
      return conta;
   }

   public static int procuraPal(String[] palavras, String palavra) {
      boolean encontrada = false; int index;
      index = 0;
      while(!encontrada && index <= palavras.length-1) {
         if(palavras[index].equals(palavra)) encontrada = true;
         else index++;
      }
      return (encontrada ? index : 0);
   }

   public static String[] removePal(String[] palavras, int index, String pal) {
     // faz "shift down" desde indice+1 até length-1
     for(int p = index+1; p <=palavras.length-1; p++)
       palavras[p-1] = palavras[p];
     return palavras;
   }

   public static void main(String[] args) {
      String[] dicionario = new String[MAXDIM];
      String palavra, resp; int indice;
      int total = lePalavras(dicionario);
      out.println("----   DICIONÁRIO ----");
      for(int p = 0; p <= total-1; p++) out.println(dicionario[p]);
      out.println("----------------------");
      do {
        out.print("Palavra a remover: "); palavra = Input.lerString();
        indice = procuraPal(dicionario, palavra);
        out.println("Indice " + indice);
        if (indice == 0)
          out.println("PALAVRA NÃO EXISTENTE !!");
        else {
            dicionario = removePal(dicionario, indice, palavra);
            total--;
            out.println("----  DICIONÁRIO ACTUAL  ----");
            for(int i = 0; i <= total-1; i++)
              out.println(dicionario[i]);
            }
       out.println("Pretende remover mais palavras (S/*) ? ");
       resp = Input.lerString();
      }
      while(resp.equals("S") || resp.equals("s"));
      out.println("--- FIM DO PROGRAMA ---");
   }
}
```

Ex8: Escrever um programa que leia para um array os vencimentos mensais brutos (íliquidos) dos 20 funcionários de uma empresa. O programa possuirá uma tabela de retenção de IRS constante, do tipo

Salário Ilíquido	% Retenção de IRS
0 a 500 Euros	5
501 a 1000 Euros	10
1001 a 2000	20
2001 a 4000	30
4001 ou mais	40

Pretende-se que o programa crie um array no qual, para o respectivo funcionário cujo vencimento bruto se encontra no array lido, sejam introduzidos as respectivas retenções para IRS. No final, o programa deve apresentar uma listagem com os vencimento bruto, retenção de IRS e vencimento líquido para os 20 funcionários.

```
/**
 * Calculo de Retenção de IRS sobre vencimentos de funcionários.
 *
 * @author F. Mário Martins
 * @version 3/2007
 */
import static java.lang.System.out;
public class Ex8Arrays {

   public static int MAXDIM = 100;

   public static double[] leSalarios(int total) {
      double[] salarios = new double[total];
      for(int i = 0; i <= total - 1; i++) {
           out.print("Vencimento " + (i+1) + " : " );
           salarios[i] = Input.lerInt();
      }
      return salarios;
   }

   public static double calcula_imposto(int[] escaloes, int[] taxas,
                                double salario) {
      int index = 0;
      while(escaloes[index] < salario && index < escaloes.length) { index++; }
      return ((double)taxas[index])/100*salario;
   }

   public static void mostra_resultados(double[] vencs, double[] impostos) {
      out.println("\n\n-------------------------------------------------------");
      out.println("\tVENCIMENTO BRUTO\t IRS\t\tLÍQUIDO");
      for(int i = 0; i <= vencs.length - 1; i++)
         out.printf("N. %2d\t\t %8.2f\t%6.2f\t\t%8.2f%n",
                  (i+1), vencs[i], impostos[i], vencs[i]-impostos[i]);
      out.println("\n-------------------------------------------------------");
   }

   public static void main(String[] args) {
      int totalFunc;
      do {
         out.print("Numero Total de funcionários < " + MAXDIM + " : ");
         totalFunc = Input.lerInt();
      }
      while(totalFunc > MAXDIM);
      // arrays de vencimentos, taxas, escalões e impostos
      double[] vencimentos = new double[totalFunc];
      double[] impostos = new double[totalFunc];
      int[] escaloes = { 500, 1000, 2000, 4000, Integer.MAX_VALUE };
```

```
      int[] taxas = { 5, 10, 20, 30, 40 };
      // cálculos de imposto
      vencimentos = leSalarios(totalFunc);
      for(int v = 0; v <= totalFunc - 1; v++) {
         impostos[v] = calcula_imposto(escaloes, taxas, vencimentos[v]);
      }
      mostra_resultados(vencimentos, impostos);

   }
}
```

Ex9: Escrever um programa que simule o jogo do Euromilhões. O programa gera aleatoriamente uma chave contendo 5 números (de 1 a 50) e duas estrelas (1 a 9).
Em seguida são pedidos ao utilizador 5 números e duas estrelas (a aposta). O programa deverá em seguida apresentar a chave gerada e o número de números e estrelas certos da aposta do utilizador. Naturalmente devem ser usados arrays para guardar os dados.

```
/**
 * Jogo simulador do Euromilhões
* @author F. Mário Martins
 * @version 3/2007
 */
import static java.lang.Math.random;
import static java.lang.System.out;
public class Ex9Arrays {

   public static int[] geraNumeros(int total, int menor, int maior) {
      int[] sorteio = new int[total];
      for(int i = 1; i <= total; i++)
         sorteio[i-1] = menor + (int) (random()*maior);
      return sorteio;
   }

   public static int[] leNums(int total, int inf, int sup) {
      int[] nums = new int[total]; boolean repetido = false; int num;
      for(int i = 0; i <= total - 1; i++) {
        do {
          out.print("Numero " + (i+1) + " (entre " + inf + " e " + sup + " ): ");
          num = lnput.lerInt();
          repetido = existe(nums, i+1, num);
          if (repetido) out.println("Não serve. É repetido !");
        }
        while(!(num >= inf && num <= sup) || repetido);
        nums[i] = num;
      }
      return nums;
   }

   public static boolean existe(int[] lista, int dim, int elem) {
      // verifica se elem existe na lista dada como parâmetro
      int index = 0; boolean existe = false;
      while(index <= dim-1 && !existe) {
         existe = (lista[index] == elem); index++;
      }
      return existe;
   }
   public static int comparaArrays(int[] lista1, int[] lista2) {
       // quantos elementos de lista2 existem em lista1
      boolean existe; int conta = 0;
      for(int i = 0; i <= lista2.length-1; i++) {
```

```
            existe = existe(lista1, lista1.length, lista2[i]);
            if (existe) conta++;
        }
        return conta;
    }

    public static void mostra_resultados(int numsOk, int estrelasOk) {
        out.println(" Acertou em " + numsOk + " números e "
                + estrelasOk + " estrelas.");
        out.println("-----------------------------------------------------");
        if( numsOk == 5 && estrelasOk == 2 )
          out.println(" VOCÊ É MULTIMILIONÁRIO !!! EXCÊNTRICO !!!");
    }

    public static void main(String[] args) {
        int[] numeros = new int[5];
        int[] estrelas = new int[2];
        int[] palpiteNums = new int[5];
        int[] palpiteEstrelas = new int[2];
        String resp; int numCertos, estrelasCertas;

        do {
          numeros = geraNumeros(5, 1, 50);
          estrelas = geraNumeros(2, 1, 9);
          palpiteNums = leNums(5, 1, 50);
          palpiteEstrelas = leNums(2, 1, 9);
          //
          numCertos = comparaArrays(numeros, palpiteNums);
          out.println("--------  CHAVE DO EUROMILHÕES  -----------");
          for(int i = 0; i < 5; i++) out.print(numeros[i] + " ");
          out.print(" -- ");
          for(int i = 0; i < 2; i++) out.print(estrelas[i] + " ");
          out.println();
          out.println("-------------------------------------------");
          estrelasCertas = comparaArrays(estrelas, palpiteEstrelas);
          mostra_resultados(numCertos, estrelasCertas);
          //
          out.println("Novo Jogo ? (S/*) : ");
          resp = lnput.lerString();
        }
        while(resp.equals("S"));
        out.println("------ SEMPRE A CRIAR EXCÊNTRICOS ..... ");
    }
}
```

Ex10: Modifique o programa do exemplo 9 de modo a que no final o programa apresente o somatório de todos os vencimentos e de todos os impostos retidos aos funcionários.

FICHA PRÁTICA 3

LABORATÓRIO DE CLASSES I

SÍNTESE TEÓRICA

Nas linguagens de PPO os objectos são divididos em duas grandes categorias de entidades: as INSTÂNCIAS (objectos computacionais puros) e as CLASSES (objectos fundamentalmente de definição, mas que também podem ser de definição e até de prestação de serviços).

CLASSES são objectos particulares que, entre outras funções, guardam a descrição da **estrutura** (*variáveis de instância*) e do **comportamento** (*métodos*) que são comuns a todas as instâncias a partir de si criadas.

As instâncias de uma classe são criadas usando a palavra reservada **new** e um método especial para criação de instâncias, designado **construtor**, que tem, obrigatoriamente, o mesmo nome da classe, podendo ter ou não parâmetros. Exemplos:

```
Triangulo tri1 = new Triangulo();
Ponto p = new Ponto(); Ponto p2 = new Ponto(5, -1);
Turma t = new Turma(); Aluno al1 = new Aluno();
```

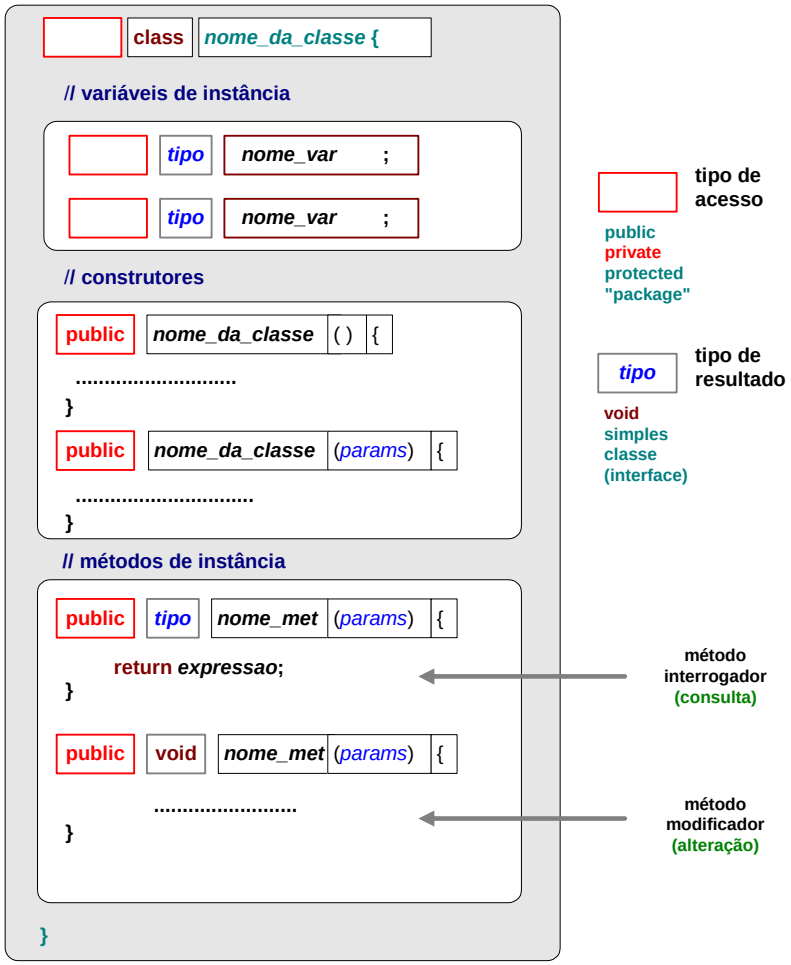


Fig.1 – Estrutura de definição de instâncias típica de uma classe.

1.- CLASSE SIMPLES COM MÉTODOS COMPLEMENTARES USUAIS

```
/**
 * Pontos descritos como duas coordenadas reais.
*/
import static java.lang.Math.abs;
public class Ponto2D {

  // Construtores usuais
  public Ponto2D(double cx, double cy) { x = cx; y = cy; }
  public Ponto2D(){ this(0.0, 0.0); }  // usa o outro construtor
  public Ponto2D(Ponto2D p) { x = p.getX(); y = p.getY(); }

  // Variáveis de Instância
  private double x, y;

  // Métodos de Instância
  public double getX() { return x; }
  public double getY() { return y; }

  /** incremento das coordenadas */
  public void incCoord(double dx, double dy) {
    x += dx; y += dy;
  }
  /** decremento das coordenadas */
  public void decCoord(double dx, double dy) {
    x -= dx; y -= dy;
  }
  /** soma as coordenadas do ponto parâmetro ao ponto receptor */
  public void somaPonto(Ponto2D p) {
    x += p.getX(); y += p.getY();
  }
  /** soma os valores parâmetro e devolve um novo ponto */
  public Ponto2D somaPonto(double dx, double dy) {
   return new Ponto2D(x += dx, y+= dy);
  }
  /* determina se um ponto é simétrico (dista do eixo dos XX o
     mesmo que do eixo dos YY */
  public boolean simetrico() {
   return abs(x) == abs(y);
  }

  /** verifica se ambas as coordenadas são positivas */
  public boolean coordPos() {
   return x > 0 && y > 0;
  }

  // Métodos complementares usuais

  /* verifica se os 2 pontos são iguais */
  public boolean igual(Ponto2D p) {
    if (p != null) return (x == p.getX() && y == p.getY());
    else return false;
  }

  // outra versão de igual(Ponto2D p)
  public boolean igual1(Ponto2D p) {
   return (p == null) ? false : x == p.getX() && y == p.getY();
  }



  /** Converte para uma representação textual  */
  public String toString() {
```

```
    return new String("Pt2D = " + x + ", " + y);
  }

  /** Cria uma cópia do ponto receptor (receptor = this) */
  public Ponto2D clone() {
    return new Ponto2D(this);
  }
}
```

2.- Regras Fundamentais para a Criação de Classes.

- As variáveis de instância devem ser declaradas como **private**, satisfazendo assim o princípio do encapsulamento;
- Pelo menos dois construtores devem ser criados: o que redefine o construtor de JAVA por omissão e o construtor para cópia (recebe um objecto da mesma classe e inicializa as variáveis de instância com os valores deste);
- Por cada variável de instância devemos ter um método de consulta **getVar()** ou equivalente, e um método de modificação **setVar()** ou equivalente;
- Devem ser sempre incluídos nas definições das classes, a menos que não se justifique por serem demasiado complexas, os métodos complementares: **equals()**, **toString()** e **clone()**;
- A classe deverá ser bem documentada, em especial os métodos, usando comentários, em particular os comentários **/** ... */** para que seja produzida automaticamente documentação de projecto através do utilitário **javadoc**.

3.- Desenvolvimento em BlueJ.

O ambiente BlueJ fornece um ambiente integrado para a criação, desenvolvimento e teste das classes a conceber. As figuras seguintes mostram, de forma simples, as facilidades de edição, de criação de instâncias e teste destas no ambiente BlueJ.

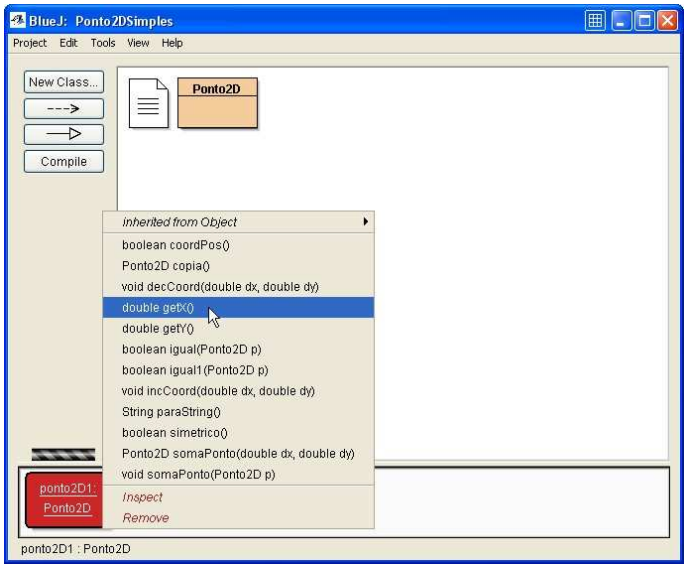


Fig.2 – API e Invocação de Método

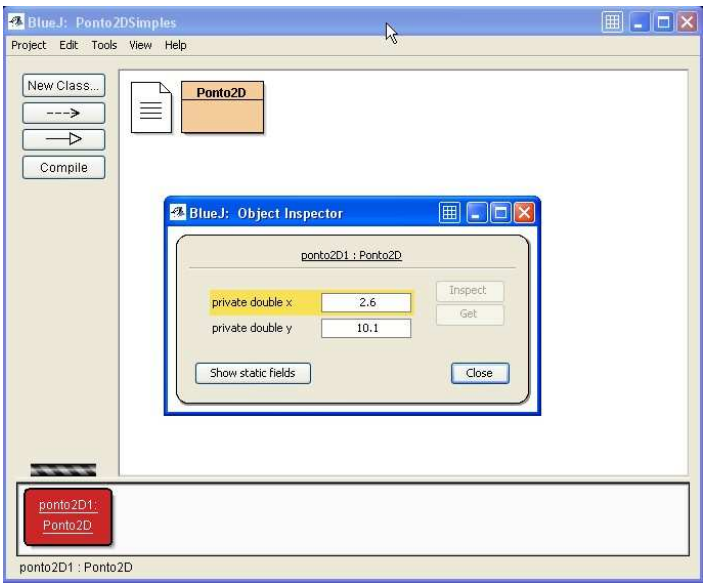


Fig.3 - Resultado de *Inspect*

Exercícios:

Ex 1: Um pixel é um ponto de coordenadas x e y inteiras mas que tem a si associada uma cor de 0 a 255. Crie uma classe Pixel que permita criar "pixels", sabendo-se que, para além das usuais operações de consulta, cada pixel deve responder às seguintes mensagens:

- Deslocar-se para cima, para baixo, para a esquerda e para a direita de um valor real;
- Mudar a sua cor para uma nova cor de número dado (entre 0 e 255);
- Sendo o espectro de cores desde o 0 - Preto a 255 – Branco, sendo o preto de 0 a 2 e o branco de 253 a 255, entre o preto e o branco situam-se o cinza, o azul, o verde, o vermelho e o amarelo, sendo o espaço das 250 cores restantes dividido igualmente entre estas 5 (50 para cada). Exemplo: de 3 a 52 são os cinza e de 53 a 102 os azuis. Escrever um método que devolva uma String correspondente à cor actual do pixel.
- Não esquecer os métodos **equals()**, **toString()** e **clone()**.

Ex 2: Um Segmento de recta é representável por dois pontos de coordenadas reais: o início e o fim do segmento. Escreva uma classe Segmento que implemente os métodos seguintes:

- Calcular o comprimento do segmento;
- Determinar o declive do segmento, cf. (y2-y1)/(x2-x1);
- Determinar se o segmento sobe ou desce a partir do seu início (devolver uma String);
- Deslocar o segmento dx em XX e dy no eixo dos YY;
- Se o segmento for o diâmetro de uma circunferência, determinar qual o perímetro desta;

Ex 3: Um veículo motorizado é caracterizado pela sua matrícula, quilometragem total (Km), pelo número de litros total do seu depósito (capacidade), pelo número de litros contidos em tal depósito (reserva incluída = 10 litros). Sabe-se ainda o seu consumo médio aos 100 Km, que raramente varia. O veículo possui ainda um contador de viagens realizadas.
Crie uma classe Veiculo que implemente métodos de instância que permitam obter os seguintes resultados:

- Determinar quantos quilómetros é possível percorrer com o combustível que está no depósito;

- Registar uma viagem de K quilómetros e actualizar os dados do veículo;
- Determinar se o veículo já entrou na reserva;
- Dado um valor médio de custo por litro, calcular o valor total gasto em combustível;
- Dado um valor médio de custo por litro, calcular o custo médio por Km;
- Meter L litros de gasolina, ou o máximo possível < L, sem transbordar.

Ex 4: Um Cartão de Cliente (actualmente tão em voga) é um cartão de compras que acumula pontos de bonificação à medida que são registadas compras, e que possui o valor total em dinheiro das compras realizadas, um código alfanumérico e um nome de titular. Num dado estabelecimento as regras são as seguintes: por cada compra efectuada em Euros o cliente recebe de bónus um número de pontos (inteiro) que é o arredondamento para baixo de 10% do valor da compra. Sempre que é atingido um valor múltiplo de 50 pontos o cliente acumula mais 5 pontos por cada 50, que são juntos aos que já tinha no cartão. Escrever uma classe `CartaoCliente` cujas instâncias exibam este comportamento, e permitam ainda:

- Descontar P pontos ao cartão devido a levantamento de um prémio;
- Modificar o titular do cartão;
- Modificar a taxa de prémio, passando, por exemplo de 10% para 11%;
- Descarregar os pontos de um cartão para o nosso cartão;
- Inserir no cartão a informação de uma nova compra de certo valor, e actualizar dados;

Ex 5: Um Rectângulo de base paralela ao eixo dos XX é representável por dois pontos de coordenadas reais, que são os extremos da sua diagonal. Desenvolva uma classe `Rectangulo` com métodos que realizem as operações seguintes:

- Calculem os comprimentos da base, da altura e da diagonal;
- Calcule o perímetro do rectângulo;
- Calcule a área do rectângulo;
- Realize o deslocamento do rectângulo em XX e em YY;

Ex 6: Um Produto de um dado stock de produtos possui as seguintes características de informação: código, nome, quantidade em stock, quantidade mínima, preço de compra e preço de venda a público. Desenvolva uma classe `Produto` e os seguintes métodos de instância:

- Alterar a quantidade de um produto, ou por saída ou por entrada de uma dada quantidade do produto no stock;
- Modificar o nome do produto;
- Modificar o preço de venda de um dado produto;
- Determinar o valor total da quantidade em stock em caso de venda;
- Determinar o lucro actual de tal stock em caso de venda total;
- Dada uma encomenda de X unidades do produto, determinar o preço total de tal encomenda;
- Verificar se um produto está já abaixo do nível mínimo de stock.

Ex 7: Um número Complexo na sua forma rectangular representa-se por `z = a + bi`, e possui uma parte real e uma parte imaginária, sendo a e b números reais que são os coeficientes real e imaginário do número. Outra forma de representar números complexos designa-se por representação polar e tem a forma `z = r(cos Φ + sen Φ i)`, sendo `r = sqrt( a*a + b*b)`, `r.cos Φ = a`, `r.sen Φ = b`, e sendo o valor de `Φ = arctan b/a`. Desenvolva uma classe `Complexo` com os seguintes métodos de instância:

- Cálculo da soma de dois complexos $z + w = (a + bi) + (c + di) = (a+c) + (b+d)i$ dando um novo número complexo como resultado;
- Cálculo do produto de dois complexos $z * w = (a + bi) * (c + di) = (ac-bd) + (bc+ad)i$ dando um novo número complexo como resultado;
- Cálculo do conjugado de um complexo, sendo $conjugado(a+ bi) = a – bi$;
- Calcular a representação polar do complexo e devolvê-la como `String`;
- Métodos `igual(Complexo c)` e `toString()`.

Ex 8: Um cronómetro marca as diferenças entre dois tempos registados (início e fim). Um cronómetro "double-split" faz ainda mais: Inicia uma contagem de tempo, faz uma primeira paragem, mas continua a contar o tempo até ser feita a segunda paragem. Criar uma classe `CronometroDS` que permita calcular:

- O tempo total em minutos, segundos e milissegundos entre o início e a primeira paragem;
- O tempo total em minutos, segundos e milissegundos entre o início e a segunda paragem;
- A diferença de tempos entre a primeira e a segunda paragem de tempo;
- Determinar o tempo absoluto em *hora-min-seg-miliseg* do arranque e de cada paragem;

Ex 9: Uma Conta Bancária a prazo é criada com um código, um titular, tem uma data de início (dia, mês e ano) de contagem de juros que é actualizada sempre que os juros são calculados e adicionados ao capital, tem um dado montante de capital depositado, com um prazo para cálculo de juros, e uma taxa de juro de t% para tal prazo, definida aquando da sua criação. Crie uma classe `ContaPrazo` que, para além dos construtores e dos métodos de consulta, permita realizar as seguintes operações:

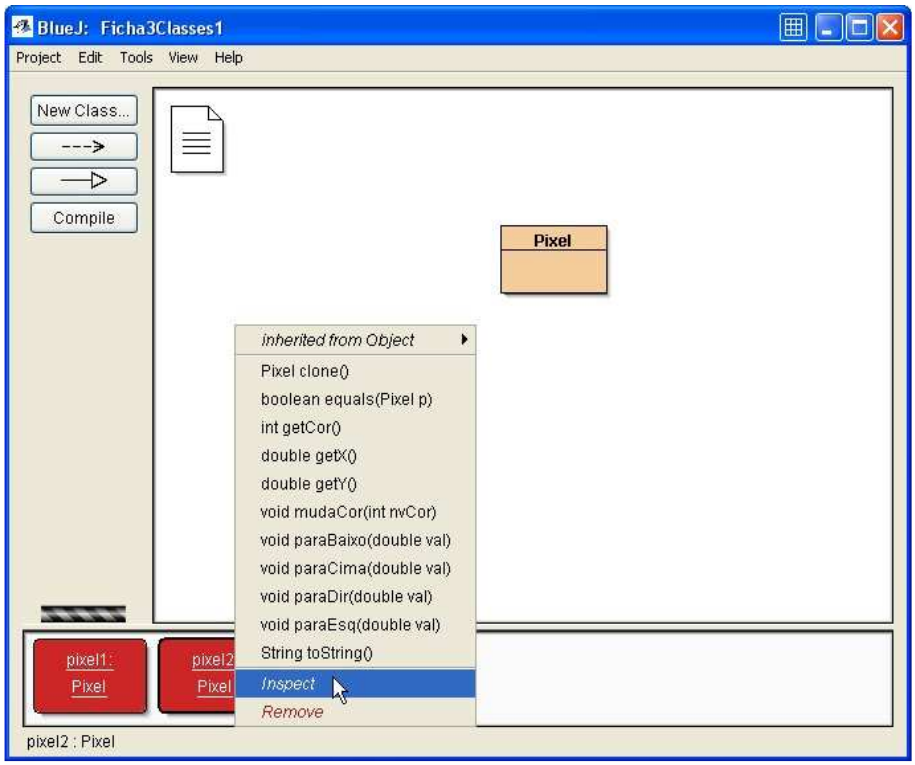
- Calcular o número de dias passados desde a abertura da conta;
- Alterar o titular da conta ou alterar a taxa de juros;
- Atingido o prazo para juros, calcular tais juros, juntá-los ao capital, e registar a nova data de cálculo de juros;
- Verificar se hoje é o dia de calcular os juros;
- Fechar a conta calculando o valor total a pagar ao titular (capital inicial + juros);

Resoluções:

Ex1:

```
/**
 * Classe que representa "pixels" e operações diversas sobre estes.
 *
 * @author FMM
 * @version 1.0/ 3-2007
 */
public class Pixel {

    // Variáveis de Instância
    private double x;   // coordenada em X
    private double y;   // coordenada em Y
    private int cor;    // entre 0 e 255

    // Construtores
    public Pixel() { x = 0.0; y = 0.0; cor = 1; }

    public Pixel(int cor) {
       x = 0.0; y = 0.0;
       this.cor = (cor >= 0 && cor <= 255) ? cor : 0;
    }

    public Pixel(double x, double y, int cor) {
       this.x = x; this.y = y;
       this.cor = (cor >= 0 && cor <= 255) ? cor : 0;
    }

    public Pixel(Pixel p) {
       x = p.getX(); y = p.getY(); cor = p.getCor();
    }

    // Métodos de Instância

    public double getX() { return x; }
    public double getY() { return y; }
    public int getCor() { return cor; }

    /** Modificação da Cor */
    public void mudaCor(int nvCor) { cor = nvCor; }

    /** Método auxiliar que resolve todas as deslocações do pixel */
    private void desloca(double dx, double dy) { x += dx; y += dy; }

    // Métodos de deslocamento do pixel

    /** Desloca o pixel val unidades para cima no eixo dos YY */
    public void paraCima(double val) { this.desloca(0.0, val); }

    /** Desloca o pixel val unidades para baixo no eixo dos YY */
    public void paraBaixo(double val) { this.desloca(0.0, -val); }

    /** Desloca o pixel val unidades para a esquerda no eixo dos XX */
    public void paraEsq(double val) { this.desloca(-val, 0.0); }

    /** Desloca o pixel val unidades para a direita no eixo dos XX */
    public void paraDir(double val) { this.desloca(val, 0.0); }
```

```
    /** Teste de igualdade de pixels */
    public boolean equals(Pixel p) {
       return x == p.getX() && y == p.getY() && cor == p.getCor();
    }

    /** toString() */
    public String toString() {
           StringBuilder s = new StringBuilder();
           s.append("Pixel = (" + x + "," + y + ", " + cor + ")\n");
           return s.toString();
    }

    /** Cópia do pixel receptor */
    public Pixel clone() {
       return new Pixel(this);
    }
}
```



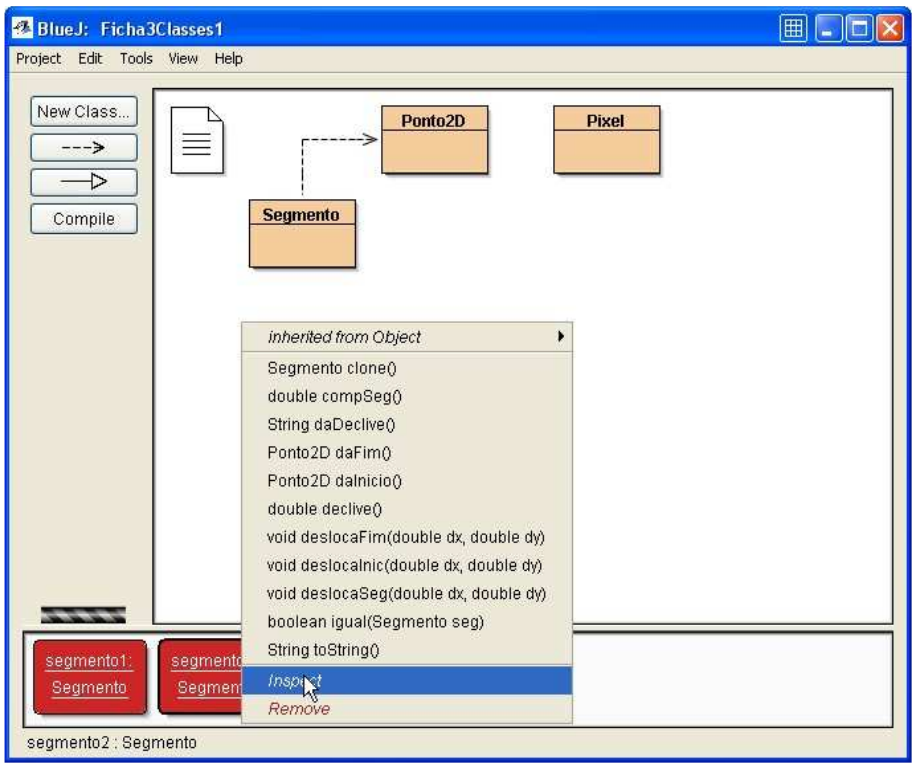
Janela do Projecto da classe Pixel

Ex2:

```
/**
 * Classe que implementa Segmentos de recta, usando 2 Ponto2D de
 * coordenadas reais que representam os limites do segmento
.*
 * @author F. Mário Martins
 * @version 2.0/ 3-2007
**/
  import static java.lang.Math.sqrt;
  public class Segmento {

    // Variáveis de Instância
    private Ponto2D p1;   // ponto início
    private Ponto2D p2;   // ponto fim
```

```
// Construtores

/** Redefine construtor por omissão */
public Segmento() { p1 = new Ponto2D(); p2 = new Ponto2D(); }

/** Construtor a partir de pontos */
public Segmento(Ponto2D pa, Ponto2D pb) { p1 = pa.clone(); p2 = pb.clone(); }

/** Construtor para cópia */
public Segmento(Segmento segm) {
   p1 = segm.daInicio(); p2 = segm.daFim(); // daInicio() e daFim() já fazem clone()
}

// Métodos de Instância

 /** Devolve o ponto início do segmento */
public Ponto2D daInicio() { return p1.clone(); }

 /** Devolve o ponto fim do segmento */
public Ponto2D daFim() { return p2.clone(); }

 /** Comprimento = raiz quadrada da soma dos quadrados das
  *  diferenças das coordenadas.
  */
public double compSeg() {
   double dx = p1.getX() - p2.getX(); double dy = p1.getY() - p2.getY();
   dx = dx*dx; dy = dy * dy;
   return sqrt(dx+dy);
}

 /** Determina o declive do segmento */
public double declive() {
   double dx = p1.getX() - p2.getX(); double dy = p1.getY() - p2.getY();
   return dy/dx;
}

 /** Dá o declive do segmento sob a forma de texto */
public String daDeclive() {
    String declive = "";
    if (this.declive() > 0) declive = "Segmento a subir ..";
    else if(this.declive() < 0) declive = "Segmento a descer ..";
         else declive = "Segmento horizontal ..";
    return declive;
}

 /** Desloca  o segmento em XX e YY */
public void deslocaSeg(double dx, double dy) {
    p1.incCoord(dx, dy); p2.incCoord(dx, dy);
}

 /** Desloca o ponto início */
public void deslocaInic(double dx, double dy) { p1.incCoord(dx, dy); }

 /** Desloca o ponto fim */
public void deslocaFim(double dx, double dy) { p2.incCoord(dx, dy); }

 /** Determina se o receptor e o parâmetro são segmentos iguais */
public boolean igual(Segmento seg) {
    return p1.igual(seg.daInicio()) && p2.igual(seg.daFim());
}
```

```
    /** Cópia do receptor */
    public Segmento clone() { return new Segmento(this); }

    /** Converte para representação como String */
    public String toString() {
       return new String("Segmento[" + p1.paraString() + ", " + p2.paraString() + "]");
    }
}
```



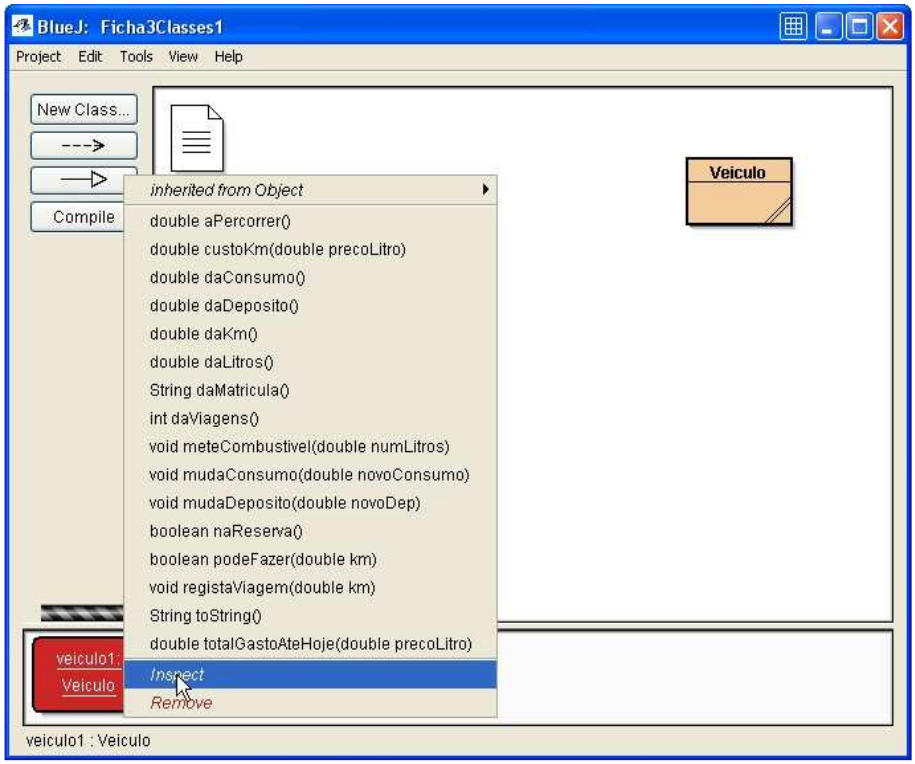
Janela do Projecto da classe Segmento

Ex3:

```
/**
 * Class Veiculo descreve veículos, seus depósitos, consumos, kms,
 * e calcula e regista dados sobre os mesmos.
 *
 * @author FMM
 * @version 1.0 3-2007
 *
 */
public class Veiculo {

   public static double RESERVA = 10;  // reserva de 10 litros

   // Variáveis de Instância
   private String matricula;
   private double kmTotal;
   private double deposito;
   private double litros;                    // <= 10 litros = reserva
   private double consumoMed;
   private int viagens;
```

```
// Construtores

/** Construtor a partir das partes */
public Veiculo(double kmInic, double dep, double numLitros,
               double consumoMedio, int numViagens, String mat) {
  kmTotal = kmInic; deposito = dep; litros = numLitros;
  consumoMed = consumoMedio; viagens = numViagens; matricula = mat;
}

/** Construtor por cópia */
public Veiculo(Veiculo veic) {
 matricula = veic.daMatricula(); kmTotal = veic.daKm();
 deposito = veic.daDeposito(); litros = veic.daLitros();
 consumoMed = veic.daConsumo(); viagens = veic.daViagens();
}

// Métodos de Instância
public String daMatricula() { return matricula; }
public double daKm() { return kmTotal; }
public double daDeposito() { return deposito; }
public double daLitros() { return litros; }
public double daConsumo() { return consumoMed; }
public int daViagens() { return viagens; }

// Modificadores

/** Altera capacidade do depósito */
public void mudaDeposito(double novoDep) { deposito = novoDep; }

/** Altera consumo médio */
public void mudaConsumo(double novoConsumo) { consumoMed = novoConsumo; }

// Operações

/** Quilómetros que ainda pode percorrer com o combustível que tem */
public double aPercorrer() { return litros/consumoMed*100; }

/** Poderá percorrer mais km quilómetros ?? */
public boolean podeFazer(double km) {
   double vaiGastar = consumoMed/100*km;
   return litros > vaiGastar;
}

/** Registo de uma nova viagem */
public void registaViagem(double km) {
   double gastou = consumoMed/100*km;
   viagens++; kmTotal += km;
   litros -= gastou;
}

/** Estará já na reserva de combustível? */
public boolean naReserva() { return litros <= RESERVA; }

/** Total de dinheiro gasto em combustível desde o início */
public double totalGastoAteHoje(double precoLitro) {
   return consumoMed/100*kmTotal*precoLitro;
}

/** Custo por quilómetro percorrido */
public double custoKm(double precoLitro) {
   return consumoMed/100*precoLitro;
}
```

```
    /** Mete combustível no veículo, sem deitar por fora !! */
    public void meteCombustivel(double numLitros) {
        double paraEncher = (deposito - litros);
        litros += numLitros > paraEncher ? paraEncher : numLitros;
    }

    /** Representação Textual do veículo */
    public String toString() {
        StringBuilder s = new StringBuilder();
        s.append("-------- VEÍCULO --------");
        s.append("Matricula: " + matricula + "\n");
        s.append("Total KM: " + kmTotal + "\n");
        s.append("Depósito: " + deposito + " litros.\n");
        s.append("Litros: " + litros + "\n");
        s.append("Consumo Médio: " + consumoMed + "\n");
        s.append("Viagens feitas: " + viagens + "\n");
        s.append("-------------------------\n");
        return s.toString();
    }
}
```



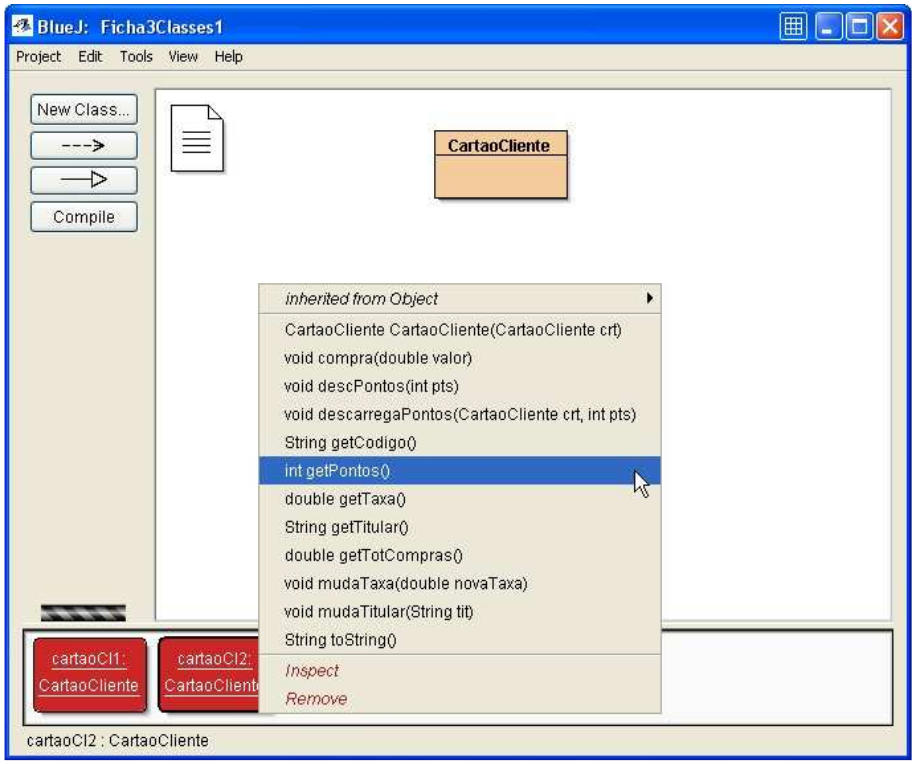
Janela do Projecto da classe Veiculo

Ex4:

```
/**
 * A classe CartaoCliente representa o funcionamento de um cartão de
 * compras que permite acumular pontos para serem posteriormente trocados
 * por prémios, etc.
 *
 * @author FMM
 * @version 1.0 - 2007
 */
```

```
public class CartaoCliente {

   // Variáveis de Instância
   private String codigo;
   private String titular;
   private double totCompras;
   private int pontos;
   private double taxaPremio;

   // Construtores
   public CartaoCliente(String cod, String tit, double taxa) {
      codigo = cod; titular = tit; taxaPremio = taxa;
      totCompras = 0; pontos = 0;
   }

   public CartaoCliente(CartaoCliente cart) {
      codigo = cart.getCodigo(); titular = cart.getTitular();
      pontos = cart.getPontos(); taxaPremio = cart.getTaxa();
      totCompras = cart.getTotCompras();
   }

   // Métodos de Instância

   /** Métodos de Consulta */
   public String getCodigo() { return codigo; }
   public String getTitular() { return titular; }
   public int getPontos() { return pontos; }
   public double getTaxa() { return taxaPremio; }
   public double getTotCompras() { return totCompras; }

   // Modificadores

   /** Altera a taxa de prémio */
   public void mudaTaxa(double novaTaxa) { taxaPremio = novaTaxa; }

   /** Altera o titular do cartão */
   public void mudaTitular(String tit) { titular = tit; }

   /** Método auxiliar que desconta pontos a um cartão. Não é PUBLIC !! */
   private void tiraPontos(int pts) { pontos -= pts; }

   // Operações

   /** Desconta pontos no cartão */
   public void descPontos(int pts) { pontos -= pts; }

   /** Efectua uma compra de dado valor e actualiza pontos e total compras */
   public void compra(double valor) {
      int quantos50 = (int) valor/50;
      totCompras += valor;
      if ( (int) totCompras/50 == quantos50)  // ultrapassou mais um múltiplo
         pontos += (int) valor/taxaPremio;
      else
          pontos += (5*quantos50) + (int) valor/taxaPremio;
   }

   /** Descarrega os pontos de um cartão para o cartão receptor */
   public void descarregaPontos(CartaoCliente crt, int pts) {
      pontos += pts; crt.tiraPontos(pts);
   }
```

```
    /** Representação textual */
    public String toString() {
        StringBuilder s = new StringBuilder();
        s.append("----- Cartão de Cliente -----\n");
        s.append("Código: " + codigo + "\n");
        s.append("Titular: " + titular + "\n");
        s.append("Total Compras: " + totCompras + " Euros\n");
        s.append("Pontos: " + pontos + "\n");
        s.append("Taxa Actual: " + taxaPremio + "\n");
        s.append("---------------------------\n");
        return s.toString();
    }

    /** Cópia do cartão */
    public CartaoCliente CartaoCliente(CartaoCliente crt) {
        return new CartaoCliente(this);
    }
}
```

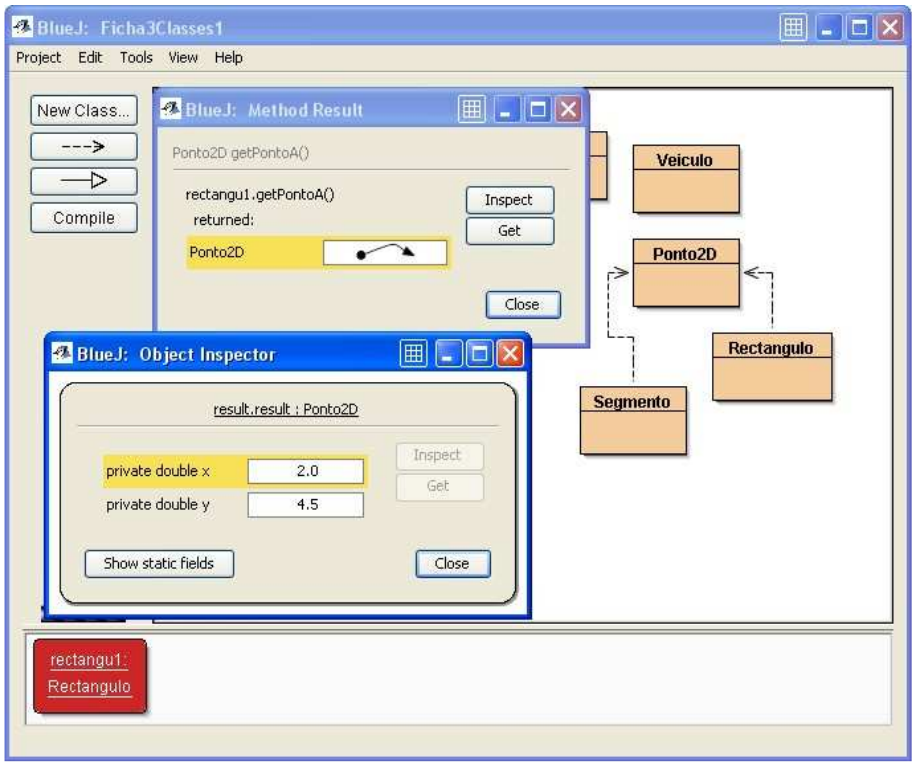


Janela do Projecto da classe CartaoCliente

Ex5:

```
/**
 * Classe que representa Rectângulos, e calcula bases, alturas,perímetros e
 * áreas, permitindo ainda deslocamentos dos pontos limite da diagonal.
 *
 * @author FMM
 * @version 1.0 - 3/3007
 */
import static java.lang.Math.*;
public class Rectangulo {

    // Variáveis de Instância
    private Ponto2D pontoA, pontoB;  // pontos limite da diagonal
```

```
    // Construtores
    public Rectangulo() {
       pontoA = new Ponto2D(); pontoB = new Ponto2D();
    }

    /** Construtor a partir das partes */
    public Rectangulo(Ponto2D p1, Ponto2D p2) {
       pontoA = p1.clone(); pontoB = p2.clone();
    }

    /** Construtor de cópia */
    public Rectangulo(Rectangulo rect) {
       pontoA = rect.getPontoA(); pontoB = rect.getPontoB();
    }

    // Métodos de Instância

    /** Método que devolve o limite 1 da diagonal */
    public Ponto2D getPontoA() { return pontoA.clone(); }

    /** Método que devolve o limite 2 da diagonal */
    public Ponto2D getPontoB() { return pontoB.clone(); }

    /** Método que desloca o limite 1 */
    public void mudaPontoA(double dx, double dy) {
       pontoA.incCoord(dx, dy);
    }
    /** Método que desloca o limite 2 */
    public void mudaPontoB(double dx, double dy) {
       pontoB.incCoord(dx, dy);
    }
    /** Devolve o valor da base do rectângulo */
    public double base() { return abs(pontoA.getX() - pontoB.getX()); }

    /** Devolve o valor da altura do rectângulo */
    public double altura() { return abs(pontoA.getY() - pontoB.getY()); }

    /** Devolve o comprimento da diagonal do rectângulo */
    public double comprimDiagonal() {
       return sqrt(pow(pontoB.getX()- pontoA.getX(), 2) +
                   pow(pontoB.getY()- pontoA.getY(), 2));
    }

    /** Devolve o valor do perímetro do rectângulo */
    public double perimetro() {
       return 2 * (this.base() + this.altura());
    }

    /** Devolve o valor da área do rectângulo */
    public double area() { return this.base() * this.altura(); }

    /** Representação textual do Rectângulo */
    public String toString() {
       StringBuilder s = new StringBuilder();
       s.append("Rectangulo [ " + pontoA.toString() + " - " +
                               pontoB.toString() + " ]\n");
       return s.toString();
    }
}
```

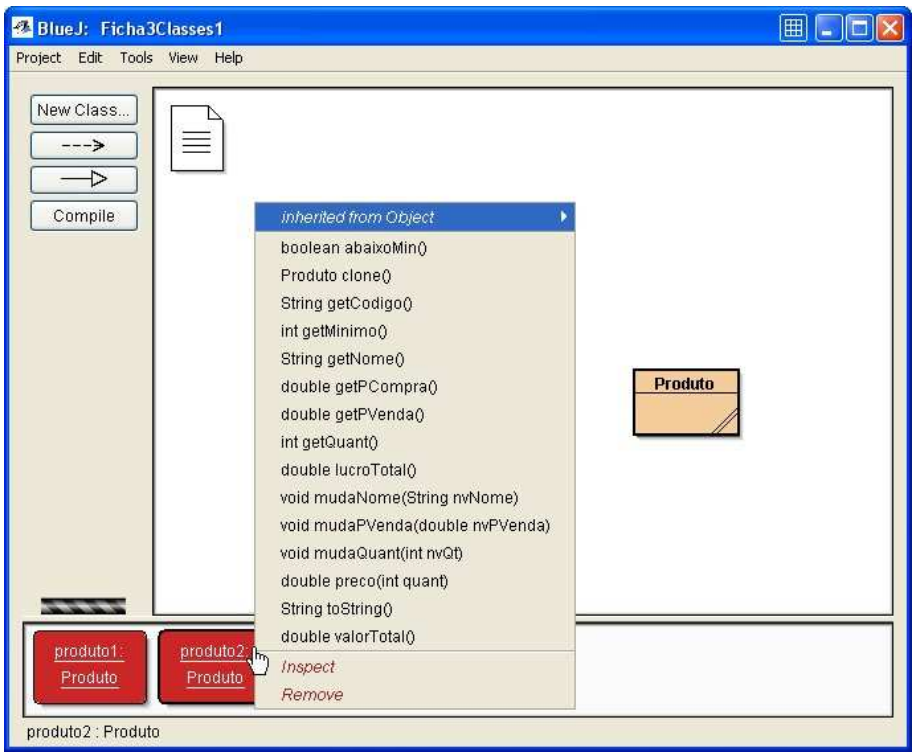


Janela do Projecto da classe Rectangulo

Ex6:

```
/**
 * Class que representa um Produto de um Stock.
 *
 * @author FMM
 * @version 1.0 3-2007
 */
public class Produto {

    // Variáveis de Instância
    String codigo;
    String nome;
    int quantidade;
    double precoCompra;
    double precoVenda;
    int minimo;

    // Construtores
    public Produto(String cod, String nom, int quant, double pCompra,
                double pVenda, int min) {
      codigo = cod; nome = nom; quantidade = quant;
      precoCompra = pCompra; precoVenda = pVenda; minimo = min;
    }


    public Produto(Produto prod) {
       codigo = prod.getCodigo(); nome = prod.getNome();
       quantidade = prod.getQuant(); precoCompra = prod.getPCompra();
       precoVenda = prod.getPVenda(); minimo = prod. getMinimo();
    }
```

```
    // Métodos de Instância
    public String getCodigo() { return codigo; }
    public String getNome() { return nome; }
    public int getQuant() { return quantidade; }
    public double getPCompra() { return precoCompra; }
    public double getPVenda() { return precoVenda; }
    public int getMinimo() { return minimo; }

    // Modificadores

    /** Muda a designacao do Produto */
    public void mudaNome(String nvNome) { nome = nvNome; }

    /** Muda a quantidade do produto */
    public void mudaQuant(int nvQt) { quantidade += nvQt; }

    /** Muda preco de venda */
    public void mudaPVenda(double nvPVenda) {
       precoVenda = nvPVenda;
     }

    /** Valor total do stock de Produto em caso de venda total */
    public double valorTotal() { return quantidade * precoVenda; }

    /** Lucro total em caso de venda total; Se < 0 e prejuizo !! */
    public double lucroTotal() {
      return quantidade*(precoVenda - precoCompra);
    }

    /** Encomenda de N unidades e valor total a pagar */
    public double preco(int quant) { return quant * precoVenda; }

    /** Verifica se esta abaixo do nivel minimo */
    public boolean abaixoMin() { return quantidade <= minimo; }

    /** Representacao como texto */
    public String toString() {
        StringBuilder s = new StringBuilder();
        s.append("-----  Produto  -----\n");
        s.append("Codigo: " + codigo + "\n");
        s.append("Nome: " + nome + "\n");
        s.append("Quant: " + quantidade + "\n");
        s.append("Preco Compra: " + precoCompra + "\n");
        s.append("Preco Venda: " + precoVenda + "\n");
        s.append("Minimo: " + minimo + "\n");
        return s.toString(); }

    /** Copia de Produto */
    public Produto clone() { return new Produto(this); }
}
```

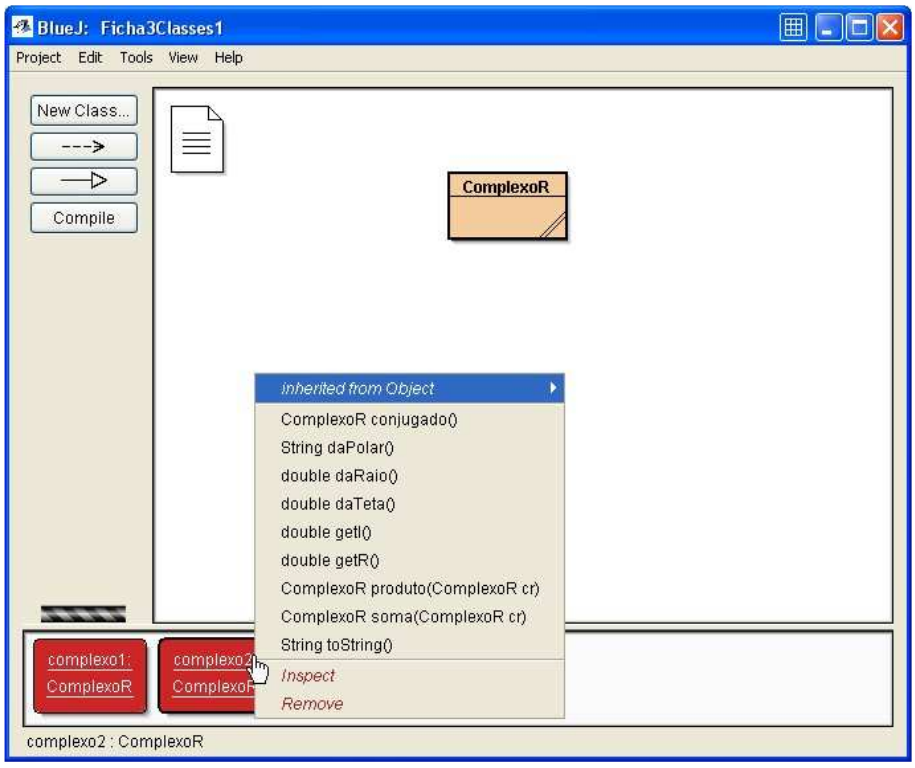


Janela do Projecto da classe Rectangulo

Ex7:

```
/**
 * ComplexoR: complexos na forma normal.
 *
 * @author F. Mário Martins
 * @version (a version number or a date)
 */
   import static java.lang.Math.*;
   public class ComplexoR {

      // Variáveis de Instância
      private double a;   //  z = a + bi
      private double b;

      // Construtores
      public ComplexoR() { a = 0.0; b = 0.0; }
      public ComplexoR(double r, double i) { a = r; b = i; }
      public ComplexoR(ComplexoR cpx) { a = cpx.getR(); b = cpx.getI(); }

      // Consulta
      public double getR() { return a; } // parte real
      public double getI() { return b; }  // p. imaginária

      /** z + w = (a + bi) + (c + di) = (a+c) + (b+d)i  */
      public ComplexoR soma(ComplexoR cr) {
       return new ComplexoR(a + cr.getR(), b + cr.getI());
      }

      /** z*w = (a + bi) * (c + di) = (ac – bd) + (bc + ad) i  */
      public ComplexoR produto(ComplexoR cr) {
        double a1 = a*cr.getR() - b*cr.getI(); double b1 = b*cr.getR() + a*getI();
        return new ComplexoR(a1, b1);
      }
```

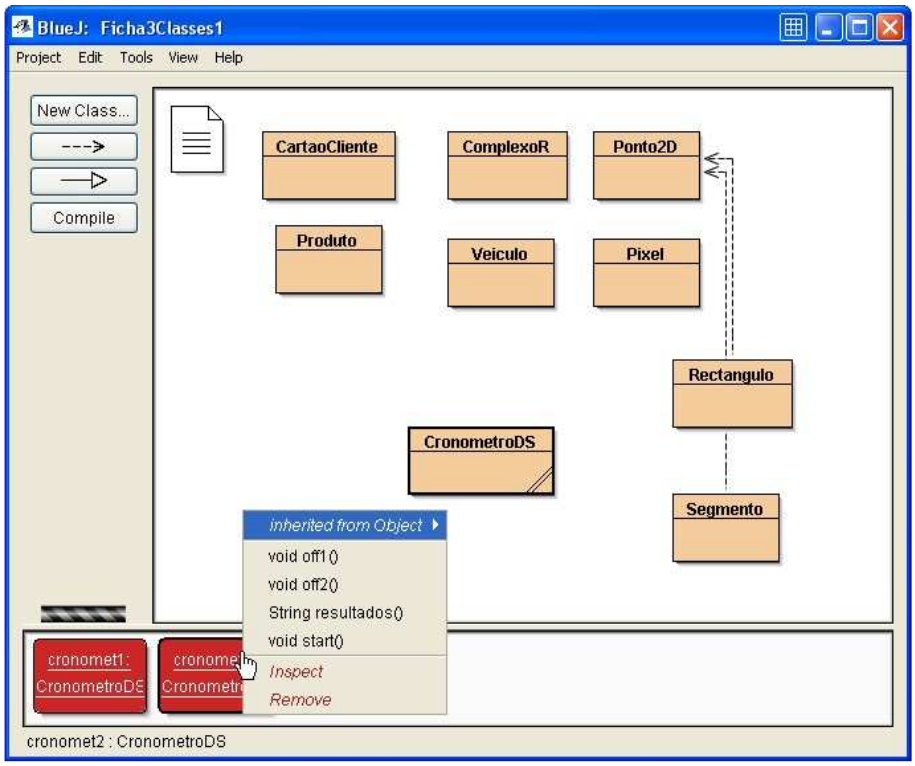
```
    /** zc = a – bi (conjugado de z) */
    public ComplexoR conjugado() {
        return new ComplexoR(a, -b);
    }

    /** Calcula raio polar */
    public double daRaio() {
     return sqrt(pow(a,2) + pow(b,2));
    }

    /** Calcula teta da representacao polar */
    public double daTeta() {
       return atan(b/a);
    }

    /** Conversão para a forma polar */
    public String daPolar() {
      StringBuilder s = new StringBuilder();
      s.append("CplxPolar = ");
      s.append(this.daRaio() + " (cos " + this.daTeta() + " + sen " +
                  this.daTeta() + " i\n");
      return s.toString();
    }

    /** Representação textual */
    public String toString() {
      StringBuilder s = new StringBuilder();
      s.append("Cplx = "); s.append(a);
      s.append( b>=0 ? "+" : ""); s.append(b +" i");
      return s.toString();
    }
}
```



Janela do Projecto da classe Complexo

Ex8:

```
/**
 * Uma pequena classe que implementa o funcionamento de um cronómetro.
 * Inicia uma contagem de tempo com START. Faz uma primeira paragem com
STOP1
 * mas o tempo continua a contar até fazer a segunda paragem com STOP2.
 * Em seguida, devolve todos os tempos registados: do início ate a primeira
 * paragem, do inicio ate a segunda e entre a primeira e a segunda.
 *
 * Os formatos para os tempos medidos será mM:sS:msMls.
 *
 * @author F. Mário Martins
 * @version 1.0/04-2004
 */

import java.util.GregorianCalendar;
public class CronometroDS {

   // Variáveis de Instância para os tempos
   private long timeStart;      // em milissegundos da ERA
   private long timeStop1;      // em milissegundos da ERA
   private long timeStop2;      // em milissegundos da ERA

   /** Construtor de Cronómetros */
   public CronometroDS() { timeStart = 0; timeStop1 = 0; timeStop2 = 0; }

   /** Método START: coloca o cronómetro a funcionar */
   public void start() {
     GregorianCalendar crono = new GregorianCalendar();
     timeStart = crono.getTimeInMillis();
   }

   /** Método STOP1: para a primeira contagem */
   public void off1() {
     GregorianCalendar crono = new GregorianCalendar();
     timeStop1 = crono.getTimeInMillis();
   }
   /** Método STOP2: para a segunda contagem */
   public void off2() {
     GregorianCalendar crono = new GregorianCalendar();
     timeStop2 = crono.getTimeInMillis();
   }

   /** Método auxuliar que calcula intervalo de tempo e coloca em String */
   private String intervaloEmStr(long fim, long inicio) {
       long totalmilis = fim - inicio;
       int totalseg = (int) totalmilis/1000; int milis = (int) totalmilis%1000;
       int seg = totalseg%60; int totalmin = totalseg/60;
       int min = totalmin%60;
       return new String(min + " M: " + seg + " S: " + milis + " ms.\n");
   }

   /**  Resultados da cronometragem */
   public String resultados() {
       StringBuilder s = new StringBuilder();
       s.append("Tempo Total: "); s.append(intervaloEmStr(timeStop2, timeStart));
       s.append("Tempo Intervalo1: "); s.append(intervaloEmStr(timeStop1, timeStart));
       s.append("Tempo Intervalo2: "); s.append(intervaloEmStr(timeStop2, timeStop1));
       System.out.println(s.toString());
       return s.toString();
     }
}
```



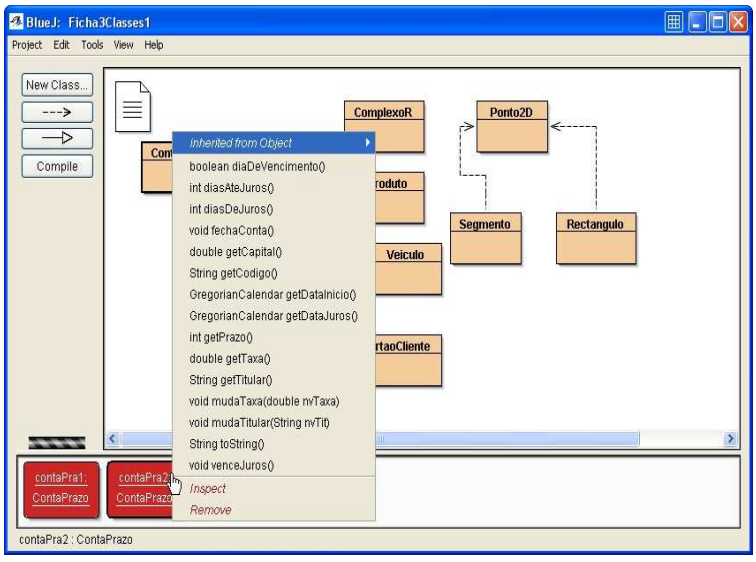
Janela do Projecto da classe CronometroDS

Ex9:

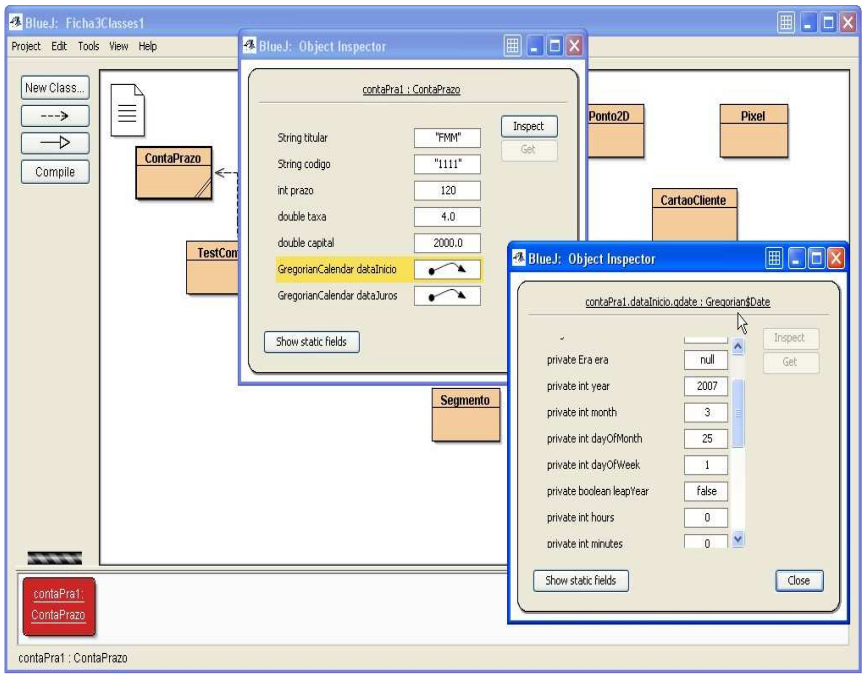
```
/**
 * ContaPrazo: classe que permite criar contas a prazo com uma dada taxa de
 * juros e realizar algumas operações sobre estas.
 *
 * @author FMM
 * @version 1.0 3-2007
 */
import java.util.GregorianCalendar;
public class ContaPrazo {

   // Variáveis de Instância
   String titular; String codigo;
   int prazo; double taxa;  // taxa para o prazo
   double capital;
   GregorianCalendar dataInicio;
   GregorianCalendar dataJuros;  // redundante: dataInicio + prazo = dataJuros

   // Construtores
   public ContaPrazo(String tit, String cod, int dias, double montante, double tax) {
      titular = tit; codigo = cod; prazo = dias;
      capital = montante; taxa = tax; dataInicio = new GregorianCalendar();
      dataJuros = (GregorianCalendar) dataInicio.clone();
      dataJuros.add(GregorianCalendar.DAY_OF_MONTH, prazo);
   }

   public ContaPrazo(ContaPrazo ctp) {
      titular = ctp.getTitular(); codigo = ctp.getCodigo();
      prazo = ctp.getPrazo(); capital = ctp.getCapital();
      taxa = ctp.getTaxa(); dataInicio = ctp.getDataInicio();
      dataJuros = ctp.getDataJuros();
   }
```

```
// Métodos de Instância
/** Devolve o titular da conta a prazo */
public String getTitular() { return titular; }

/** Devolve o codigo da conta a prazo */
public String getCodigo() { return codigo; }

/** Devolve o prazo da conta a prazo */
public int getPrazo() { return prazo; }

/** Devolve o montante actual da conta a prazo */
public double getCapital() { return capital; }

/** Devolve a taxa de juros actual da conta a prazo */
public double getTaxa() { return taxa; }

/** Altera o titular da conta */
public void mudaTitular(String nvTit) { titular = nvTit; }

/** Altera a taxa de juros */
public void mudaTaxa(double nvTaxa) { taxa = nvTaxa; }

/** Devolve a data de abertura da conta. Não se faz clone() pelo facto de que
 * o método clone() de GC não devolve um GC pelo que temos que converter
 */
public GregorianCalendar getDataInicio() {
    return (GregorianCalendar) dataInicio.clone();
}

/** Devolve a data de calculo dos juros. Não se faz clone() pelo facto de que
 * o método clone() de GC não devolve um GC pelo que havera que converter
 */
public GregorianCalendar getDataJuros() {
    return (GregorianCalendar) dataJuros.clone();
}

/** Dias entre duas datas */
private int diasEntreDatas(GregorianCalendar inicio, GregorianCalendar fim) {
    long totalmilis = fim.getTimeInMillis() - inicio.getTimeInMillis();
    long milis = totalmilis%1000; long totalseg = totalmilis/1000;
    int totalmin = (int) totalseg/60; long seg = totalseg%60;
    int min = totalmin%60; int totalhoras = totalmin/60;
    int horas = totalhoras%60; int totaldias = totalhoras/24;
    return totaldias;
}

/** Dias entre hoje e o calculo dos juros */
public int diasAteJuros() {
  return this.diasEntreDatas( new GregorianCalendar(), dataJuros);
}

/** Dias passados desde a abertura da conta */
public int diasDeJuros() {
   return this.diasEntreDatas(dataInicio, new GregorianCalendar());
}
```

```
    /** Calcular juros actuais, somar ao capital e calcular nova data de calculo de
     * juros
     */
    public void venceJuros() {
        int diasDeJuros = this.diasEntreDatas(dataInicio, dataJuros);
        double juros = capital * (taxa/100/prazo) * diasDeJuros;
        capital += juros; dataInicio = new GregorianCalendar();
        dataJuros = (GregorianCalendar) dataInicio.clone();
        dataJuros.add(GregorianCalendar.DAY_OF_MONTH, prazo);
    }

    /** Fecho da conta com calculos */
    public void fechaConta() {
        this.venceJuros(); dataInicio = null; dataJuros = null;
    }

    /** Determinar se hoje é dia de calcular juros */
    public boolean diaDeVencimento() {
        return this.diasEntreDatas(dataInicio, dataJuros) == 0;
    }

    /** Data em String cf. DIA-MES-ANO */
    private String dataToString(GregorianCalendar data) {
        int ano = data.get(GregorianCalendar.YEAR);
        int mes = data.get(GregorianCalendar.MONTH) + 1;
        int dia = data.get(GregorianCalendar.DAY_OF_MONTH);
        return "" + dia + " - " + mes + " - " + ano;
    }

    /** toString() */
    public String toString() {
        StringBuilder s = new StringBuilder();
        s.append("------------ CONTA A PRAZO ------------\n");
        s.append("Codigo: " + codigo + "\t\tTitular: " + titular + "\n");
        s.append("Capital: " + capital + "\t\tPrazo: " + prazo + "\n");
        s.append("Taxa de Juro: " + taxa + "% a " + prazo + " dias\n");
        s.append("Data Inicio: " + dataToString(dataInicio) + "\n");
        s.append("Data Juros: " + dataToString(dataJuros) + "\n");
        return s.toString();
    }
}
```



Janela do Projecto da classe ContaPrazo



Janela do Projecto da classe ContaPrazo - Inspect

FICHA PRÁTICA 4

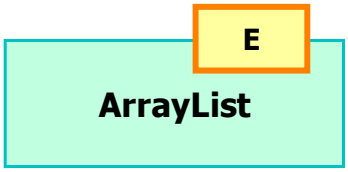
LABORATÓRIO DE COLECÇÕES I

ARRAYLIST<E>

SÍNTESE TEÓRICA

Em JAVA5, tal como em algumas outras linguagens de programação por objectos, certas estruturações de objectos (colecções) são *parametrizadas*, ou seja, aceitam um tipo parâmetro a ser posteriormente substituído por um tipo concreto (p.e., uma classe).

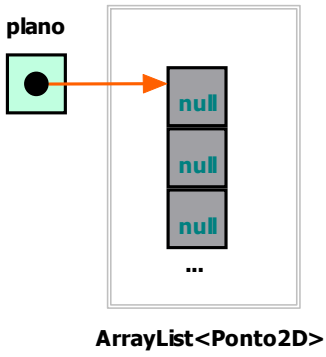
Uma colecção de JAVA5 de extraordinária utilidade na agregação de objectos e sua estruturação sob a forma de uma lista, é **ArrayList<E>**. Tendo por suporte um *array* de características dinâmicas, ou seja, capaz de aumentar ou diminuir de dimensão ao longo da execução de um programa, um ArrayList<E>, implementa listas de objectos, em que E representa a classe/tipo parâmetro de cada um dos objectos nele contidos. Um *arraylist* é pois, ao contrário de um array, um objecto e de dimensão dinâmica, possuindo um grande conjunto de métodos a si associados para realizar as operações mais usuais sobre listas, não podendo porém conter valores de tipos primitivos (apenas objectos).



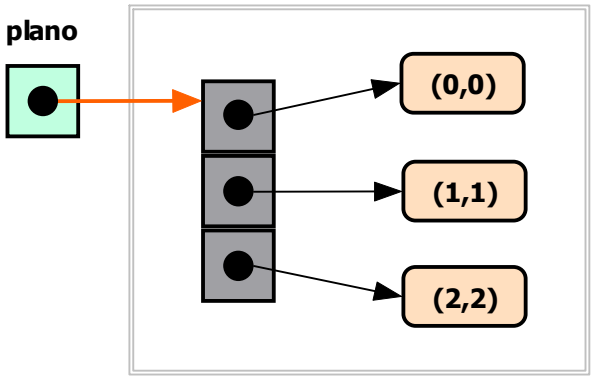
A declaração de um arraylist concreto, consiste em definir-se qual o tipo concreto para o seu tipo parâmetro, numa declaração normal mas paramétrica, onde, em geral, usando um **construtor**, é criado imediatamente um *arraylist* inicialmente vazio (atenção ao **construtor** que também é parametrizado), com dimensão inicial ou não.

```
ArrayList<String> nomes = new ArrayList<String>(100);
ArrayList<Ponto2D> plano = new ArrayList<Ponto2D>();
```

Em ambos os casos o estado inicial do arraylist deverá ser visto como sendo uma lista de apontadores a **null**, já que nenhum elemento foi ainda criado e inserido, cf.



Posteriormente, e à medida que cada um dos Ponto2D forem inseridos usando os diversos métodos que apresentaremos em seguida, cada posição do arraylist (iniciadas no índice 0), referenciará uma instância de Ponto2D ou manter-se-á a **null**, tal como se ilustra na figura seguinte.



ArrayList<Ponto2D>

A API da classe **ArrayList<E>** é apresentada em seguida, encontrando-se as várias operações disponíveis agrupadas por funcionalidades para melhor compreensão. Todos os parâmetros representados como sendo do tipo *Collection* devem ser assumidos como podendo ser uma qualquer colecção de JAVA, ainda que neste momento estejamos limitados a trabalhar apenas com **ArrayList<E>**.

SINTAXE ESSENCIAL

Categoria de Métodos	API de ArrayList<E>

Construtores	`new ArrayList<E>()`
	`new ArrayList<E>(int dim)`
	`new ArrayList<E>(`*`Collection`*`)`

Inserção de elementos	`add(E o); add(int index, E o);` `addAll(`*`Collection`*`); addAll(int i, `*`Collection`*`);`
Remoção de elementos	`remove(Object o); remove(int index);` `removeAll(`*`Collection`*`); retainAll(`*`Collection`*`)`

Consulta e comparação de conteúdos	`E get(int index); int indexOf(Object o);`
	`int lastIndexOf(Object o);`
	`boolean contains(Object o); boolean isEmpty();`
	`boolean containsAll(`*`Collection`*`); int size();`

Criação de Iteradores	`Iterator<E> iterator();`
	`ListIterator<E> listIterator();`
	`ListIterator<E> listIterator(int index);`

Modificação	`set(int index, E elem); clear();`
Subgrupo	`List<E> sublist(int de, int ate);`
Conversão	`Object[] toArray();`
Outros	`boolean equals(Object o); boolean isEmpty();`

Quadro – API de `ArrayList<E>`

NOTA:

Os parâmetros de tipo **Collection** serão explicados mais tarde. De momento apenas será necessário que se compreenda que são colecções de JAVA. De momento apenas podemos usar colecções que sejam `ArrayList<T>` e em que `T` seja um tipo compatível com o tipo `E` (por exemplo, sendo mesmo `T = E`).

1.- EXEMPLOS SIMPLES

Consideremos um exemplo simples cujo objectivo é a apresentação da semântica dos principais métodos da API de **ArrayList<E>**, através da sua utilização.

Vamos criar duas listas de nomes, uma contendo os nomes de amigos e outra contendo os nomes de pessoas que conhecemos e são músicos. Serão, naturalmente dois **ArrayList<String>**. Vamos em seguida realizar a sua declaração completa, limitando o número de músicos a uns 50.

DECLARAÇÕES

```
ArrayList<String> amigos =  new ArrayList<String>();  // vazio
ArrayList<String> musicos = new ArrayList<String>(50);  // capacidade = 50
```

INSERÇÕES NO FIM USANDO add(E elem)

```
amigos.add("Jorge"); amigos.add("Ivo"); amigos.add("Rino");
// ERRO DE COMPILAÇÃO => amigos.add(new Ponto2D(0.0, 2.0));
musicos.add("Rino"); musicos.add("Zeta");
```

NÚMERO ACTUAL DE ELEMENTOS DE UMA LISTA

```
int numAmigos = amigos.size();
```

CONSULTA DO ELEMENTO NO ÍNDICE i <= amigos.size() - 1

```
String amigo = amigos.get(10);
String nome = amigos. get(i);
```

ITERAÇÃO SOBRE TODOS OS ELEMENTOS DE UM ARRAYLIST<STRING>

```
// Imprimir os nomes de todos os amigos – Solução A: Ciclo for
for(int i = 0; i <= amigos.size()-1; i++)
   out.printf("Nome: %s%n", amigos.get(i);

// Imprimir os nomes de todos os amigos – Solução B: Criando um Iterador
// Depois de criado, o Iterator<String> é percorrido até it.hasNext() ser falso,
// sendo cada elemento devolvido usando it.next()
for(Iterador<String> it = amigos.iterator(); it.hasNext();)
   out.printf("Nome: %s%n", it.next());

// Imprimir os nomes de todos os amigos – Solução C: Usando for(each)
for(String nome : amigos)
   out.printf("Nome: %s%n", nome);
```

PROCURA DE UM ELEMENTO => USAR ITERATOR<E> E WHILE

```
String chave = Input.lerString();
encontrado = false; int index = 0;
Iterator<String> it = amigos.iterator();  // Iterator<String> criado
while(it.hasNext() && !encontrado) {      // iteração
   encontrado = it.next().equals(chave); index++;
}
if (encontrado)
  out.printf("O nome %s está na posição %2d%n", chave, index-1);
else
  out.println("O nome " + chave + " não existe !\n");
```

PROCURA DE UM ELEMENTO (VERSÃO SIMPLES)

```
String chave = Input.lerString();
int index = amigos.indexOf(chave);
if (index = -1)
  out.println("Não existe tal nome !");
else
  out.println("Está no índice: " + index);
```

ARRAYLIST ESPARSO (NÃO CONTÍGUO): UTILIZAÇÃO CORRECTA

A) INICIALIZAR A **null** AS POSIÇÕES QUE SE PRETENDEM ACEDER;

B) USAR **set(int index, E elem)** SENDO **index <= size()-1;**

```
ArrayList<String> cantores = new ArrayList<String>(DIM);  // capac = DIM
for(int i = 0; i <= DIM-1; i++) cantores.add(null);
// inserção de nomes em qualquer índice até DIM-1 !!
cantores.set(10, "Rui"); cantores.set(2, "Ana");
```

ELEMENTOS COMUNS A DOIS ARRAYS

```
// Determinar quais os amigos que também são músicos
//
// 1) Cria um arraylist e copia o arraylist amigos na criação
ArrayList<String> temp = new ArrayList<String>(amigos);
// ou, em alternativa equivalente usando temp1
ArrayList<String> temp1 = new ArrayList<String>();
for(String am : amigos) temp1.add(am);
// 2) Remove da lista temp todos os que não são musicos, ou seja,
// retém os que também estão na lista musicos.
temp.retainAll(musicos);
// imprime os nomes
out.println("\n--- AMIGOS MÚSICOS ---\n");
for(String name : temp) out.printf(" %s%n", name);
// o mesmo para a lista de amigas
temp = new ArrayList<String>(amigas); temp.retainAll(musicos);
out.println("\n--- AMIGAS MÚSICAS ---\n");
for(String n : temp) out.printf(" %s%n", n);

// Remover da lista de amigos e da lista de amigas os que são músicos;
//
// As listas amigos e amigas são modificadas caso alguns sejam músicos !!
// Outra hipótese seria usar, tal como acima, um arraylist temporário.
amigas.removeAll(musicos); amigos.removeAll(musicos);
// imprime os nomes de amigos e amigas restantes
out.println("\n--- AMIGOS NÃO MÚSICOS ---\n");
for(String n : amigos) out.printf(" %s%n", n);
out.println("\n--- AMIGAS NÃO MÚSICOS ---\n");
for(String n : amigas) out.printf(" %s%n", n);

// Criar um arraylist que seja a reunião das listas de amigos e amigas;
//
// Esta operação é semelhante a uma reunião matemática, mas não faz
// qualquer filtragem de elementos em duplicado (também não devem
// existir neste caso !!).
ArrayList<String> todos = new ArrayList<String>(amigos);
todos.addAll(amigas);
out.println("--- Todos : ---\n");
for(String nom : todos) out.printf("Nome: %s%n", nom);
```

EXERCÍCIOS:

Ex 1: Desenvolva uma classe Plano que represente um conjunto de pontos 2D de um plano cartesiano num ArrayList<Ponto2D> (a representação correcta seria um conjunto mas como ainda não estudamos como representar conjuntos em JAVA usaremos um arraylist). Desenvolva, para além dos construtores e métodos usuais, métodos que implementem as seguintes funcionalidades:

- Determinar o número total de pontos de um plano;
- Adicionar um *novo* ponto ao plano e remover um ponto caso exista;
- Dado um ArrayList de Pontos2D, juntar tais pontos ao plano receptor;
- Determinar quantos pontos estão mais à direita ou mais acima (ou ambos) relativamente ao Ponto2D dado como parâmetro;
- Deslocar todos os pontos com coordenada em XX igual a cx, dada como parâmetro, de dx unidades, igualmente dadas como parâmetro (alterar os pontos portanto);
- Dado um plano como parâmetro, determinar quantos pontos são comuns aos dois planos;
- Criar a lista contendo os pontos comuns ao plano receptor e ao plano parâmetro;
- Criar uma lista contendo todos os pontos do plano com coordenada em XX inferior a um valor dado como parâmetro (atenção, pretende-se obter cópias dos originais);
- Criar um novo plano que contenha os pontos comuns entre o plano receptor e um plano dado como parâmetro;
- Não esquecer os métodos **equals()**, **toString()** e **clone()**.

Ex 2: Uma ficha de informação de um país, FichaPais, contém 3 atributos: nome do país, continente e população (real, em milhões). Crie uma classe ListaPaises que permita criar listas de FichaPais, por uma ordem qualquer, e implemente os seguintes métodos:

- Determinar o número total de países;
- Determinar o número de países de um continente dado;
- Dado o nome de um país, devolver a sua ficha completa, caso exista;
- Criar uma lista com os nomes dos países com uma população superior a um valor dado;
- Determinar a lista com os nomes dos continentes dos países com população superior a dado valor;
- Determinar o somatório das populações de dado continente;
- Dada uma lista de FichaDePais, para cada país que exista na lista de países alterar a sua população com o valor na ficha; caso não exista inserir a ficha na lista;
- Dada uma lista de nomes de países, remover as suas fichas;

Ex 3: Uma Stack (ou pilha) é uma estrutura linear do tipo LIFO ("last in first out"), ou seja, o último elemento a ser inserido é o primeiro a ser removido. Uma stack possui assim apenas um extremo para inserção e para remoção. Implemente uma Stack de nomes, com as usuais operações sobre stacks:

- String top(): que determina o elemento no topo da stack;
- void push(String s): insere no topo;
- void pop(): remove o elemento do topo da stack, se esta não estiver vazia;
- boolean empty(): determina se a stack está vazia;
- int length(): determina o comprimento da stack;

Ex 4: Cada e-mail recebido numa dada conta de mail é guardado contendo o endereço de quem o enviou, a data de envio, a data de recepção, o assunto e o texto do mail (não se consideram anexos, etc.). Crie a classe Mail que represente cada um dos mails recebidos.
Em seguida crie uma classe designada MailList que permita guardar todos os actuais e-mails existentes numa dada conta, e implemente as seguintes operações sobre os mesmos:

- Determinar o total de mails guardados;
- Guardar um novo mail recebido;
- Determinar quantos mails têm por origem um dado endereço;
- Criar uma lista contendo os índices dos mails que no assunto contêm uma palavra dada como parâmetro (qualquer que seja a posição desta);
- O mesmo que a questão anterior, mas criando uma lista contendo tais mails;
- Eliminar todos os e-mails recebidos antes de uma data que é dada como parâmetro;
- Criar uma lista dos mails do dia;
- Dada uma lista de palavras, eliminar todos os mails que no seu assunto contenham uma qualquer destas (anti-spam);
- Eliminar todos os mails anteriores a uma data dada;

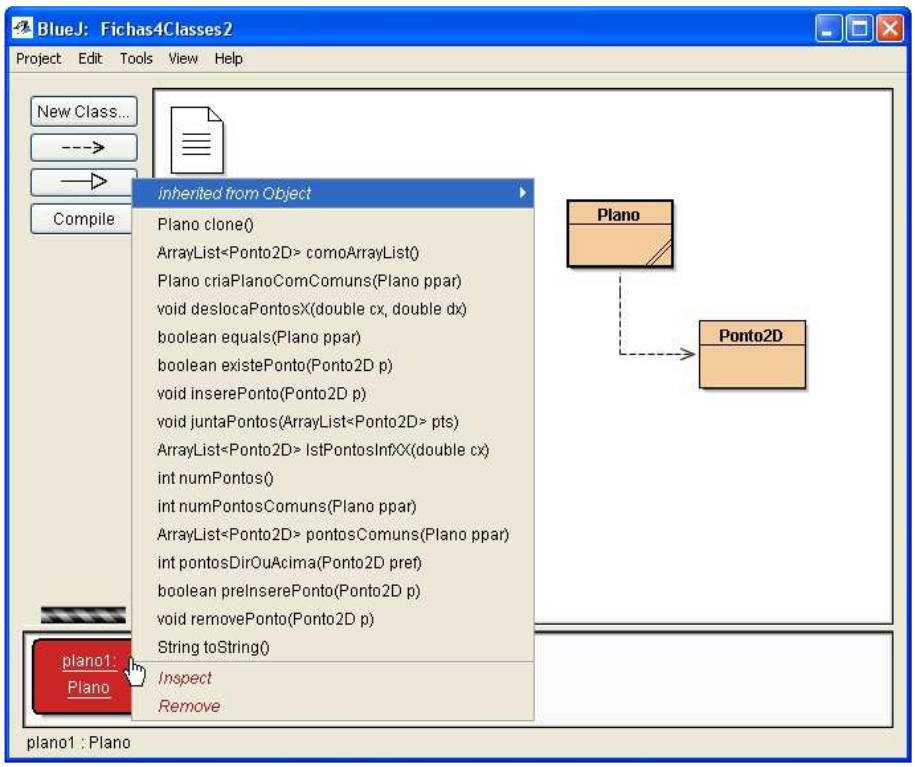
RESOLUÇÕES:

Ex1:

```
/**
 * Classe Plano: representa um plano cartesiano usando um ArrayList<Ponto2D>,
 * ou seja, um arraylist contendo pontos.
 *
 * @author F. Mário Martins
 * @version 1.0 4-2007
 */
import java.util.ArrayList;
public class Plano {

   // NOTA: this é um PLANO; plano é um ArrayList<Ponto2D>;
   // NÃO CONFUNDIR CLASSE com REPRESENTAÇÃO !!

   // Variáveis de Instância
   private ArrayList<Ponto2D> plano;

   // Construtores
   public Plano() { plano = new ArrayList<Ponto2D>(); }

   public Plano(ArrayList<Ponto2D> lpontos) {
      plano = new ArrayList<Ponto2D>();
      for(Ponto2D p : lpontos) plano.add(p.clone());
   }

   public Plano(Plano pcart) {
      plano = new ArrayList<Ponto2D>();
      for(Ponto2D p : pcart.comoArrayList()) plano.add(p.clone());
   }

   // Métodos de Instância

    /** Devolve o número actual de pontos do plano */
   public int numPontos() { return plano.size(); }

    /** Verifica se um ponto já existe no plano */
   public boolean existePonto(Ponto2D p) { return plano.contains(p); }

    /** Acrescenta um novo ponto ao plano */
   public void inserePonto(Ponto2D p) { plano.add(p.clone()); }

    /** Remove um ponto do plano */
   public void removePonto(Ponto2D p) { plano.remove(p); }

    /** Junta ao plano os pontos da lista parâmetro, sem duplicados */
   public void juntaPontos(ArrayList<Ponto2D> pts) {
     for(Ponto2D p : pts)
       if (!this.existePonto(p)) plano.add(p.clone());
   }

    /** Número de pontos à direita e/ou acima do ponto dado */
   public int pontosDirOuAcima(Ponto2D pref) {
      int conta = 0;
      for(Ponto2D p : plano)
        if (p.getX() > pref.getX() || p.getY() > pref.getY()) conta++;
      return conta;
   }
```

```
 /** Lista dos pontos do plano: Converte Plano em ArrayList<Ponto2D> */
public ArrayList<Ponto2D> comoArrayList() {
   ArrayList<Ponto2D> res = new ArrayList<Ponto2D>();
   for(Ponto2D p : plano) res.add(p.clone());
   return res;
}

 /** Desloca todos os pontos de coordenada em X igual a cx de dx unidades */
public void deslocaPontosX(double cx, double dx) {
   for(Ponto2D p : plano)
     if(p.getX() == cx) p.incCoord(dx, 0);
}

 /** Número de pontos comuns aos dois planos */
public int numPontosComuns(Plano ppar) {
   ArrayList<Ponto2D> comuns = this.comoArrayList();
   comuns.retainAll(ppar.comoArrayList());
   return comuns.size();
}

 /** Lista de pontos comuns aos dois planos */
public ArrayList<Ponto2D> pontosComuns(Plano ppar) {
   ArrayList<Ponto2D> comuns = this.comoArrayList();
   comuns.retainAll(ppar.comoArrayList());
   return comuns;
}

 /** Lista de pontos com XX inferior a cx dado como parâmetro */
public ArrayList<Ponto2D> lstPontosInfXX(double cx) {
   ArrayList<Ponto2D> res = new ArrayList<Ponto2D>();
   for(Ponto2D p : plano)
     if(p.getX() < cx) res.add(p.clone());
   return res;
}

 /** Plano formado pelos pontos comuns aos dois planos: solução 1 */
public Plano criaPlanoComComuns(Plano ppar) {
  ArrayList<Ponto2D> pontosPar = ppar.comoArrayList();
  ArrayList<Ponto2D> pontosThis = this.comoArrayList();
  pontosThis.retainAll(pontosPar); // pontosThis são os comuns
  return new Plano(pontosThis);
}

 /** Plano formado pelos pontos comuns aos dois planos: solução 2 */
public Plano criaPlanoComComuns2(Plano ppar) {
  return new Plano(this.pontosComuns(ppar));
}

 /** Igualdade de dois planos: mesmos pontos pela mesma ordem */
public boolean equals(Plano ppar) {
   if (this == ppar) return true;
   if ( (ppar == null) || (this.getClass() != ppar.getClass()) ) return false;
   ArrayList<Ponto2D> lstPtsPar = ppar.comoArrayList();
   ArrayList<Ponto2D> lstPtsThis = this.comoArrayList();
   if (lstPtsThis.size() != lstPtsPar.size()) return false;
   int i = 0; boolean iguais = true;
   while(i<= lstPtsPar.size()-1 && iguais) {
      iguais = lstPtsPar.get(i).equals(lstPtsThis.get(i)); i++;
   }
   return iguais;
}
```

```
  public String toString() {
     StringBuilder s = new StringBuilder("--- Plano ---\n");
     for(Ponto2D p : plano) s.append(p.toString() + "\n");
     return s.toString();
  }

  public Plano clone() { return new Plano(this); }

}
```



Janela do Projecto da classe Plano

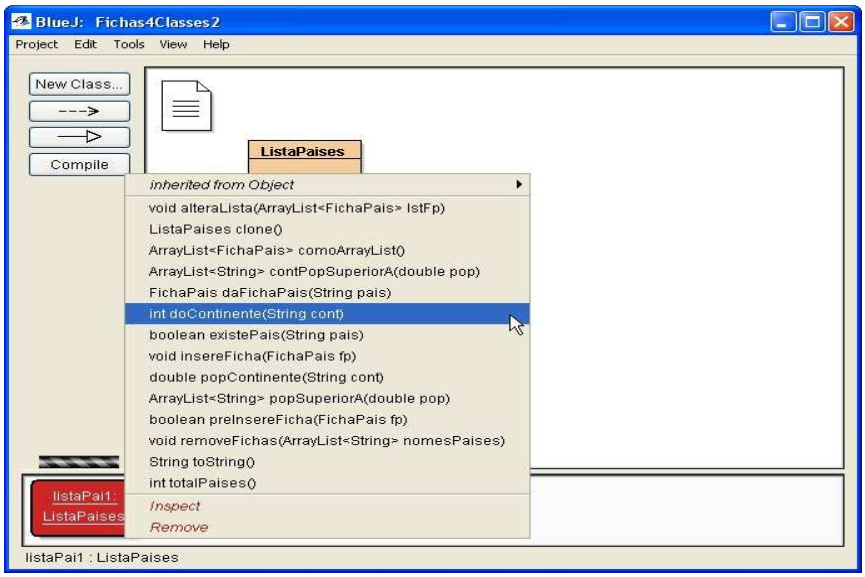
Ex2:

```
/**
 * FichaPais: ficha contendo a informação de um país:
 * Nome, Continente e População
 *
 * @author F. Mário Martins
 * @version 1.0 4-2007
 */
public class FichaPais {

   // Variáveis de Instância
   String pais;
   String continente;
   double populacao;

   // Construtores
   public FichaPais(String nomePais, String cont, double pop) {
      pais = nomePais; continente = cont; populacao = pop;
   }
```

```
    public FichaPais(FichaPais fichap) {
        pais = fichap.getPais(); continente = fichap.getCont();
        populacao = fichap.getPop();
    }

    // Métodos de Instância
    public String getPais() { return pais; }
    public String getCont() { return continente; }
    public double getPop() { return populacao; }

    public void mudaPop(double pop) { populacao = pop; }

    public String toString() {
        StringBuilder s = new StringBuilder("--- Ficha Pais ---\n");
        s.append("Pais: " + pais + "\n");
        s.append("Continente: " + continente + "\n");
        s.append("Populacao: " + populacao + "\n");
        s.append("------------------------------");
        return s.toString();
    }

    public FichaPais clone() { return new FichaPais(this); }

}


/**
 * ListaPaises: uma lista contendo as fichas informativas de países.
 * Cada ficha de país contém Nome, Copntinente e População de cada país.
 *
 * @author F. Mário Martins
 * @version 1.0 4-2007
 */
import java.util.ArrayList;
public class ListaPaises {

    // Variáveis de Instância
    private ArrayList<FichaPais> lstPaises;

    // Construtores
    public ListaPaises(ArrayList<FichaPais> lstfp) {
        lstPaises = new ArrayList<FichaPais>();
        for(FichaPais fp : lstfp) lstPaises.add(fp.clone());
    }
    public ListaPaises(ListaPaises lpaises) {
        lstPaises = new ArrayList<FichaPais>();
        for(FichaPais fp : lpaises.comoArrayList()) lstPaises.add(fp.clone());
    }

    // Métodos de Instância

    /** Número total de países */
    public int totalPaises() { return lstPaises.size(); }

    /** Devolve um arraylist com todas as fichas copiadas */
    public ArrayList<FichaPais> comoArrayList() {
        ArrayList<FichaPais> res = new ArrayList<FichaPais>();
        for(FichaPais fp : lstPaises) res.add(fp.clone());
        return res;
    }
```

```
/** Verifica se uma ficha já existe */
public boolean preInsereFicha(FichaPais fp) {
   return lstPaises.contains(fp);
}

/** Insere uma nova ficha de país na lista */
public void insereFicha(FichaPais fp) {
   lstPaises.add(fp);
}

/** Verifica se um país de nome dado já tem ficha */
public boolean existePais(String pais) {
   int comp = lstPaises.size();
   boolean encontrado = false;
   int conta = 0;
   while(!encontrado && conta <= comp-1) {
      encontrado = lstPaises.get(conta).getPais().equals(pais);
      conta++;
   }
   return encontrado;
}

/** Devolve a ficha de um país de nome dado ou null caso este não exista */
public FichaPais daFichaPais(String pais) {
   int comp = lstPaises.size();
   boolean encontrado = false;
   int conta = 0;
   while(!encontrado && conta <= comp-1) {
      encontrado = lstPaises.get(conta).getPais().equals(pais);
      conta++;
   }
   return encontrado ? lstPaises.get(conta-1) : null;
}

/** Número de países de um dado continente */
public int doContinente(String cont) {
   int conta = 0;
   for(FichaPais fp : lstPaises)
     if(fp.getCont().equals(cont)) conta++;
   return conta;
}


/** Lista de nomes de países com população superior à dada */
public ArrayList<String> popSuperiorA(double pop) {
   ArrayList<String> res = new ArrayList<String>();
   for(FichaPais fp : lstPaises)
     if(fp.getPop() > pop) res.add(fp.getPais());
   return res;
}

/** Lista de nomes de continentes com países com população > a dada */
public ArrayList<String> contPopSuperiorA(double pop) {
   ArrayList<String> res = new ArrayList<String>();
   for(FichaPais fp : lstPaises)
     if(fp.getPop() > pop)
        if(!res.contains(fp.getCont())) res.add(fp.getCont());
   return res;
}
```

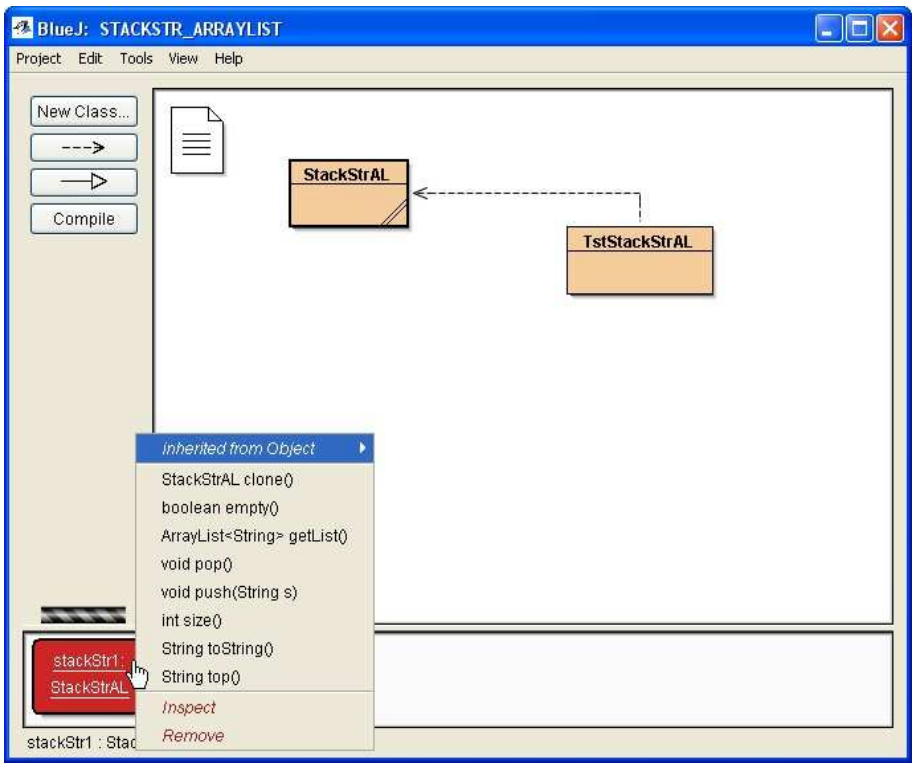
```
    /** Total da população de um dado continente */
    public double popContinente(String cont) {
        int total = 0;
        for(FichaPais fp : lstPaises)
            if(fp.getCont().equals(cont)) total += fp.getPop();
        return total;
    }

    /** Para cada país da lista parâmetro existente na Lista de Países,
        actualiza a sua população */
    public void alteraLista(ArrayList<FichaPais> lstFp) {
        FichaPais fp = null;
        for(FichaPais fichap : lstFp) {
            fp = this.daFichaPais(fichap.getPais());
            if (!(fp == null))
                    fp.mudaPop(fichap.getPop());
            else
                    this.insereFicha(fichap);
        }
    }

    /** Remover as fichas dos países com nomes na lista parâmetro */
    public void removeFichas(ArrayList<String> nomesPaises) {
        FichaPais fp = null;
        for(String np : nomesPaises) {
            fp = this.daFichaPais(np);
            if (!(fp == null)) lstPaises.remove(fp);
        }
    }

    public String toString() {
        StringBuilder s = new StringBuilder("------ PAISES ------\n");
        for(FichaPais fp : lstPaises) s.append(fp.toString() + "\n");
        s.append("---------------------------\n");
        return s.toString();
    }

    public ListaPaises clone() { return new ListaPaises(this); }
}
```



Janela do Projecto da classe ListaPaises

Ex3:

```
/**
 * Stack de STRING implementada usando um ArrayList<String>
* @author FMM
 * @version 1/03-2006
 */
import java.util.ArrayList;
public class StackStrAL {

   // Representação da stack
   private ArrayList<String> stack = new ArrayList<String>();

   // Construtores
   public StackStrAL() { }

   public StackStrAL(String s) { stack.add(s); }

   public StackStrAL(ArrayList<String> listStr) {
      stack.clear(); stack.addAll(listStr); // são strings. OK !!
   }

   public StackStrAL(StackStrAL stk) {
      stack.clear(); for(String s : stk.getList()) stack.add(s);
   }

   // Métodos de instancia usuais para uma STACK

   /** Inserção de um elemento */
   public void push(String s) { stack.add(s); }

   /** Determina se stack está vazia  */
   public boolean empty() { return stack.size() == 0; }

   /** Determina comprimento da stack */
   public int size() { return stack.size(); }

   /** Determina qual o topo da stack (se não estiver vazia !!) */
   public String top() { return (stack.get(this.size()-1)); }

   /** Remove o topo da stack (se não estiver vazia !!) */
   public void pop() { stack.remove(this.size()-1); }

   /** Devolve um ArrayList<String> com os elementos da stack */
   public ArrayList<String> getList() {
      ArrayList<String> elems = new ArrayList<String>();
      for(String s : stack) elems.add(s); // strings não necessitam de clone()
      return elems;
   }

   /** toString() */
   public String toString() {
      StringBuilder s = new StringBuilder();
      s.append("--- STACK ---\n");
      for(int i = stack.size()-1; i >= 0; i--)
         s.append(stack.get(i) + "\n");
      return s.toString();
   }

   /** clone */
   public StackStrAL clone() { return new StackStrAL(this); }

}
```



Janela do Projecto da classe StackStrAL

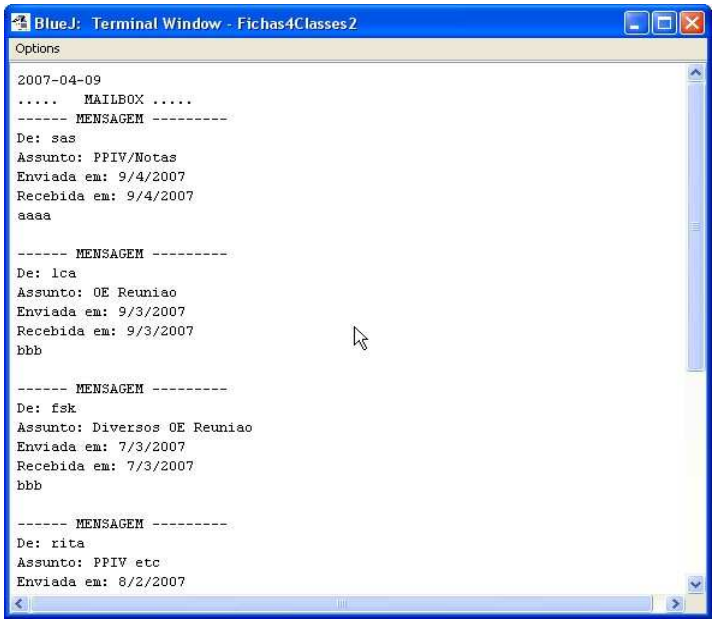
Ex4:

```
/**
 * Mail: classe que descreve a estrutura de um e-mail simples e as
 * operações simples que com o mesmo podem ser feitas
* @author F. Mário Martins
 * @version 1.0 4-2007
 */
import java.util.GregorianCalendar;
import java.util.Calendar;
public class Mail {

   // Construtores
   public Mail(Mail m) {
      endereco = m.getEndereco(); assunto = m.getAssunto();
      data_envio = (GregorianCalendar) m.getEnvio().clone();
      data_recepcao = (GregorianCalendar) m.getRecepcao().clone();
      texto = m.getTexto();
   }

   public Mail(String end, String asst, GregorianCalendar envio,
               GregorianCalendar recep, String txt) {
      endereco = end; assunto = asst; data_envio = envio;
      data_recepcao = recep; texto = txt;
   }

   // Variáveis de Instância

   private String endereco;
   private String assunto;
   private GregorianCalendar data_envio;
   private GregorianCalendar data_recepcao;
   private String texto;
```

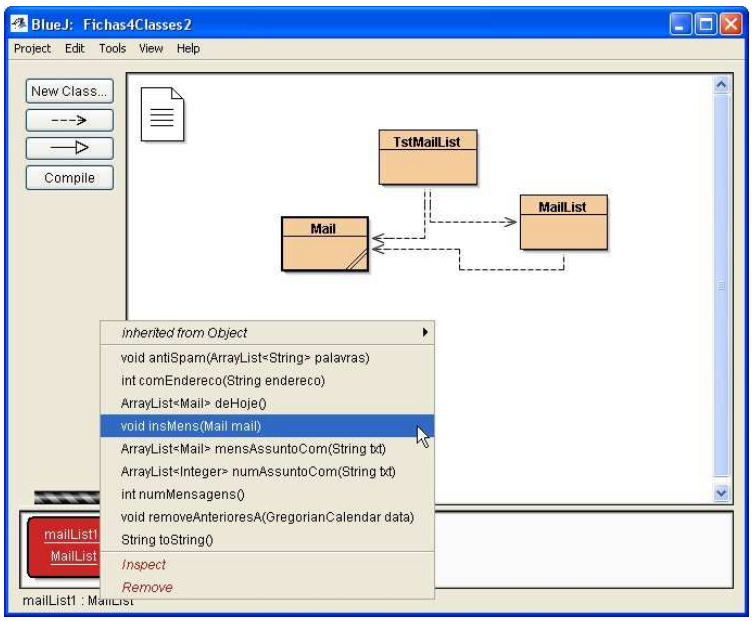
```
    // Métodos de Instância

    public String getEndereco() { return endereco; }
    public String getAssunto() { return assunto; }
    public GregorianCalendar getEnvio() {
       return (GregorianCalendar) data_envio.clone();
    }
    public GregorianCalendar getRecepcao() {
       return (GregorianCalendar) data_recepcao.clone();
    }

    public String getTexto() { return texto; }

    public String toString() {
       StringBuilder s = new StringBuilder("------ MENSAGEM ---------\n");
       s.append("De: " + endereco + "\n");
       s.append("Assunto: " + assunto + "\n");
       s.append("Enviada em: " + data_envio.get(Calendar.DAY_OF_MONTH) + "/" +
                           (data_envio.get(Calendar.MONTH) + 1) + "/" +
                           data_envio.get(Calendar.YEAR) + "\n");
       s.append("Recebida em: " + data_recepcao.get(Calendar.DAY_OF_MONTH) + "/" +
                           (data_recepcao.get(Calendar.MONTH) + 1) + "/" +
                           data_recepcao.get(Calendar.YEAR) + "\n");
       s.append(texto + "\n");
       return s.toString();
    }

    public boolean equals(Object o) {
      if (this == o) return true;
      if ((o == null) || (this.getClass() != o.getClass())) return false;
      //
     Mail umMail = (Mail) o;
     return (  this.endereco.equals(umMail.getEndereco())
             && this.assunto.equals(umMail.getAssunto())
            && this.data_envio.equals(umMail.getEnvio())
            && this.data_recepcao.equals(umMail.getRecepcao())
            && this.texto.equals(umMail.getTexto()));
    }

   public Mail clone() { return new Mail(this); }

}

/**
 * MailList: classe que guarda instâncias de Mail e implementa um conjunto
 * de operaçoes para consulta e gestão das mensagens guardadas.
 *
 * @author F. Mário Martins
 * @version 1.0 4-2007
 */

import java.util.ArrayList;
import java.util.GregorianCalendar;
import java.util.Calendar;
public class MailList {

    // Variáveis de Instância
    ArrayList<Mail> mailBox = new ArrayList<Mail>();
```

```
// Métodos de Instância

/** Número total de mensagens */
public int numMensagens() { return mailBox.size(); }

/** Insere uma nova mensagem de mail */
public void insMens(Mail mail) { mailBox.add(mail.clone()); }

/** Total de mensagens recebidas de um dado endereço */
public int comEndereco(String endereco) {
   int conta = 0;
   for(Mail m : mailBox)
     if(m.getEndereco().equals(endereco)) conta++;
   return conta;
}

/** Números das mensagens que contêm uma dada string no seu assunto */
public ArrayList<Integer> numAssuntoCom(String txt) {
   ArrayList<Integer> nums = new ArrayList<Integer>(); int index = 0;
   for(Mail m : mailBox) {
     if (m.getAssunto().contains(txt)) nums.add(index+1); // !!??
     index++;
   }
   return nums;
}

/** Lista das mensagens que contêm uma dada string no seu assunto */
public ArrayList<Mail> mensAssuntoCom(String txt) {
   ArrayList<Mail> mensgs = new ArrayList<Mail>();
   for(Mail m : mailBox)
     if (m.getAssunto().contains(txt)) mensgs.add(m.clone());
   return mensgs;
}

/** Remove todas as mensagens anteriores a uma dada data */
public void removeAnterioresA(GregorianCalendar data) {
   int[] indices = new int[mailBox.size()];   // array das posições a remover
   int conta = 0; int index = 0;
   for(Mail m : mailBox) {
     if (m.getRecepcao().before(data)) {
        indices[conta] = index; conta++;
     }
     index++;
   }
   // remove mails nas posições dadas por indices[i]
   for(int i = 0; i <= conta-1; i++)
        mailBox.remove(indices[i]);
}

/** Listas das mensagens recebidas hoje */
public ArrayList<Mail> deHoje() {
   ArrayList<Mail> mailsDeHoje = new ArrayList<Mail>();
   GregorianCalendar hoje = new GregorianCalendar();
   GregorianCalendar dataRecep = null;
   int diaHoje = hoje.get(Calendar.DAY_OF_MONTH);
   int mesHoje = hoje.get(Calendar.MONTH); int anoHoje = hoje.get(Calendar.YEAR);
   for(Mail m : mailBox) {
     dataRecep = m.getRecepcao();
     int diaRecep = dataRecep.get(Calendar.DAY_OF_MONTH);
     int mesRecep = dataRecep.get(Calendar.MONTH);
     int anoRecep = dataRecep.get(Calendar.YEAR);
     if (diaHoje == diaRecep && mesHoje == mesRecep && anoHoje == anoRecep)
```

```
            mailsDeHoje.add(m.clone());
        }
        return mailsDeHoje;
    }

    /** Remove todas as mensagens cujo assunto contem qq palavra da lista */
    public void antiSpam(ArrayList<String> palavras) {
        for(Mail m : mailBox) {
            int index = 0; boolean contem = false;
            while(index <= palavras.size()-1 && !contem) {
                contem = m.getAssunto().contains(palavras.get(index));
                if (contem) mailBox.remove(m);
                else index++;
            }
        }
    }

    /** toString() */
    public String toString() {
        StringBuilder s = new StringBuilder(".....   MAILBOX .....\n");
        for(Mail m : mailBox) { s.append(m.toString() + "\n"); }
        return s.toString();
    }

}
```



Resultado de toString() de MailList



Janela do Projecto da classe MailList

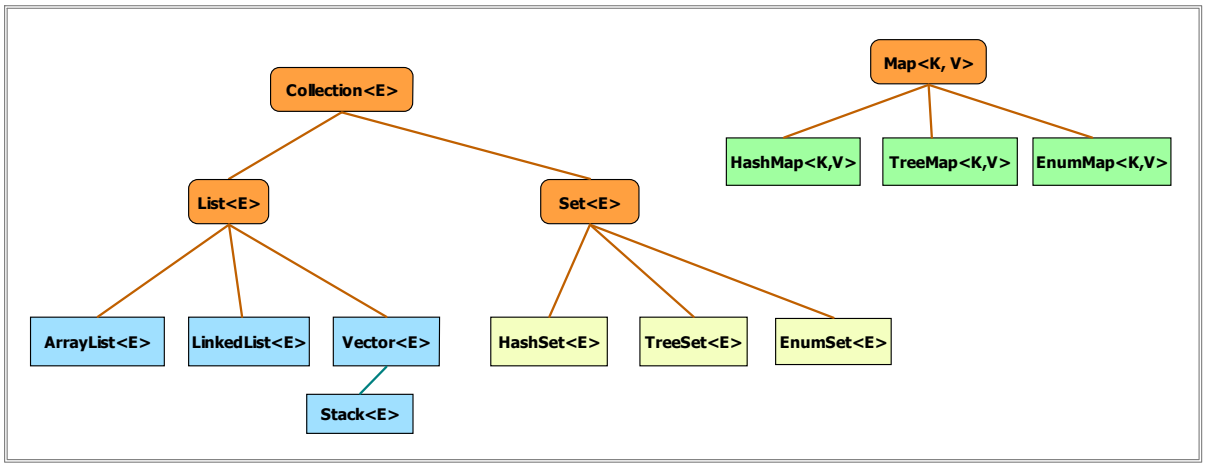
FICHA PRÁTICA 5

LABORATÓRIO DE COLECÇÕES

LIST<E>, SET<E>, MAP<K,V>

SÍNTESE TEÓRICA

O JFC ("JAVA5 Collections Framework") é uma sofisticada arquitectura de classes e tipos (interfaces) de JAVA, que, em síntese, oferece ao utilizador três **tipos** de organizações parametrizadas de objectos: *listas* (do tipo List<E>), *conjuntos* (do tipo Set<E>) e *correspondências* (do tipo Map<K, V>), e, para cada tipo, diversas classes concretas de implementação, tal como se apresenta na figura seguinte.



Tipos de Colecções e Classes de Implementação

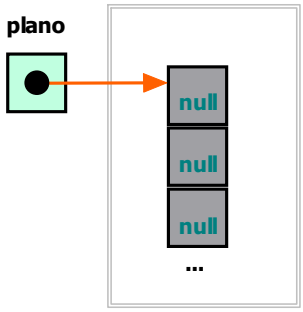
A colecção **ArrayList<E>** foi já apresentada e usada anteriormente, podendo passar a ser agora livremente utilizada nos exemplos seguintes com colecções.

As colecções do tipo **Set<E>** são implementações de *conjuntos de objectos*, pelo que os seus elementos não são indexados ou acessíveis por índice, nem podem ocorrer em duplicado (cf. noção matemática de conjunto). Das três implementações propostas no JCF vamos de momento centrar a nossa atenção em **HashSet<E>** e em **TreeSet<E>**. A colecção HashSet<E> implementa conjuntos usando uma *tabela de hashing* e *algoritmos de hashing*, enquanto que TreeSet<E> é uma implementação de conjuntos baseada numa *árvore binária ordenada*.

A declaração de um *hashset* ou de um *treeset* concreto, consiste em definir-se qual o tipo concreto para o seu tipo parâmetro, numa declaração normal mas paramétrica, onde, em geral, usando um **construtor**, é criada imediatamente uma *colecção* inicialmente vazia (atenção ao **construtor** que também é parametrizado), com dimensão inicial ou não.

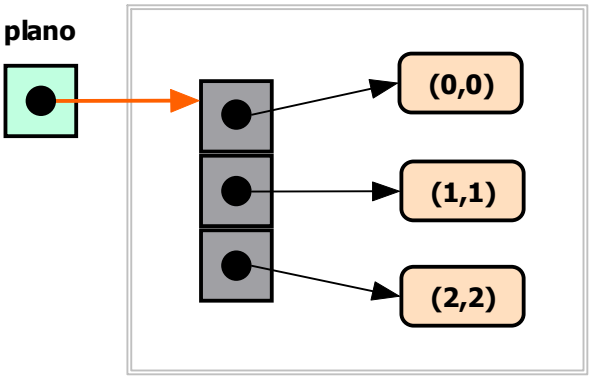
```
HashSet<String> codigos = new HashSet<String>();
HashSet<Ponto2D> caminho = new HashSet<Ponto2D>();
TreeSet<String> nomes = new TreeSet<String>();
TreeSet<Ponto2D> plano = new TreeSet<Ponto2D>(50);
```

Qualquer colecção é, como vimos já, uma agregação de apontadores para objectos do tipo do parâmetro da colecção, que inicialmente tomam o valor **null**, já que nenhum elemento foi ainda inserido, cf.



HashSet<Ponto2D>

Posteriormente, e à medida que cada um dos Ponto2D for inserido, usando os diversos métodos que apresentaremos em seguida, cada posição referenciará uma instância de Ponto2D. Porém, nos conjuntos não há indexação, ou seja, não há acesso aos elementos por índice. Tal significa que a API de Set<E> é um subconjunto da API de List<E> sem as operações por índice.



HashSet<Ponto2D>

A API do tipo **Set<E>**, comum a ambas as implementações, é apresentada em seguida.
Todos os parâmetros representados como sendo do tipo *Collection* devem ser assumidos como podendo ser uma qualquer colecção de JAVA do tipo **List<E>** ou **Set<E>**.

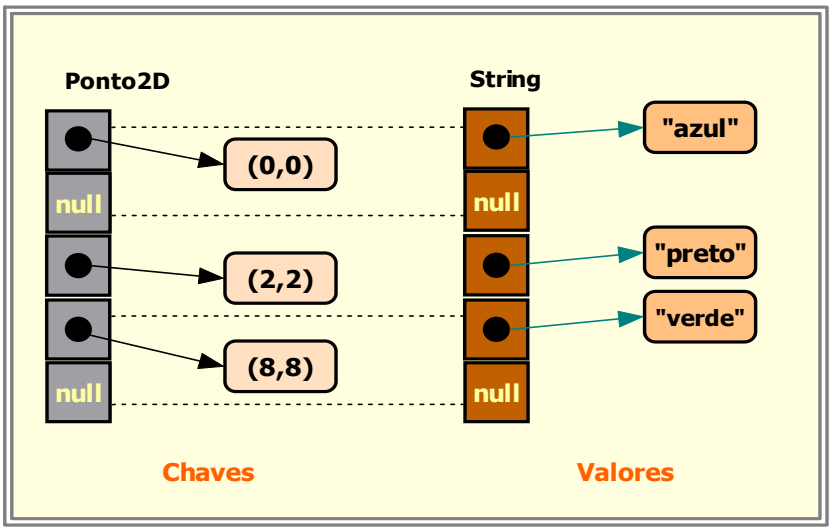
Categoria de Métodos	API de Set<E>
Inserção de elementos	`add(E o);` `addAll(Collection); addAll(int i, Collection);`
Remoção de elementos	`remove(Object o); remove(int index);` `removeAll(Collection); retainAll(Collection)`
Consulta e comparação de conteúdos	`boolean contains(Object o); boolean isEmpty();` `boolean containsAll(Collection); int size();`
Criação de Iteradores	`Iterator<E> iterator();`
Modificação	`clear();`
Subgrupo	`List<E> sublist(int de, int ate);`
Conversão	`Object[] toArray();`
Outros	`boolean equals(Object o); boolean isEmpty();`

Quadro – API de Set<E>

As estruturações de tipo **Map<K,V>** são correspondências finitas, um para um, entre os objectos de um tipo/classe **K** (chaves) e objectos do tipo/classe **V** (valores). Em informática é muito comum que a um dado código único se faça corresponder uma ficha de informação, por exemplo, a ficha de um aluno dado o seu código, a ficha de um produto dado o seu código, etc. Estas estruturações de informação correspondem a **Maps** de códigos para as respectivas fichas.

As duas principais classes de implementação do tipo **Map<K,V>** são **HashMap<K,V>** e **TreeMap<K,V>**, devendo as **TreeMaps** ser usadas sempre que pretendermos que as suas chaves se mantenham ordenadas (seja por um algoritmo de ordenação automático pré-definido de JAVA para esse tipo de chaves (exº String, Integer, etc.), seja através de um algoritmo de ordenação fornecido pelo programador ao respectivo construtor de TreeMaps (a ver mais tarde).

```
HashMap<Ponto2D, String> pixels =  new HashMap<Ponto2D, String>();
```



Exemplo de representação de um HashMap<Ponto2D, String>

A API do tipo **Map<K, V>**, comum a ambas as implementações, é apresentada em seguida.

Categoria de Métodos	API de Map<K,V>
Inserção de elementos	`put(K k, V v);` `putAll(Map<? extends K, ? extends V> m);`
Remoção de elementos	`remove(Object k);`

Consulta e comparação de conteúdos	`V get(Object k);`
	`boolean containsKey(Object k);`
	`boolean isEmpty();`
	`boolean containsValue(Object v); int size();`

Criação de Iteradores	`Set<K> keySet();`
	`Collection<V> values();`
	`Set<Map.Entry<K, V>> entrySet();`

Outros	`boolean equals(Object o); Object clone()`

Quadro – API de Map<K,V>

TreeMap<K, V> métodos adicionais
`TreeMap<K, V>()`
`TreeMap<K, V>(Comparator<? super K> c)`
`TreeMap<K, V>(Map<? extends K, ? extends V> m)`
`K firstKey()`
`SortedMap<K, V> headMap(K toKey)`
`K lastKey()`
`SortedMap<K, V> subMap(K fromKey, K toKey)`
`SortedMap<K, V> tailMap(K fromKey)`

Quadro – API parcial de TreeMap<K,V>

Algumas Regras Básicas de Utilização de Colecções Em Projectos:

- Escolher com critério se a colecção a criar deve ser uma lista ou um conjunto (duplicados ou não) ou então uma correspondência entre chaves e valores;
- Escolher para *sets* e *maps* uma classe de implementação adequada, cf**. Hash** (sem ordem especial) ou **Tree** (com ordem);
- Nunca usar os métodos pré-definidos **addAll()** ou **putAll()**. EM vez destes, usar antes o iterador **for** e fazer **clone()** dos objectos a copiar para a colecção cópia (ver exemplos);
- Sempre que possível, os resultados dos métodos devem ser generalizados para os tipos List<E>, Set<E> ou Map<K,V> em vez de devolverem classes específicas como ArrayList<E>, HashSet<E> ou TreeSet<E> ou HashMap<K,V>. Aumenta-se assim a abstracção.

Exercícios:

Ex 1: Desenvolva uma classe Produto que represente a informação básica de um produto existente no armazém de uma dada empresa. Sobre cada produto pretende ter-se a seguinte informação:

```
private String codigo;   // código
private String nome;     // descrição
private int quant;       // quantidade em stock
private int alarme;      // mínimo de alerta – valor mínimo aceitável
```

Crie em seguida uma classe Empresa contendo o nome da empresa em questão e uma representação do stock da empresa, associando a cada código de produto a sua ficha de informação.
Para além dos construtores e métodos usuais, a classe Empresa deverá definir ainda os seguintes métodos de instância:

- **Método que devolve todos os actuais códigos de produtos em stock;**
- **Método que insere um novo produto no stock da empresa;**
- **Método que remove do stock o produto de código dado;**
- **Método que altera a quantidade de um produto em stock de um valor dado;**
- **Método que devolve a quantidade total de produtos em stock;**
- **Método que verifica se o produto de código dado existe;**
- **Método que cria uma lista com os códigos dos produtos com quantidade <= alarme;**
- **Método que devolve uma cópia do stock;**
- **Método que devolve a informação de um produto de código dado;**

Ex 2: Crie uma classe Paises que estabeleça uma correspondência entre o nome de um dado país e a informação sobre a sua capital (FichaDeCapital), designadamente: nome da cidade, população, número de veículos, salário médio (real) e custo de vida mensal médio (real).
Implemente os seguintes métodos na classe Paises:

- Determinar o número total de países;
- Devolver os nomes dos países com capitais com população acima de um valor dado;
- Dado o nome de um país, devolver ficha completa da sua capital;
- Alterar a população da capital de um dado país;
- Inserir a informação de um novo país;
- Criar uma listagem com os nomes de todas as capitais registadas;

- Determinar o somatório de todas as populações das capitais;
- Dada um Map de nome de país para FichaDeCapital, para cada país que exista na lista de países alterar a sua ficha de capital e para cada país novo inserir a sua informação.
- Dada um conjunto de nomes de países, remover as suas fichas de capital;

Ex 3: Considerando a classe ContaPrazo anteriormente desenvolvida, crie agora uma classe Banco que associe a cada código de conta uma ContaPrazo. A classe Banco deverá implementar métodos que realizem as seguintes operações:

- Inserir uma nova conta;
- Determinar o conjunto de códigos das contas pertencentes a dado titular;
- O mesmo que o anterior mas para um conjunto de nomes de titulares;
- Determinar os códigos das contas com capital superior a um valor dado;
- Criar um Map das contas com taxa de juro superior a um valor dado;
- Conjunto dos códigos das contas que vencem juros no dia de hoje;
- Dada uma lista de códigos de contas incrementar as suas taxas de juro de um valor X;
- Devolver os nomes de todos os titulares de contas;
- Criar um Map que associe a cada nome de titular existente o valor total do capital investido nas suas várias contas (use métodos auxiliares).

Ex 4: Cada e-mail recebido numa dada conta de mail é guardado contendo o endereço de quem o enviou, a data de envio, a data de recepção, o assunto e o texto do mail (não se consideram anexos, etc.). Crie uma classe MailMap que associe a cada endereço de envio todos os mails recebidos (cf. classe Mail) e implemente as seguintes operações:

- Determinar o total de endereços a partir dos quais se recebeu mail;
- Guardar um novo mail recebido;
- Determinar quantos mails têm por origem um dado endereço;
- Criar uma lista contendo todos os endereços que enviaram mails contendo no seu assunto uma lista de palavras dada como parâmetro;
- O mesmo que a questão anterior, mas criando um conjunto contendo os mails;
- Eliminar todos os e-mails recebidos antes de uma data que é dada como parâmetro;
- Criar uma lista dos endereços que hoje enviaram mails;
- Dada uma lista de palavras, eliminar todos os mails de um dado endereço que no seu assunto contenham uma qualquer destas (anti-spam);
- Eliminar todos os mails anteriores a uma data dada;
- Criar uma listagem com todos os endereços de mail oriundos de Portugal;

FICHA PRÁTICA 6

LABORATÓRIO DE HERANÇA E POLIMORFISMO