A Codificación

Os algoritmos deben ser codificados utilizando unha linguaxe do programación. *Dentro do mesmo paradigma* as distintas linguaxes de programación son moi similares. Unha vez deseñado o algoritmo a codificación e unha tarefa bastante directa, xa que basicamente so e necesario o coñecemento da sintaxe concreta a utilizar.

Neste módulo utilizaremos a linguaxe Java para a codificación dos nosos algoritmos. Java é unha linguaxe:

- *Compilada a Bytecode:* O compilador traduce o código fonte a un código numérico intermedio (***bytecode***) que é executado por un intérprete especial: A máquina virtual Java (JVM)
- *Orientado a Obxectos*: Java é unha linguaxe puramente orientada a obxectos, sen embargo e posible utilizalo para implantar algoritmos non orientados a obxectos (aínda que non é a mellor aproximación).
- *Tipado Forte*: Java é unha linguaxe de tipado forte. E necesario definir as variables previamente ao seu uso e indicar exactamente que tipo de información se vai a almacenar nelas.
- *Multiplataforma*: Un programa Java funcionará sen cambios en calquera sistema que dispoña dunha JVM compatible.

Principios da Orientación a Obxectos

Aínda que nun inicio non imos a utilizar orientación a obxectos é importante ter en conta uns conceptos básicos cando se traballa cunha linguaxe puramente orientada a obxectos como Java.

A orientación a obxectos e un sistema de análise dos problemas que se diferencia moito do análise clásico. As linguaxes orientadas a obxectos, simplemente proporcionan as ferramentas para implantar os problemas analizados deste xeito.

No ***análise clásico*** analizamos a solución ós problemas buscando unha serie de pasos (simples ou complexos) para acadar a solución. Si utilizamos deseño descendente (Top-Down) en primeiro lugar consideramos pasos complexos. Posteriormente cada paso complexo o analizamos de novo por separado (dando lugar moi posiblemente a unha función) e así sucesivamente.

No ***análise orientado a obxectos*** consideramos que todos os problemas están formados por unha serie de obxectos que teñen unhas características ou atributos concretos (propiedades) e unhas capacidades de acción, e dicir, un conxunto de accións que é capaz de levar a cabo (métodos). Polo tanto, si identificamos cada un dos obxectos cos seus atributos e métodos e identificamos a interacción entre eles, teremos solucionado o problema.

*Unha **clase** e unha definición* das características e funcionalidades que terá calquera obxecto de esa clase.

Un programa orientado a obxectos consiste na definición dun conxunto de clases, a creación dos obxectos necesarios e a descrición da interacción entre eles (programa).

A orientación a obxectos permite solucións robustas e reutilizables xa que:

- Os atributos e métodos pertencen a cada obxecto, polo tanto a súa modificación non afecta a nada externo a eles. As accións levadas a cabo fora do obxecto tampouco afectan a súa funcionalidade (encapsulación)
- As clases describen un conxunto de atributos e funcionalidades que poden ser comúns a outros problemas, podendo ser reutilizados de xeito moi simple si se deseñan correctamente.
- Se pode dividir a codificación da solución entre varios programadores, xa que podemos encargar a cada programador a elaboración dun conxunto de clases cunhas funcionalidades e atributos concretas. Os programadores non necesitan coñecer o problema completo, so implantar a clase.
- Os erros son mais fáciles de corrixir, xa que se coñece a clase que implanta a funcionalidade que falla e basta corrixila sen necesidade de tocar o resto do programa.

As linguaxes orientadas a obxectos proporcionan un conxunto de clases de uso común que podemos empregar para definir obxectos necesarios para o noso programa denominadas ***libraría de clases do sistema.***

As clases son unicamente definicións, non fan nada “por si mesmas” si non que é necesario crear un obxecto da clase (*instanciación*) e invocar a funcionalidade desexada.

Unha clase é simplemente un tipo de datos especialmente complexo a partir do que podemos crear obxectos (instancias) que teñen as propiedades e métodos definidos.

Tanto os atributos como as funcionalidades definidas na clase pertencen a cada un dos obxectos que creemos. O valor dos atributos dun obxecto nun instante determinado se coñece como ***estado do obxecto***. Cada obxecto mantén o seu propio estado.

Sin embargo, é posible definir tanto atributos como métodos “especiais” de xeito que pertenzan á clase, en lugar de a cada un dos obxectos. Deste xeito poderíamos utilizar funcionalidade sen necesidade de crear ningún obxecto, ou manter un estado común a todos os obxectos da clase. Son os ***métodos e atributos estáticos*** (static). Obviamente, dende un método estático (da clase) non podo acceder ós métodos e atributos non estáticos (do obxecto) a non ser que teñamos un obxecto.

Se debe ser coidadoso sobre a decisión de cando facer un atributo ou método estático. Soamente deberíamos facer un método estático si a súa funcionalidade non depende de ningún atributo, e deberíamos evitar a creación de atributos estáticos si non están moi xustificados xa que se rompe a encapsulación.

Si optamos por facer sempre métodos estáticos, todos os métodos pertencerán a clase e se poderán usar sen necesidade de crear obxectos. Nese caso, aínda que a linguaxe sexa orientada a obxectos non estaremos usando programación orientada a obxectos. Utilizaremos isto para a introdución ós algoritmos, pero non é un bo método de análise e pode crear malos hábitos de programación.

Java é unha linguaxe orientada a obxecto pura. Todo o código que escribamos debe estar definido obrigatoriamente dentro dunha clase.

Ola Mundo en Java

O “ola mundo” é normalmente o primeiro programa que se realiza cando se comeza cunha nova linguaxe de programación:

```
package ud2_algoritmos;

public class OlaMundo {
	public static void main(String[] args) {
		System.out.println(“Ola Mundo!”);
	}
}
```

Examinemos este código con detalle, xa que se ven moitas características importantes.

- ***package ud2_algoritmos***
  Definición do espazo de nomes ud2_algoritmos.
  Unha aplicación está composta de moitas clases (que teñen un nome que as identifica). Moitas de esas clases son clases elaboradas por outras organizacións / empresas ou da propia libraría de clases do sistema, polo que é bastante posible que cando deseñemos as nosas clases utilicemos un nome que xa utiliza outra clase do programa.

  Para evitar isto as linguaxes de programación utilizan espazos de nomes, que en Java se denominan ***packages***. Un *package* define tanto o lugar onde debe residir a clase a partir da carpeta raíz (calquera das carpetas que estean na variable CLASSPATH) como o nome completo a utilizar polas clases. Por exemplo, o nome completo da clase HolaMundo do exemplo sería ***ud2_algoritmos.HolaMundo***

  Co obxecto de conseguir espazos de nomes únicos no mundo, habitualmente as organizacións utilizan o nome do dominio invertido. Por exemplo, *com.iesrodeira.utilidades*

  Existe un package por defecto (default) ao que pertencen as clases básicas da libraría Java. Para utilizar clases de outros packages, si son diferentes ao package onde se atopa o inicio do programa, é preciso ou ben antepoñer o nome do package a clase (com.iesrodeira.utilidades.Mayusculas) ou importar a clase mediante a sentencia ***import***

  Si lanzamos a aplicación do exemplo HolaMundo.class dende a carpeta “Exemplos”, o ficheiro debe estar dentro dunha carpeta ud2_algoritmos, e invocala mediante *java ud2_algoritmos.HolaMundo*

  ```
  Exemplos/
  └── ud2_algoritmos/
      └── OlaMundo.java
  ```

- **public class OlaMundo {**
  Definición da clase OlaMundo
  - A clase é public, polo que se pode utilizar dende dentro de calquera espazo de nomes (package). Si non se especifica ***public*** a clase so poderá ser usada dende as clases definidas no mesmo paquete.
  - As chaves indican o comezo dun bloque de código (o bloque de código da clase).

- **public static void main(String[] args) {**
  Definición dun método que ten como nome “*main*”.
  - **public: É público** se pode utilizar dende fora dos obxectos da clase
  - **static: É estático** polo que pertence a clase en lugar de pertencer aos obxectos creados e se pode usar sen necesidade de crear ningún obxecto da clase OlaMundo
  - **Recibe como argumento String[] args**. É un “array”, e dicir, un conxunto de valores do mesmo tipo, neste caso de tipo String (cadea de texto).

Si quixeramos chamar a este método poderíamos facelo dende outro punto do programa do seguinte xeito:

```
String[] datos=new String[]{“un”,”dous”};
OlaMundo.main(datos)
```

Con new String[] creamos un obxecto da clase Array que e capaz de almacenar obxectos de clase String e que se inicializa cos elementos “un” e “dous”.

Sin embargo os métodos definidos como **public static void main(String[] args)** son “especiais”. Cando chamamos á JVM con *java ud2_algoritmos.HolaMundo a JVM* busca este método e o executa. ***E o punto de inicio do programa.***

A JVM lle pasa como argumentos os que especifiquemos na chamada, por exemplo:

java ud2_algoritmos.HolaMundo un dous

pasará no parametro args o array {“un”, “dous”}.

A JVM inicia a aplicación executando o método main da clase que se utilice como principal.

- **System.out.println(“Ola Mundo!”);**
  Aquí estamos utilizando a clase da libraría estándar de Java “System”. Esta clase ten un atributo estático (pertencente a clase en lugar de os obxectos creados) chamado **out**. Este obxecto perntence a clase da libraría estándar de Java “PrintStream”, que entre outros métodos (accións que é capaz de levar a cabo) está ***println*****,** que visualiza un texto en pantalla.

Os elementos dun programa Java

package

Definición do espazo de nomes onde se agrupan as clases relacionadas. Permite evitar conflitos de nomes entre clases e organizar o código de forma lóxica.

definición de clases

As clases son modelos ou plantillas para crear obxectos. En Java defínense coas palabras clave `class` e opcionalmente `public` si queremos que se podan utilizar dende outros packages.

public class OlaMundo { ... }

definición de atributos e variables: Os tipos de datos

Os atributos son características dun obxecto e as variables locais son contedores de datos temporais.
En Java hai tipos primitivos (int, double, char, boolean) e tipos complexos (Obxectos).
Os tipos de datos indican o tipo de información pode almacenarse nunha variable. En Java podemos distinguir entre tipos primitivos (que existen por pura eficiencia) e obxectos.

- **As variables de tipos primitivos** almacenan información do tipo indicado codificada en binario.
- **As variables de tipo Obxecto** almacenan a posición na memoria onde se garda o estado do obxecto

Tipos Primitivos

Tipos de datos de números enteiros:
byte: 8 bits, de -128 a 127.
short: 16 bits, de -32.768 a 32.767.
int: 32 bits, de -2.147.483.648 a 2.147.483.647.
long: 64 bits, de -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807.

Tipos de datos de números en punto flotante:
float: 32 bits, para números con decimais de precisión simple.
double: 64 bits, para números con decimais de precisión dobre. É o tipo por defecto para os decimais.

Tipo de datos para caracteres:
char: 16 bits, para almacenar un único carácter Unicode.

Tipo de datos booleano:
boolean: Representa un valor de true (verdadeiro) ou false (falso).

Obxectos

Os obxectos son elementos pertencentes a unha clase, e están compostos dunha serie de características (atributos) e unha serie de funcionalidades que son capaces de levar a cabo (métodos).
En Java, todos os tipos primitivos teñen unha clase da que se poden crear obxectos (Integer, Float, Double, Character.... etc), pero en xeral e preferible o uso dos tipos primitivos por motivo de eficiencia.

String texto = "Ola"; *// texto almacena a dirección onde se atopa o obxecto da clase String que almancena "Ola"*
int num =10; // Valor binario
Integer numero = 10; *// Wrapper dun int. numero Almacena a dirección onde se atopa este obxecto da clase Integer con valor 10*

A clase String é unha clase da libraría estándar de Java que almacena unha cadea de caracteres ou texto (ten un conxunto de char como atributo, entre outros) e dispón numerosas funcionalidades que podemos levar a cabo sobre a mesma.

Unha característica importante da clase String ou das clases correspondentes cos tipos primitivos (*Integer, Float...*) e que **son inmutables**. Iso quere dicir que unha vez creados non se pode variar o valor dos seus atributos (o seu valor). ***Calquera transformación sobre eles da lugar a un novo obxecto*** en lugar de modificar simplemente o valor almacenado nos seus atributos.

Mentres que nas variables de tipos primitivos almacenan un valor codificado en binario, as variables de obxectos almacenan a referencia (numero do byte de memoria) onde se almacena o estado do obxecto (o valor dos seus atributos).

Isto último ten importantes consecuencias no momento de comparar variables que almacenan obxectos, xa que cos operadores de comparación se compara simplemente o contido binario almacenado na variable. Tamén ten implicacións no operador de asignación, xa que con ese operador simplemente almacenamos un valor na variable, no caso dos obxectos, a referencia ao obxecto, non os atributos do obxecto.

String x="Hola"; String y=x; *// En x non teño unha copia de "Hola", é o mesmo texto "Hola". As variables x e y almacenan a mesma referencia*

Java é unha linguaxe puramente orientada a obxectos. Todo o código necesariamente ten que pertencer a unha clase. Non se pode escribir código fora das clases. A execución da aplicación comezará polo método **public static void main(String[] args)** da clase invocada pola JVM java (denominada clase principal).

É unha práctica moi común escribir en cada clase un método **public static void main(String[] args)** aínda que non sexa a clase principal da aplicación co obxecto de escribir código que simplemente comprobe o correcto comportamento da clase. Este tipo de probas de clases de xeito individual se denominan ***test unitarios***. Sen embargo, hoxe en día existen ferramentas como JUnit que permiten facer estas tarefas de comprobación de xeito máis eficiente.

Definición de métodos

Un método describe unha acción que pode executar unha clase ou un obxecto. Si o método pertence á clase (static) se poderá utilizar sen necesidade de crear ningún obxecto, noutro caso, o método pertence a un obxecto concreto que debemos crear con anterioridade

```
public class Calculadora {
    // Método non estático
    public int sumar(int a, int b) {
        return a + b;
    }

    // Método estático
    public static int restar(int a, int b) {
        return a - b;
    }
}
```

Neste caso, poderíamos facer int r=Calculadora.restar(9,3); e se almacenaría o valor 6 en r. Sin embargo, a sentencia int s=Calculadora.sumar(9,2); non sería correcta, xa que o método sumar non pertence a clase, se non a os obxectos que creemos. Deberíamos facer:

```
Calculadora c=new Calculadora()        // Creamos un obxecto Calculadora
int s=c.sumar(9,2);                    // Almacenaría 11 en s
```

Neste exemplo podemos observar tamén os **parámetros** que son a definición das variables para almacenar os datos que necesita o método para poder facer o seu traballo.

As variables que reciben os parámetros e todas as variables declaradas no corpo do método son variables locais ao método, e non teñen existencia fora do mesmo.

Tamén podemos ver na definición dos métodos o ***tipo de retorno*** que indica qué tipo de resultado vai a retornar o método mediante a sentencia ***return***. Si o método non vai retornar nada se pode indicar como tipo **void**

As Sentencias

Expresións aritméticas e lóxicas: Operadores

Para crear expresións aritméticas (dan como resultado un valor numérico) utilizaremos os operadores aritméticos.

Par crear expresións lóxicas (dan como resultado verdadeiro ou falso), utilizaremos os operadores lóxicos e relacionais.

```
int a = 10;
double b = a/3.15+20;
boolean result=a < b && b < 30;                // O resultado será verdadeiro (true) ou falso (false)
```

Operadores aritméticos básicos

Operador	Nome	Exemplo	Resultado
`+`	Suma	`5 + 3`	`8`
`-`	Resta	`10 - 4`	`6`
`*`	Multiplicación	`7 * 2`	`14`
`/`	División enteira ou decimal	`10 / 3` `10.0 / 3`	`3` `3.3333`
`%`	Módulo (resto)	`10 % 3`	`1`

Se ambos operandos son enteiros (int), o resultado dunha división tamén será enteiro.
Para obter decimais, polo menos un dos operandos debe ser de tipo double ou float.

Operadores unarios

Operador	Nome	Exemplo	Resultado
`+`	Identidade	`+x`	Valor de `x`
`-`	Negación	`-x`	Oposto de `x`
`++`	Incremento	`x++` ou `++x`	Aumenta `x` en 1
`--`	Decremento	`x--` ou `--x`	Resta 1 a `x`

Diferenza entre prefixo e sufixo:

- ++x → incrementa antes de usar o valor.
- x++ → usa o valor e logo incrementa.

Precedencia dos operadores aritméticos

En Java, igual que nas matemáticas, algúns operadores teñen maior prioridade ca outros:

1. () → parénteses
2. ++ -- → incremento/decremento
3. * / % → multiplicación, división, módulo
4. + - → suma e resta

Operadores lóxicos básicos

Operador	Nome	Exemplo	Resultado
`&&`	AND lóxico (conxunción)	`true && false`	`false`
`||`	OR lóxico (disxunción)	`true || false`	`true`
`!`	Negación lóxica	`!true`	`false`

Operadores relacionais

Operador	Nome	Exemplo	Resultado
==	Igualdade	5 == 5	true
!=	Diferente	5 != 3	true
>	Maior ca	7 > 4	true
<	Menor ca	2 < 8	true
>=	Maior ou igual ca	6 >= 6	true
<=	Menor ou igual ca	3 <= 5	true

Precedencia dos operadores lóxicos

En Java existe unha orde de avaliación dos operadores lóxicos:

1. `()` → `paréntese`s
2. `!` (negación)
3. `&&` (AND)
4. `\|\|` (OR)

Secuencia

Consiste na execución de instrucións en orde. En Java podemos chamar a métodos, instanciar clases (crear obxectos), facer asignacións, ou avaliar expresións aritméticas e lóxicas ademais do uso das sentencias de control.

```
int a = 5;
int b = 3;
int suma = a + b;
System.out.println(suma);
```

Selección

Elección segundo condicións (resultado da avaliación dunha expresión lóxica) :

```
if (a > b) {
    System.out.println("a é maior");
} else {
    System.out.println("b é maior ou igual");
}
```

Iteración

Repetición de bloques de código segundo o resultado da avaliación dunha expresión lóxica:

```
// Bucle for
for (int i=0; i<5; i++) {
   System.out.println("i = " + i);
}

// Bucle while
int x = 0;
while (x < 5) {
   System.out.println("x = " + x);
   x++;
}

// Bucle do while. Neste caso se executa unha única vez
int x = 5;
do {
   System.out.println("x = " + x);
   x++;
} while (x < 5)
```

Entrada de Datos Básica

Para a entrada de datos básica, a libraría estándar de clases Java dispón dunha clase denominada ***System*** que ten un atributo estático da clase ***InputStream*** denominado **in** que está asociado co teclado, de modo que as pulsacións de teclas xeran un fluxo de información que se pode obter de ***System.in.***

Esta información é puramente numérica, correspondendo cos códigos numéricos asociados as distintas teclas. Para poder obter información intelixible nos programas, a libraría estándar de Java nos proporciona a clase ***Scanner*** no package ***java.util***. Podemos facer uso dun obxecto desta clase creado a partir do obxecto estático System.in para obter información do teclado no formato desexado.

```
import java.util.Scanner; // Importamos a clase Scanner

public class ExemploScanner {
   public static void main(String[] args) {
      // Creamos un obxecto Scanner para ler desde a entrada estándar (teclado)
      Scanner sc = new Scanner(System.in);

      // Pedimos ao usuario que introduza o seu nome
      System.out.print("Introduce o teu nome: ");
      String nome = sc.nextLine(); // Lemos unha liña completa de texto

      // Pedimos ao usuario que introduza a súa idade
      System.out.print("Introduce a túa idade: ");
      int idade = sc.nextInt(); // Lemos un número enteiro

      // Amosamos os datos introducidos
      System.out.println("Ola " + nome + "! Tes " + idade + " anos.");
   }
}
```

Neste exemplo podemos observar que o operador + actúa como operador de concatenación cando un dos operandos é un String

Saida de Datos Básica

De xeito análogo a ***System.in,*** a clase System ten un atributo estático da clase ***PrintStream*** que está asociado coa pantalla do terminal. Calquera información que enviemos mediante este obxecto será visualizada na pantalla.
A clase PrintStream define moitos métodos, sendo o máis común ***print*** e ***println*** como vimos en exemplos anteriores.

Consultar o **API público da libraría estándar de Java** é clave para programar de forma eficiente e evitar erros. Por exemplo:

- **`PrintStream`** (`System.out`) permite mostrar información na pantalla, imprimir valores e formatear saída.
- **`Scanner`** facilita ler datos do teclado e converter automaticamente a distintos tipos (`int`, `double`, `String`).