Tema 1: Conceptos básicos de Programación

Introducción a la Programación
Grado en Ingeniería Informática, EPI Gijón

{jdiez,oluaces,juanjo}@uniovi.es

Departamento de Informática - Universidad de Oviedo en Gijón

Objetivos

- Comprender los objetivos y principios de la Programación.
- Entender las características principales de los paradigmas y los lenguajes de programación.
- Conocer las fases en la construcción de un programa.
- Diferenciar los conceptos de algoritmo y programa.
- Comprender las propiedades fundamentales que un programa debe cumplir.
- Conocer el objetivo de las pruebas de software, qué son los requisitos funcionales y cómo se especifica formalmente un programa.
- Ser capaz de escribir programas en Java que impriman en pantalla.

Contenidos

Programación

¿Qué es la Programación?

Programación

Disciplina que se ocupa de estudiar y definir las técnicas y metodologías más apropiadas para la elaboración de programas informáticos.

Estudiaremos muchos aspectos de los programas:

1. las características deseables que deben tener,
2. la forma de abordar su realización,
3. las fases en las que se suele dividir su elaboración,
4. los lenguajes en los que se pueden escribir, y los tipos de lenguajes que existen,
5. los diferentes paradigmas de programación que se pueden utilizar, y
6. estudiaremos las bases de uno de ellos, la Programación Orientada a Objetos, en un lenguaje concreto, Java.

Programas

¿Tienes claro qué es un programa?

Programa

Texto con instrucciones que deben poder ser entendidas y ejecutadas por un ordenador.

Analizando esa definición, podemos resaltar varios aspectos:

- Es un **texto** legible por un programador que conozca el lenguaje en el que está escrito (lenguaje de programación). A este texto, también se le denomina **código fuente**.
- Consta de un conjunto de **instrucciones** precisas que sirven para realizar la tarea que hace el programa.
- Puede ser **ejecutado** por un ordenador (gracias a otros programas que lo traducen o interpretan llamados compiladores e intérpretes).

Paradigmas de Programación

Diferentes formas de hacer un Programa

Paradigma de Programación

Representa un enfoque particular o filosofía para la construcción de un programa.

Es importante resaltar que:

- no existe un paradigma mejor que los demás,
- cada uno tiene sus características,
- todos tienen ventajas y desventajas, y
- sin embargo, hay situaciones donde un paradigma resulta más apropiado.

Paradigmas de Programación

Clasificación

1. **Imperativo**: concibe la programación como una secuencia instrucciones que se ejecutan paso a paso (algoritmo) y van cambiando el estado de los datos del programa hasta alcanzar la solución (ejemplo, lenguaje C).
   - **Procedimental**, los programas se resuelven descomponiéndolos en procedimientos o módulos (C, Pascal).
2. **Declarativo**: se basa en describir cómo es la solución, no en cómo se hace paso a paso.
   - **Funcional**: entiende la programación como la evaluación de funciones matemáticas (Lisp).
   - **Lógico**, se basa en definir reglas lógicas que, combinadas con un motor de inferencia, resuelven los problemas (Prolog).
3. **Orientado a Objetos**: un programa se compone de objetos que cooperan y se comunican entre sí (Smalltalk, Java, C++).

Declarativo vs. Imperativo

Dos enfoques opuestos

Programa Declarativo - Lenguaje Prolog

```
1 superior(X,Y)  :-  jefe(X,Y).
2 superior(X,Y)  :-  jefe(X,Z), superior(Z,Y).
```

Programa Imperativo - Lenguaje C++

```
1  bool superior(individuo X, individuo Y, conjunto C) {
2     if ( jefe(X,Y) ) return true;
3     else  {
4        C = sacar(C,X); C = sacar(C,Y);
5        while ( !vacio(C) )  {
6           Z = buscarEn(C);
7           if ( jefe(X,Z) && superior(Z,Y) )  return true;
8           C = sacar(C,Z);
9        }
10       return false;
11    }
12 }
```

Los Lenguajes de Programación

¿Cómo es el texto que tiene un Programa?

Lenguaje de Programación

Idioma formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen la estructura y el significado de las instrucciones de que consta el lenguaje.

En realidad no deja de ser un "lenguaje", como lo puedan ser el español o el inglés, pero:

- son mucho más limitados, constan de muchísimas menos palabras (generalmente unas pocas decenas),
- son sintácticamente mucho más estrictos, de hecho un programa no puede tener ni un solo error sintáctico, y
- no permiten las ambigüedades del lenguaje natural, cada instrucción tiene una semántica clara y precisa.

Lenguajes de Programación

¿Por qué hay tantos?

- Proliferación: hay muchos lenguajes por varias razones, intereses empresariales, nuevas necesidades, etc.
- Eso divide el nivel de utilización de cada uno de ellos: no hay un lenguaje claramente dominante.
- Hay muchos lenguajes parecidos entre sí, tienen instrucciones equivalentes (misma semántica, aunque diferente sintaxis).
- Lo que más te interesa conocer como programador:
  1. las metodologías y paradigmas que existen,
  2. los lenguajes que soportan cada metodología,
  3. centrándonos en los lenguajes imperativos, las instrucciones que todos ellos poseen: asignación, condicionales, bucles,..., y
  4. las construcciones o algoritmos típicos que se pueden hacer con ellas.

Importante

Aprender nuevos lenguajes es sencillo si se adquieren los conocimientos básicos sobre la programación y sus metodologías

Lenguajes de Programación

Tipo de lenguajes

1. Según la complejidad de sus instrucciones:
   - Lenguajes de bajo nivel: sus instrucciones son las básicas que entiende el procesador (Fundamentos de Computadoras y Redes).
   - Lenguajes de alto nivel: instrucciones más complejas que suelen implicar varias instrucciones básicas.
2. Según la forma de ejecutar los programas que con ellos se escriben:
   - Lenguajes compilados: el compilador traduce el código fuente y genera un programa ejecutable (por el sistema operativo).
   - Lenguajes interpretados: partiendo del código fuente, se genera un código intermedio que requiere de un programa llamado intérprete para poder ser ejecutado.

El lenguaje Java que emplearemos en este curso es un lenguaje de alto nivel pseudointerpretado.

El lenguaje Java

¿Por qué Java?

1. Es un lenguaje de alto nivel: la tendencia en la industria es a usar lenguajes de alto nivel, el uso de los lenguajes de bajo nivel es marginal,
2. es un lenguaje orientado a objetos: introducir el paradigma Orientado a Objetos es el objetivo de esta asignatura,
3. es un lenguaje relativamente sencillo, con menos complejidad conceptual que otros, y
4. es posiblemente el lenguaje más utilizado hoy en día.

Programa: Hola, Mundo

Enunciado

Enunciado

Realizar un programa que escriba en la pantalla el mensaje "Hola, Mundo".

El programa es evidentemente el más simple que haremos en todo el curso, pero nos servirá para presentar:

1. la estructura general de un programa en Java,
2. nuestra primera clase en Java,
3. el método `main()`, y
4. cómo imprimir mensajes de texto en la pantalla.

Hola, Mundo

Código fuente: HolaMundo.java

HolaMundo.java

```
1  /** Muestra por pantalla el mensaje "Hola, Mundo"
2   *  @author los profesores de IP  */
3  public class HolaMundo {
4
5      public static void main(String[ ] args) {
6          //Imprimimos el mensaje Hola, Mundo
7          System.out.print("Hola, Mundo");
8      }
9
10 }
```

Las clases

Java es un lenguaje Orientado a Objetos

- Todo programa en Java debe contener una **clase**.
- Una clase con el mismo nombre que el del fichero donde está escrita (en nuestro ejemplo **HolaMundo**).
- Para declarar la clase, hay que escribir las palabras reservadas `public class` y después el nombre de la clase (línea 3).
- El contenido de la clase va entre llaves:

Declaración de una clase

```
public class nombre {
    elementos que contiene la clase (atributos+métodos)
}
```

- Estudiaremos todo lo relacionado con las clases en los Temas 3 y 6.

El método main()

El punto de inicio de la ejecución del programa

- En este caso, la clase sólo tiene el **método** `main()`.
- Sería engorroso explicar tan pronto todo lo que tiene su declaración, así que de momento la escribiremos tal como aparece en la línea 5:

El método `main()`

```
public static void main (String [ ] args) {
    secuencia de intrucciones
}
```

- Lo más importante que se debe saber es que lo primero que se hará al ejecutar el programa serán las instrucciones que contenga el método `main()`, de ahí su importancia.
- El programa podría tener parámetros (variable `args`).

Imprimir en pantalla

Los métodos print() y println()

- En Java, la pantalla está asociada al objeto `System.out`.
- Para imprimir podemos usar su método `print()`.
- El método `print()` imprime en la pantalla una cadena de texto.
- La cadena se pasa entre **comillas dobles**.

Enunciado

Realizar un programa que escriba en la pantalla las cadenas "Bienvenido a" e "Introducción a la Programación", cada una en una línea.

- En este caso debemos usar el método `println()`.
- También imprime una cadena de texto, pero después salta a la línea siguiente.

Bienvenido a IP

Código fuente: Bienvenido.java

Bienvenido.java

```
1  /** Muestra en consola un texto de bienvenida a IP
2   *  @author los profesores de IP  */
3  public class Bienvenido {
4      public static void main(String[ ] args) {
5          /* Imprimimos las cadenas "Bienvenido a" e
6          "Introducción a la Programación" en dos líneas */
7          System.out.println("Bienvenido a");
8          System.out.print("Introducción a la Programación");
9      }
10 }
```

Los comentarios

Explicaciones útiles sobre el funcionamiento del programa

Comentarios

Explicaciones útiles, escritas dentro del programa, para aclarar la lógica de funcionamiento del mismo. Sirven para ayudar a entender mejor el programa.

En Java se pueden escribir tres tipos de comentarios:

1. //: el comentario empieza tras las dos barras y acaba al final de la línea,
2. /*: el comentario empieza con /* y acaba cuando encuentra */ (generalmente en otra línea), y
3. /**: similar al anterior, empieza en /** y acaba en */. La diferencia es que estos comentarios, procesados con algunas herramientas, permiten generar automáticamente documentación del programa (**Javadoc**).

Propiedades de un programa

Tus programas deben cumplir todas!

1. **correcto** o eficaz, que produzca resultados coherentes con su especificación,
2. **eficiente**, que optimice el consumo de recursos:
   - tiempo: tarde en ejecutarse lo menos posible,
   - memoria: use la menor cantidad de memoria posible para sus datos,
   - tiempo y memoria son dos variables a veces imposibles de optimizar juntas, para que un programa tarde menos debe ocupar más memoria y viceversa,
3. **legible** o claro, fácilmente comprensible por el autor (tiempo después de su diseño) y por cualquier otro programador que lea el código fuente (junto con la documentación aportada),
4. **reutilizable**, algunos trozos (clases o funciones, según la metodología) deberían poder ser utilizados en otros casos, y
5. **modificable**, a lo largo de su ciclo de vida.

Que vamos a exigir que cumplan tus programas

Correctos, eficientes, legibles

1. **correcto**: todos los programas hacen algo, lo difícil es lograr que hagan exactamente las cosas para las que se diseñaron, "tienen que funcionar".
2. **eficiente**: entre todos los programas que funcionan, uno es el más eficiente (óptimo). Cuanto más se aproxime un programa a ese óptimo, mejor.
3. **legible**: pero no bastará con que sea correcto y eficiente, además tendrá que estar bien escrito y documentado (muy importante):
   - **Sangrados**: bien sangrado para facilitar su lectura.
   - **Comentarios**: deben ayudar a entender cómo funciona internamente el programa, cuál es su lógica de funcionamiento.
   - **Documentación**: otros documentos (distintos al código fuente) necesarios para que otros programadores puedan entender los detalles técnicos del programa.

Ciclo de vida de un Programa

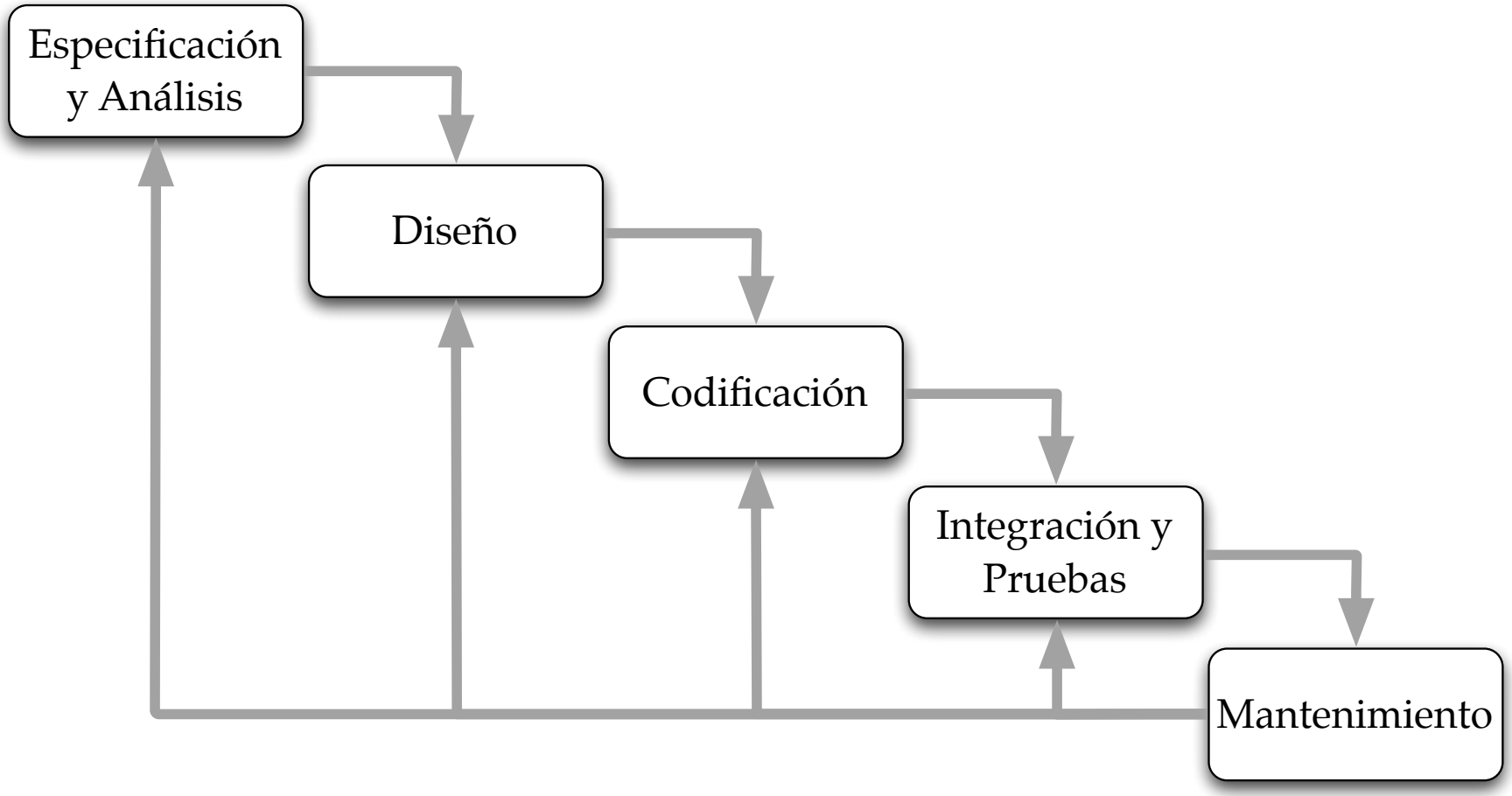
Fases en la realización de un Programa

Cuando se acomete un programa, sus programadores NO se lanzan a escribir el código del mismo.

1. Analizan qué debe hacer el programa (**Especificación**) y determinan las partes en que se puede descomponer (**Análisis**),
2. se diseña claramente cómo va a ser internamente cada parte (**Diseño**),
3. solamente tras realizar el Análisis y un Diseño, el programa se escribe (**Codificación**),
4. en aplicaciones de cierta entidad, compuestas de muchas partes, sería necesario juntar todas ellas (**Integración**),
5. cuando el programa es operativo, se prueba si cumple todos los requisitos y se detectan sus fallos (**Pruebas**), y
6. durante su vida útil, el programa se corrige o se le añaden nuevas funcionalidades (**Mantenimiento**).

Ciclo de vida

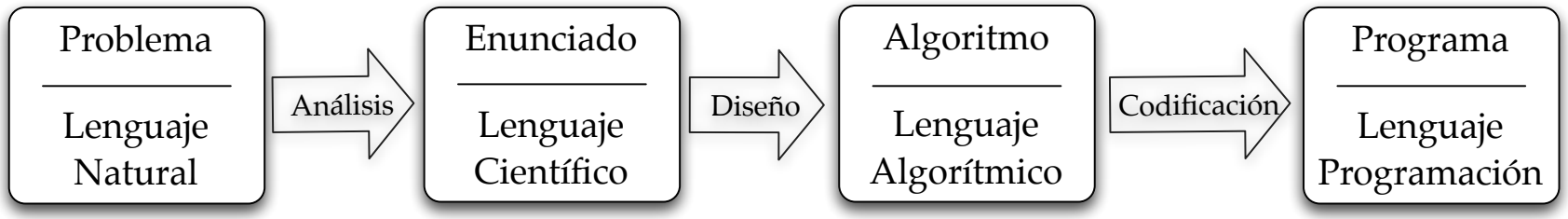
Un proceso iterativo



Cada fase usa un lenguaje distinto

Etapas en un programa simple

- Dado que nuestros programas serán simples, todas esas fases resultarán más sencillas y cortas.
- Partiremos siempre de un enunciado preciso, que indicará lo que el programa debe hacer. Esto hará el análisis más sencillo.
- Pero siempre se aplicarán todas las fases. Por ejemplo, **hay que probar muy bien los programas**.
- Cada fase tiene su propio lenguaje.



Análisis

Cómo descomponer un Programa

Análisis

El objetivo del Análisis es descomponer el programa en unidades más simples, facilitando así su realización.

Pero:

- No se trata de partir el programa como si fuera una tarta.
- Es más bien dividirlo en unidades funcionales básicas, con una entidad propia, clara y bien definida.
- Este proceso permite varias cosas:
  1. entender mejor las distintas partes que componen el programa,
  2. propiciar la división del trabajo, y
  3. favorecer las fases posteriores de la elaboración del programa (es más sencillo hacer el diseño o realizar las pruebas).

Análisis

Programación Modular vs. Programación Orientada a Objetos

Las dos formas más extendidas para descomponer y simplificar la construcción de programas son:

1. **Programación Modular**: la unidad básica es la **función**, el programa se compondrá de un conjunto de funciones.
2. **Programación Orientada a Objetos**: la unidad básica es la **clase**, el programa se formará por la unión de varias clases, relacionadas entre sí por herencia y composición.

Se van a estudiar en:

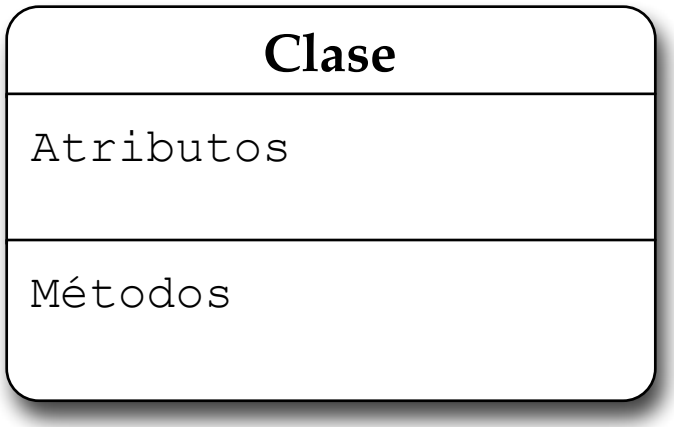
- Fundamentos de Informática: Programación Modular.
- Introducción a la Programación: Programación Orientada a Objetos (construir una clase, composición).
- Metodología de la Programación: Programación Orientada a Objetos (jerarquías de clases, herencia, polimorfismo).

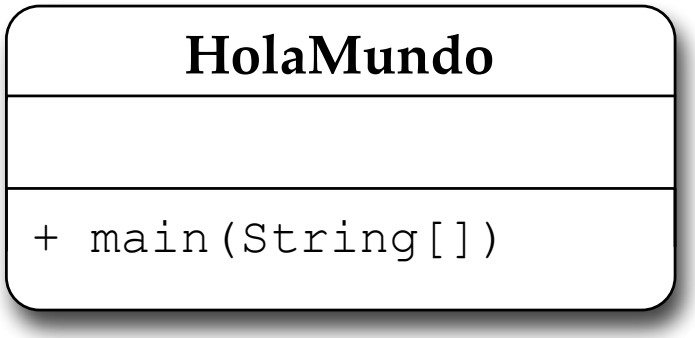
Diseño - Notación UML

Qué tienen las clases de nuestro Programa

Para explicar cómo van a ser las clases que tendrán nuestros programas usaremos la notación UML (Unified Modeling Language). Sirve para indicar los elementos que tiene una clase:

1. sus atributos (los datos que tiene internamente cada objeto de la clase), y
2. las funciones (métodos) que los objetos de esa clase pueden realizar.





Diseño - Algoritmos y Pseudocódigo

Cómo son los métodos de la clase

Para detallar cómo van a ser internamente las acciones (métodos) de las clases que hagamos, indicaremos sus **algoritmos** expresados en **lenguaje algorítmico** o **pseudocódigo**.

Algoritmo

Conjunto finito de instrucciones ordenadas que permite realizar una acción mediante pasos sucesivos. Dado un estado inicial y unos datos, siguiendo dichos pasos sucesivos se llega a un estado final (solución).

Lenguaje algorítmico o Pseudocódigo

Permite describir a alto nivel un algoritmo, empleando una mezcla de lenguaje natural y algunas convenciones sintácticas propias de los lenguajes de programación, como asignaciones, bucles y condicionales.

Algoritmo 1 HolaMundo - método `main()`

Escribir en pantalla "Hola, Mundo"

Pruebas de software

Garantizar la calidad de los Programas

Pruebas de Software

Las pruebas de software son todos los procesos que permiten evaluar y verificar la calidad de un producto software.

Tipos de pruebas:

- **Funcionales**
- No funcionales
- **Unitarias**
- De integración
- De rendimiento
- De carga
- Alfa
- Beta
- De seguridad
- De compatibilidad
- De usabilidad y accesibilidad
- De aceptación

Nos centraremos en las pruebas funcionales:

1. Detectar y corregir los errores que contiene el programa
2. Comprobar que el programa funciona correcta y eficientemente

Requisitos funcionales y pruebas

¿Qué debe hacer el programa?

Requisito funcional

Es cada una de las funcionalidades que tiene un programa o un sistema software.

Ejemplo: Programa de tratamiento de imágenes:

RF1 Abrir y leer ficheros de imágenes en formato jpg, tiff y bmp.
RF2 Convertir imágenes en color a escala de grises.
RF3 Grabar una imagen a fichero en cualquier de los formatos soportados, independientemente del formato de la misma.

Prueba 1.1 Abrir un fichero en formato jpg y comprobar que el programa lo abre correctamente.
Prueba 1.2 Idem en formato tiff.
Prueba 1.3 Idem en formato bmp.
Prueba 1.4 Abrir un fichero en uno formato que no sea ni jpg, ni tiff, ni bmp. El programa debe dar un mensaje de error.

Especificación Formal de un Programa

Qué debe hacer y bajo qué condiciones debe funcionar un Programa

Enunciado

Realizar un programa que permita al usuario introducir dos números enteros por teclado y muestre su suma en la pantalla.

- Siempre partiremos de enunciados claros, que indican lo que los programas deben hacer.
- Es importantísimo saber LEER y ENTENDER los enunciados de los programas.
- Es imprescindible para que los programas hagan lo que deben.
- Los enunciados siempre tienen dos partes:
  1. el resultado que el programa debe producir o hacer, y
  2. los datos que usa y todos los condicionantes que el programa debe tener en cuenta.

Precondición y Postcondición

Condiciones iniciales y resultado final

Precondición

Indica el conjunto de condiciones para las que el programa producirá una salida correcta.

Ejemplo: viene dado por la frase "permita al usuario introducir dos números enteros". El programa funcionará cuando el usuario introduzca dos números enteros, no otra cosa.

Postcondición

Indica el resultado que el programa producirá si se cumplen las condiciones que indica la precondición.

Ejemplo: "muestre su suma en la pantalla". Al final, el programa mostrará la suma de ambos números, esa es la postcondición.

Precondición y Postcondición

Condiciones iniciales y resultado final

Importante

Un programa solamente deberá funcionar correctamente, es decir, deberá alcanzar su postcondición, cuando se cumplan las condiciones indicadas en su precondición

Terminal — bash — 78×9

```
aic9:SumaDosNumeros juanjo$ java -Dfile.encoding=UTF8 SumaDosNumeros
Introduce dos números: 12.1   14.7
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:840)
        at java.util.Scanner.next(Scanner.java:1461)
        at java.util.Scanner.nextInt(Scanner.java:2091)
        at java.util.Scanner.nextInt(Scanner.java:2050)
        at SumaDosNumeros.main(SumaDosNumeros.java:15)
aic9:SumaDosNumeros juanjo$
```

¿Está mal el programa?

- NO, hace lo que se pedía.
- Si se deseaba poder sumar números reales, la especificación (su enunciado) tendría que haber sido otro.