ENTORNOS DE DESARROLLO

UNIDAD 1

INDICE

1.DESARROLLO DE SOFTWARE

En esta primera parte de la unidad vamos a aprender a reconocer elementos y herramientas que se usan para desarrollar un programa informático, así como las características y fases que tiene que pasar hasta su puesta en funcionamiento.

Definiremos conceptos como qué es el software y su ciclo de vida. Analizaremos los distintos tipos de lenguajes de programación con sus características y veremos las fases de desarrollo de la ingeniería del software.

1.1. EL SOFTWARE DEL ORDENADOR

Antes de comenzar con la definición de software, es necesario aclarar la diferencia entre hardware y software. Elordenador está compuesto por dos partes: la parte física, que llamamos **hardware** y que está compuesta por elementos físicos como el teclado, el ratón, el monitor, los discosduros o la placa base, entre otros elementos. En definitiva, lo forman todos aquellos componentes que podemos very tocar. Por otro lado, el ordenador posee otra parte lógicallamada **software**, encargada de dar instrucciones al hardware y hacer funcionar la computadora.

En este apartado veremos conceptos básicos para el desarrollo del software.

Además de dar instrucciones al hardware, el software también almacenará los datos necesarios para ejecutar los programas y contendrá los datos almacenados del ordenador.

Podemos dividir el software en dos categorías: según las **tareas que realiza** y según su **método de distribución**.

1.1.1. SOFTWARE BASADO EN EL TIPO DE TRABAJO QUE REALIZAN

Según esta clasificación, podemos distinguir tres tipos de software:

- **Software de sistema**: es el que hace que el hardware funcione. Está formado por programas que administranla parte física e interactúa entre los usuarios y el hardware. Algunos ejemplos son los sistemas operativos, loscontroladores de dispositivos, las herramientas de diagnóstico, etcétera.
- **Software de aplicación**: aquí tendremos los programas que realizan tareas específicas para que el ordenador seaútil al usuario, por ejemplo, los programas ofimáticos, elsoftware médico o el de diseño asistido, entre otros.
- **Software de programación o desarrollo**: es el encargado de proporcionar al programador las herramientas necesarias para escribir los programas informáticos y para hacer uso de distintos lenguajes de programación.Entre ellos encontramos los entornos de desarrollo integrado (IDE).

1.1.2. SOFTWARE BASADO EN EL MÉTODO DE DISTRIBUCIÓN

Mediante esta clasificación, también distinguimos trestipos de software:

- **Shareware**: el usuario puede evaluar de forma gratuitael producto, pero con limitaciones en algunas características. Si realiza un pago, podrá disfrutar del software sinlimitaciones. Por ejemplo, Malwarebytes.
- **Freeware**: donde los usuarios de software pueden des-cargar el aplicativo de forma gratuita, pero que mantienelos derechos de autor. Sería el caso de Firefox.
- **Adware**: es un aplicativo donde se ofrece publicidad incrustada, incluso durante la instalación de este. Por ejemplo, CCleaner.

1.1.3. LICENCIAS DE SOFTWARE. SOFTWARE LIBRE Y PROPIETARIO

Una **licencia** es un contrato entre el desarrollador de un software y el usuario final. En él se especifican los derechos y deberes de ambas partes. Es el desarrollador el que especifica qué tipo de licencia distribuye. Existen dos tipos de licencias:

- **Software libre**: el autor de la licencia concede libertadesal usuario, entre ellas están:
  - Libertad para usar el programa con cualquier fin.
  - Libertad para saber cómo funciona el programa y adaptar el código a nuestras propias necesidades.
  - Libertad para poder compartir copias con otros usuarios.
  - Libertad para poder mejorar el programa y publicar lasmodificaciones realizadas.

  Hay que dejar claro que la palabra “libre” en este contexto no significa que sea gratis. El **software libre** es unconcepto de libertad, no de precio. Nos referimos a la libertad de los usuarios para ejecutar, copiar, distribuir,cambiar, reutilizar y mejorar el software.
- **Software propietario**: este software no nos permitirá acceder al código fuente del programa y, de forma general, nos prohibirá la redistribución, la reprogramación, la copia o el uso simultáneo en varios equipos.

Como ejemplo, tenemos programas de antivirus como Kaspersky o programas ofimáticos como Microsoft Office.

La licencia que más se usa en el software libre es la licencia **GPL** (*general public license* o licencia pública general), que nos dejará usar y cambiar el programa, con el único requisito de que se hagan públicas las modificaciones realizadas.

BUSCA EN LA WEB

Para encontrar más información sobre licencia GPL, podremos visitar la siguiente dirección:

www.gnu.org/licenses/license-list.es.html#SoftwareLicenses

ponte a prueba

¿En qué tipo de método de distribución estaría el siguiente software?

a) Adware
b) Shareware
c) Freeware
d) Jailware

El software libre puede ser vendido.

a) Verdadero
b) Falso

1.2. CONCEPTO DE PROGRAMA INFORMÁTICO

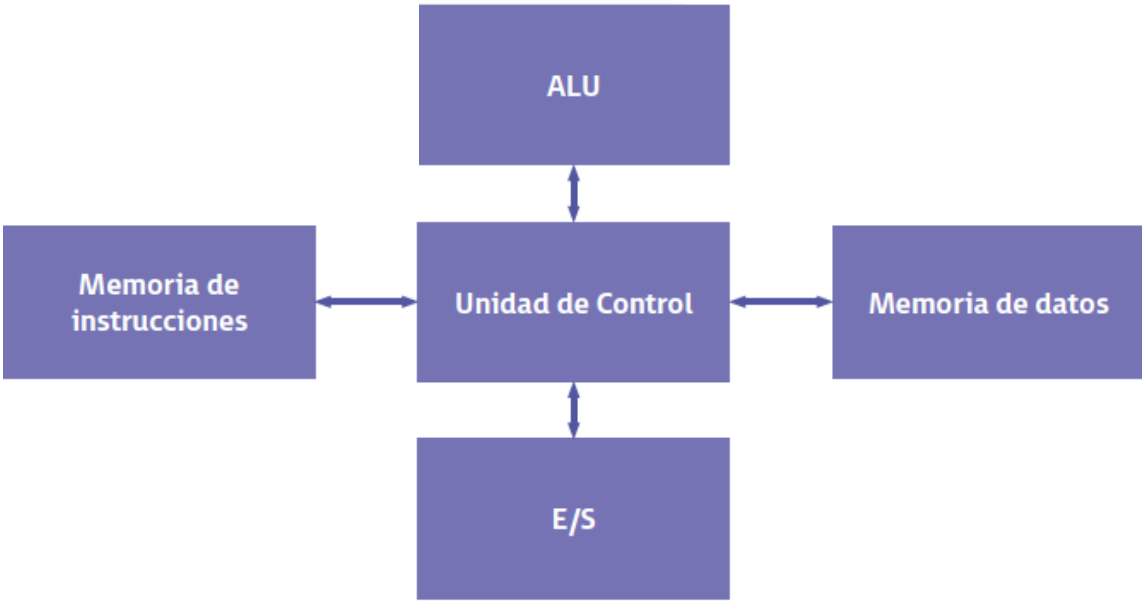
Un **programa informático** es un fragmento de software formado por una secuencia de instrucciones y procesos construido con el fin de cumplir un objetivo concreto. Estos programas informáticos están escritos utilizando lenguajes de programación como Java, C# o Python y deben sertraducidos al lenguaje máquina para que puedan ser pro-cesados por nuestro ordenador.

1.2.1. PROGRAMA Y COMPONENTES DEL SISTEMA INFORMÁTICO

El diseño de la arquitectura de nuestra CPU (en inglés, *central processing unit*) está basado en la arquitectura de **VonNeumann**[1].

Por el contrario, tenemos otra estructura que es la llamada**Harvard**[2]. La característica fundamental es que tanto la memoria RAM como la memoria de instrucciones no comparten características comunes.

La unidad de control (UC) es el centro de la arquitectura yconecta con la unidad aritmético-lógica (UAL), los dispositivos de entrada y salida y con ambas memorias mencionadas anteriormente.



1 MacTutor History of Mathematics Archive: https://mathshistory.st-andrews.ac.uk/Biographies/Von_Neumann/

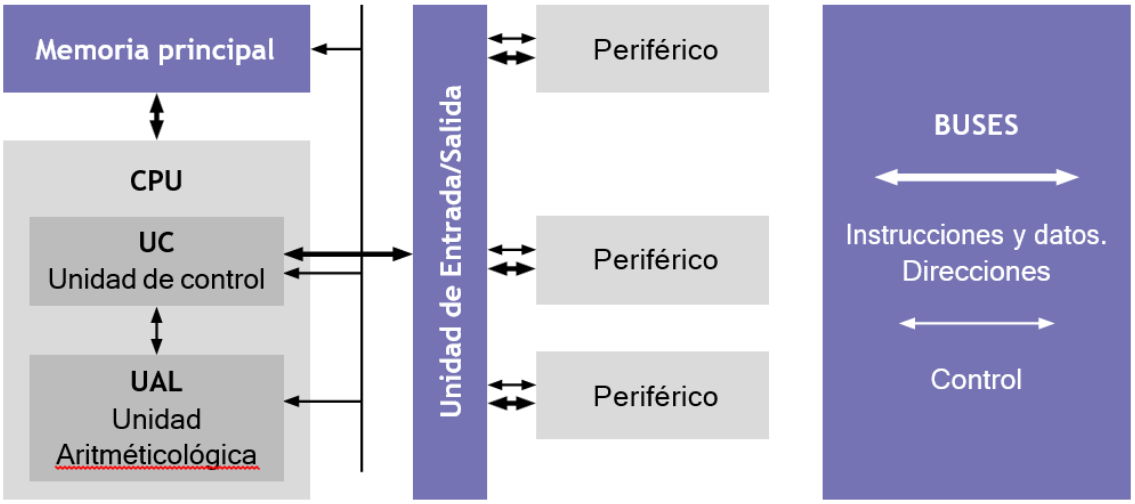
2 García Ortega, Víctor H.; Sosa Savedra, Julio C.; Ortega S., Susana; Tovar, Rubén H. (2009) *Microprocesador didáctico de arquitectura RISC implementado en un FPGA*. E-Gnosis. https://www.redalyc.org/pdf/730/73012215014.pdf

En la arquitectura de Von Neumann, puede estar leyendouna instrucción o bien escribir un dato en la memoria, peroambos no pueden ocurrir a la vez, ya que utilizan el mismo sistema de buses.

En la arquitectura Harvard, la CPU puede tanto leer una instrucción como realizar un acceso de memoria de datos.

Si vemos la arquitectura Von Neumann, entenderemos cómo funcionan los componentes que conforman la CPU:

- **La unidad de control (UC)**: se encarga de interpretar y ejecutar las instrucciones que se almacenan en la memoria principal y, además, genera las señales de controlnecesarias para ejecutarlas.
- **La unidad aritmético-lógica (UAL)**: es la que recibe los datos y ejecuta operaciones de cálculo y comparaciones,además de tomar decisiones lógicas (si son verdaderas ofalsas), pero siempre supervisada por la unidad de control.
- **Los registros**: son aquellos que almacenan la información temporal. Es el almacenamiento interno de la CPU.



A continuación, vamos a ver los diferentes registros queposee la UC:

- **Contador de programa (CP)**: contendrá la dirección dela siguiente instrucción para realizar. Su valor será actualizado por la CPU después de capturar una instrucción.
- **Registro de instrucción (RI)**: es el que contiene el código de la instrucción, se analiza dicho código. Consta de dos partes: el código de la operación y la dirección dememoria en la que opera.
- **Registro de dirección de memoria (RDM)**: tiene asignada una dirección correspondiente a una posición de memoria que va a almacenar la información mediante elbus de direcciones.
- **Registro de intercambio de memoria (RIM)**: recibe o envía, según si es una operación de lectura o escritura,la información o dato contenido en la posición apuntadapor el RDM.

- **Decodificador de instrucción (DI)**: extrae y analiza el código de la instrucción contenida en el RI.
- **El reloj**: marca el ritmo del DI y nos proporciona unos impulsos eléctricos con intervalos constantes a la vez que marca los tiempos para ejecutar las instrucciones.
- **El secuenciador**: son órdenes que se sincronizan con elreloj para que ejecuten correctamente y de forma ordenada la instrucción.

Cuando ejecutamos una instrucción podemos distinguir dos fases:

1. **Fase de búsqueda**: se localiza la instrucción en la memoria principal y se envía a la unidad de control parapoder procesarla.
2. **Fase de ejecución**: se ejecutan las acciones de las instrucciones.

Para que podamos realizar operaciones de lectura y escritura en una celda de memoria, se utilizan el RDM, el RIMy el DI. El decodificador de instrucción es el encargado deconectar la celda RDM con el registro de intercambio RIM,el cual posibilita que la transferencia de datos se realice enun sentido u otro según sea de lectura o escritura.

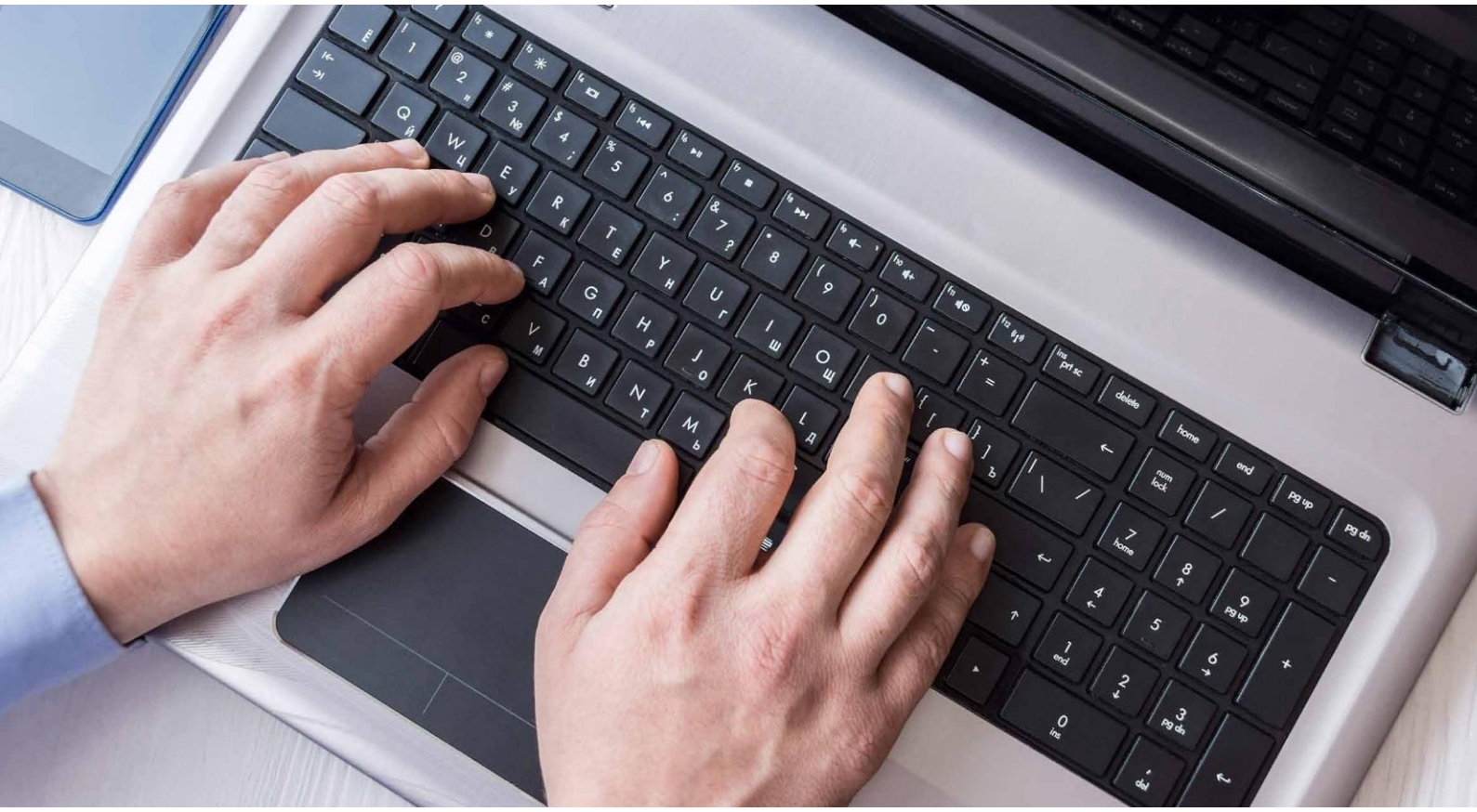
ponte a prueba

¿Qué tipo de lenguaje de programación es Python?

a) Alto nivel
b) Bajo nivel
c) Ensamblador
d) Ninguna de las respuestas es correcta

¿Qué función realiza la ALU?

a) Operaciones aritméticas
b) Decisiones lógicas
c) Operaciones de comparación
d) Todas las opciones son correctas



1.3. CÓDIGO FUENTE, CÓDIGO OBJETO Y CÓDIGO EJECUTABLE. MÁQUINAS VIRTUALES

En la etapa de diseño construimos las herramientas de software capaces de generar un código fuente en lenguaje deprogramación. Estas herramientas pueden ser diagramas de clase (realizados con el lenguaje de modelado **UML**), pseudocódigo o diagramas de flujo.

La etapa de codificación es la encargada de generar el código fuente, y pasa por diferentes estados.

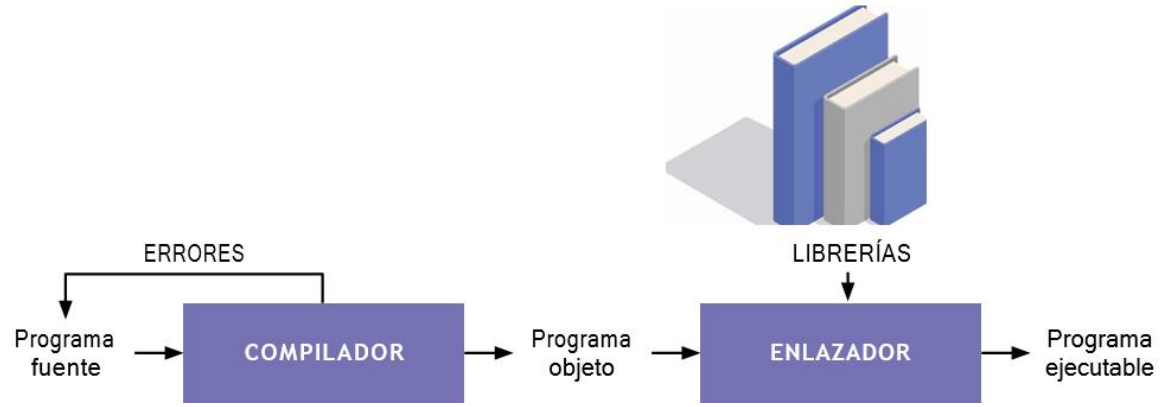
1.3.1 TIPOS DE CÓDIGO

Cuando escribimos un código, pasa por distintos estadoshasta que se ejecuta:

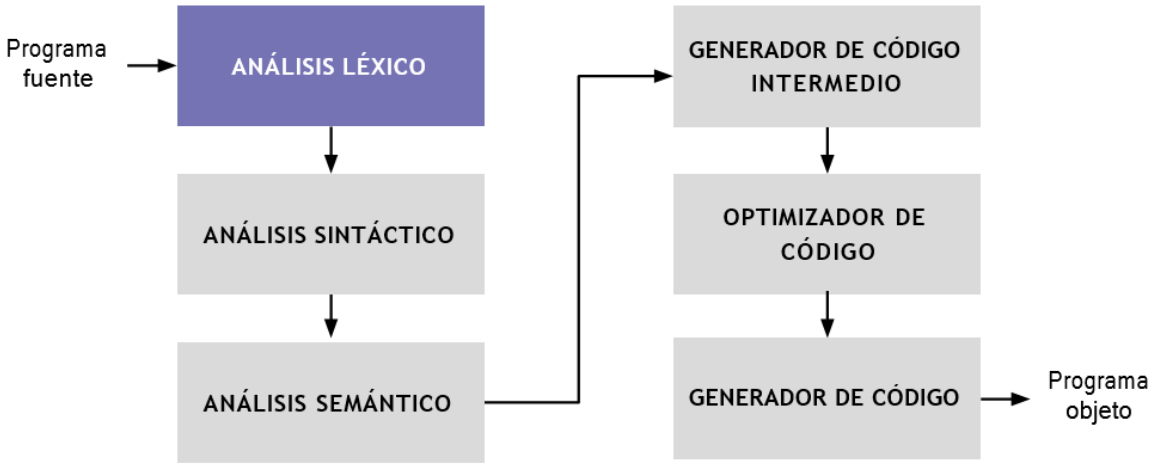
- **Código fuente**: es el código realizado por los programadores usando algún editor de texto o herramienta deprogramación. Posee un lenguaje de alto nivel y para escribirlo se parte de, por ejemplo, diagramas de clases. No se puede ejecutar directamente en el ordenador.
- **Código objeto**: es el código que se crea tras realizar la compilación del código fuente. Este código no es entendido ni por el ordenador ni por nosotros. Es una re-presentación intermedia de bajo nivel.
- **Código ejecutable**: este código se obtiene tras unir el código objeto con varias librerías para que así pueda serejecutado por el ordenador.

1.3.2. COMPILACIÓN

La compilación es el proceso a través del cual se convierteun programa en lenguaje máquina a partir de otro programa de computadora escrito en otro lenguaje. La compilaciónse realiza a través de dos programas: el compilador y el en-lazador. Si en el **compilador** se detecta algún tipo de errorno se generará el código objeto y tendremos que modificarel código fuente para volver a pasarlo por el compilador.



Dentro del compilador, tendremos varias **fases** en las que se realizan distintas operaciones:



- **Análisis léxico**: se lee el código obteniendo unidades decaracteres llamados *tokens*. Ejemplo: la instrucción resta
    - = 2 – 1, genera 5 *tokens:* resta, =, 2, –, 1.
- **Análisis sintáctico**: recibe el código fuente en forma de *tokens* y ejecuta el análisis para determinar la estructuradel programa, se comprueba si cumplen las reglas sin- tácticas.
- **Análisis semántico**: revisa que las declaraciones sean correctas, los tipos de todas las expresiones, si las operaciones se pueden realizar, si los *arrays* son del tamañocorrecto, etcétera.

- **Generación de código intermedio**: después de analizarlo todo, se crea una representación similar al códigofuente para facilitar la tarea de traducir al código objeto.
- **Optimización de código**: se mejora el código intermedioanterior para que sea más fácil y rápido a la hora de interpretarlo la máquina.
- **Generación de código**: se genera el código objeto.

El **enlazador** insertará en el código objeto las librerías necesarias para que se pueda producir un programa ejecutable. Si se hace referencia a otros ficheros que contengan las librerías especificadas en el código objeto, se combinacon dicho código y se crea el fichero ejecutable.

1.3.3. MÁQUINAS VIRTUALES

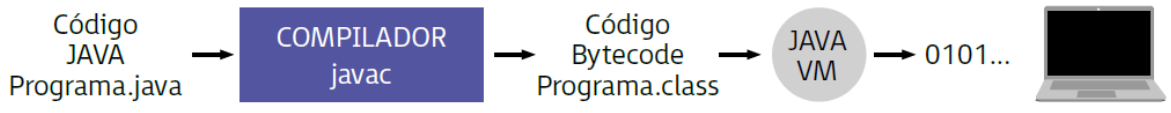
Una máquina virtual es un tipo de software capaz de ejecutar programas como si fuese una máquina real. Se clasificanen dos categorías:

- **Máquinas virtuales de sistema**. Nos permiten virtualizar máquinas con distintos sistemas operativos en cada una. Un ejemplo son los programas VMware Workstation o Virtual Box, que podremos usar para probar nuevos sistemas operativos o ejecutar programas.
- **Máquinas virtuales de proceso**. Se ejecutan como un proceso normal dentro de un sistema operativo y solo so-portan un proceso. Se inician cuando lanzamos el proceso y se detienen cuando este finaliza. El objetivo es proporcionar un entorno de ejecución independiente del hardwarey del sistema operativo y permitir que el programa sea ejecutado de la misma forma en cualquier plataforma. Ejemplo de ello es la máquina virtual de Java (JVM).

Las máquinas virtuales requieren de grandes recursos, porlo que hay que tener en cuenta dónde las vamos a ejecutar. Los procesadores tienen que ser capaces de soportardichas máquinas para que no se ralentice o colapse el restodel sistema.

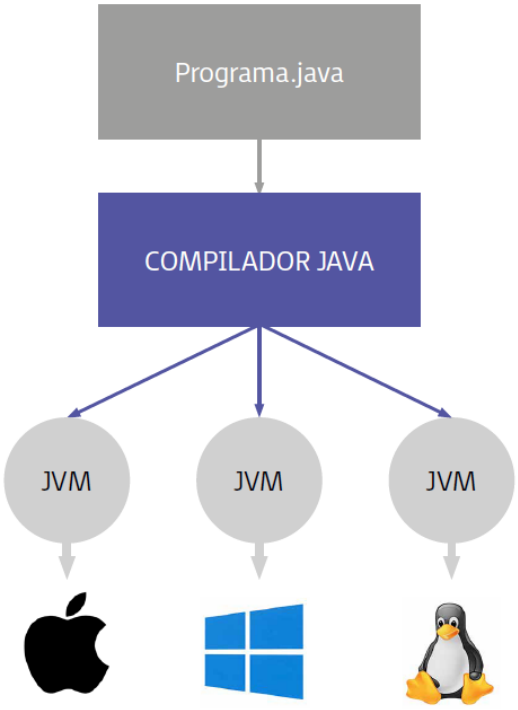
La máquina virtual de Java

Los programas que se compilan en lenguaje Java son ca-paces de funcionar en cualquier plataforma (UNIX, Mac, Windows, Solaris, etc.). Esto se debe a que el código no loejecuta el procesador del ordenador sino la propia *Máquina Virtual de Java* (JVM).



El funcionamiento básico de la máquina virtual es el siguiente:

1. El código fuente estará escrito en archivosde texto planos con la extensión .java.
2. El compilador ***javac*** generará uno o varios archivos, siempre que no se produzcan errores, y tendrán la extensión .class.
3. Este fichero .class contendrá un lenguajeintermedio entre el ordenador y el sistema operativo y se llamará *bytecode*.
4. La JVM coge y traduce mediante un compilador JIT (siglas en inglés de compilación en tiempo de ejecución) el *bytecode* en código binario para que el procesador de nuestro ordenador sea capaz de reconocerlo.
5. Los ficheros .class podrán ser ejecutadosen múltiples plataformas.



La máquina virtual de Java contiene, entre otras, las instrucciones para las siguientes tareas:

- Carga y almacenamiento de datos.
- Excepciones (errores en tiempo de ejecución).
- Operaciones aritméticas.
- Conversiones de tipos de datos.
- Llamadas a métodos y devolución de datos.
- Creación y manejo de objetos.

Una de las desventajas de usar este tipo de lenguajes que se basan en una máquina virtual puede ser que son más lentos que los lenguajes ya compilados, debido a la capa intermedia.

1.4. TIPOS DE LENGUAJES DE PROGRAMACIÓN. CLASIFICACIÓN Y CARACTERÍSTICAS DE LOS LENGUAJES MÁS DIFUNDIDOS

Como hemos definido anteriormente, un programa informático es un conjunto de instrucciones escritas en un lenguaje de programación. Asimismo, lenguaje de programación hace referencia al conjunto de caracteres, reglas yacciones combinadas y consecutivas que un equipo debeejecutar.

Constará de los siguientes elementos:

- **Alfabeto o vocabulario**: conjunto de símbolos permitidos.
- **Sintaxis**: reglas para realizar correctamente construcciones con los símbolos.
- **Semántica**: reglas que determinan el significado deconstrucción del lenguaje.

1.4.1. CLASIFICACIÓN Y CARACTERÍSTICAS

Podemos clasificar los lenguajes de programación basándonos en los siguientes criterios:

Según su nivel de abstracción:	Lenguajes de bajo nivel. Lenguajes de nivel medio. Leguajes de alto nivel.
Según la forma de ejecución:	Lenguajes compilados. Lenguajes interpretados.
Según el paradigma de programación:	Lenguajes imperativos. Lenguajes funcionales. Lenguajes lógicos. Lenguajes estructurados. Lenguajes orientados a objetos.

Según su nivel de abstracción

- **Lenguajes de bajo nivel**: el lenguaje de más bajo nivelpor excelencia es **el lenguaje máquina**, el que entiendedirectamente la máquina. Utiliza el lenguaje binario (0 y1) y los programas son específicos para cada procesador.

  Al lenguaje máquina le sigue el **lenguaje ensamblador**. Es complicado de aprender y es específico para cada procesador. Cualquier programa escrito en este lenguaje tiene que ser traducido al lenguaje máquina para que sepueda ejecutar. Se utilizan nombres mnemotécnicos y las instrucciones trabajan directamente con registros de memoria física.

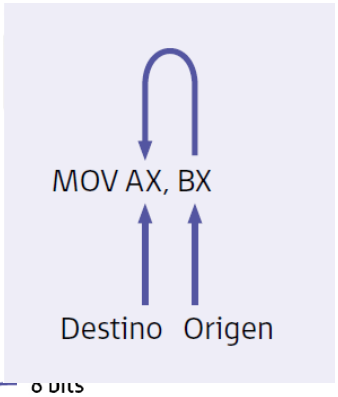
  En el lenguaje ensamblador, podemos encontrar:

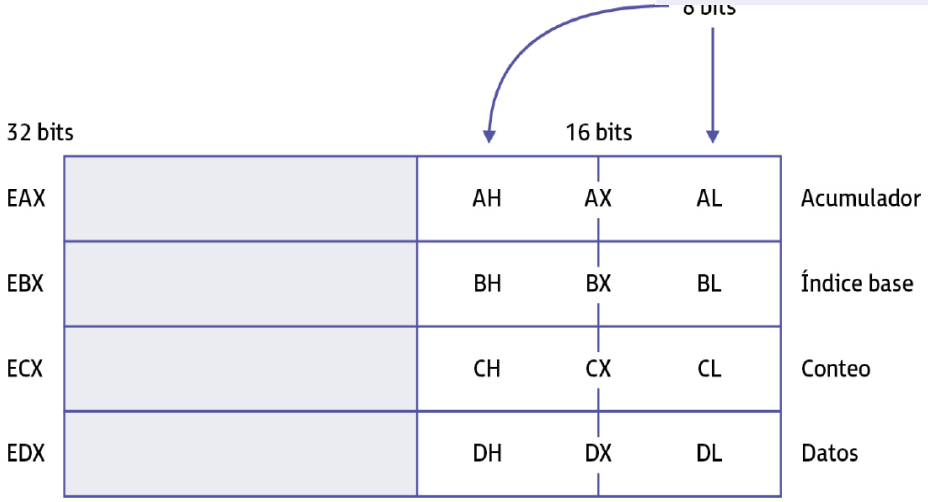
  – Los registros extendidos de 32 bits, como EAX (registro acumulador), EBX (registro de índice base), ECX (registro de conteo) o EDX (registro de datos).

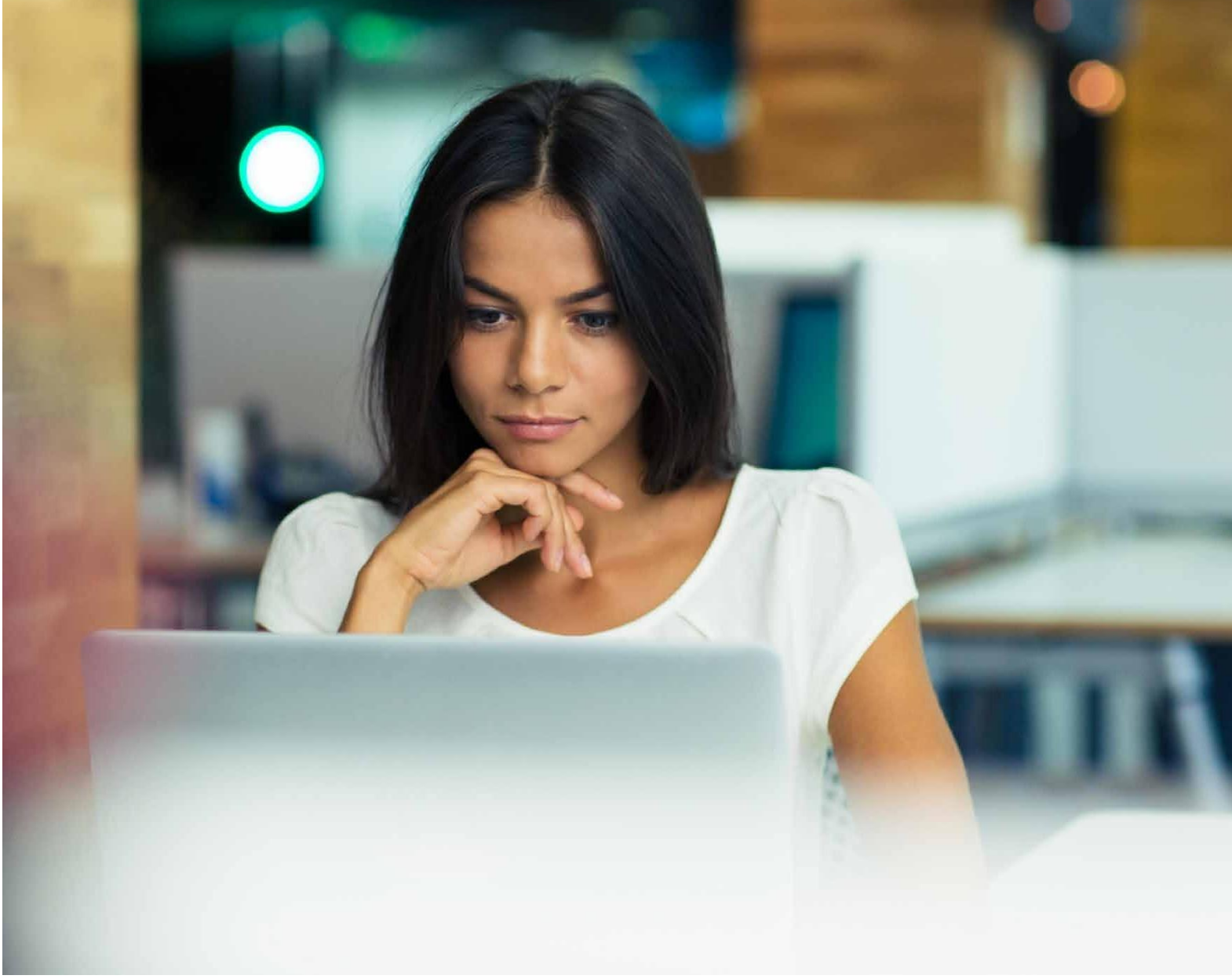
  – Estos registros se dividen en registros de menor tamaño, como AX, BX, CX y DX, de 16 bits, y estos, a su vez,en otros de 8 bits, como AH, AL, BH, BL, CH, CL, DH o DL.

    - **Registro EAX**: el acumulador se utiliza para instrucciones tales como multiplicación o división.
    - **Registro EBX**: guarda la dirección de desplazamientode una posición en el sistema de memoria.
    - **Registro ECX**: es un registro de propósito general queguarda la cuenta de varias instrucciones. Realiza funciones de contador.
    - **Registro EDX**: es un registro de propósito general quealmacena datos de, por ejemplo, aplicaciones aritméticas como el divisor antes de hacer una división.

Podemos realizar operaciones con estos registros. Porejemplo, una instrucción ADD ECX, EBX suma el con-tenido de 32 bits de EBX con el de ECX (solo ECX cambiadebido a esta instrucción) o mover el contenido de unregistro a otro MOV AX, BX (el contenido de BX se alma-cenaría en el registro AX).







- **Lenguajes de nivel medio**: poseen características de ambos tipos de nivel, tanto del nivel bajo como del alto, yse suele usar para la creación de sistemas operativos. Unlenguaje de nivel medio es el lenguaje C.
- **Lenguajes de alto nivel**: este tipo de lenguaje es más fácil a la hora de aprender, ya que estos lenguajes utilizannuestro lenguaje natural. El idioma que se suele emplear es el inglés y, para poder ejecutar lo que escribamos, necesitaremos un compilador para que traduzca al lenguaje máquina las instrucciones.

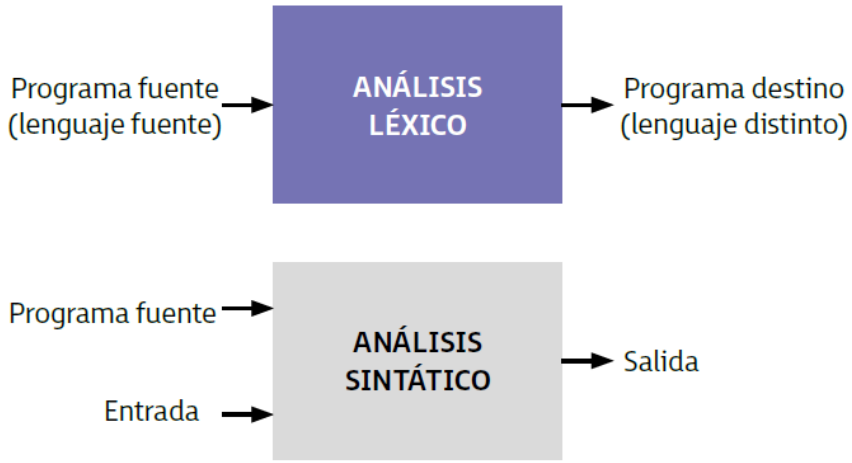
  Este lenguaje es independiente de la máquina, ya que no depende del hardware del ordenador.

  Algunos ejemplos de lenguajes de alto nivel son ALGOL, C++, C#, Clipper, COBOL, Fortran, Java, Logo y Pascal.

Según la forma de ejecución

- **Lenguajes compilados**: al programar en alto nivel, hay que traducir ese lenguaje a lenguaje máquina a través de compiladores.

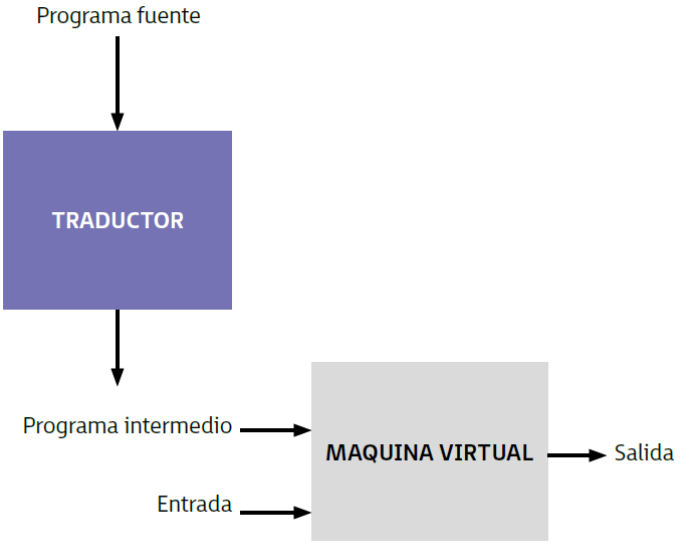
Los compiladores traducen desde un lenguaje fuente a unlenguaje destino. Devolverán errores si el lenguaje fuente está mal escrito y lo ejecutarán si el lenguaje destino es ejecutable por la máquina. Por ejemplo: C, C++, C#, Objetive-C.



- **Lenguajes interpretados**: son otra variante para traducirprogramas de alto nivel. En este caso, nos da la aparienciade ejecutar directamente las instrucciones del programa fuente con las entradas proporcionadas por el usuario. Cuando ejecutamos una instrucción, se debe interpretary traducir al lenguaje máquina.

  El compilador es, de forma general, más rápido que unintérprete al asignar las salidas. Sin embargo, al usar elintérprete evitaremos tener que compilar cada vez que hagamos alguna modificación. Ejemplos de algunos lenguajes son: PHP, JavaScript, Python, Perl, Logo, Ruby, ASP y Basic.

  El lenguaje Java usa tanto la compilación como la interpretación. Un programa fuente en Java puede compilarse primero en un formato intermedio, llamado *bytecodes*, para que luego una máquina virtual lo interprete.



Según el paradigma de programación

El paradigma de programación nos detalla las reglas, los patrones y los estilos de programación que usan los lenguajes. Cada lenguaje puede usar más de un paradigma, el cual resultará más apropiado que otro según el tipo de problema que queramos resolver.

Existen diferentes categorías de lenguaje:

- **Lenguajes imperativos**: al principio, los primeros lenguajes imperativos que se usaron fueron el lenguaje máquina y, más tarde, el lenguaje ensamblador. Ambos lenguajes consisten en una serie de sentencias que establecen cómo debe manipularse la información digital presente en cada memoria o cómo se debe enviar o recibir la información en los dispositivos.

  A través de las estructuras de control podemos establecer el orden en que se ejecutan y modificar el flujo del pro- grama según los resultados de las acciones. Facilitan las operaciones por medio de cambios de estado, siendo este la condición de una memoria de almacenamiento.

  Algunos ejemplos de estos lenguajes son: Basic, Fortran, Algol, Pascal, C, Ada, C++, Java, C#. Casi todos los **lenguajes de desarrollo de software comercial** son imperativos.

  Dentro de esta categoría, podremos englobar:

  - Programación estructurada**.**
  - **Programación modular**.
  - **Programación orientada a objetos** (usa objetos y sus interacciones para crear programas).

- **Lenguajes funcionales**: están basados en el concepto de función y estarán formados por definiciones de funciones junto con sus argumentos.

  Entre sus características destaca que no existe la operación de asignación. Las variables almacenan definiciones a expresiones. El resultado de un cálculo es la entrada del siguiente, y así sucesivamente hasta que se consigue el valor deseado.

  Sería el caso de tipos de lenguaje como Lisp, Scheme, ML, Miranda o Haskell. Apenas se usan para el software comercial.

- **Lenguajes lógicos**: están basados en el concepto de razonamiento, ya sea de tipo deductivo o inductivo. A partir de una base de datos consistente en un conjunto de entidades, propiedades de esas entidades o relaciones entre entidades, el sistema es capaz de hacer **razonamientos**.

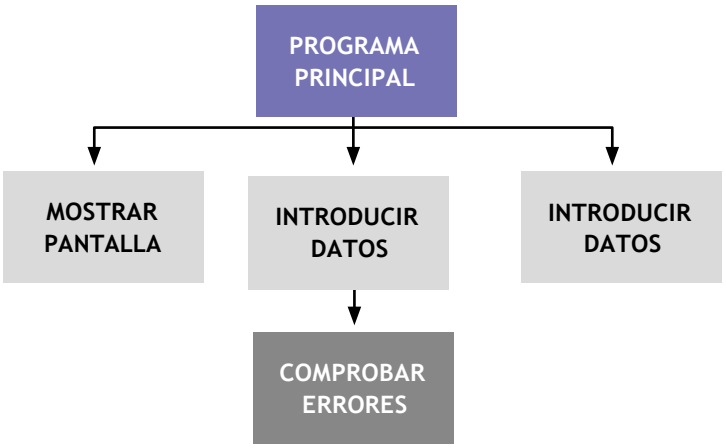
  Los programas escritos en este lenguaje suelen tener forma de una **base de datos**, la cual está formada por declaraciones lógicas, es decir, que son ciertas o falsas, que podremos consultar. La ejecución será en forma de consultas hacia esa base de datos.

  El lenguaje lógico más importante es Prolog, especial- mente preparado para sistemas expertos, demostración de teoremas, consultas de bases de datos relacionales y procesamiento de lenguaje natural.

- **Lenguajes estructurados**: utilizan las tres construcciones lógicas nombradas anteriormente y resultan fáciles de leer. El inconveniente de estos programas estructura-dos es el código, que está centrado en un solo bloque, lo que dificulta el proceso de hallar el problema.

  Cuando hablamos de programación estructurada nos estamos refiriendo a programas creados a través de módulos, es decir, pequeñas partes más manejables que, unidas entre sí, hacen que el programa funcione. Cada uno de los módulos poseen una entrada y una salida y deben estar perfectamente comunicados, aunque cada uno de ellos trabaja de forma independiente.

  A continuación, vemos un programa estructurado en módulos:



  La evolución a esta programación mediante módulos se le denomina **programación modular** y posee las siguientes ventajas:

  – Al dividir el programa en módulos, varios programado-res podrán trabajar a la vez en cada uno de ellos.

  – Estos módulos pueden usarse para otras aplicaciones.

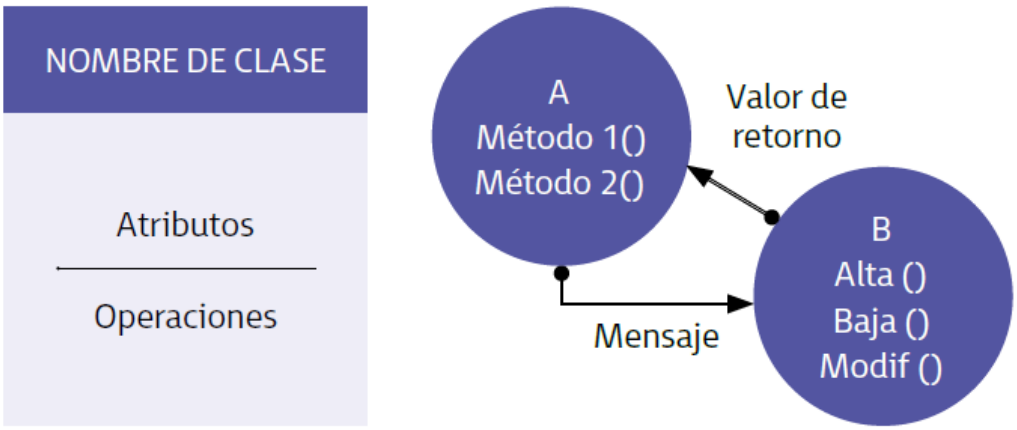
  – Si surge algún problema será más fácil y menos costo- so detectarlo y abordarlo, ya que se puede resolver de forma aislada.

  Algunos ejemplos de estos lenguajes son Pascal, C, Fortran y Modula-2.

- **Lenguajes orientados a objetos**: estos lenguajes estarán definidos por un conjunto de objetos en vez de por módulos, como hemos visto anteriormente.

  Estos objetos están formados por una estructura de da-tos y por una colección de métodos que interpretan esos datos. Los datos que se encuentran dentro de los objetos son sus **atributos**, y las operaciones que se realizan sobre los objetos cambian el valor de uno o más atributos.

La comunicación entre objetos se realiza a través de mensajes, como se plasma en la siguiente figura:



Una clase es una plantilla para la creación de objetos. Alcrear un objeto, se ha de especificar a qué clase pertenece para que el compilador sepa qué características posee.

Entre las ventajas de este tipo de lenguaje hay que des-tacar la facilidad para reutilizar el código, el trabajo enequipo o el mantenimiento del software.

Una desventaja es que el concepto de un programadorpuede ser distinto a otros, por lo que se realiza una división entre objetos distinta.

Los lenguajes orientados a objetos más comunes son C++, Java, Ada, Smalltalk y Ruby, entre otros.

ponte a prueba

¿Qué capacidad (en bits) tiene el registro EAX?

a) 8 bits
b) 16 bits
c) 32 bits
d) El registro EAX no existe

¿Cuál de los siguientes lenguajes no son de alto nivel?

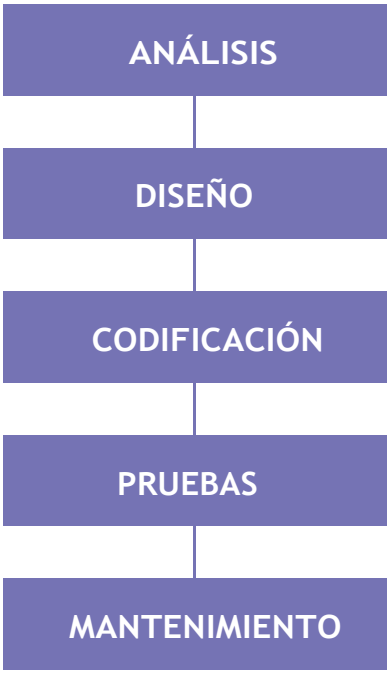
a) Python
b) Java
c) C
d) Ensamblador

1.5. FASES DEL DESARROLLO DE UNA APLICACIÓN: ANÁLISIS, DISEÑO, CODIFICACIÓN, PRUEBAS, DOCUMENTACIÓN, MANTENIMIENTO Y EXPLOTACIÓN

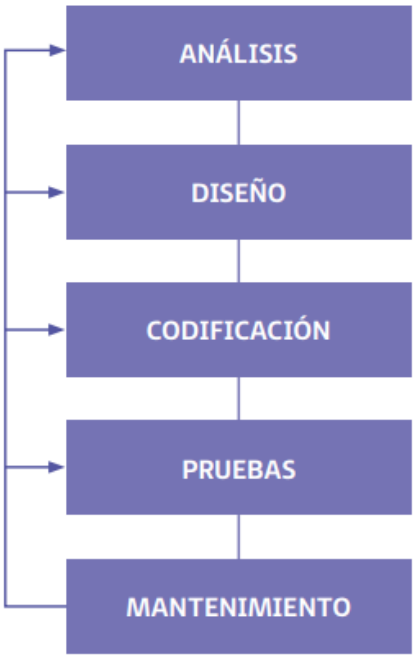
Cuando queramos realizar un proyecto de software, antesdebemos crear un ciclo de vida en el que examinemos lascaracterísticas para elegir un modelo de desarrollo u otro.

Modelo en cascada

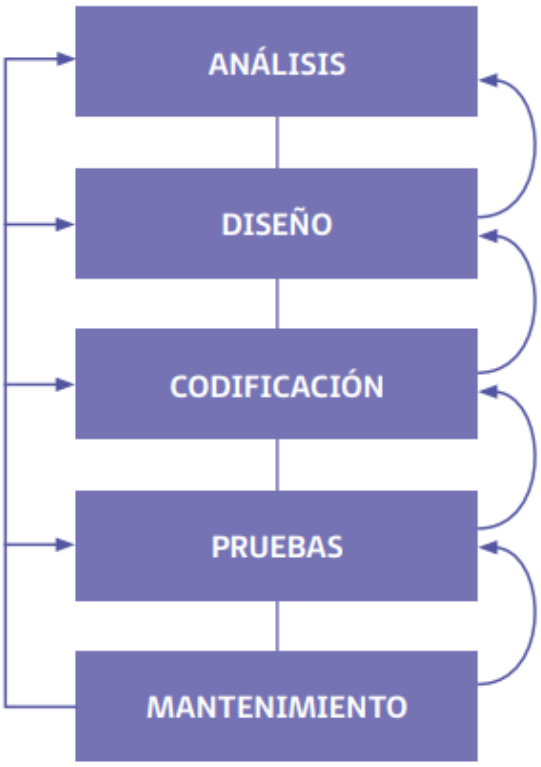
En este modelo, las etapas para el desarrollo de softwaretienen un orden, de tal forma que, para empezar una etapa, es necesario finalizar la etapa anterior. Después de cada etapa se realiza una revisión para comprobar si se puedepasar a la siguiente.



Este modelo permite hacer iteraciones. Por ejemplo, si elcliente requiere una mejora durante la etapa de mantenimiento del producto, esto implica que hay que modificar algo en el diseño, lo cual significa que habrá que hacer cambios en la codificación y se tendrán que realizar de nuevo las pruebas. Es decir, si se tiene que volver a una de las etapas anteriores, hay que recorrer de nuevo el resto delas etapas.



Este modelo tiene distintas variantes, una de la más utiliza-das es el **modelo en cascada con realimentación**, que produce una realimentación entre etapas. Supongamos que en cualquiera de las etapas se detectan fallos (los requisitos han cambiado, han evolucionado, ambigüedades en la definición de estos, etcétera), entonces, será necesario retornar a la etapa anterior para realizar los ajustes pertinentes. A esto se le conoce como realimentación, pudiendo volver de una etapa a la anterior o incluso de varias etapas a otra anterior.



Ventajas	Inconvenientes
Fácil de comprender, planificar yseguir.	La necesidad de tener todos los requisitosdefinidos desde el principio.
La calidad del producto resultante esalta.	Es difícil volver atrás si se cometen errores enuna etapa (es un modelo inflexible)
Los recursos que se necesitan sonmínimos.	El producto no está disponible para su uso hasta que no está completamente terminado.
Se recomienda cuando:	
• El proyecto es similar a alguno que ya se haya realizado con éxito anteriormente.	
• Los requisitos son estables y están bien comprendidos.	
• Los clientes no necesitan versiones intermedias.	

Modelo iterativo incremental

El modelo incremental está basado en varios ciclos en cas- cada realimentados aplicados repetidamente. Este modelo entrega el software en partes pequeñas, pero utilizables, llamadas incrementos o prototipos. En general, cada incremento se construye sobre aquel que ya ha sido entregado.

A continuación, se muestra un diagrama del modelo bajoun esquema temporal donde se observa de forma iterativael modelo en cascada para la obtención de un nuevo incremento mientras progresa el tiempo en el calendario.

Al desarrollar el software de manera incremental, resulta más barato y fácil realizar cambios en el software conforme este se va desarrollando.

Cada incremento del aplicativo incorpora algunas de las funciones que necesita el cliente. Esto significa que el cliente puede evaluar el desarrollo del sistema en una etapa temprana.

Ventajas	Inconvenientes
No necesitan conocer todos los requisitos. Se reduce, por tanto, el coste de adaptar los requerimientos cambiantes del cliente.	Es difícil estimar el esfuerzo y el coste final necesarios.
Permite la entrega temprana al cliente de partes operativas del software.	Se tiene el riesgo de no acabar nunca.
Las entregas facilitan la realimentación de los próximos entregables.	No es recomendable para desarrollo de sistemas de tiempo real, de alto nivel de seguridad, de procesamiento distribuido y/o de alto índice de riesgos.
	La incorporación de muchos cambios hace que el software se vuelva inestable.
Se recomienda cuando:	
• Los requisitos o el diseño no están completamente definidos y es posible que haya grandes cambios.	
• Se están probando o introduciendo nuevas tecnologías.	

Modelo en espiral

Este modelo combina el modelo en cascada con el modelo iterativo de construcción de prototipos. El proceso de desarrollo del software se representa como una espiral donde en cada ciclo se desarrolla una parte de este. Cada ciclo está formado por cuatro fases y, cuando se termina, produce una versión incremental del software con respecto al ciclo anterior. En este aspecto, se parece al modelo iterativo incremental, con la diferencia de que en cada ciclo se tiene en cuenta el análisis de riesgos.

Durante los primeros ciclos, la versión incremental podríaestar compuesta de maquetas en papel o modelos de pantallas (prototipos de interfaz); en el último ciclo, se tendríaun prototipo operacional que implementa algunas funciones del sistema.

Para cada ciclo, los desarrolladores siguenestas fases:

1. **Determinar objetivos**: cada ciclo de la espiral comienza con la identificación de los objetivos, las alternativaspara alcanzar los objetivos y las restricciones impuestas a la aplicación de las alternativas.
2. **Análisis del riesgo**: a continuación, hay que evaluar las alternativas en relación con los objetivos y limitaciones. Con frecuencia, en este proceso se identifican los riesgos involucrados y la manera de resolverlos (requisitos no comprendidos, mal diseño, errores en la implementación, etcétera). Se aconseja realizar un análisis minucioso para reducir los riesgos. Utiliza la construcción de prototipos como mecanismo de reducción de riesgos.

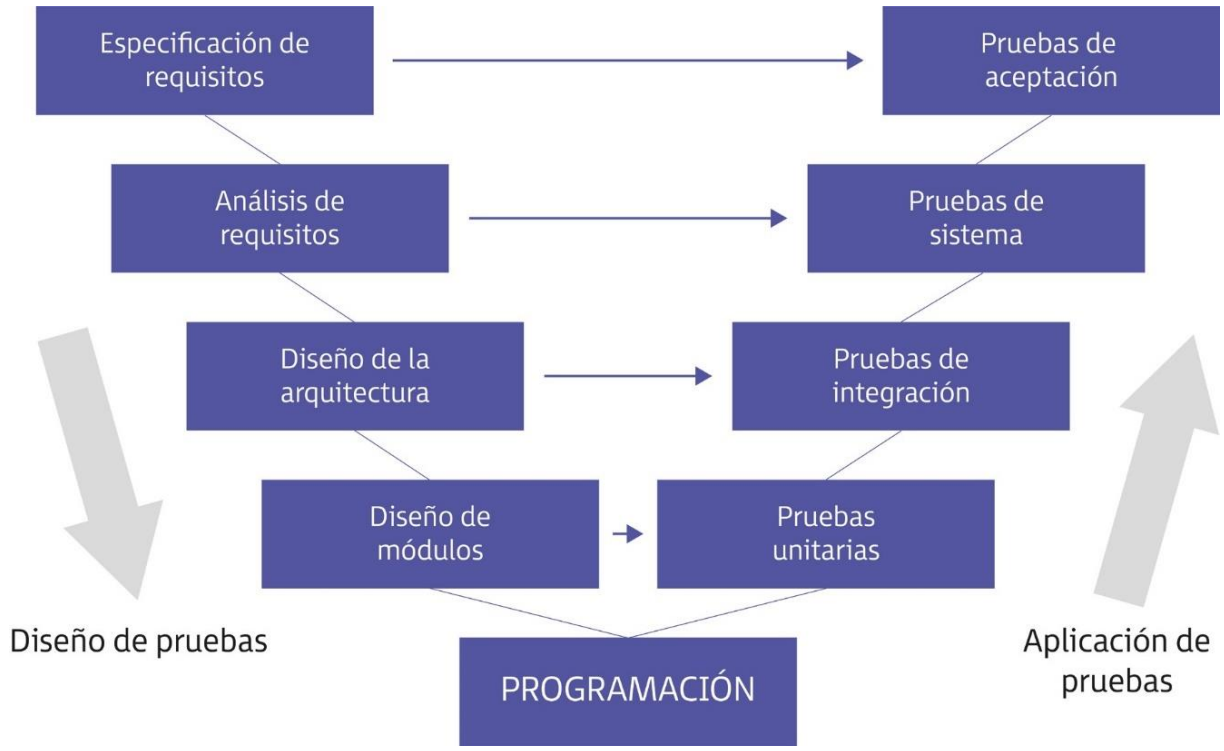
3. **Desarrollar y probar**: desarrollar la solución al problema en este ciclo y verificar que es aceptable. Por ejemplo, si existen riesgos en una interfaz de usuario, podríamos ir creando prototipos hasta conseguir un enfoque deseable.
4. **Planificación**: revisar y evaluar todo lo que se ha hechoy, con ello, decidir si se continúa; entonces hay que planificar las fases del ciclo siguiente.

Ventajas	Inconvenientes
No requiere una definición completa de los requisitos para empezar a funcionar.	Es difícil evaluar los riesgos.
Análisis del riesgo en todas las etapas.	El costo del proyecto aumenta a medida que la espiral pasa por sucesivas itera-ciones.
Reduce riesgos del proyecto.	El éxito del proyecto depende en gran medida de la fase de análisis de riesgos.
Aumento de la productividad.	Es difícil hacer ver al cliente que este enfoque evolutivo es controlable.
Se recomienda para:	
• Proyectos de gran tamaño y que necesitan constantes cambios.	
• Proyectos donde sea importante el factor riesgo.	

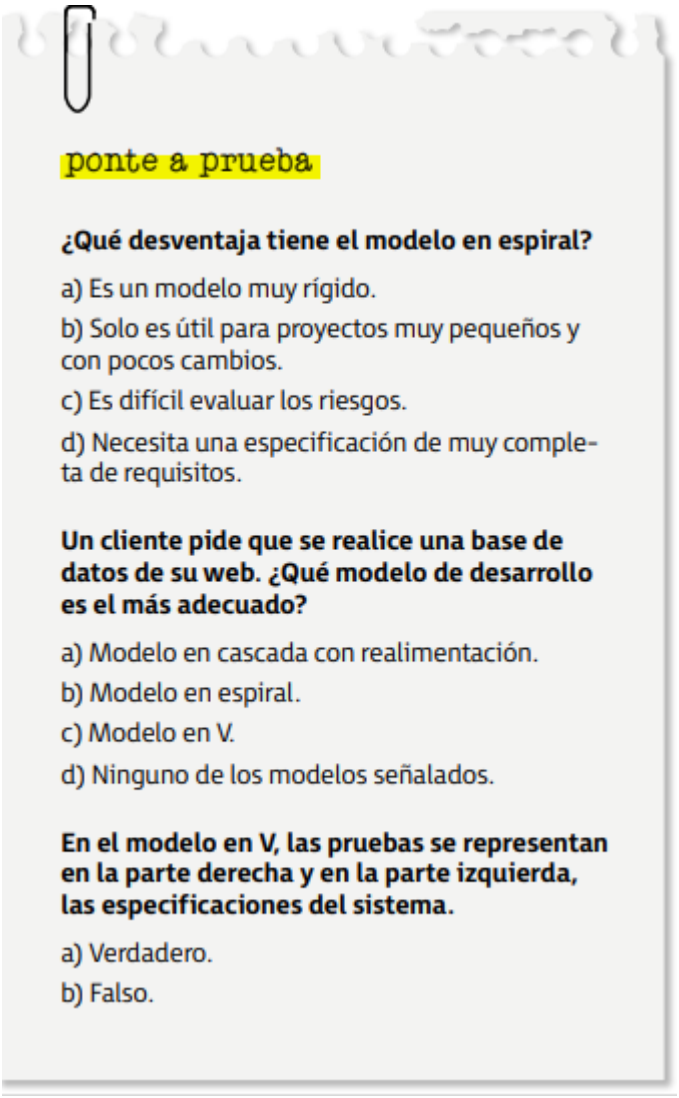
Modelo en V

Es un proceso que representa la secuencia de pasos en eldesarrollo del ciclo de vida de un proyecto. En él se des- criben las actividades y resultados que deben producirse durante el desarrollo del producto. El **lado izquierdo** de la V representa la descomposición de las necesidades y la creación de las especificaciones del sistema. El **lado derecho** de la V representa la integración de las piezas y suverificación. Es muy similar al modelo en cascada, ya quees muy rígido y contiene una gran cantidad de iteraciones.

Indistintamente del modelo que escojamos, deberemos seguir una serie de etapas:



Ventajas	Inconvenientes
Facilita la localización de fallos.	Las pruebas pueden llegar a ser costosas.
Modelo muy sencillo.	El cliente debe tener paciencia hasta el producto final.
El cliente está involucrado en las pruebas.	Pueden no estar bien definidos los requisitos del cliente.
Se recomienda para:	
• Ser aplicado en sistemas sencillos pero de confiabilidad alta (transacciones en bases de datos).	



ponte a prueba

¿Qué desventaja tiene el modelo en espiral?

a) Es un modelo muy rígido.

b) Solo es útil para proyectos muy pequeños y con pocos cambios.

c) Es difícil evaluar los riesgos.

d) Necesita una especificación de muy completa de requisitos.

Un cliente pide que se realice una base de datos de su web. ¿Qué modelo de desarrollo es el más adecuado?

a) Modelo en cascada con realimentación.

b) Modelo en espiral.

c) Modelo en V.

d) Ninguno de los modelos señalados.

En el modelo en V, las pruebas se representan en la parte derecha y en la parte izquierda, las especificaciones del sistema.

a) Verdadero.

b) Falso.

1.5.1. ANÁLISIS

Al realizar un proyecto, la parte más importante es entender qué se quiere realizar y analizar las posibles alternativas y soluciones. Por ello, es **fundamental** analizar los requisitos que el cliente ha solicitado.

Por tanto, el análisis consiste en la especificación de las características operativas del software, indica cuál es la interfaz que ha de desarrollarse y marca las restricciones de este.

Aunque pueda parecerlo, no es una tarea fácil, ya que a menudo el cliente es poco claro y durante el desarrollo pueden surgir nuevos requerimientos. Habrá que tener una buena comunicación entre el cliente y los desarrolladores para evitar futuros problemas. Para la obtención de estosrequisitos, se usarán distintas técnicas:

- **Entrevistas**: técnica tradicional en la que hablamos con el cliente. En un ambiente más relajado (por ejemplo, compartiendo un café fuera de la oficina), el cliente tiende a expresarse con más claridad.
- **Desarrollo conjunto de aplicaciones (JAD, *join appli- cation design*)**: entrevista de dinámica de grupo (talleres) en la que cada integrante aporta su conocimiento (usuarios, administradores, desarrolladores, analistas, etcétera).
- **Planificación conjunta de requisitos (JRP, *joint re- queriments planning*)**: el objetivo de estas sesiones es involucrar a la dirección para obtener mejores resultados en el menor tiempo posible. La diferencia con el JAD es la participación del nivel más alto de la organización en la visión general del negocio (director, promotor, especia-listas de alto nivel, entre otros).
- ***Brainstorming*:** reuniones en las que se intentan crear ideas desde distintos puntos de vista (tormenta de ideas). Idónea para el comienzo del proyecto.
- **Prototipos**: versión inicial del sistema en el que se puede ver el problema y sus posibles soluciones. Se puede des- echar o usar para añadir más cosas.
- **Casos de uso**: "Los casos de uso simplemente son una ayuda para definir lo que existe fuera del sistema (acto-res) y lo que debe realizar el sistema (casos de uso)" (Ivar Jacobson, 2005). Este tipo de diagramas son fundamentales en la ingeniería de requisitos.

Podemos clasificar los requisitos en:

- **Requisitos funcionales**: nos describen al detalle la función que realiza el sistema, la reacción ante determinadas entradas y cómo se comporta en distintas situaciones.
- **Requisitos no funcionales:** son limitaciones sobre la funcionalidad que ofrece el sistema. Estos requerimientos se refieren a las propiedades emergentes del sistema, como la fiabilidad o la capacidad de almacena-miento. Incluyen restricciones impuestas por el estándar del aplicativo.

A la hora de representar estos requisitos, podemos hacerlocon distintos modelos:

- **Modelo basado en el escenario**: se basa en el punto devista de los actores del sistema. Lo podemos representar con los casos de uso o las historias de usuario.

UC-NUM	LOGIN	
Versión	1.0 (dd/mm/yyyy)	
Autores		
Dependencias	· [OBJ-0001] Objetivo Principal · [OBJ-0009] Gestión de usuarios	
Descripción	El sistema deberá comportarse tal como se describe en el siguiente caso de uso cuando *el usuario invitado entra en la aplicación*	
Precondición	El usuario está registrado	
Secuencia normal	**Paso**	**Acción**
	1	El sistema *pide nickname y password*
	2	El actor Invitado (ACT-0001) *introduce los datos y selecciona "entrar"*
	3	El sistema *comprueba en la base de datos que los datos son válidos y el caso de uso finaliza*
Postcondición	El usuario está dentro de la aplicación	
Excepciones	**Paso**	**Acción**
	3	Si *los datos no son válidos*, el sistema *no permite el acceso*, a continuación este caso de uso *queda sin efecto*

Ejemplo de caso de uso: registro de usuario.

- **Modelo de datos**: muestra el entorno y la información del problema. Lo podemos representar con el diccionario de datos.
  - **Diccionario de datos (DD)**: descripción detallada de los datos utilizados por el sistema que gráficamente están representados por los flujos de datos y almacenespresentes sobre el conjunto de DFD.

    En esta primera fase de análisis, es fundamental quetodo lo que se realice quede plasmado en el documento Especificación de Requisitos de Software (ERS). Debe ser un documento completo, sin ambigüedades, sencillo de usar a la hora de verificarlo, modificarlo o de identificar el origen y las consecuencias de los requisitos. Cabe destacar que nos servirá para la siguiente faseen el desarrollo.

EJEMPLO

Denominación = tratamiento formal + nombre + apellido

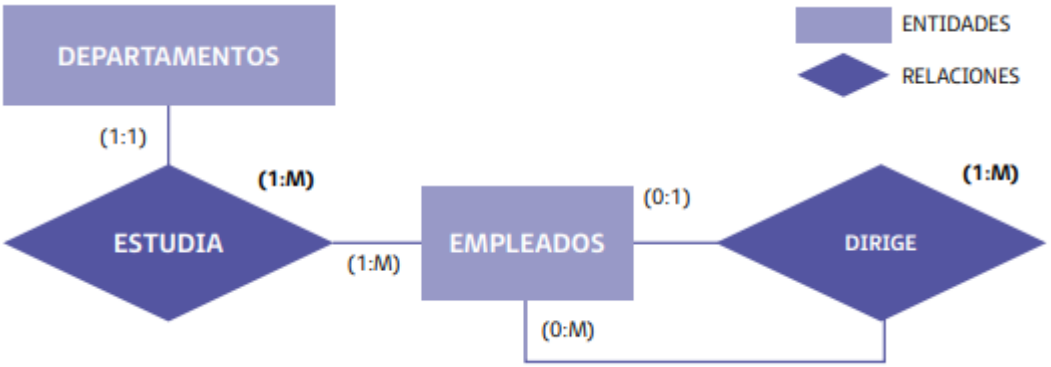
Tratamiento formal = [Don|Doña|Sr.|Sra.]

Nombre = {carácter}

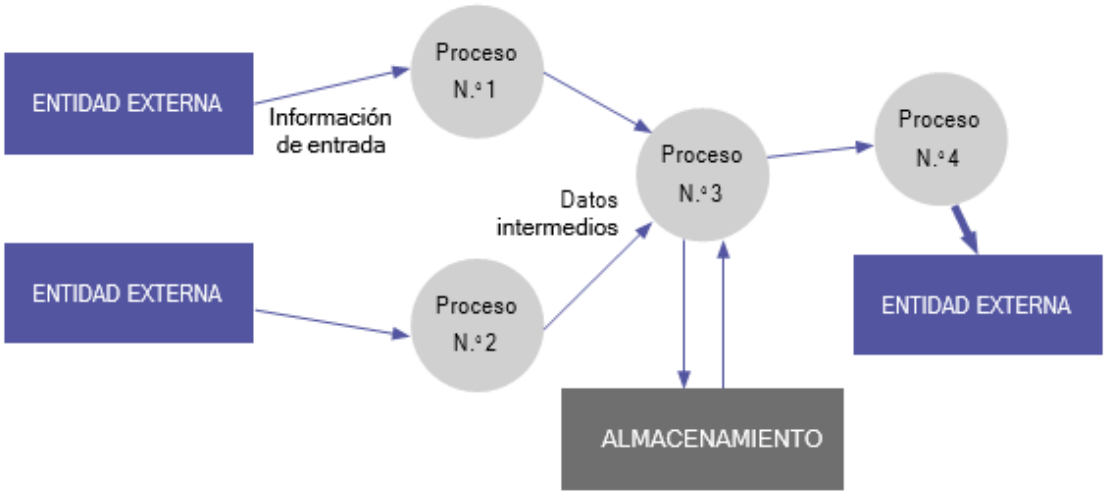
Apellido = {carácter}

DNI = {carácter}

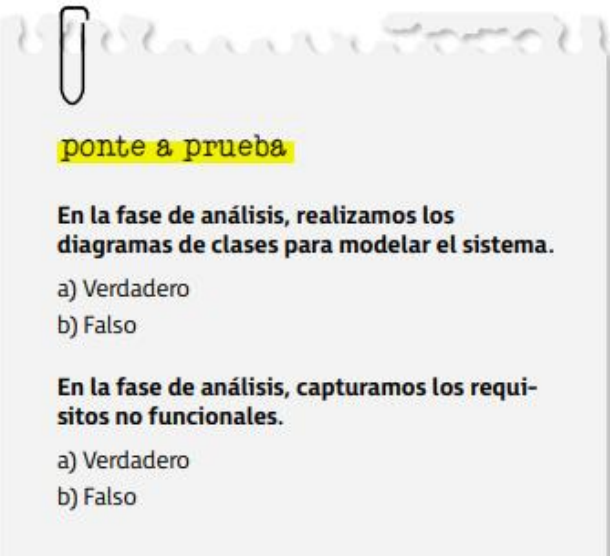
- **Diagrama entidad/relación (DER):** se usa para representar datos y sus relaciones. Representa los datos que se introducen, almacenan y transforman en el sistema.



- **Modelos orientados al flujo**: representan los elementos funcionales del sistema de tal forma que reflejan cómo se transforman los datos a medida que avanzan dentro del aplicativo.
  - **Diagramas de flujo de datos (DFD):** nos va a representar el flujo de datos entre procesos, entidades externas (componentes que no son del sistema) y almacenes del sistema (datos desde el punto de vista estático):
    - **Procesos** → burbujas ovaladas o circulares.
    - **Entidades externas** → rectángulos.
    - **Almacenes** → dos líneas horizontales y paralelas.
    - **Flujo de datos** → flechas.



- **Diagramas de flujo de control (DFC)**: similar al anterior, pero en vez de flujo de datos muestra el flujo de control. Un gran número de aplicaciones son causadas por eventos y no por datos, producen información de control y procesan información con mucha atención altiempo y al rendimiento.
- **Diagramas de transición de estados (DTE):** representa el comportamiento del sistema dependiente del tiempo. Se aplican, sobre todo, en sistemas en tiempo real.



ponte a prueba

En la fase de análisis, realizamos los diagramas de clases para modelar el sistema.

a) Verdadero
b) Falso

En la fase de análisis, capturamos los requisitos no funcionales.

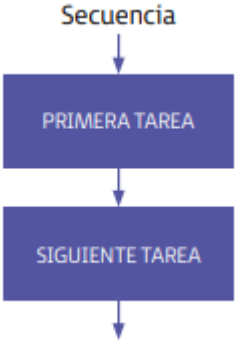
a) Verdadero
b) Falso

1.5.2. DISEÑO

Una vez que hemos identificado los requerimientos necesarios, ahora tendremos que componer la forma para solucionar el problema. Traduciremos los requisitos funcionales y los no funcionales en una representación de software. “Las preguntas acerca de si el diseño es necesarioo digno de pagarse están más allá de la discusión: el diseñoes inevitable. La alternativa al buen diseño es el mal diseño, no la falta de diseño” (Douglas Martin,1994).

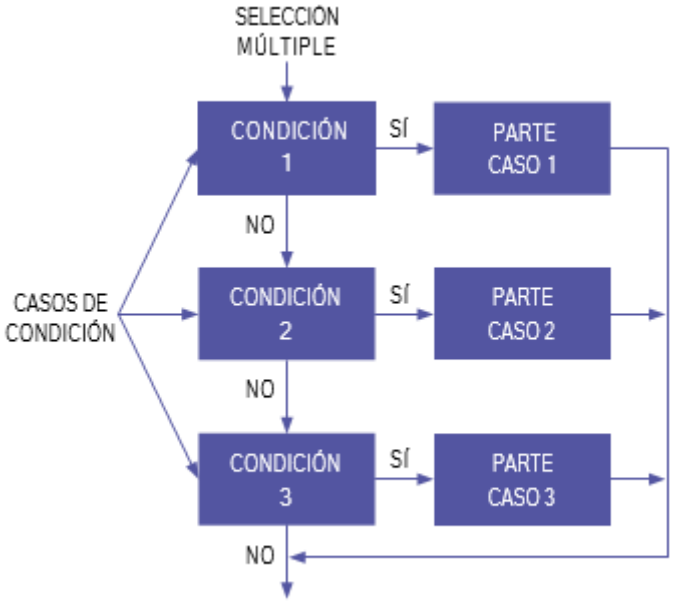
Existen dos tipos de diseño:

- **Diseño estructurado**: basado en el flujo de datos a través del sistema. Dentro de la programación estructurada, existen una serie de construcciones (bucles) que son fundamentales para el diseño a nivel de componentes. Estas construcciones son: secuencial, condicional y repetitiva:
  - **Construcción secuencial**: se refiere a la ejecución sentencia por sentencia de un código fuente, es decir, hasta que no termine la ejecución de una sentencia, no pasará a la siguiente.



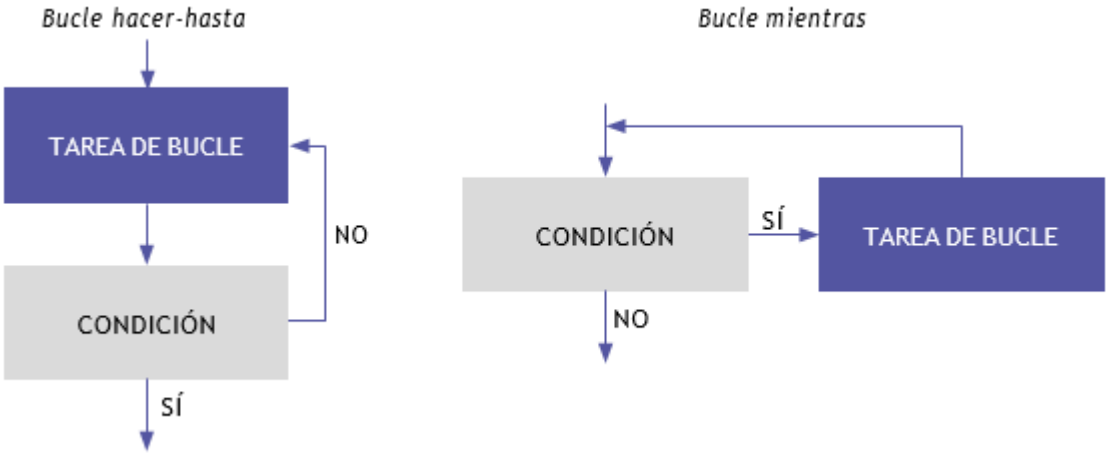
– **Construcción condicional**: selecciona un proceso u otro según una condición lógica (rombo). Si se cumple,se realiza la parte SI, si no, se realiza la parte NO.

  La selección múltiple es una extensión de la estructura*Si entonces si-no;* un parámetro se prueba por continuas decisiones hasta que alguna es verdadera yejecuta según ese camino.



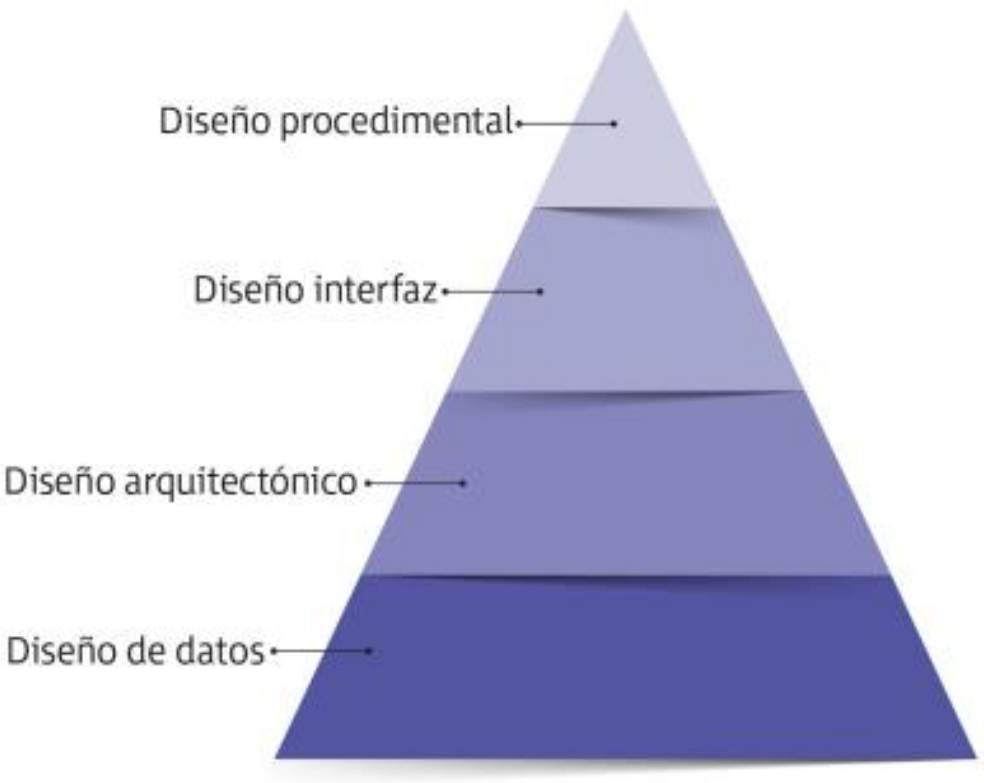
– **Construcción repetitiva:**

  - ***Hacer hasta****:* se ejecuta una primera vez la tarea y alfinalizarla se comprueba la condición. Si esta no secumple, se realiza de nuevo hasta que se cumpla la condición y finalice la tarea. Se realiza al menos una vez.
  - ***Mientras****:* en este caso, se comprueba antes la condición y, después, se realiza la tarea continuamente siempre que se cumpla la condición. Se finaliza cuan-do la condición no se cumpla.



El diseño estructurado podemos dividirlo en:

- **Diseño de datos**. Transforma la información relativa al análisis en estructuras de datos para su posteriorimplementación mediante diferentes lenguajes de pro gramación. Por ejemplo, esquemas lógicos de datos.
- **Diseño arquitectónico**. Es un esquema similar al planode una casa. Se centra en la representación de la estructura de los componentes del software, sus propiedades y sus interacciones. Este diseño se basa en la información del entorno del aplicativo que realizar y de los modeladosde requerimientos, como DFD o DFC.
- **Diseño de la interfaz**. Detalla la comunicación que rea- liza el software consigo mismo, los sistemas que operancon él y los usuarios. El resultado es la creación de formatos de pantalla. Los elementos importantes son la interfaz de usuario (UI) e interfaces externas con otrossistemas, dispositivos o redes.
- **Diseño a nivel de componentes (diseño procedimental).**Es similar a los planos de cada habitación de una casa. Convierte elementos estructurales de la arquitectura delsoftware en una descripción procedimental de los componentes del software. El resultado será el diseño de cada componente con el detalle necesario para que sirva deguía en la generación del código fuente. Se realiza mediante diagramas de flujo, diagramas de cajas, tablas dedecisión, pseudocódigo, etcétera.

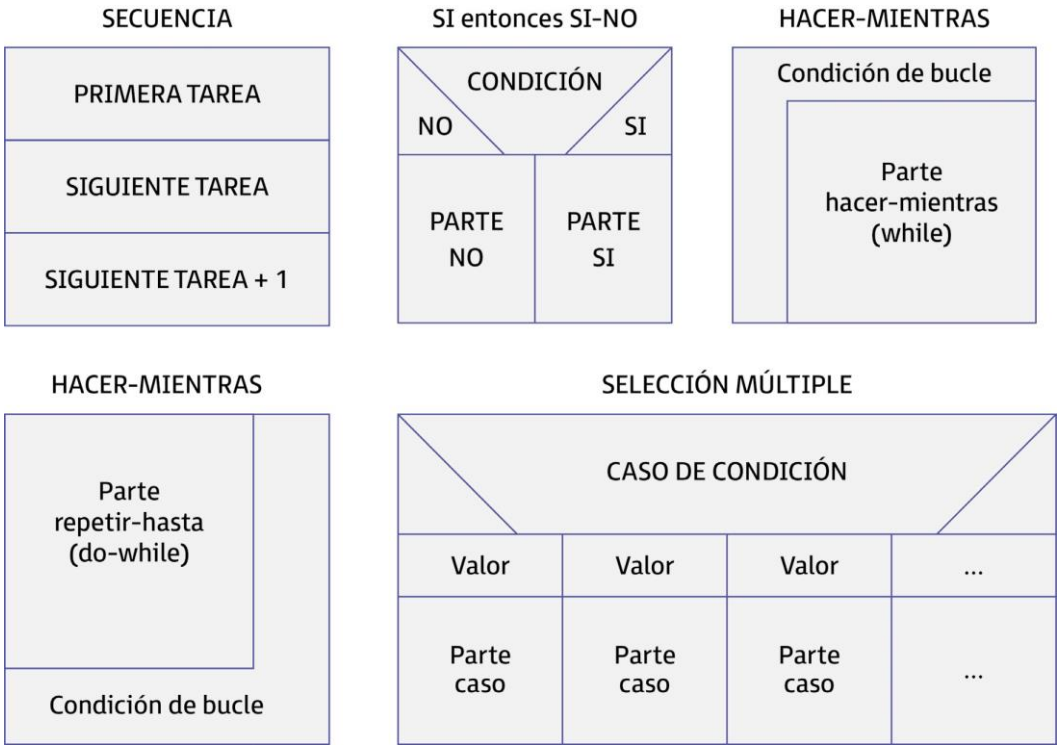


Notaciones gráficas para el diseño procedimental

Al representar el diseño, usaremos algunas herramientasbásicas, como los diagramas de flujo, los diagramas de cajas, las tablas de decisión o el pseudocódigo.

- **Diagramas de flujo**: herramienta muy usada para el di- seño procedimental, donde:
  - **Caja**: paso del proceso.
  - **Rombo**: condición lógica.
  - **Flechas**: flujo de control.

- **Diagrama de cajas**: otra representación de nuestro diseño estructurado son los diagramas de cajas, en los que:
  - **Secuencia**: varias cajas seguidas.
  - **Condicional**: una caja para la parte SI y otra para la par-te NO. Encima indicamos la condición.
  - **Repetitiva**: proceso que se repite, se encierra en unacaja que se sitúa dentro de otra en la que indicamos lacondición del bucle en la parte superior *(while)* o inferior *(dowhile)*.
  - **Selección múltiple**: la parte superior indica el caso de condición, mientras que en la parte inferior se definen tantas columnas como valores se quieran



- **Tablas de decisión**: nos permiten representar en una tabla las condiciones y las acciones que se llevarán a cabo al combinar esas condiciones. Proporcionan una notación que traduce las acciones y condiciones a una forma estructurada.

  Se dividirá en dos:

  - **Condiciones**: son las combinaciones posibles de nuestro sistema.
  - **Acciones**: sentencias que se ejecutan cuando se cumplen determinadas condiciones.

	REGLAS						
CONDICIONES	**1**	**2**	**3**	**4**			**n**
Condición N.º 1	✓			✓			
Condición N.º 2		✓	✓				
Condición N.º 3	✓			✓			
ACCIONES							
Acción N.º 1	✓			✓			
Acción N.º 2	✓	✓					
Acción N.º 3		✓	✓	✓			
Acción N.º 4			✓	✓			

En las diferentes columnas podemos implementar las reglas que se deben cumplir para una determinada acción.

Al construir la tabla, seguimos los siguientes pasos:

1 Crear una lista con todas las acciones y todas las condiciones.

2 Relacionar los conjuntos con las acciones específicas, eliminando las combinaciones imposibles.

3 Determinar las reglas indicando la acción o accionesque ocurren para un conjunto de condiciones.

- **Pseudocódigo:** esta herramienta utiliza un texto descriptivo para crear el diseño de un algoritmo. Se asemeja al lenguaje de programación, ya que mezcla el lenguaje natural con la sintaxis de la programación estructurada, incluyendo palabras clave.

No existe un estándar definido y, al no ser un lenguaje deprogramación, no puede compilarse. La representaciónen pseudocódigo de las estructuras básicas de la programación estructurada es:

Secuencial	Instrucción 1 Instrucción 2 … Instrucción n
Condicional	Si <Condición> Entonces <Instrucciones> Si no <Instrucciones> Fin Si
Condicional múltiple	Según sea <Variable> Hacer Caso valor 1: <Instrucciones >Caso valor 2: <Instrucciones >Caso valor 3: <Instrucciones >Otro caso: <Instrucciones> Fin Según
Hacer mientras	Hacer <Instrucciones >Mientras <Condición>
Mientras	Mientras <Condición> Hacer <Instrucciones >Fin Mientras

A continuación, vemos un ejemplo de pseudocódigo:

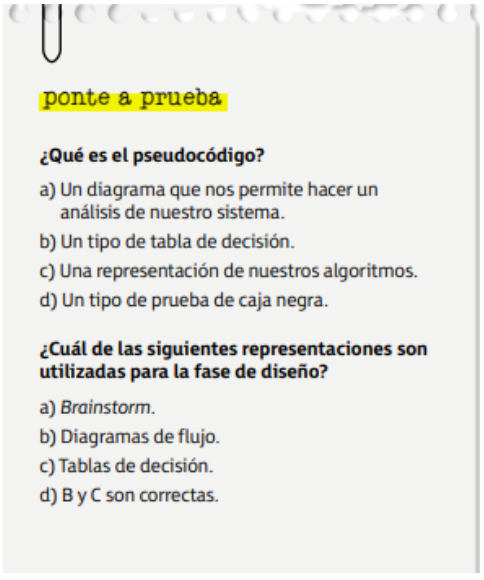
```
Inicio
    Abrir Archivo
    Leer Datos del Archivo
    Mientras no sea Fin de Archivo Hacer
        Procesar Datos Leído
        Leer Registro del Archivo
    Fin mientras
    Cerrar Archivo
Fin
```

- **Diseño orientado a objetos (DOO)**: conforme el diseñoevoluciona, deberemos definir un conjunto de clases paraafinar los detalles de nuestro sistema, lo que nos permitirá implementar una infraestructura que apoye nuestrasolución del producto.

  Podemos especificar cinco clases de diseño:

  - Clases de interfaces: definimos todas las interacciones entre el usuario y la máquina.
  - Clases de domino de negocio: identificamos las clasesy los métodos que se necesitan para implementar elementos del dominio.
  - Clases de proceso: implementación a bajo nivel parala gestión de las clases de domino de negocio.
  - Clases de persistencia: definimos el almacenamiento de los datos (por ejemplo, una base de datos). Estasclases se mantendrán más allá de la ejecución de undeterminado software.
  - Clases de sistema: se definen funciones que permitenal sistema comunicarse con el exterior.

  En el DOO utilizamos un UML (lenguaje de modelado unificado). Es un lenguaje de modelado basado en diagramas para expresar modelos anteriores.



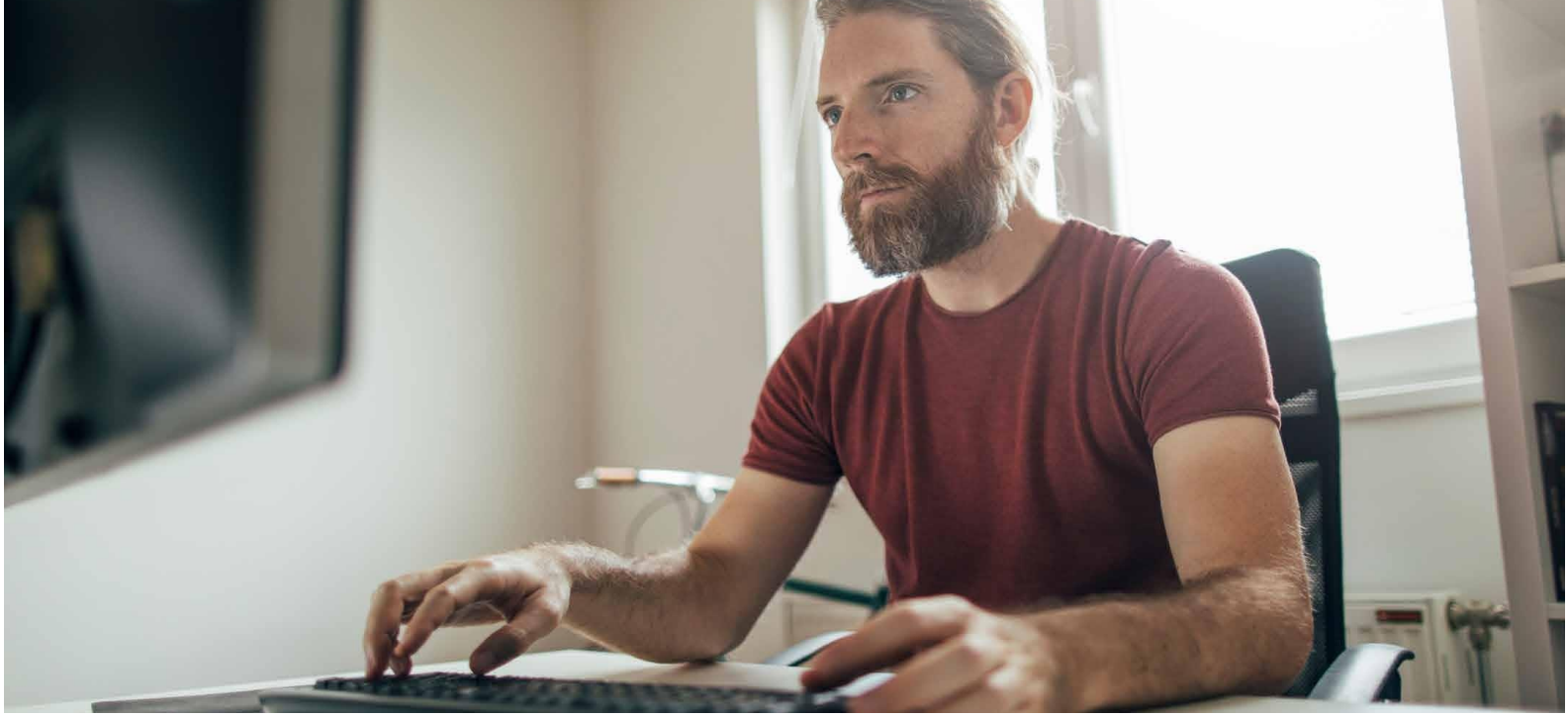
ponte a prueba

¿Qué es el pseudocódigo?

a) Un diagrama que nos permite hacer un análisis de nuestro sistema.
b) Un tipo de tabla de decisión.
c) Una representación de nuestros algoritmos.
d) Un tipo de prueba de caja negra.

¿Cuál de las siguientes representaciones son utilizadas para la fase de diseño?

a) *Brainstorm*.
b) Diagramas de flujo.
c) Tablas de decisión.
d) B y C son correctas.



1.5.3. CODIFICACIÓN

La tercera fase, una vez realizado el diseño, consistirá en elproceso de codificación. Aquí el programador se encargade recibir los datos del diseño y transformarlo en lenguaje de programación. A estas instrucciones las llamaremos **código fuente**.

En cualquier proyecto en que se trabaje con un grupo depersonas, habrá que tener unas normas de codificación yestilo que sean **sencillas, claras y homogéneas**, las cuales nos facilitará la corrección en caso de que sea otra personala que lo ha realizado.

A continuación, veremos algunas series de normas en código **Java**:

- **Nombre de ficheros**: los archivos de código fuente tendrán como extensión .java, y los archivos compilados.class.
- **Organización de ficheros**: cada archivo deberá tener unaclase pública y podrá tener otras clases privadas e inter- faces que irán definidas después de la pública y estaránasociadas a esta. El archivo se dividirá en varias secciones:
  - **Comentarios**: cada archivo debe empezar con un comentario en el que se indique el nombre de la clase, lainformación de la versión, la fecha y el aviso de derechos de autor.
  - **Sentencias de tipo *package* e *import*:** se sitúan después de los comentarios en este orden: primero la sentencia *package* y después la de *import*.
  - **Declaraciones de clases e interfaces**: consta de las siguientes partes:
    - Comentario de documentación (/**…*/) acerca de laclases o interfaz.
    - Sentencia tipo *class* o interfaz.
    - Comentario de la implementación (/*…*/) de la claseo interfaz.
    - Variables estáticas, en este orden: públicas, protegidas y privadas.
    - Variables de instancia en este orden: públicas, protegidas y privadas.
    - Constructores.
    - Métodos.

- **Indentación:**
  - Se usarán cuatro espacios (como recomendación) como unidad de indentación.
  - Longitud de líneas de código no superior a 80 líneas.
  - Longitud de líneas de comentarios no superior a 70 líneas.
  - Si una expresión no cabe en una sola línea, se deberá romper antes de una coma o un operador y se alineará al principio de la anterior.
- **Comentarios**: contendrán solo información que sea relevante para la lectura y comprensión del programa. Habrá dos tipos: de documentación y de implementación.

  Los primeros describen la especificación del código como las clases Java, interfaces, constructores, métodos y campos. Se situarán antes de la declaración. La herramienta **Javadoc** genera páginas HTML partiendo de este tipo de comentarios. Un ejemplo sería:

```
/**
* Esta clase Prueba nos proporciona…
*/
public class Prueba (…)
```

  Los comentarios de implementación sirven para hacer algún comentario sobre la aplicación en particular. Pueden ser de tres tipos:

  – De bloque:

```
/*
*/ Esto es un comentario de bloque
*/
```

  – De línea:

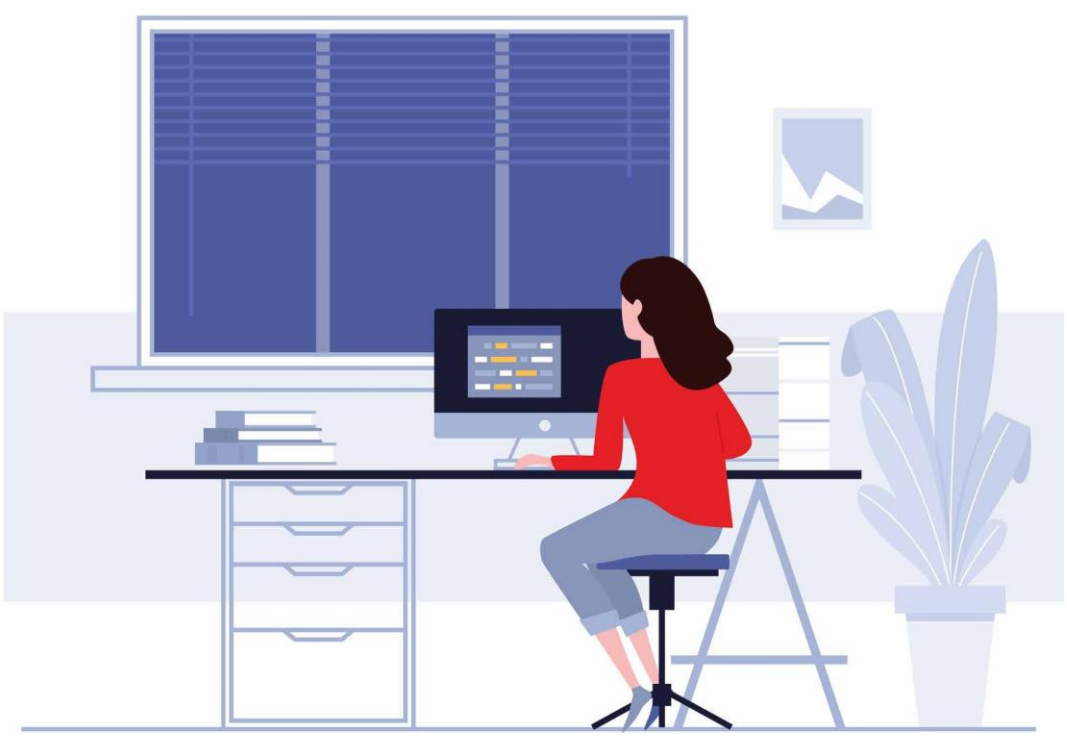
```
/* Comentario de línea */
```

  – Corto:

```
// Comentario corto
```

- **Declaraciones:**
  - Declarar una variable por línea.
  - Inicializar una variable local al comienzo, donde se declara, y situarla al comienzo del bloque.

- En clases o interfaces:
  - No poner espacios en blanco entre el nombre del mé-todo y el paréntesis "(".
  - Llave de apertura "{", situarla en la misma línea que elnombre del método.
  - Llave de cierre "}", situarla en una línea aparte y enla misma columna que el inicio del método. Exceptocuando esté vacío.
  - Métodos separados por una línea en blanco.



- **Sentencias:**
  - Cada línea contendrá una sentencia.
  - Si es un bloque, debe estar sangrado con respecto a loanterior y entre llaves, aunque solo tenga una sentencia.
  - Sentencias *if-else, if else-if else*. Nos definen bloques y tendrán todos los mismos niveles de sangrado.
  - Bucles. Tendrán las normas anteriores y, si están vacío,no irán entre llaves.
  - Sentencias *return* no irán entre paréntesis.
- **Separaciones**: hacen más legible el código. Se utilizarán(como recomendación):
  - **Dos líneas en blanco** entre definiciones de clases e interfaces.
  - **Una línea en blanco** entre métodos, definición de variables locales y la primera instrucción, antes de un comentario, entre secciones lógicas dentro de un método.
  - **Un carácter en blanco** entre una palabra y un paréntesis, después de una coma, de operadores binarios menos el punto, de expresiones del *for* y entre un *cast* y la variable.

- **Nombres**: los nombres de las variables, métodos, clases,etcétera, hacen que los programas sean más fáciles a lahora de leerlos. Las normas que hay que seguir para asignar nombres son:
  - **Paquetes**: se escriben en minúscula. Se podrán utilizarpuntos para algún tipo de organización jerárquica, por ejemplo, java.io.
  - **Clases e interfaces**: deben ser sustantivas o descriptivas, según lo que estemos creando. Si están compuestaspor varias palabras, la primera letra de cada palabra iráen mayúscula.
  - **Métodos**: se usarán verbos en infinitivo. Si están formados por varias palabras, el verbo estará en minúsculay la siguiente palabra empezará con mayúscula.
  - **Variables**: deben ser cortas y significativas. Si están formadas por varias palabras, la primera debe ir en minúscula.
  - **Constantes**: el nombre debe ser descriptivo. Será totalmente escrito en mayúscula y, si son varias palabras,separadas por un carácter de subrayado.

Vamos a ver un ejemplo sencillo en Java:

1. Creamos la clase *Persona* con sus atributos privados ysu instancia.

```
package Ejemplo;

/**
 * Clase utulizada para crear nuevas personas
 * Pueden contener una edad y un nombre
 * @autor Martin Rivero
 * @version 1.0
 */
public class Persona {
    /**
     * Variable privada para guardfar el nombre de la Persona
     */
    private String nombre;
    /**
     * Variable privada para guardar la edad de la Persona
     */
    private int edad;

    /**
     * Instancia una nueva persona sin nombre y sin edad.
     */
    public Persona() {
    }

    /**
     * Instancia una nueva PErsona.
     *
     * @param nombre el nombre
     * @param edad la edad
     */
     public Persona(String nombre, int edad) {
         this.nombre = nombre;
         this.edad = edad;
     }
```

2. Posteriormente, creamos los *getters* y *setters.*

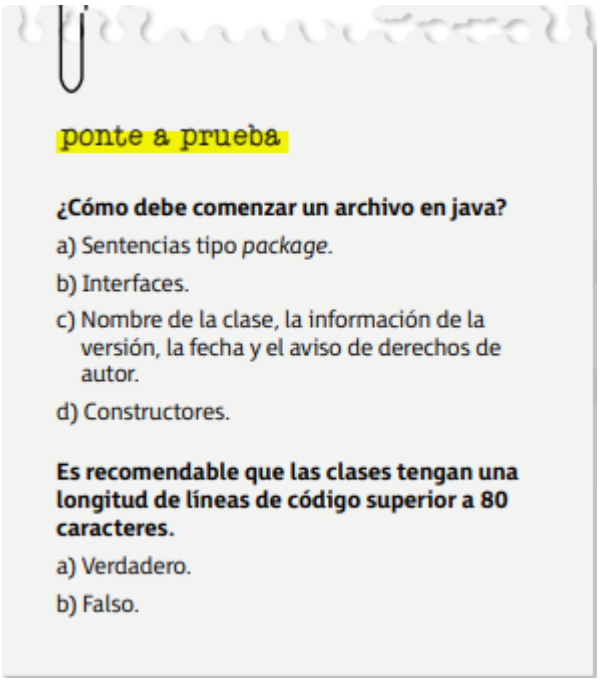
```
/**
 * Recupera el nombre.
 *
 * @return nombre
 */
public String getNombre() {
    return nombre;
}


/**
 * Establece el nombre.
 *
 * @param nombre Nombre de la Persona
 */
public void setNombre(String nombre) {
    this.nombre = nombre;
}


/**
 * Recupera la edad.
 *
 * @return edad
 */
public int getEdad() {
    return edad;
}


/**
```

3. Por último, en el código, creamos un método de tipo *string* para recuperar el nombre y la edad de la persona.

```
/**
 * Recupera la Persona en formato String..
 *
 * @return string de Persona
 */
@Override
public String toString() {
    return "persona{" +
            "nombre='" + nombre + '\'' +
            ", edad=" + edad +
```

Cuando hemos terminado de escribir el código, lo tendremos que traducir al lenguaje máquina a través de **compiladores** o **intérpretes**. El resultado será el códigoobjeto, aunque este no será todavía ejecutable hasta que lo enlacemos con las librerías para obtener así el **código ejecutable**. Una vez obtenido este código, tendremos que comprobar el programa para ver si cumple con nuestrasespecificaciones en el diseño.

Junto con el desarrollo del código, deberemos escribir **manuales** técnicos y de referencia, así como la parte inicial delmanual de usuario. Estas partes serán esenciales para la etapa de prueba y mantenimiento, así como para la entregadel producto.



ponte a prueba

¿Cómo debe comenzar un archivo en java?

a) Sentencias tipo *package*.

b) Interfaces.

c) Nombre de la clase, la información de la versión, la fecha y el aviso de derechos de autor.

d) Constructores.

Es recomendable que las clases tengan una longitud de líneas de código superior a 80 caracteres.

a) Verdadero.

b) Falso.

1.5.4. PRUEBAS

Al iniciar la etapa de pruebas, ya contaremos con el software, por lo que trataremos de encontrar errores en lacodificación, en la especificación o en el diseño. Duranteesta fase, se realizan las siguientes tareas:

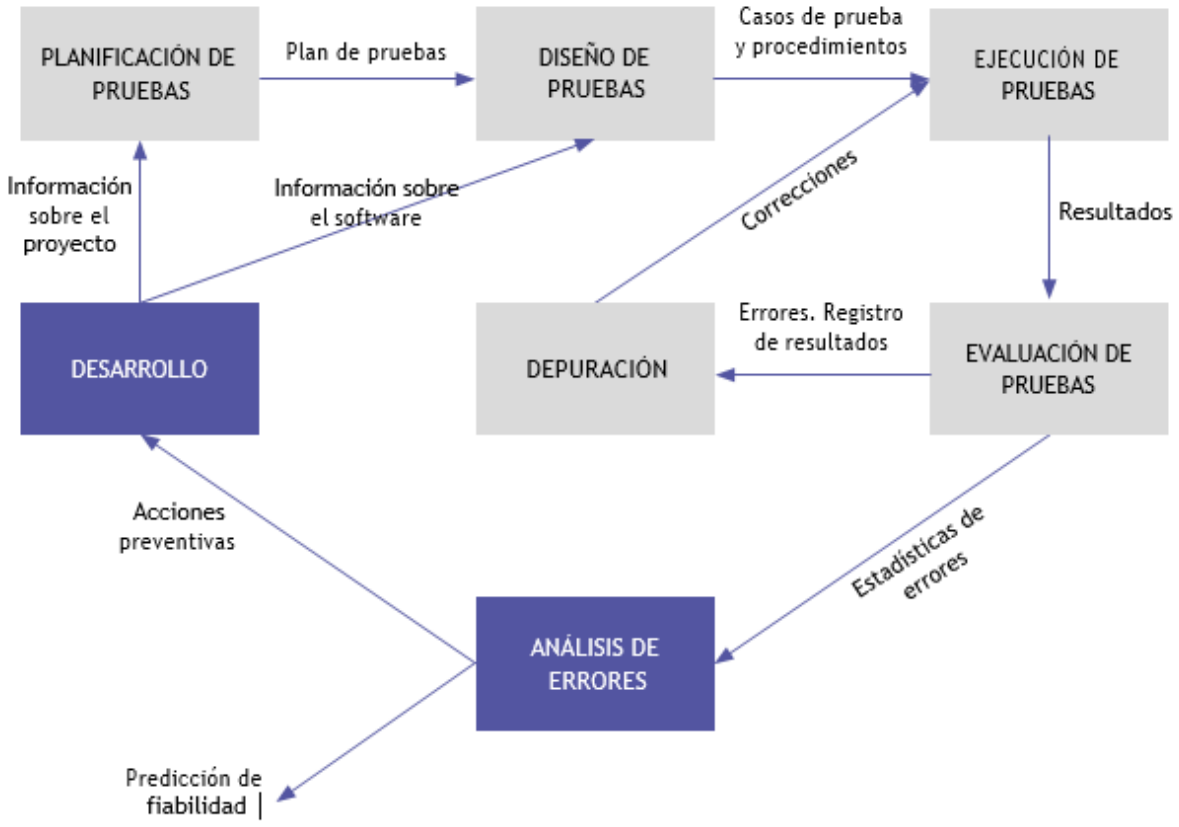
- **Verificación**: probar que el software cumple con los requerimientos. Esto quiere decir que, en el documento derequerimientos, debe haber, como mínimo, una pruebapor cada uno de ellos.
- **Validación**: encontrar alguna situación donde el software sea incorrecto o no cumple con las especificaciones.

En esta etapa trataremos de comprobar los distintos tiposde errores, haciéndolo en el menor tiempo y con el menor esfuerzo posibles. Una prueba tendrá éxito si encontramos algún error no detectado anteriormente.

- Las recomendaciones para llevar a cabo las pruebas son lassiguientes:
- Cada prueba definirá los resultados de la salida esperados.
- Evitar que el programador pruebe sus propios programas.

- Comprobar cada resultado en profundidad.
- Incluir todo tipos de datos, tanto válidos y esperados como inválidos e inesperados.
- Comprobar que el software hace lo que debe y lo que no debe hacer.
- No hacer pruebas que no estén documentadas.
- No suponer que en las pruebas no se van a cometer errores.
- Cuantas más pruebas se realicen, mayor es la probabilidad de encontrar errores y, una vez solucionados, tendremos una capacidad mayor de poder perfeccionarnuestro sistema.

El flujo del proceso a la hora de probar el software es elsiguiente:



1 Primero **generamos un plan de pruebas** a partir de la documentación del proyecto y de la documentación delsoftware que debemos probar.

2 Después se **diseñan las pruebas**: qué técnicas vamosa utilizar.

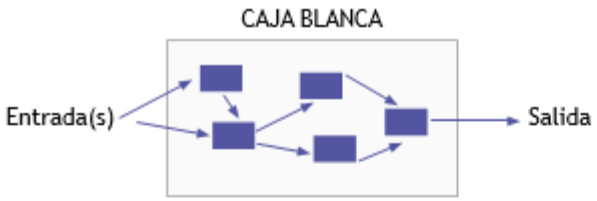
3 Ejecución de las pruebas.

4 **Evaluación**: identificación de posibles errores.

5 **Depuración**: localizar y corregir errores. Testar de nuevo tras encontrarse un error para verificar que se hacorregido.

6 **Análisis de errores**: analizar la fiabilidad del software ymejorar los procesos de desarrollo.

Para la realización del diseño de prueba se usan dos técnicas **prueba de caja blanca y prueba de caja negra**. Laprimera valida la estructura interna del sistema, y la segunda, los requisitos funcionales sin observar el funcionamientointerno del programa. No son pruebas excluyentes y podemos combinarlas para descubrir distintos tipos de error.



ponte a prueba

Estamos realizando las pruebas de un método que realiza el factorial de un número. Estamos introduciendo el número 4 y nos da como salida 24. ¿Qué pruebas estamos llevando a cabo?

a) Prueba de caja blanca
b) Prueba de caja negra
c) Pruebas de integración del sistema
d) Pruebas de seguridad

1.5.5. DOCUMENTACIÓN

Cada etapa del desarrollo tiene que quedar perfectamente documentada. Para ello, necesitaremos reunir los documentos generados y hacer una clasificación según el niveltécnico de sus descripciones.

Estos documentos:

- Deben actuar como medio de comunicación para que los miembros del equipo se puedan comunicar entre sí.
- Deben ser un almacén de información del sistema parapoder ser utilizado por personal de mantenimiento.
- Proporcionan información para facilitar la planificación de gestión del presupuesto y programar el proceso de desarrollo del software.
- Algunos documentos deben especificar al usuario cómodebe usar y administrar el sistema.

La documentación se puede dividir en dos clases:

1 **Documentación del proceso**. En estos documentos seregistra el proceso de desarrollo y mantenimiento. Así,se incluyen planes, estimaciones y horarios que se usanpara predecir y controlar el proceso de software.

2 **Documentación del producto**. Este documento des-cribe el producto que está siendo desarrollado e incluyela documentación del sistema y la documentación delusuario.

Mientras que la documentación del proceso se usa paragestionar todo el proceso de desarrollo del software, ladocumentación del producto se usará una vez que elsistema ya esté funcionando.

Documentación del usuario

Hay dos tipos de documentación:

- **Orientada a usuarios finales**: describe la funcionalidad del sistema con el objetivo de que el usuario pueda interactuar con este.
- **Orientada a administradores del sistema**: describela gestión de los programas utilizados por los usuarios finales. Como administradores del sistema, podemos encontrar gestores de redes o técnicos especializados enresolver problemas a los usuarios finales.

Al existir distintos tipos de usuario, tendremos que entregarun tipo de documento a cada uno. En el siguiente esquemase muestran los diferentes documentos:

Manual introductorio	Destinado a usuarios noveles.	Explicaciones sencillas de cómo empezar a usar el sistema. Solución de errores.
Manual de referencia del sistema	Destinado a usuarios experi-mentados.	Descripción detallada del sistemay lista completa de errores.
Guía del administrador del sistema	Destinado a administradores.	Para sistemas en los que hay comandos, describir las tareas,los mensajes producidos y las respuestas del operador.

Documentación del sistema

Serán los documentos que describen el sistema, desde la especificación de los requisitos hasta las pruebas de aceptación, y serán esenciales para entender y mantener el software.

Deberán incluir:

Fundamentos del sistema	Se describen los objetivos del sistema.
El análisis y especifica-ción de requisitos	Información exacta de los requisitos.
Diseño	Se describe la arquitectura del sistema.

Implementación	Descripción de la forma en que se expresa el sistema y acciones del programa en forma de comentarios.
Plan de pruebas del sistema	Evaluación individual de las unidades del sistema y las pruebas que se realizan.
Plan de pruebas de aceptación	Descripción de pruebas que el sistema debe pasar antesde que los usuarios las acepten.
Los diccionarios de datos	Descripciones de los términos que se relacionan con el software en cuestión.

En ocasiones, cuando los sistemas son más pequeños, seobvia la documentación.

Es necesario, como mínimo, incluir la especificación delsistema, el documento de diseño arquitectónico y el códigofuente del programa.

El mantenimiento de esta documentación muchas veces sedescuida y se deja al usuario desprotegido ante cualquierproblema sobre el manejo y errores de la aplicación.

Estructura del documento

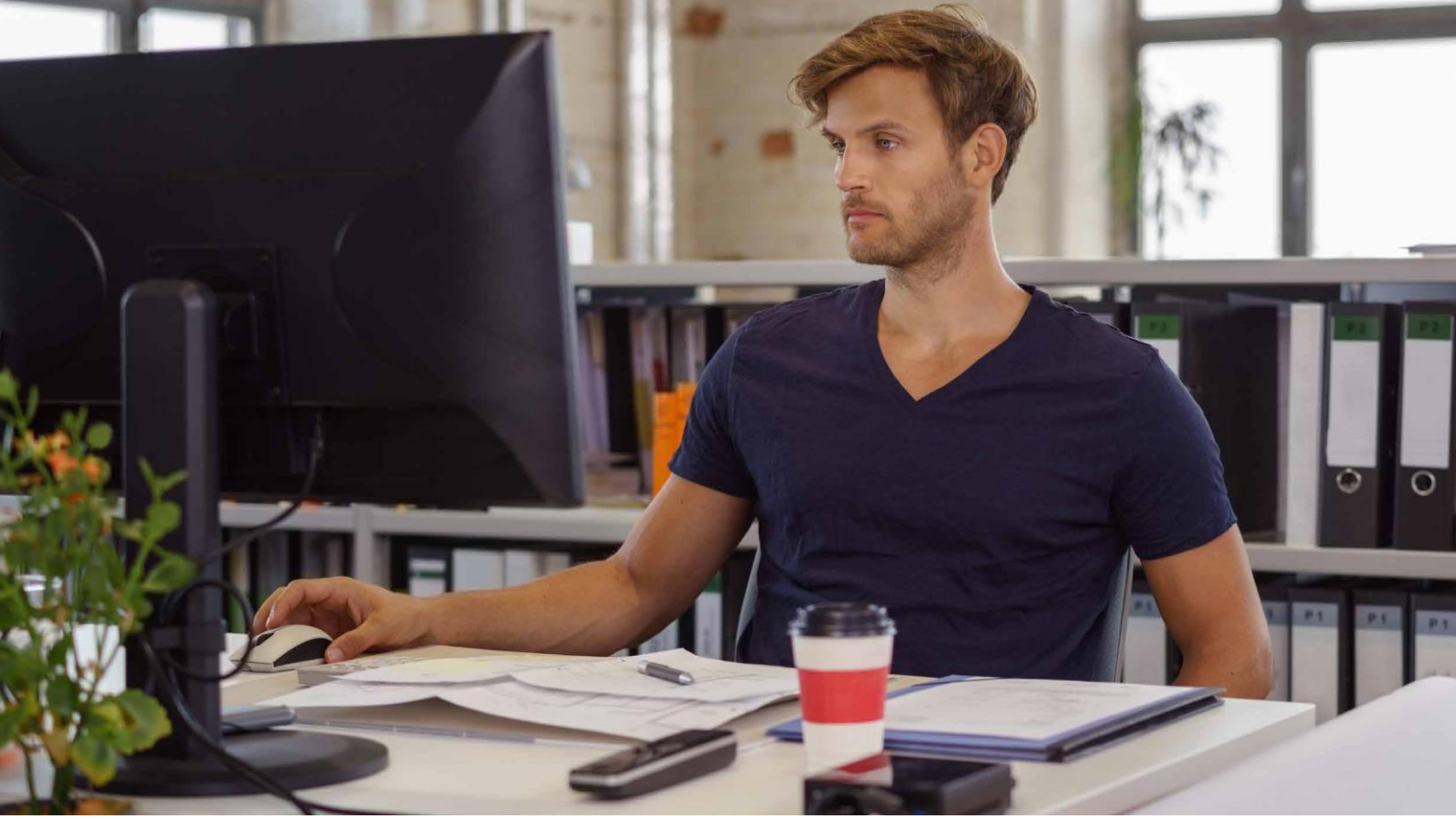
El documento debe tener una organización para que, a lahora de consultarlo, se localice fácilmente la información,por lo que tendrá que estar dividido en capítulos, seccionesy subsecciones. Algunas pautas son las siguientes:

- Deben tener una portada, tipo de documento, autor, fecha de creación, versión, revisores, destinatarios del documento y la clase de confidencialidad del documento.
- Debe poseer un índice con capítulos, secciones y subsecciones.
- Incluir al final un glosario de términos.

Según el estándar IEEE std 1063-2001, la documentaciónde usuario tiene que estar formada por los siguientes elementos:

Componente	Descripción
Datos de identificación	Título e identificación.
Tabla de contenidos	Capítulo, nombre de sección y número de página. Es obligatoria en documentos de más de ocho páginas.

Lista de ilustraciones	Números de figura y títulos (optativo).
Introducción	Propósito del documento y breve resumen.
Información para el uso de la documentación	Sugerencia sobre cómo usar la documentación de forma eficaz.
Conceptos de las operaciones	Explicación del uso del software.
Procedimientos	Instrucciones sobre cómo utilizar el software.
Información sobre los comandos software	Descripción de cada uno de los comandos del software.
Mensajes de error y resolución de problemas	Descripción de los mensajes de error y los tiposde resolución.
Glosario	Definiciones de términos especiales.
Fuentes de información relacionadas	Enlaces a otros documentos para proporcionar información adicional.
Características de navegación	Permite encontrar su ubicación actual y moverse por el documento.
Índice	Lista de términos clave y páginas a las que estos hacen referencia.
Capacidad de búsqueda	Forma de buscar términos específicos (en documentos electrónicos).



1.5.6. EXPLOTACIÓN

En esta etapa se lleva a cabo la instalación y puesta en marcha de producto. Se deberán indicar las tareas programadas (como la gestión de *backups*), la monitorización ola gestión de la capacidad. Se llevarán a cabo las siguientes tareas:

- **Gestión de *backups***: descripción detallada de la políticade *backups* del sistema. Se incluirá cada cuánto tiempose realiza un respaldo, un almacenamiento de datos y la gestión de versiones del aplicativo.
- **Carga y descarga de datos**: se llevará un control de lacarga y descarga masiva de datos y se detallarán las situaciones por las cuales se han llevado a cabo.
- Monitorización y mapeo del aplicativo.
- **Estrategia para la implementación del proceso**: se definen procedimientos para recibir, registrar, solucionar y hacer un seguimiento de los problemas y probar el producto.
- **Soporte al usuario**: dar asistencia y consultoría al usuario. Las peticiones y acciones que se hagan deberán registrarse y supervisarse.

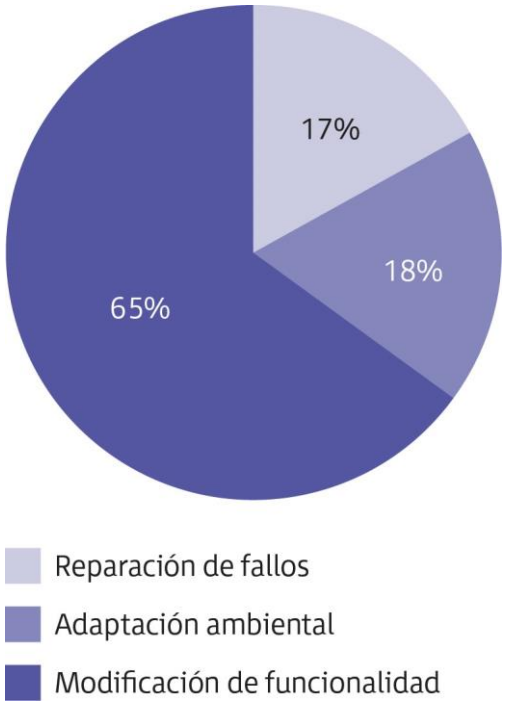
1.5.7. MANTENIMIENTO

La fase de mantenimiento consiste en la modificación delproducto software después de la entrega al usuario/clientepara corregir fallos, mejorar el rendimiento o adaptar el producto a un entorno modificado.

Existen tres tipos de mantenimiento:

- **Recuperación de fallos**: los errores de codificación son los menos costosos. Los errores de diseño son más caros porque pueden verse afectados muchos componentes del software.
- **Adaptación ambiental**: mantenimiento respecto del entorno del sistema, como el hardware, el sistema operativo, etcétera.
- **Incremento de funcionalidad**: varían los requisitos del sistema debido, por ejemplo, a un cambio empresarial o en la organización.

La distribución del esfuerzo de mantenimiento es el siguiente:



En general, el coste de agregar una nueva funcionalidad después de haber realizado el aplicativo es mayor que añadir esta funcionalidad durante el desarrollo de este. Esto se debe sobre todo a:

- **La estabilidad del equipo**: una vez que el proyecto fina-liza, por término general, el equipo de trabajo se separa y se encarga de nuevos proyectos. El nuevo equipo de mantenimiento debe emplear un tiempo en conocer la estructura y los componentes del sistema.
- **Diferencia de contratos**: generalmente, el desarrollo y el mantenimiento suelen ir en contratos diferentes, incluso el mantenimiento puede ser encargado a una compañía diferente. Por tanto, ese equipo de mantenimiento apenas cuenta con estímulos suficientes.
- **Habilidades del equipo de mantenimiento**: por término general, el equipo de mantenimiento suele ser más inexperto y no está familiarizado con los entornos de desarrollo del sistema. De hecho, pueden existir sistemas antiguos de lenguajes de programación obsoletos (como Basic o sistemas MS-DOS) en los que no se tenga experiencia.

- **Antigüedad del programa**: según se van realizando cambios, la estructura del programa tiende a degradarse. Los programas van envejeciendo y resultan más complicados de escalar y mantener. La documentación puede estar obsoleta si no ha sido actualizada o los programas se desarrollaron con técnicas de programaciónya extintas.

Por tanto, intentar predecir el número de solicitudes decambio para un sistema requiere un sobrecoste de entendimiento de la relación entre el sistema y su relación con el exterior.

Con base en la experiencia, podemos intentar predecir:

- ¿Cuántas peticiones cambio suelen producirse?
- ¿Qué partes del sistema suelen ser las más afectadas porlos cambios?
- ¿Qué es lo más costoso en el mantenimiento?
- ¿Cómo se distribuye el coste en el sistema?
- ¿Cómo evolucionan esos costes a lo largo de vida del proyecto?

1.6.METODOLOGÍAS ÁGILES

Las metodologías ágiles son métodos de gestión que permiten adaptar la forma de trabajo al contexto y naturalezade un proyecto, basándose en la flexibilidad y la inmediatez y teniendo en cuenta las exigencias del mercado y de los clientes. Los pilares fundamentales de las metodologíaságiles son el trabajo colaborativo y en equipo.

Trabajar con metodologías ágiles conlleva las siguientesventajas:

- Ahorrar tanto en tiempo como en costes (son baratas ymás rápidas).
- Mejorar la satisfacción del cliente.
- Mejorar la motivación e implicación del equipo de desarrollo.
- Mejorar la calidad del producto.
- Eliminar aquellas características innecesarias del producto.

- Alertar rápidamente tanto de errores como de problemas.

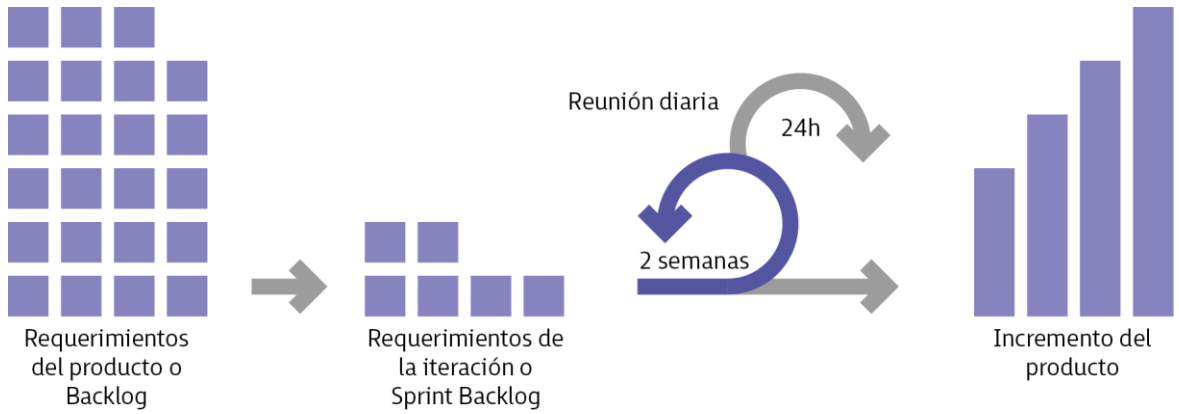
Aunque hay muchas ventajas a la hora de aplicar estas metodologías, estos principios son difíciles a veces de cumplir:

- Aunque la idea de que el cliente esté involucrado en todoel proceso es un incentivo positivo, los clientes tienenque ocuparse de otros procesos de gestión y están sometidos a presiones muy diferentes.
- La idea de que el equipo de trabajo esté cohesionado es,en ocasiones, ideal. No todos los componentes del equipo tienen la misma personalidad ni el mismo compromiso.
- Los cambios son complicados con demasiados participantes.
- Mantener esta línea de trabajo a veces es complicado porrazones externas: plazos de entrega, cada componentedel equipo trabaja a un ritmo, etcétera.
- Las grandes compañías están sujetas a muchos cambiosa lo largo del tiempo, por lo que muchos métodos de trabajo se pueden ver modificados y pueden existir muchas alteraciones en las metodologías.

En este punto, vamos a destacar cuatro metodologías:

SCRUM

El enfoque de esta metodología se basa en un trabajoiterativo.



Existen tres fases de esta metodología:

- **Planificación**: donde se establecen los objetivos genera-les del proyecto y cómo será la arquitectura.
- **Ciclos *(sprints)***: en cada uno de estos ciclos se desarrolla un incremento o iteración.
- **Documentación**: donde se desarrolla la ayuda del sistema y los manuales de usuario.

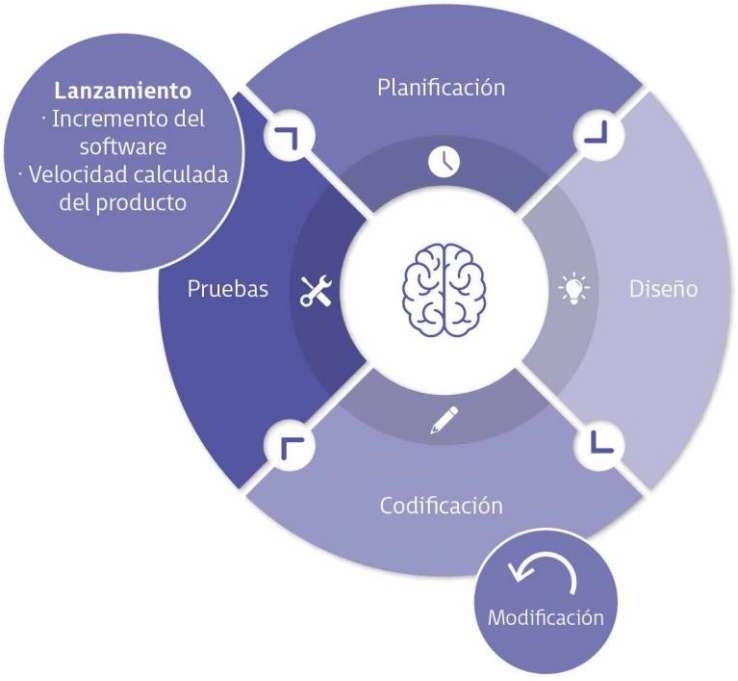
Por tanto, aporta una estrategia de desarrollo incremental en lugar de la planificación y ejecución completa del producto. La calidad del resultado se basa principalmente en el conocimiento innato de las personas en equipos autoorganizados antes que en la calidad de los procesosempleados.

Características específicas de SCRUM:

Una de las bases de las metodologías ágiles es el ciclo devida iterativo e incremental. El ciclo de vida iterativo o incremental es aquel en el que se va liberando el productopor partes, periódicamente, iterativamente, poco a pocoy, además, cada entrega es el incremento de funcionalidad respecto de la anterior. Cada periodo de entrega esun *sprint*. Estos *sprints* tienen una longitud fija de entre dos y cuatro semanas.

- Reunión diaria. Máximo, 15 minutos. Se trata de ver qué se hizo ayer, qué se va a hacer hoy y qué problemas se hanencontrado.
- Reunión de revisiones del *sprint*. Al final de cada *sprint*, se trata de analizar qué se ha completado y qué no.
- Retrospectiva del *sprint*. También se realiza al final de*sprint*, y sirve para que los implicados den sus impresiones sobre *sprint*. Se utiliza para la mejora del proceso.
- Durante la fase de cómo va a ser el *sprint*, se asignan lasprioridades del proyecto y riesgos de este.
- A diferencia de XP, SCRUM no hace indicaciones concretas sobre cómo detallar los requerimientos.

Programación extrema (XP)



Metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollodel software, promoviendo el trabajo en equipo, pre- ocupándose por el aprendizaje de los desarrolladores ypropiciando un buen clima de trabajo.

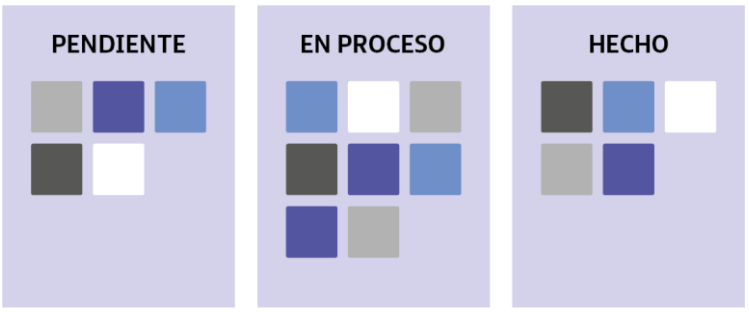
XP se basa en la retroalimentación continua entre clientey el equipo de desarrollo. La XP es especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes.

Características específicas de la XP:

- Los requerimientos se expresan como escenarios (también llamados historias de usuario).
- Las liberaciones que se van haciendo del sistema se basan en esas historias de usuario.
- Todos los procesos se basan en la **programación a pares**:trabajo colectivo del grupo.
- Existe, por tanto, una refactorización constante.
- Se valora al individuo y las interacciones del equipode desarrollo sobre el proceso y las herramientas. La gente es el principal factor de éxito de un proyecto desoftware.
- Se trata de desarrollar un software que funcione, más queconseguir una buena documentación.
- Se propone que exista una interacción y colaboración constante entre el cliente y el equipo de desarrollo.
- La habilidad de responder a los cambios que puedan surgir a lo largo del proyecto determina también el éxito o elfracaso de este. La planificación no debe ser estricta, sinoflexible y abierta.

Kanban

Es una palabra japonesa que significa “tarjetas visuales”. Esta técnica se creó en Toyota, y se utiliza para controlar el avance del trabajo en el contexto de una línea de producción. Actualmente, está siendo aplicado en la gestión deproyectos software.



Características específicas de Kanban

- Visualiza tu trabajo dividido en bloques.
- Tiene varias barras de progreso donde la tarea va avanzando según su estado.
- Tener una visión global de todas las tareas pendientes,de las tareas que estamos realizando y de las tareas que hemos realizado.

- Se pueden hacer cálculos de tiempo con las tareas finalizadas y tareas similares que vamos a realizar, para teneruna idea del tiempo que cada uno puede tardar a realizarlas.
- Limita tu WIP *(work in progress).* Asigna límites en lo querespecta a cuántos elementos puedan estar en progresoen cada estado.

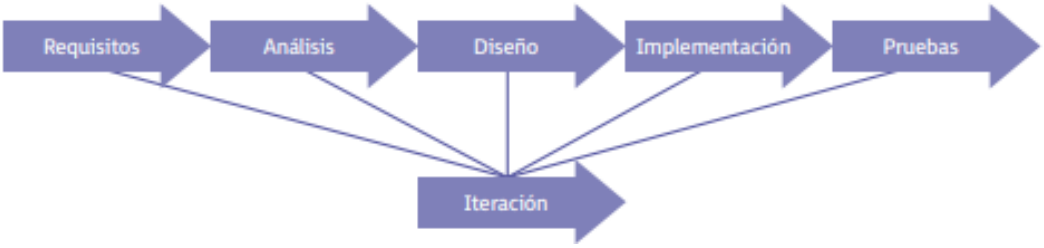
Rational unified process (RUP)

El objetivo es estructurar y organizar el desarrollo del software. Las características son:

Está dirigido por los casos de uso.

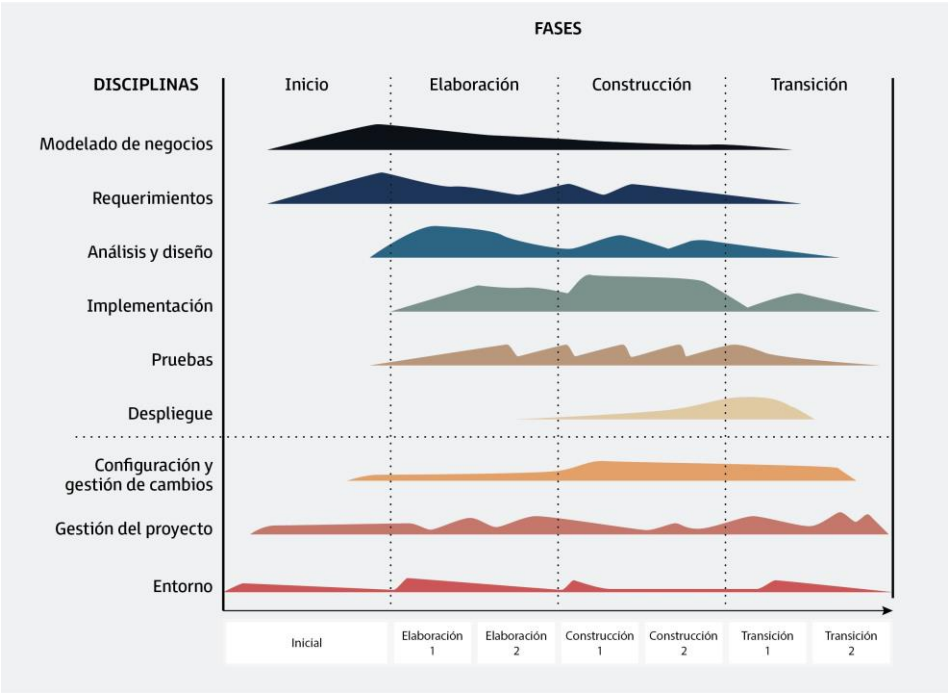
Centrado en la arquitectura del SW. Se centra en las diferentes vistas del sistema correspondiente a las fases explicadas anteriormente: análisis, diseño e implementación. Analiza el sistema como un todo y a la vez analizacada una de sus partes.

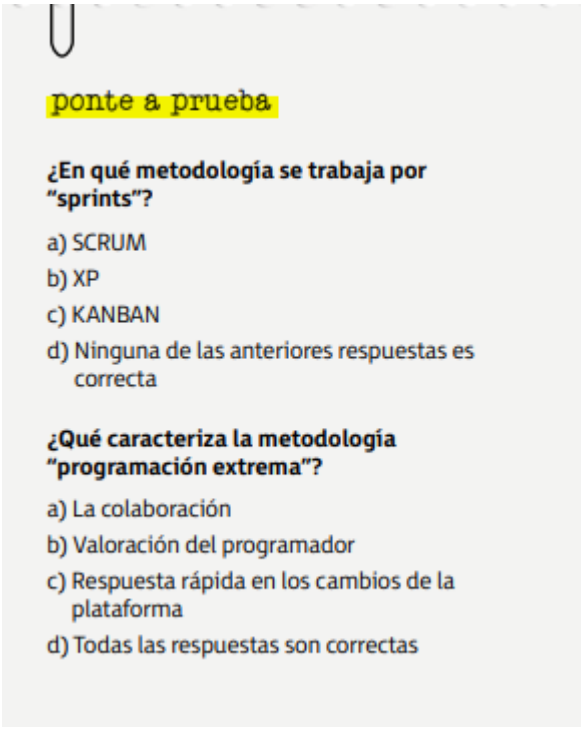
Es un proceso iterativo e incremental: Se divide en pequeños proyectos donde se va incrementando en funcionalidad.



La estructura de RUP se pone en práctica en tres perspectivas:

1. La dinámica: contendrá fases del sistema sobre el tiempo de desarrollo.
2. La estática: donde se muestran todas las actividadesdel proceso.
3. La práctica: donde se ponen de manifiesto las buenasprácticas de desarrollo (gestión de los requisitos, modelado en UMlL, verificación de calidad, etcétera).





ponte a prueba

¿En qué metodología se trabaja por "sprints"?

a) SCRUM
b) XP
c) KANBAN
d) Ninguna de las anteriores respuestas es correcta

¿Qué caracteriza la metodología "programación extrema"?

a) La colaboración
b) Valoración del programador
c) Respuesta rápida en los cambios de la plataforma
d) Todas las respuestas son correctas

1.7.PROCESO DE OBTENCIÓN DE CÓDIGO A PARTIR DEL CÓDIGO FUENTE. HERRAMIENTAS IMPLICADAS

La generación de código fuente se lleva a cabo en la etapa de codificación. En esta etapa, el códigopasa por diferentes estados.

1.7.1. TIPOS DE CÓDIGO:

- **Código fuente**: es el código escrito por los programadores, utilizando algún editor de texto oalguna herramienta de programación. Se utilizaun lenguaje de alto nivel.
- **Código objeto**: es el código resultante de compilar el código fuente. No es ejecutable por elordenador ni entendido por la máquina, es un código intermedio de bajo nivel.
- **Código ejecutable**: es el resultado de enlazar elcódigo objeto con una serie de rutinas y librerías,obteniendo así el código que es directamente ejecutable por la máquina.

1.7.2. OBTENCIÓN Y EDICIÓN DEL CÓDIGO EJECUTABLE.HERRAMIENTAS:

- **Librerías**: son archivos que contienen diferentes funcionalidades. Pueden ser llamados desde cualquier aplicación de un sistema operativo (porejemplo, Windows está compuesto de librerías de enlaces dinámicos llamadas DLL que generan extensiones como .exe o .fon para las fuentes).

- **Editor**: es una herramienta para crear código fuente en un lenguaje de programación. Puede marcar la sintaxis de dicho lenguaje y realizar tareas de autocompletado. Ejemplos de editores son IntelliJ IDEA, Notepad++ o Sublime Text.
- **Compilador**: es la herramienta que nos proporciona el lenguaje máquina entendible por nuestro ordenador. A este proceso de traducción se le denomina compilación.
- **Enlazador**: el objetivo de esta herramienta escoger los objetos que se crean en los primeros pasos de la compilación, quitar los recursos que no necesita y enlazar el código objeto con sus bibliotecas. El resultado es un archivo ejecutable.