DISEÑO DE UN PROGRAMA

PROGRAMACIÓN

Tabla de contenido

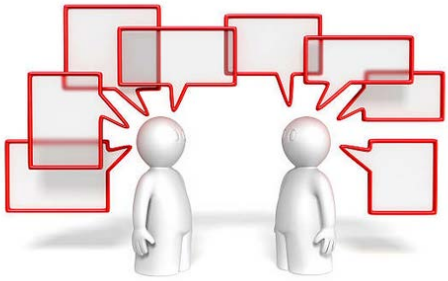
DISEÑO DE UN PROGRAMA

A la hora de afrontar la realización de un programa debemos tener claro qué debemos hacer. Es un error comenzar a crear software a lo loco ya que solo conseguiremos dedicar a este menester más tiempo del necesario. Además, el proceso de creación no solo conlleva picar código, sino que una vez se ha generado un código ejecutable este debe ser probado, de forma que se detecten posibles fallos de ejecución para su corrección antes de promocionarlo entre los usuarios finales.

Así, a la hora de construir un programa se deben seguir una serie de pautas o fases:

- Análisis del problema.
- Diseño del algoritmo.
- Codificación.
- Prueba y depuración.
- Documentación.

FASE DE ANÁLISIS DEL PROBLEMA



En esta fase se debe analizar con detenimiento el problema que se plantea. A la hora de realizar un programa debemos tener clara cuál es su finalidad y a quien va dirigido. Así, debemos tener una perspectiva cercana a la exactitud de qué elementos debe incluir y que tareas debe realizar. Cuando el software es para nuestro uso personal nosotros mismo especificamos las pautas y decidimos su funcionalidad, en cambio, cuando debe ser usado por otras personas, para conocer sus necesidades debemos entrevistamos con ellos.

Así, normalmente, cuando una persona individual o una empresa nos encarga la realización de un programa debemos concertar una entrevista de forma que de ella obtengamos una visión particular de las necesidades del usuario o la empresa y de las tareas que debe realizar el software.

La fase de análisis en una parte fundamental del proceso de desarrollo de un programa, una de las más importantes, ya que un mal análisis puede hacer que un programa ya codificado y probado deba realizarse de nuevo.

En qué tipo de ordenadores esperamos ejecutar el software. Sistema operativo para el que se diseña, microprocesador y memoria mínimos, etc. Las aplicaciones en función de su codificación serán más o menos óptimas de forma que necesitarán ordenadores de mayor o menor rendimiento. Debemos conocer esta característica justo en este punto para en la fase de codificación adaptar las tareas a los recursos de los que vamos a disponer.

A quién va dirigido el software. No es lo mismo diseñar un programa para personas acostumbradas a usar todo tipo de aplicaciones y otras que utilicen mínimamente las nuevas tecnologías.

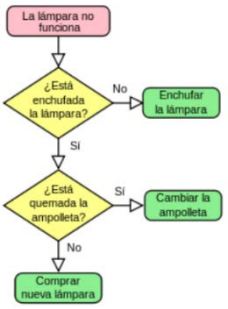
Cuáles serán los datos de entrada. A la hora de usar el programa, cuáles serán los datos que se van a procesar, por ejemplo, si vamos a diseñar una aplicación para una cafetería los datos a introducir serán las características de los diferentes alimentos que tendremos de venta a nuestros clientes.

Cuáles serán los datos de salida. En función de los datos de entrada y las operaciones que realicemos con ellos cuáles serán los datos de salida esperados. En el caso de la cafetería podría ser un ticket impreso con los diferentes productos consumidos.

Cuáles serán las operaciones a realizar para que a partir de unos datos de entrada dados obtengamos la salida que hemos analizado con anterioridad.

FASE DE DISEÑO DEL ALGORITMO

Una vez se tengan claras las tareas a realizar, los datos que se van a usar, en qué sistemas se ejecutará el software, etc., pasamos a desarrollar el algoritmo que representa de forma técnica cómo abordar el problema inicial.

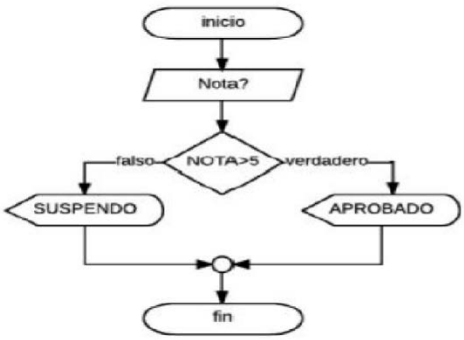


En la fase de algoritmo usaremos **diagramas de flujo** y **pseudocódigo.**

En la fase anterior concretábamos todos los elementos que el programa debía tener en cuenta y las tareas a realizar, en esta se identifican soluciones a los problemas planteados, así como pasos de ejecución para concretar las tareas.

La fase de diseño, es una especificación detallada que facilita posteriormente la programación en un lenguaje de programación concreto.

Diagrama de Flujo



Pseudocódigo

```
Algoritmo Aprobado;
Leer nota;
si nota>5 hacer
    Escribir "Aprobado";
sino
    Escribir "Suspenso";
Fin
```

Pseudocódigo

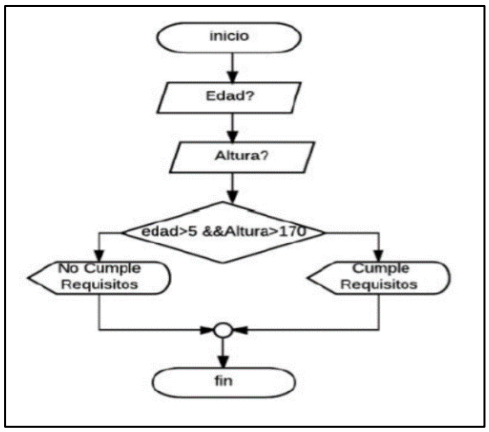
***PROGRAMA**: notaAsignatura*
DECLARACIONES**: Nota es número **entero

INICIO PROGRAMA

escribir "Introduce nota"
leer Nota
SI** Nota < 5 **ENTONCES
escribir "Suspenso"
SINO
escribir "Aprobado"
FINSI

FIN PROGRAMA

Fase de diseño, diagrama de flujo + pseudocódigo




```
Algoritmo Requisitos;
Leer edad ;
Leer Altura;
si edad>5 y altura>170
hacer
   Escribir "Cumple Requisitos"
sino
 Escribir "No Cumple Requisitos";
Fin
```

Pseudocódigo

***PROGRAMA**: Requisitos*
DECLARACIONES**: edad y altura **entero

INICIO PROGRAMA

escribir "Introduce tu edad"
leer edad
escribir "Introduce tu altura"
leer altura

SI** edad > 5 y altura >170 **ENTONCES
escribir "Suspenso"
SINO
escribir "Aprobado"
FINSI

FIN PROGRAMA

Existen unas pautas a seguir en el desarrollo de diagramas de flujo y pseudocódigo, estudiaremos estas en siguiente es apartados.

FASE DE PROGRAMACIÓN (CODIFICACIÓN)

Es en esta fase donde se usará un lenguaje de programación concreto de forma que el algoritmo anterior se convierta realmente en un programa que pueda ser ejecutado. Llegados a este punto solo tendremos que ver línea a línea el algoritmo anteriormente descrito y convertir estas a los símbolos admitidos por el lenguaje de programación que vayamos a usar. En ocasiones no será tan fácil la conversión, ya que existirán líneas que no puedan plasmarse con el significado que se colocaron en el algoritmo usando el lenguaje de programación escogido tendremos que buscar alternativas.

Pseudocódigo

```
Algoritmo Aprobado;
Leer nota;
si nota>5 hacer
    Escribir "Aprobado";
sino
    Escribir "Suspenso";
Fin
```

Lenguaje C

```
#include <stdio.h>
int main(){
int nota;
printf("Nota ?");
scanf("%i%",&nota);
if (nota==5){
    printf("Sacaste 5");
}
else {
    printf("No sacas 5");
}
}
```

Lenguaje Python

```
nota=input("Nota? ")
if (nota==5):
   print "Sacaste 5"
else:
  print "No sacas 5"
```

1. Se establece las variables en la "Declaración", es decir, los elementos de almacenamiento de información (variables) que debemos usar. En nuestro programa a este componente se llama **Nota.**
2. Se escribe en pantalla el mensaje *"Introduce una nota"*. C y C++ usan la función **printf** para mostrar información al usuario mediante el display o dispositivo hardware de salida por defecto, en Python la sentencia/método es **print**().
3. Llegados a este punto el programa se detiene a la espera de que el usuario escriba la información pedida. Se recoge el dato en el componente **Nota** antes declarado. Una de las funciones que utilizan C y C++/Python para la lectura de información es **scanf/imput().** Esta función necesita saber qué tipo de información va a recoger, es decir, si será un número, un texto, un valor decimal o un carácter, de ahí el **%i.**

```
Algoritmo Requisitos;
Leer edad ;
Leer Altura;
si edad>5 y altura>170
hacer
   Escribir "Cumple Requisitos"
sino
 Escribir "No Cumple Requisitos";
Fin
```

Lenguaje C

```
#include <stdio.h>
int main(){
int edad;
int altura;
printf("Edad ?");
scanf("%i",&edad);
printf("Altura ?");
scanf("%i%",&altura);
if (edad>18 &&altura>170)
  {
     printf("Cumple");
  }
else {
      printf("No Cumple");
    }
}
```

Código de Pyhton

```
edad=input("Dime la edad: ")
altura=input("Dime la altura: ")
if (edad>18 and altura>170):
   print "Cumple"
else:
   print "No Cumple"
```

NOTA: C++ es un lenguaje de programación orientado a objetos y en él solemos usar normalmente otro tipo de elementos para realizar la función de mostrar o recoger información al o del usuario. Encontraremos líneas similares a **cout « Mensaje** cuando queremos mostrar información al usuario

imprimir » variable cuando queremos realizar la operación contraria.

4. A continuación, el programa realizará una pequeña comprobación, en función de la cual, realizará una acción u otra. C y C++ utilizan la palabra reservada **if** para evaluar una condición. Se traduce más o menos a **si la condición se cumple...** Así, si el valor especificado por el usuario es menor que 5 se mostrará en pantalla el texto **Suspenso** y en caso contrario **(else)** el texto **Aprobado.**

FASE DE PRUEBAS Y DEPURACIÓN

Una vez el programa se crea debemos comenzar a probarlo antes de distribuirlo al usuario final. Si bien es cierto que cuando se crea un software, este se prueba repetidamente, se abordan los fallos más relevantes, pero cuando pasa a ser "probado" por el usuario final es posible que surjan nuevos fallos de los que no nos hemos percatado. Así, cuando se crea un programa y se saca al mercado es normal encontrar cada cierto tiempo actualizaciones del mismo, estas proporcionan nuevas funciones, pero además subsanan errores no controlados.

Pseudocódigo

```
PROGRAMA: notaAsignatura
DECLARACIONES: Nota es número entero

INICIO PROGRAMA

        escribir "Introduce nota"
        leer Nota
        SI Nota > 5 ENTONCES
                escribir "Suspenso"
        SINO
                escribir "Aprobado"
        FINSI

FIN PROGRAMA
```

Durante las pruebas del software se suponen todas las situaciones en las que se puede encontrar el programa y se observa cómo actúa ante ellas. La depuración consiste en el uso de valores de prueba para la detección de posibles errores, corrección de los mismos, y así elaborar una versión del software libre de fallos. Así la **depuración es el proceso de detección de errores en tiempo de ejecución** y **corrección de los mismos.**

Cuando iniciamos la fase de depuración y escogemos como valor de prueba 8, es decir, ***nota*** *=* ***8,*** **¿qué sucederá?**

Al llegar a la condición vemos que esta dice algo como "si nota es mayor que 5 escribe Suspenso", nuestra nota es 8, mayor que 5, así la palabra Suspenso es lo que se visualizará, pero, **¿es correcta la solución?**

Evidentemente no, ya que una nota de 8 es muy aceptable y para nada se considera suspenso. Dada esta situación, el desarrollador del programa sustituirá el símbolo > por < y seguirá realizando nuevas pruebas.

Cuando todos los valores de entrada posible se hayan analizado y se hayan observado todas las salidas obtenidas, si estas son correctas daremos por finalizada la fase de depuración y pruebas.

FASE DE DOCUMENTACIÓN

Para finalizar el proceso de creación de un programa, crearemos documentación necesaria relacionada con este.

Es necesario que el usuario final conozca…

qué requisitos debe poseer el hardware,

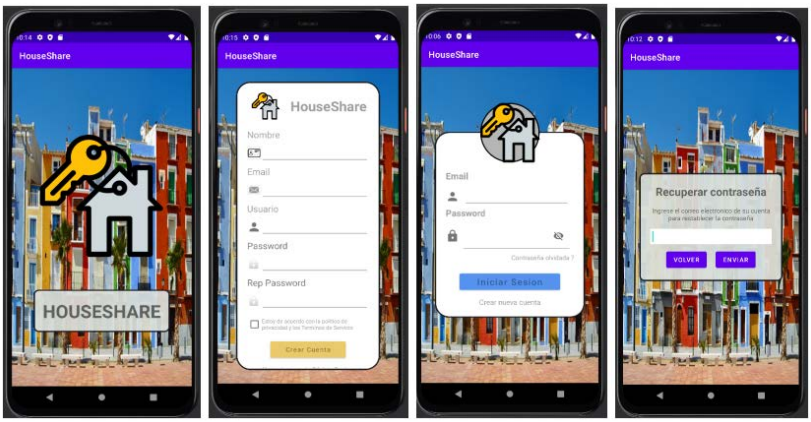
cómo podemos ejecutar el software (formato de ejecución o posibles parámetros a usar)

cuáles son los valores de entrada que acepta,

uso de cada uno de los apartados,

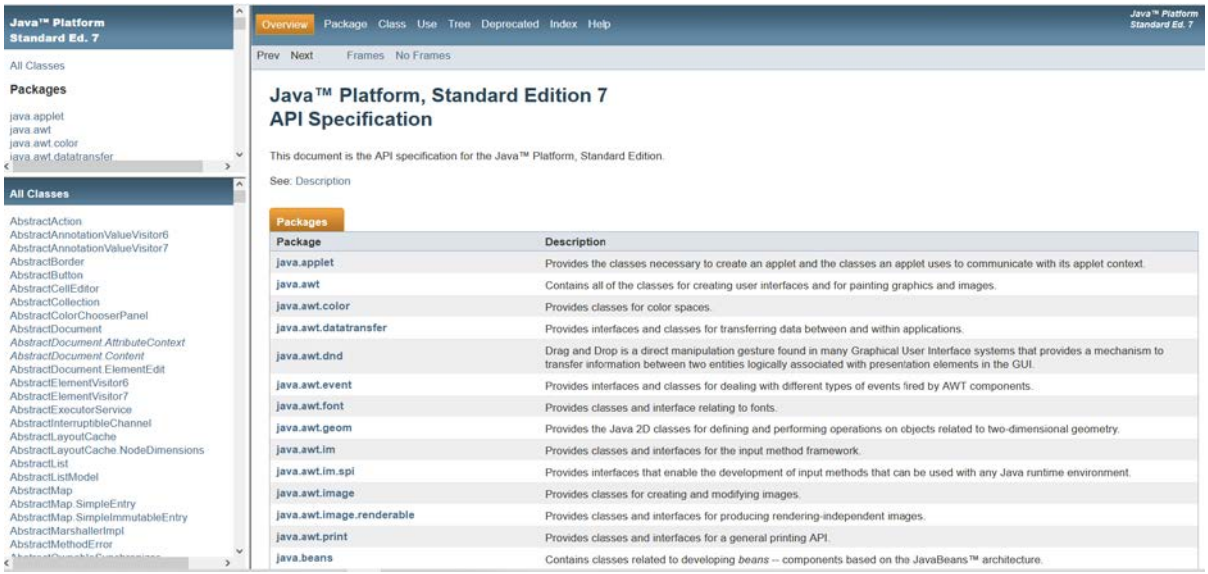
etc…

En el caso de la creación de una aplicación apk como se muestra, debemos crear el manual de usuario.



En el caso de la creación de librerías, APIs “interfaz de programación de aplicaciones”, Clases, Códigos, etc. deberemos crear la documentación necesaria para que los programadores manejen el código proporcionado.

Ejemplo, librería API para las ayudas de programación en JAVA.



¿Qué significa API?
API significa “interfaz de programación de aplicaciones”. En el contexto de las API, la palabra aplicación se refiere a cualquier software con una función distinta. La interfaz puede considerarse como un contrato de servicio entre dos aplicaciones. Este contrato define cómo se comunican entre sí mediante solicitudes y respuestas. La documentación de su API contiene información sobre cómo los desarrolladores deben estructurar esas solicitudes y respuestas.

¿Cómo funcionan las API?
La arquitectura de las API suele explicarse en términos de cliente y servidor. La aplicación que envía la solicitud se llama cliente, y la que envía la respuesta se llama servidor. En el ejemplo del tiempo, la base de datos meteorológicos del instituto es el servidor y la aplicación móvil es el cliente.

EJEMPLO DE CREACIÓN DE UN PROGRAMA

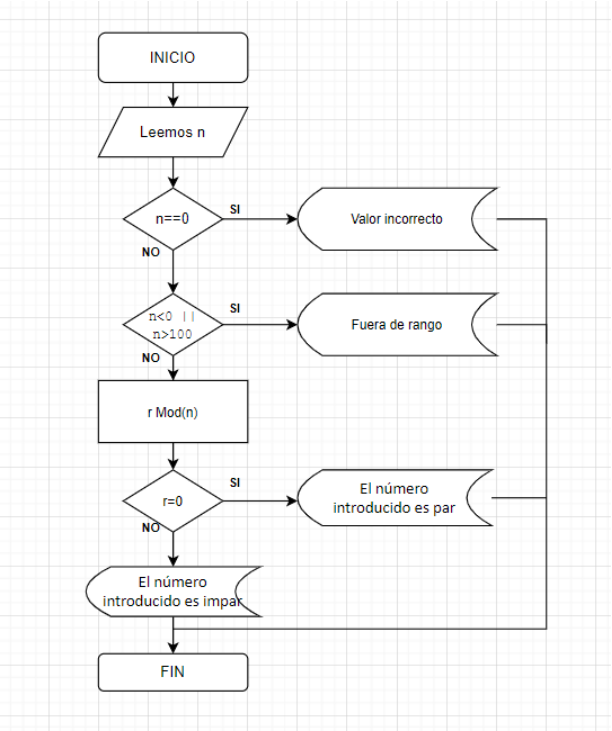
En este apartado vamos a ejemplificar las fases anteriormente descritas, teniendo en cuenta un problema real. Supongamos que una empresa se pone en contacto con nosotros y nos pide realizar un software que debe detectar si un número introducido es par o no.

FASE 1. ANÁLISIS DEL PROBLEMA

Concertamos una entrevista con la persona encargada en la empresa que pueda hablamos sobre el tipo de software que necesita. En esta entrevista preguntaremos qué tipos de datos de entrada se deben usar, una vez se procese un dato qué salida se debe obtener, si se produce un error qué se espera visualizar, etc.

Tras el encuentro queda claro que:

1. El programa solicitará un número. Solo se pueden comprobar los 100 primeros números. El cero no es un valor válido.
2. En caso de que el número sea par se visualizará en la pantalla “El número introducido es par”.
3. En caso de que el número sea impar se visualizará en la pantalla “El número introducido es impar”.
4. Si el número a comprobar es el cero se visualizará por pantalla el mensaje “Valor incorrecto, el cero no es un valor válido”.
5. Si el número introducido es menor que 0 o mayor de 100 se mostrará en pantalla el mensaje “Número introducido fuera de rango. Rango de número válido para comprobar 1 -100”.



FASE 2. DISEÑO DEL ALGORITMO

Ya tenemos los datos necesarios para comenzar a trabajar. Sabemos qué desea el usuario final que ocurra en determinadas situaciones de ejecución así que construiremos el diagrama de flujo del programa y su pseudocódigo. Ejemplos de ambos elementos podrían ser los expuestos a continuación.

PROGRAMA: NumeroPar

***ENTORNO**: numero, modulo: entero*

***ALGORITMO**:*

```
escribir "Introduce un numero"
leer numero
SI numero == 0   ENTONCES
        escribir "Valor incorrecto, el cero no es un válido"
SI numero < 0 || numero > 100 ENTONCES
        escribir "Número introducido fuera de rango. Rango de número válido para comprobar 1 - 100"
SINO
        modulo= Resto de Numero/2
        SI modulo =        0 ENTONCES
                escribir "El número introducido es par"
        SINO
                escribir "El número introducido es impar"
        FINSI
FINSI
FIN PROGRAMA
```

FASE 3. CODIFICACIÓN

La base anterior es una de las más complejas ya que concretamos exactamente las partes del algoritmo. En esta todo es más sencillo ya que solo debemos "traducir" el algoritmo a un lenguaje de programación concreto. En realidad, esta fase será más o menos compleja en función del lenguaje de programación que usemos.

Veamos línea a línea como será el código de nuestro programa teniendo en cuenta que usaremos C++ como lenguaje de programación.

PROGRAMA: numeroPar	void numeroPar () {
ENTORNO: Numero, Modulo son números entero	**int** Numero, Modulo;
ALGORITMO: Escribir "Introduce un numero" leer Numero	**printf** ("Introduce un numero"); **scanf** ("%i",Numero);

SI Numero = 0 **ENTONCES**
escribir "Valor incorrecto, el cero no es un valor válido"
	if (Numero == 0) {
	printf ("Valor incorrecto, el cero no es un valor válido");
	}

SI Numero < 0 **||** Numero > 100 **ENTONCES**
escribir "Número introducido fuera de Rango de número válido para comprobar 1 - 100".
	else if (Numero < 0 **||** Numero > 100) {
	printf("Número introducido fuera de rando. Rango de número válido para comprobar 1 - 100");
	}

SINO
Modulo = Resto de Numero/2
SI Modulo = 0 **ENTONCES**
escribir "El número introducido es par"
SINO
escribir "El número introducido es impar"
FINSI
FINSI
	else {
	Modulo = Numero mod 2;
	if (Modulo == 0) {
	printf ("El número introducido es par");
	)
	else {
	printf ("El número introducido es impar");
	}
	}

FIN PROGRAMA	}

```
void numeroParO {
        //Declaraciones
        int Numero, Modulo;

        //INICIO
        printf ("Introduce un numero");
        scanf ("%i",Numero);
        if (Numero = = 0 )  {
                printf ("Valor incorrecto, el cero no es unvalor válido");
        } elseif (Numero < 0 || Numero > 100) {
                printf("Número introducido fuera de rando de número válido para comprobar 1 - 100") ;
        } else    {
                Modulo = Numero mod 2;
                if (Modulo == 0){
                        printf ("El número introducido es par");
                }
                else {
                        printf ("El número introducido es impar");
                }
        }
}
```

Esta es la función completa escrita en C++. Se ha comprobado que está libre de fallos sintácticos, léxicos y semánticos.

FASE 4. PRUEBAS Y DEPURACIÓN

A partir de ahora comenzamos a ejecutar el software. Se debe comprobar que ante determinadas entradas las salidas son las esperadas. Las entradas que pueden presentar problemas son:

Número 0.

Número negativo.

Número superior a 100.

Al ejecutar la aplicación visualizaremos un entorno en modo texto como el que se observa en la

Escribiremos un cero.
Debe aparecer el texto "Valor incorrecto, el cero no es un valor válido"

Realizaremos este proceso con todos los valores de entrada posibles.

Observaremos las salidas, y si son las que se plantearon en el algoritmo el programa estará listo para ser usado.

Si se produce algún fallo, o alguna salida no es la esperada para el valor introducido volveremos al código fuente y localizaremos el lugar donde se produce el error en ejecución.

Ya estudiaremos cómo visualizar los cambios que se dan en los valores introducidos durante la ejecución mediante la depuración en los diferentes entornos de programación que utilizaremos.

El código planteado realiza las funciones tal y como la empresa detalló de forma que pasaremos a la fase de documentación.

FASE 5. DOCUMENTACIÓN

Antes de entregar el software a la empresa que nos lo encargó realizaremos un breve manual sobre el mismo. Este podría ser similar al que se muestra a continuación.

PROCESO DE COMPILACIÓN

Cuando estudiábamos los tipos de lenguajes de programación veíamos que en función de cómo se ejecutaban, teníamos lenguajes de programación compilados y lenguajes de programación interpretados.

En este apartado vamos a estudiar el proceso de compilación de un software. La palabra **compilación** refiere a la traducción de un programa en código fuente a código máquina o bytecode. **Compilador** será el nombre de la herramienta software que es capaz de realizar la traducción.

Un compilador está formado por dos partes fundamentales, cada una de ellas se divide en otras tantas necesarias para cubrir todos los pasos del proceso.

Así, dispone de: **Análisis** y **Síntesis.**

La parte de **análisis** se encarga de la comprobación de las líneas de código, análisis léxico, sintáctico y semántico. La partede **síntesis** es la encargada de la obtención del código objeto.

Esta, a su vez, se divide en diferentes fases donde se genera código intermedio y se optimiza.

La fase de análisis está compuesta por:

- **Análisis léxico.** Este análisis se encarga de comprobar si las palabras introducidas son correctas. Se eliminan espacios en blanco, líneas vacías, comentarios, etc. El código fuente se lee de izquierda a derecha formando tokens, que no son más que consecución de letras que forman una palabra reservada válidas o bien un dato.
- **Análisis sintáctico.** Pasado el análisis léxico y habiéndose comprobado que todas las palabras introducidas son propias del lenguaje, realizaremos el análisis sintáctico del código. En el análisis sintáctico se comprueba la estructura de las frases, si son correctas y están formadas según el lenguaje de programación que hayamos usado.
- **Análisis semántico.** Este análisis se encarga de deducir si todas las instrucciones poseen un significado semántico correcto. Por ejemplo, si tenemos una línea de código donde se asigna una cadena de caracteres a un valor entero, se producirá un error semántico ya que el lugar donde se va a almacenar el dato debería estar preparado para números y no letras.

La fase de síntesis de divide en:

- **Generación de código intermedio.** Existen compiladores que generan un código intermedio. Esta representación intermedia no es aún código máquina de forma que puede ser usada por cualquier hardware, es algo así como un código capaz de ser utilizado por diversos tipos de hardware. Llegados a este punto se han superado las fases de análisis con lo que si existían **errores de compilación** estos se han subsanado.
- **Optimización de código.** La optimización, como su nombre indica, procurará mejorar el código intermedio, de forma qué se consiga un código objeto más rápido de ejecutar.
- **Generación de código objeto.** Tras la etapa de optimización se obtendrá el código objeto preparado para ser usado en una máquina concreta.

Todas las partes descritas están siempre relacionadas con **la tabla de símbolos** y **el manejador de errores.**

Básicamente la **tabla de símbolos** no es más que una estructura que contiene información de todos los tokens que pueden usarse en un lenguaje de programación concreto. Los analizadores léxico, sintáctico y semántico agregarán información a la misma ya que existirán identificadores y otro tipo de elementos que no forman parte de las palabras reservadas del lenguaje.

En cuanto al **manejador de errores,** es la parte encargada de tratar los posibles fallos producidos durante toda la compilación en las diferentes etapas.

Supongamos que hemos usado un programa preparado para escribir código en un lenguaje de programación concreto. Hemos generado un fichero con código fuente y queremos proceder a su compilación para poder ejecutarlo. Este fichero contiene la instrucción **int a=b+2;.** Es una línea muy sencilla, con ella estamos creando un lugar donde guardar información entera llamado a, concretamente el valor a almacenar es la suma de b más 2.

Una vez se han descrito las partes de un compilador, teniendo en cuenta nuestro código fuente, el proceso de compilación se podría reducir a:

1. Recibimos el código fuente, **int a=b+2;** y comenzamos la fase de análisis. Primera etapa a superar, el analizador léxico.
2. Se separa la instrucción formando diferentes tokens. Obtendremos: **int, a, =, b, +, 2,**
3. Los tokens analizados son correctos. Algunos como **int,** =, + se encuentran en la tabla de símbolos con anterioridad, el resto se agrega a esta.
4. Al ser correcto el análisis léxico pasamos al análisis sintáctico.

En este paso debemos aseguramos de que la frase o instrucción sea correcta. Se genera un **árbol sintáctico** por el que se deduce si la instrucción está bien formada en el lenguaje de programación que estamos usando. En todo momento el manejador de errores estará alerta por si se produce algún tipo de error en alguna etapa.

5. Si no se han producido fallos en la estructura de la frase pasaremos a comenzar el análisis semántico. Los errores sintácticos son errores de estructura, frases o instrucciones mal formadas en un lenguaje concreto, por ejemplo si nuestra instrucción hubiera sido **int a==b+2;** se hubiera producido un error sintáctico, ya que en una asignación tras el igual debe aparecer valor o variable y no otro igual. El análisis semántico analizará el significado de la frase. La palabra **int** indica que estamos creando una variable de tipo entera, así, **b** debe serlo para que alrealizar la suma el resultado sea entero y pueda almacenarse en **a.**
6. Si la fase semántica es superada correctamente se generará un código intermedio.
7. A continuación se producirá la optimización del código intermedio.
8. Para finalizar el compilador obtendrá el código objeto, código máquina, listo para ser linkado" o unido a diferentes librerías, y así obtener el fichero final a ejecutar.

JAVA EN EL PROCESO DE CREACIÓN DE CÓDIGO OBJETO

Los lenguajes de programación actuales utilizan un modelo híbrido para la obtención del código objeto. Cuando generamos un código máquina tenemos en cuenta unas características hardware concretas, es decir, estamos obteniendo un código objeto para un procesador concreto. Así, si queremos que un código fuente sea ejecutado en varias arquitecturas será preciso compilar estas para cada una de ellas.

Con el fin de ser más versátiles y puedan generar código fuente traducido a objeto accesible por un mayor número de arquitecturas evitando el proceso de compilación, los lenguajes de programación de hoy día generan un código intermedio. Exactamente crean una máquina virtual para la que desarrolla el código compilado que posteriormente será interpretado por el hardware real en el que se ejecutará.

Así, podemos decir que existen lenguajes de programación híbridos tales que:

- Compilan el código fuente para obtener un código intermedio.
- El código intermedio es interpretado cuando pasa a ser ejecutado sobre una máquina

concreta.