

## 1. INTRODUCCIÓN

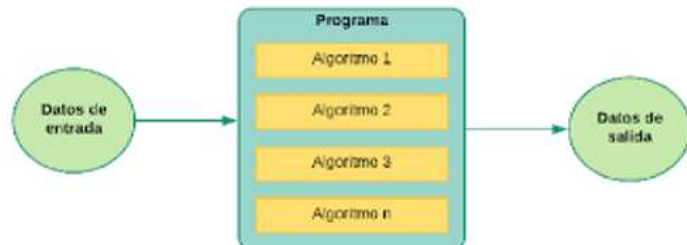
Nesta primeira unidade estudiaranse conceptos xenéricos relacionados coa programación para familiarizarse co vocabulario relacionado e con distintas metodoloxías de análise, deseño e desenvolvemento.

Ao día de hoxe, case todo o que nos rodea está relacionado dunha ou outra maneira coa programación, os programas e o tratamento de datos.

Os programas son a maneira en que distintos dispositivos informáticos (ordenadores, *tablets*, *smartphones*, etc.) levan a cabo un algoritmo que está implementado mediante unha linguaxe de programación.

Un **algoritmo** é un conxunto de pasos finitos que resollen un problema, e cando estes pasos se realizan mediante unha **linguaxe de programación** que é traducido a código executable por un procesador, entón falamos dun **programa informático**.

Outra cousa a resaltar é que todo programa colle datos de entrada para o seu proceso e xera datos de saída como resultado da execución dun algoritmo.



Os programas créanse utilizando una linguaxe de programación que facilitará a implementación dos algoritmos co obxectivo de que poidan se executados nun ordenador.

A **linguaxe de programación** que utilizaremos para implementar os programas será **Java**, un dos linguaxes máis utilizados en todo tipo de soportes informáticos e sistemas operativos.



Para poder crear os nosos programas precisaremos dun software denominado **contorno de desenvolvemento** (IDE - *Integrated Development Environment*) que nos facilitará a escritura do código, a creación de librerías e módulos, así como a depuración e a creación do programa executable correspondente.

**Visual Studio Code**  
Ligero y potente, disponible  
para Mac, Windows y Linux.



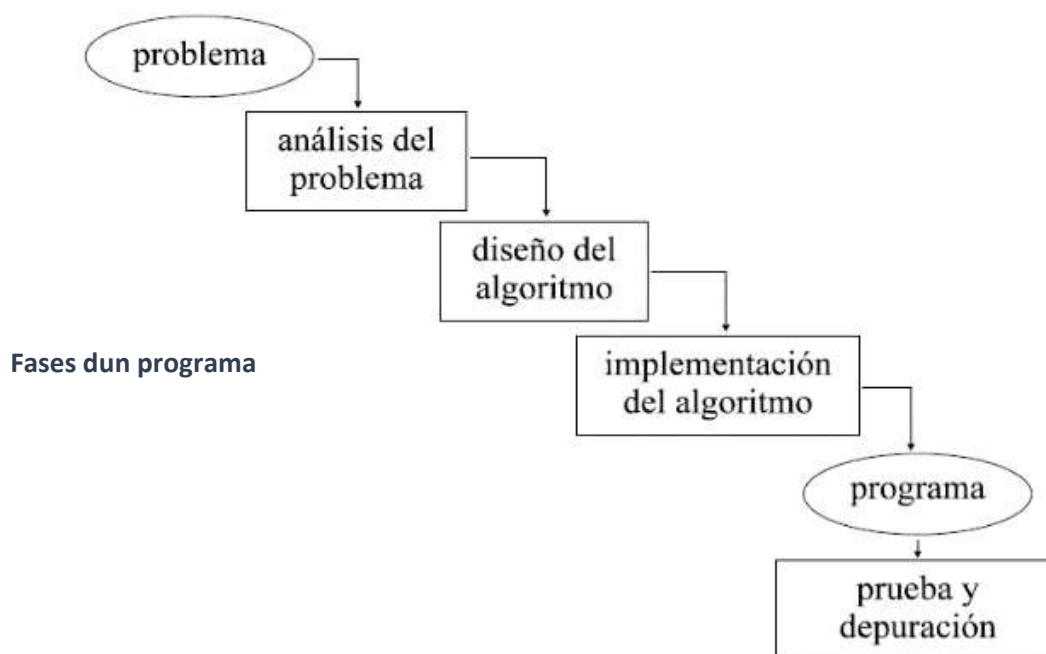
**Eclipse**  
Editor de código abierto, cuenta  
con gran cantidad de complementos.



## 2. FASES DUN PROGRAMA

Un algoritmo é un conxunto de pasos finitos que resollen un problema, e cando estes pasos se realizan mediante unha linguaxe de programación que é traducido a código executable por un procesador, entón falamos dun **programa informático**.

O desenvolvemento dun programa inclúe distintas fases que se mostran a continuación:




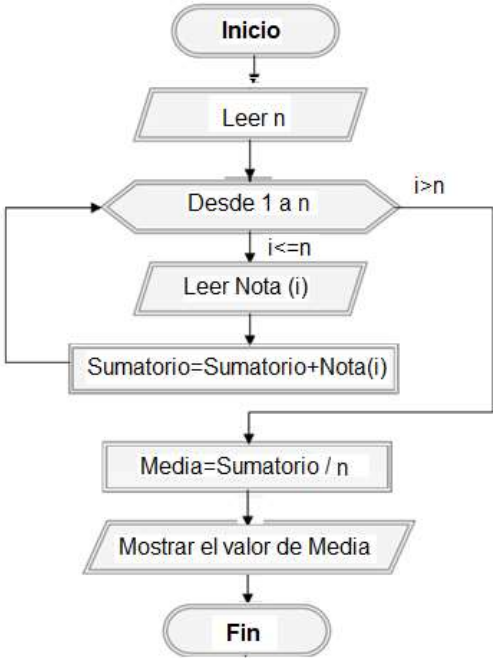
Na seguinte táboa podemos ver a relación das fases coa vida real

| <b>Resolución de problemas</b>                           |   |
|--|---|
| <b>Na vida real...</b>                                   | <b>En Programación...</b>   |
| <b>Observación da situación ou do problema.</b>          | <b>Análise do problema:</b> require que o problema sexa definido e comprendido claramente para que poida ser analizado con todo detalle.                                  |
| <b>Pensamos nas posibles solucións.</b>                  | <b>Deseño do algoritmos:</b> procedemento paso a paso para solucionar o problema dado.  |
| <b>Actuamos segundo a solución mellor.</b>               | <b>Implementación do algoritmo elixido na computadora:</b> consiste en converter o algoritmo en programa, executalo e comprobar que soluciona verdadeiramente o problema. |
| <b>Probamos e corriximos o método se hai algún erro.</b> | <b>Proba e depuración:</b> probamos o programa e corriximos erros de compilación e de execución.  |

*Un programa se está ben deseñado e implementado deberá resolver o problema de forma correcta e facelo no menor tempo posible, ademais de facer un uso óptimo dos recursos do sistema.*

## EXEMPLO

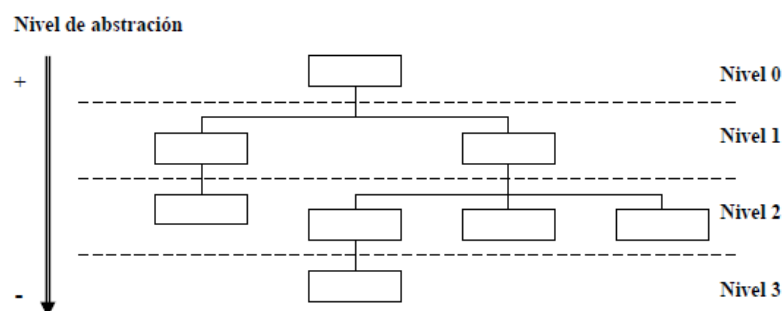
Supoñemos un programa que ten que *calcular a nota media* dun módulo dun curso determinado.

|                |  |
|----------------|--|
| <b>Análise</b> | <ul style="list-style-type: none"><li>• Que valores de entrada imos necesitar? <i>O número de alumnado e a nota de cada un deles.</i></li><li>• Que resultados temos que obter? <i>A nota media do módulo.</i></li><li>• Que procesos imos ter que realizar? <i>Sumar todas as notas e dividir o resultado polo número total de alumnado.</i></li></ul> <div data-bbox="624 607 1193 734"><pre>graph LR; Entrada --&gt; Proceso; Proceso --&gt; Saída;</pre></div>   |
| <b>Diseño</b>  | <p>Con <b>organigramas</b>:</p> <div data-bbox="662 869 1157 1525"><pre>graph TD; Inicio([Inicio]) --&gt; LeerN[/Leer n/]; LeerN --&gt; Desde1aN{Desde 1 a n}; Desde1aN -- "i &lt;= n" --&gt; LeerNotaI[/Leer Nota (i)/]; Desde1aN -- "i &gt; n" --&gt; Media[Media = Sumatorio / n]; LeerNotaI --&gt; Sumatorio[Sumatorio = Sumatorio + Nota(i)]; Sumatorio --&gt; Media; Media --&gt; MostrarMedia[/Mostrar el valor de Media/]; MostrarMedia --&gt; Fin([Fin]);</pre></div> <p>Con <b>pseudocódigo</b>:</p> <ol style="list-style-type: none"><li><b>1. Inicio [Algoritmo Nota Media]</b></li><li><b>2. Leer n</b></li><li><b>3. Desde i = 1 hasta n Hacer</b><ol style="list-style-type: none"><li>2.1 Leer Nota(i)</li><li>2.2 Sumatorio = Sumatorio + Nota(i)</li></ol></li><li><b>Siguiente</b><ol style="list-style-type: none"><li>3. Media = Sumatorio / n</li><li>4. Mostrar "La nota media de la clase es", Media</li></ol></li><li><b>5. Fin</b></li></ol> |

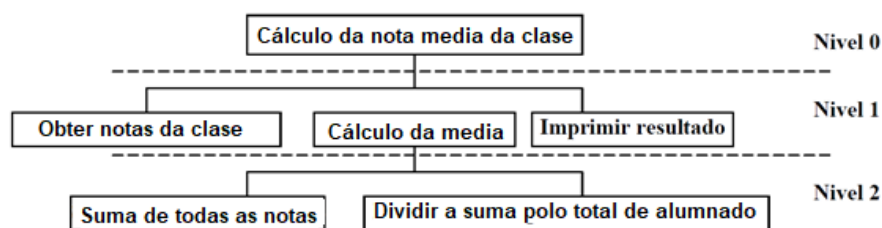
|                |  |
|----------------|--|
| Implementación | <p>Nesta fase teríamos que utilizar unha linguaxe de programación para implementar as instrucións e o fluxo de programa representado na fase de deseño.</p> <pre> import java.util.Scanner;  public class Nota_Media {      public static void main(String[] args) {         Scanner sc = new Scanner(System.in);         int n;         double nota, suma = 0, media;          System.out.println("Nota media de una clase");         System.out.println("-----");          do {             System.out.print("Introduce numero de alumnos: ");             n = sc.nextInt(); //n contiene el número de alumnos         } while (n &lt;= 0); //repetir mientras que el valor de n sea &lt;= 0          for (int i = 1; i &lt;= n; i++) { // leemos la nota de cada alumno             System.out.print("Nota del alumno " + i + ": ");             nota = sc.nextDouble();             suma = suma + nota; // la sumamos al acumulador         }         media=suma/n;          // mostrar resultados         System.out.printf("Nota media: %.2f %n", media);     } } </pre> |
|----------------|--|

### 3. TÉCNICAS DE PROGRAMACIÓN

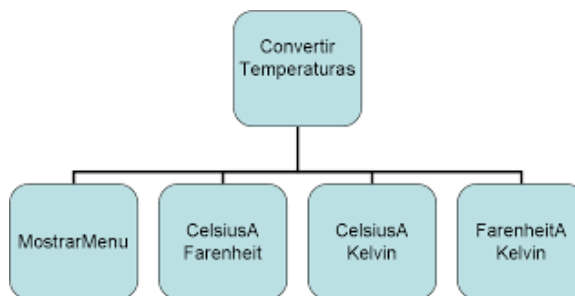
- **Diseño descendente:** Técnica que ten como obxectivo descompoñer un problema complexo en problemas mais sinxelos, ata chegar ao mínimo nivel de abstracción, no cal pódense codificar as operacións nunha linguaxe de programación.



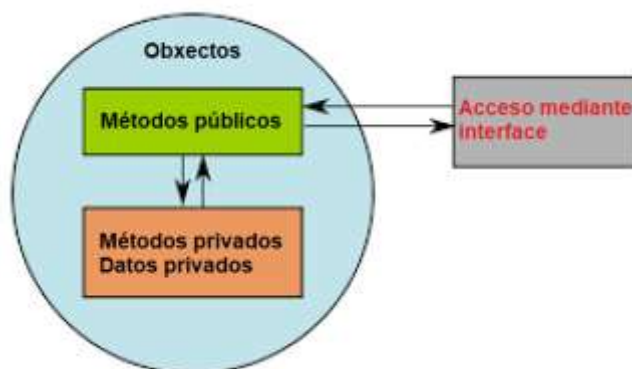
O *Nivel 0* é o máis abstracto, é a descrición do problema. O último nivel é o que permite a codificación nunha linguaxe de programación concreto



- **Modularidad:** As instrucións dos algoritmo organizaranse en módulos independentes, o que facilitará a reutilización do código e o control da súa estrutura. Por exemplo, se temos un módulo que se encarga de calcular o IVA, este módulo poderase reutilizar en distintas partes do programa, incluso en diferentes programas e poderase modificar sen que afecte a outros módulos.



- **Encapsulación:** O código que está encapsulado permite o seu uso mediante unha interface específica que se pode utilizar sen coñecer como está implementado, as modificación que se fagan no código non teñen porque afectar ao uso de súa interface.




---

## 4. PARADIGMAS DA PROGRAMACIÓN

---

Existen diferentes paradigmas de programación que se foron desenvolvendo condicionados pola evolución tecnolóxica e os diferentes problemas que tiñan que resolver os programas.

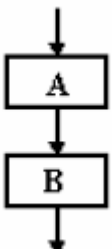
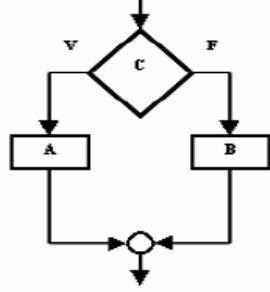
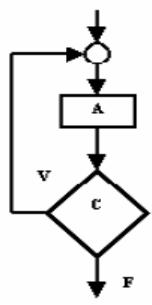
### 1. Paradigma imperativo.

Os programas consisten nunha sucesión de instrucións o conxunto de sentencias, o programador vai describindo paso a paso o que fai o programa. Este paradigma céntrase en *como se debe calcular o problema o resolver*.

Os seguintes tipos de programación responden ao paradigma imperativo:

- **Programación convencional.** Este modelo de programación utiliza instrucións sen seguir normas que faciliten a reutilización do código nin a súa modificación. Caracterízase por usar a instrución GOTO que modifica o fluxo do programa facendo que o código sexa moi pouco claro. Unha linguaxe que utilizaba con este tipo de programación era *BASIC*.

- **Programación estruturada.** Resolve os problemas da programación convencional. Este modelo contempla o uso exclusivo das estruturas de control que se mostran na seguinte táboa.

|   |   |
|---|---|
| <p><b>Estruturas secuenciais</b></p> <p>As instrucións se executan unha despois da outra.</p>   |  <pre> graph TD     Start(( )) --&gt; A[A]     A --&gt; B[B]     B --&gt; End(( )) </pre>  |
| <p><b>Estruturas condicionais</b></p> <p>Dada unha condición, o fluxo do programa vai por un conxunto de instrucións ou por outro. Normalmente correspóndese coa instrución IF.</p> |  <pre> graph TD     Start(( )) --&gt; C{C}     C -- V --&gt; A[A]     C -- F --&gt; B[B]     A --&gt; Join(( ))     B --&gt; Join     Join --&gt; End(( )) </pre> |
| <p><b>Estruturas iterativas</b></p> <p>Unha instrución ou grupo delas repítense ata que se cumpra unha condición determinada que ten que variar no bloque que se itera.</p>         |  <pre> graph TD     Start(( )) --&gt; Join(( ))     Join --&gt; A[A]     A --&gt; C{C}     C -- V --&gt; Join     C -- F --&gt; End(( )) </pre>                 |

- **Programación modular,** a cal permite crear bloques de código con entidade que poden ser reutilizados no mesmo programa ou en outros distintos. Exemplos de linguaxes de programación estruturada e modular son: *PASCAL, COBOL e C/C++*.

## 2. Paradigma declarativo.

Este paradigma non precisa definir algoritmos, a súa metodoloxía consiste en describir *que se debe calcular, sen explicar como*.

As sentencias que se utilizan describen o problema que se quere solucionar pero non as instrucións necesarias para levalas a cabo.

O paradigma declarativo subdivídese en outros dous:

- **Programación lóxica.** Neste tipo de programación, especificase o *que hai que facer pero non como facelo*. Utilízase en *intelixencia artificial*. Un exemplo desde modelo de programación e a linguaxe Prolog.
- **Programación funcional:** Este modelo de programación ten como *elemento principal funcións independentes unhas de outras* que fan operacións específicas. Lisp, Scala e Miranda son linguaxes que permiten a programación funcional.

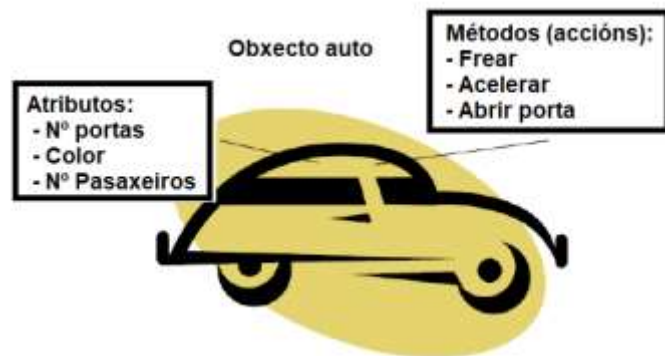
### 3. Programación orientada a obxectos.

Este paradigma ten un enfoque baseado en **obxectos** a diferenza dos esquemas tradicionais nos que a solución especificase en pasos. No esquema da POO (*Programación Orientada a Obxectos*) pártese da premisa “O mundo real está composto por obxectos que interactúan” e non é un mundo procedural no que todo ocorre secuencialmente.

Exemplos de linguaxes POO son Java, C++, PHP, C#, etc.

A POO ten como eixo central o uso de obxectos, que en realidade son estruturas lóxicas que representan elementos ou entidades no mundo real e que se definen por medio de **clases**. Esta representación baséase nos seguintes aspectos:

- Uso de **identificadores** que permiten facer unha rápida asociación co o obxecto a que se refiren. Por exemplo, pódese definir un obxecto ou entidade denominado *auto* que represente un coche no mundo real.
- Para representar o estado do obxecto utilízanse variables denominadas **atributos** ou **propiedades** que almacenan diferentes valores que poden ir cambiando no tempo. Por exemplo, el *auto* pode estar representado por *velocidade* ou *potencia*.
- Para describir o comportamento do obxecto e como este interactúa con outros do seu mesmo tipo ou de outro tipo defínense **métodos**, que se corresponden a accións específicas do obxecto, simulando o mundo real. Por exemplo, *avanzar*, *frear*, *abrir a porta dianteira dereita*, etc.



A **modularidade** e a **encapsulación** son dous conceptos estreitamente ligados a POO.

Na POO para encapsular obxectos utilízanse os **TAD** (*Tipos Abstractos de datos*) que **permiten traballar cos datos sen ter en conta como están construídos**, de tal forma que si se producen cambios na implementación do tipo de dato, estes non afectarán ao programa que os utiliza. Para poder facer uso destes datos sen coñecer a súa implementación, defínese unha interface que especifica como se pode interactúa con eses datos e obxectos.

A outra gran **vantaxe do esquema de POO** é a posibilidade de **reutilizar** partes dunha solución para resolver un segundo problema, onde se comparte parte da lóxica resolta no primeiro problema gracias a que todo o código está estruturado en módulos que, en xeral, denomínanse **clases** que poden relacionarse entre si.

*Consideremos o caso dun coche.* Somos capaces de conducir distintos coches, de distintas marcas e modelos, sen necesitar saber como funciona cada un dos seus compoñentes. De feito, non todos os coches teñen os mesmos compoñentes. Mesmo, os compoñentes que son comúns, a pesar de ter a mesma funcionalidade, difiren en modelos, rendemento, custo, etc. En resumo, a implementación é distinta pero teñen a mesma funcionalidade.

De maneira similar, unha implementación particular dun tipo de datos é independente dos programas que os usan. O único intocable é a interface. Explicaremos este concepto de interface volvendo ao exemplo do coche. Todas as marcas e modelos posúen uns elementos comúns que determinan o uso do coche (forma de usar o volante, o acelerador, o freo, etc.). Ninguén compraría un coche en que o freo se atope á dereita do acelerador, ou que o volante funcione virando o coche á dereita cando é move á esquerda.

Resumindo, a POO caracterízase por:

- O deseño modular e a reutilización do software.
- Os TAD (Tipo Abstracto de Datos)
- A representación da solución do problema polas súas clases e as súas relacións

---

## 5. TIPOS DE LINGUAXES DE PROGRAMACIÓN

---

### CLASIFICACIÓN SEGUNDO A SÚA PROXIMIDADE AO HARDWARE

En función dos procesos que ten que facer un programa desde que o escribe o programador ata que é interpretado por un procesador podemos diferenciar diferentes tipos de linguaxes de programación.

- Linguaxes de baixo nivel.
- Linguaxes de alto nivel.

Antes de nada temos que recordar que

- Os procesadores traballan con bits.
- Unidade de memoria básica: Byte (8 bits).
- Nun byte pódense representar dous números hexadecimais.

01011011 → 0101 1011 → 5B

### Linguaxes de baixo nivel

As linguaxes de baixo nivel comunícanse co procesador con mínimos procesos de tradución.

As linguaxe de baixo nivel son a linguaxe **máquina** e a linguaxe **ensamblador**.

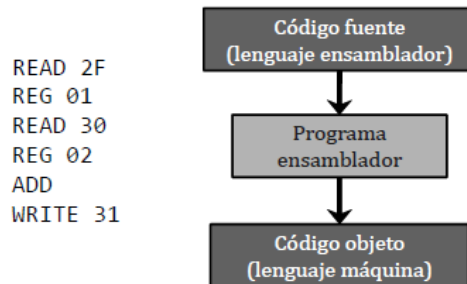
- **Linguaxe máquina:**
  - As instrucións en linguaxe máquina escríbense en linguaxe hexadecimal.
  - Utiliza a mesma linguaxe que utiliza o procesador.
  - Depende ao 100% da arquitectura do procesador.

| Instrucións en | Significado  |
|----------------|--|
| A0 2F          | Acceder a posición de memoria 2F                           |
| 3E 01          | Copialo ao rexistro 1 da Unidade Aritmético o Lóxica (ALU) |
| A0 30          | Acceder a posición de memoria 30                           |
| 3E 02          | Copialo ao rexistro 2 da ALU                               |
| 1D             | Sumar o contido dos rexistros                              |
| B3 31          | Gardar o resultado da suma na posición de memoria 31       |



- **Linguaxe ensamblador:**

- As súas instrucións consisten en nomes nemotécnicos para os códigos hexadecimais.
- Cada instrución en ensamblador tradúcese en 1 ou 2 instrucións máquina.
- Ten unha elevada dependencia do procesador.



## Linguaxes de alto nivel

As linguaxes de programación de alto nivel caracterízanse por:

- Son próximos ao linguaxe natural e matemático o que facilita moito o desenvolvemento de programas.

```
resultado = dato1 + dato2;
```

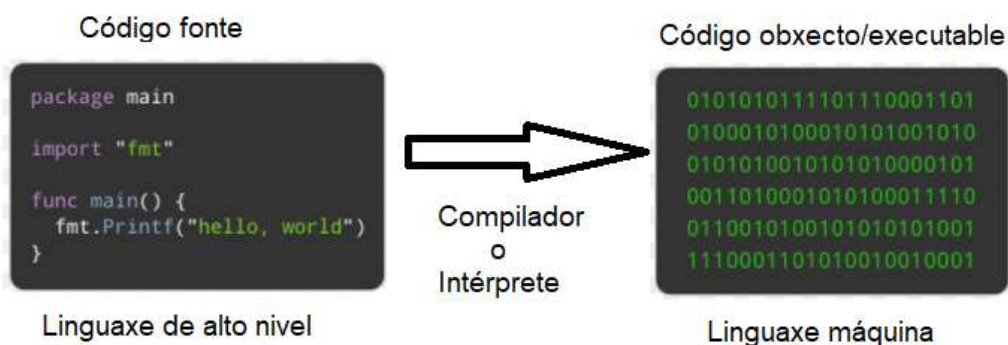
- Teñen un gran independencia da máquina.
- Cada instrución xera moitas instrucións máquina.

As linguaxes de alto nivel permiten especificar de forma abstracta estruturas de datos e escribir instrucións que levan a cabo o algoritmo que traballa con eses datos.

Por todo isto, cando se vai desenvolver un programa temos que ter en conta a estrutura dos datos que se corresponderán coas entradas e saídas do problema a resolver.

Exemplos de linguaxes de alto nivel son: C++, Java, C#, Python, etc.

Os programas codificados en linguaxes de alto nivel teñen que pasar por un proceso de tradución que teñen como obxectivo traducir as instrucións en linguaxe máquina para que poda ser executado polo procesador.



## CLASIFICACIÓN SEGUNDO A TÉCNICA DE CONVERSIÓN A BINARIO

También pódense clasificar as linguaxes de programación segundo o método que se utiliza para converter o código fonte a binario. Así podemos falar de:

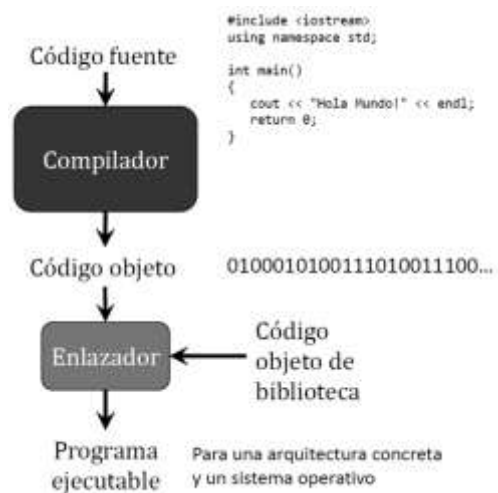
- Linguaxes compilados.
- Linguaxes interpretados.
- Linguaxes que usan máquina virtual.

### Linguaxes compilados

Son linguaxes que precisan un compilador para converter o código fonte en código máquina, xerando un arquivo co código obxecto que precisa un *enlazador* para crear o código executable. Cando se executa o código, non precisa de ningunha tradución en tempo real.

Cando se programa con utilizando unha linguaxe compilada xéranse os seguintes arquivos:

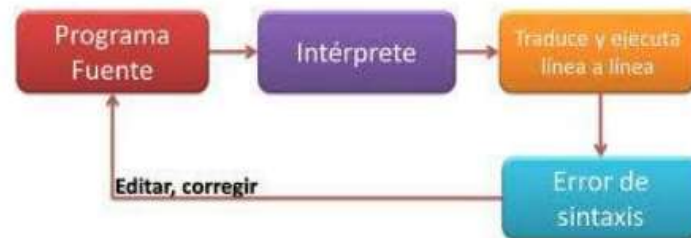
1. **Programa fonte:** é o código tal e como o escribe o programador, para facer esta operación utilízase un **editor**.
2. **Programa obxecto:** é o programa fonte traducido a linguaxe máquina.
3. **Programa executable:** é o obxecto que está enlazado con librerías e outros elementos necesarios para a súa execución; esta tarefa é realizada por un montador (*linker*) que xera o programa executable que xa pode ser executado directamente polo procesador.



Exemplos de linguaxes compilados son: C++, Fortran, Pascal, Visual Basic, etc.

## Linguaxes interpretados

O código fonte é traducido polo intérprete a código máquina á vez que se executa. Só hai código fonte e código executable. Cada vez que se executa o código é preciso unha tradución en tempo real.

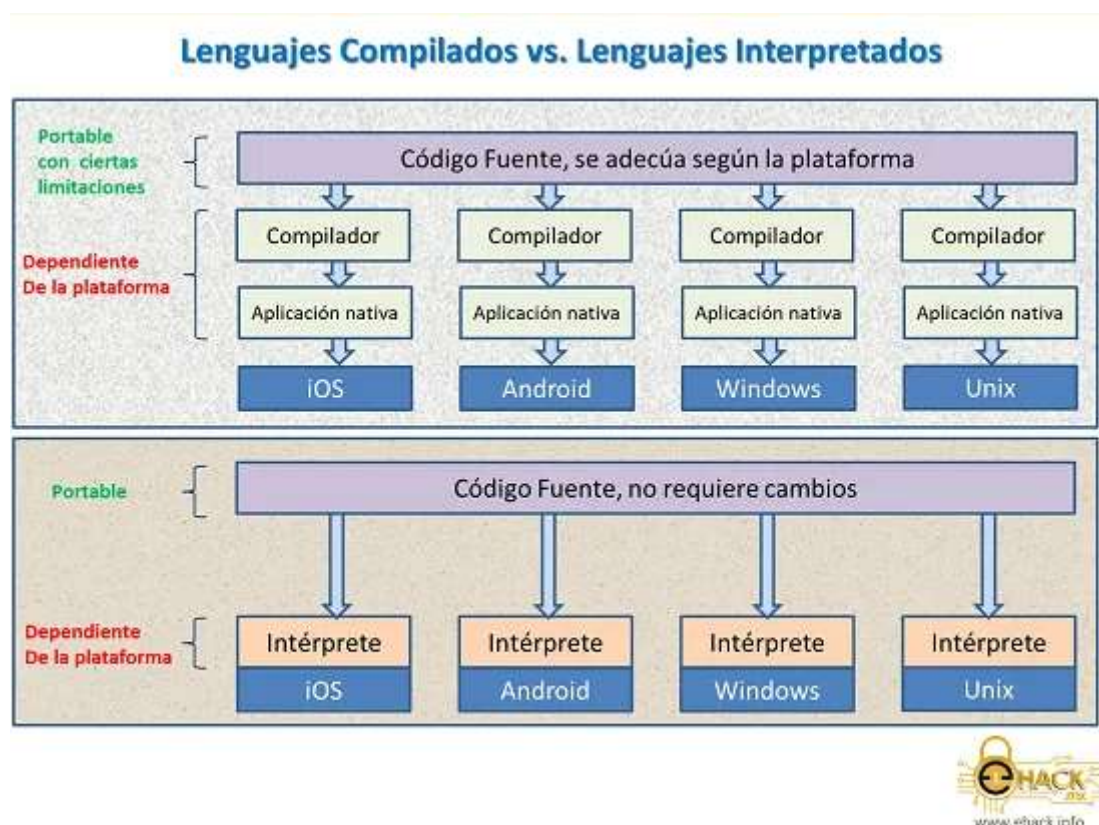


Exemplos de linguaxes interpretados son: Ruby, Python, PHP, JavaScript, Perl, Smalltalk, MATLAB, Mathematica.

## Linguaxes compilados vs linguaxes interpretados

Unha das **vantaxes das linguaxes interpretados fronte as linguaxes compilados** é a súa independencia da plataforma onde se executa.

Unha das **vantaxes das linguaxes compilados fronte as linguaxes interpretados** é que o código executable é mais eficiente e rápido.



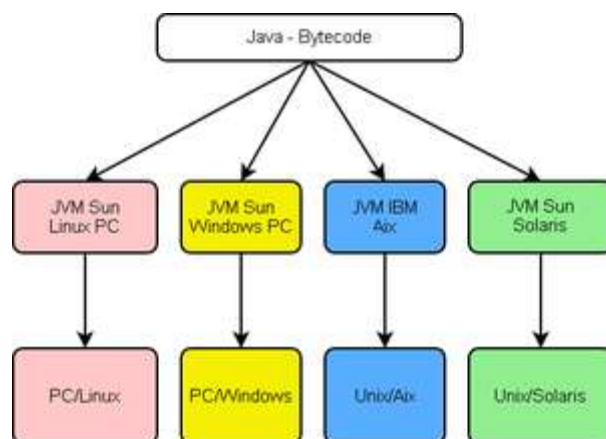
## Linguaxes que usan máquina virtual

As linguaxes que utilizan máquina virtual precisan de unha capa de software chamado máquina virtual que traduce o código obxecto do programa a código máquina da arquitectura específica onde se executa.

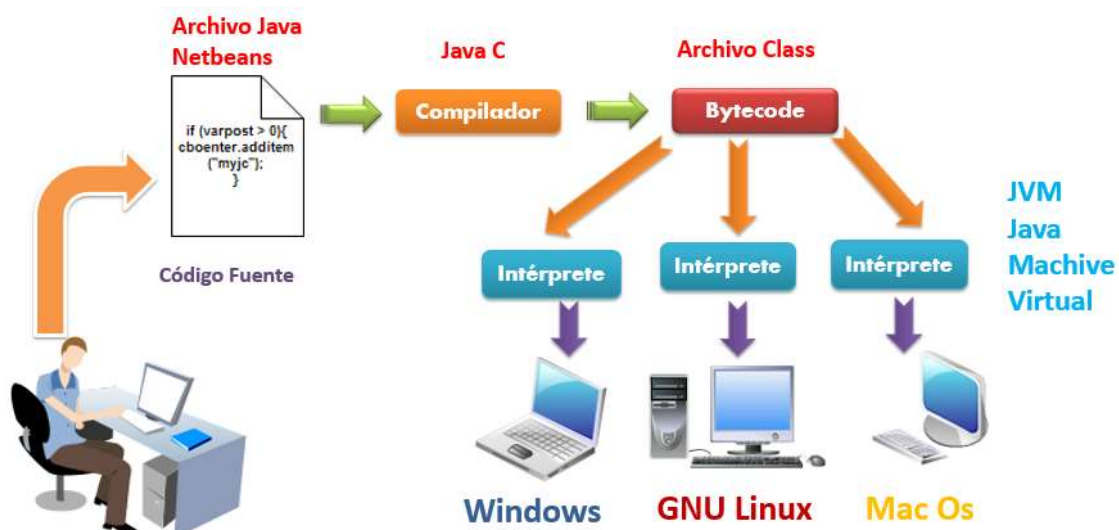
Explicaremos aquí o funcionamento da máquina virtual Java, pero de forma similar funcionan as máquinas virtuais dos contornos .NET (C#, Visual Basic .NET), e a programación para os sistemas Android e iOS.

Unha máquina virtual:

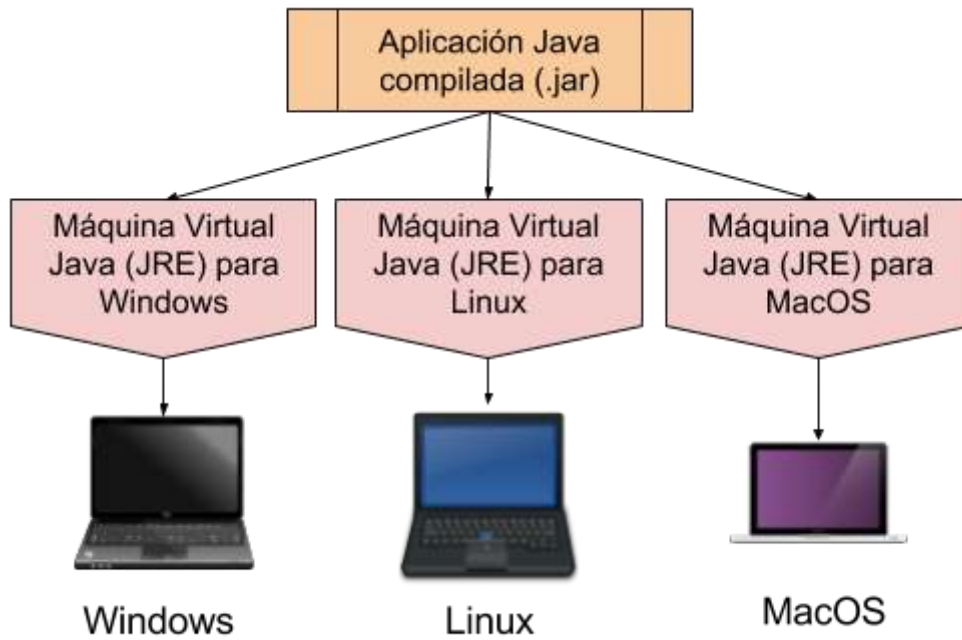
- é un software que pode executar programas coma se fose unha máquina física,
- instálase sobre un sistema operativo,
- cada sistema operativo precisa dunha máquina virtual específica,
- os programas son codificados nun código que toda máquina virtual do mesmo linguaxe de programación comprende, por exemplo, a linguaxe de máquina virtual de Java é o *bytecode*.
- a máquina virtual instalada é a responsable de *traducir* o código do programa a código executable pola máquina/sistema operativo específico.



Funcionamento de la máquina virtual de Java, unha de las máquinas virtuais de proceso máis populares [Fonte da imaxe: Wikipedia]



[Fonte da imaxe: IncanatoIT.com]



[Fonte da imaxe: javiergarciaescobedo.es]