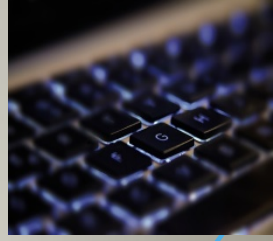
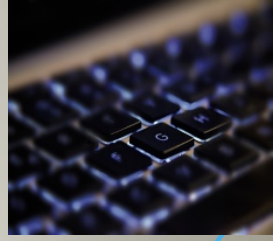


Clase 6
Introducción a la Programación Orientada a Objetos (POO)

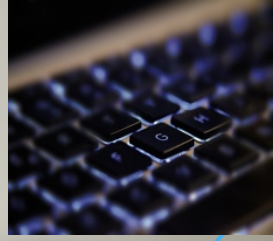
Conceptos previos a la POO: Tipos abstractos de Datos



- Definiciones básicas
  - Objetos
  - Mensajes y métodos
  - Clases, subclases y objetos
  - Herencia

- Conceptos claves
  - Encapsulamiento
  - Abstracción
  - Polimorfismo
  - Persistencia

¿Cómo evolucionó la programación?



PROGRAMACIÓN NO ESTRUCTURADA

- Flujo de programas de difícil seguimiento. Uso indiscriminado sentencia GOTO.
- Años 60. Ejemplos: Fortran, Basic, primeras versiones Cobol.

PROGRAMACIÓN ESTRUCTURADA

- Ordenación del flujo de programas. Restricción de la sentencia GOTO.
- Años 70. Ejemplos: Pascal, C, Ada.

PROGRAMACIÓN MODULAR

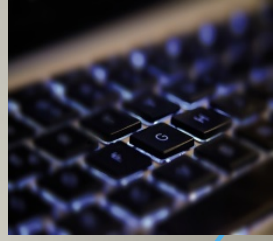
- Ordenación del código en módulos, funciones, procedimientos.
- Años 80. Admitida por la mayoría de los lenguajes.

PROGRAMACIÓN QUE USA OBJETOS

- Existencia de objetos con propiedades y métodos disponibles para el programador.
- Años 90. Ejemplos: Visual Basic.

PROGRAMACIÓN ORIENTADA A OBJETOS

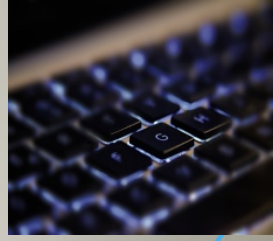
- Los programas se conciben como un conjunto de objetos representación del problema real.
- Años 90 y 2000. Introducida por numerosos lenguajes. Visual Basic.NET, Java.



Conceptos previos a la POO

Caracterizar los problemas del mundo real significa reconocer los objetos del mundo real y abstraer sus aspectos fundamentales y su comportamiento, de modo de representarlos sobre una computadora.

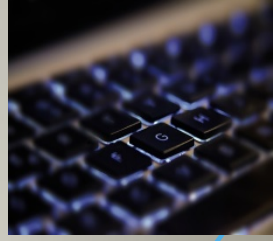
La utilidad fundamental de abstraer y modelizar objetos del mundo real es re-usar soluciones en diferentes problemas.



Conceptos previos a la POO

Los ***tipos de datos*** son necesarios para identificar valores y operaciones posibles para variables y expresiones.

Las nociones de estructuras de datos, variables y constantes que se han estudiado son abstracciones para tratar de acercar al mundo real la especificación de los datos de problemas.

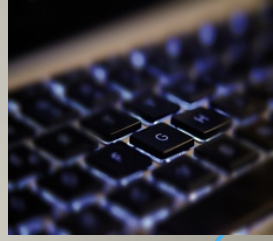


Conceptos previos a la POO

- Los lenguajes de programación y los sistemas operativos se encargan de manejar las conversiones entre las definiciones abstractas y la representación interna en memoria.

- Hasta ahora se han considerado tipos de datos simples, estructurados y definidos por el usuario.

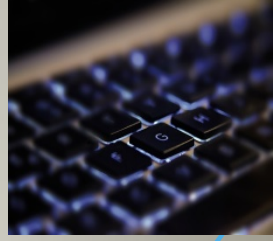
-



Conceptos previos a la POO

La mayoría de los lenguajes de programación modernos impone la obligación de declarar todos los nombres de variables, asociándoles un tipo.

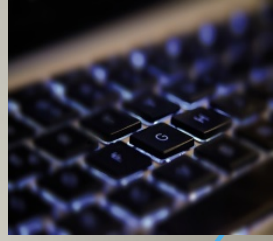
Los lenguajes también verifican que las operaciones se hagan entre datos del mismo tipo. La mezcla de tipos en general es detectada y considerada un error por los lenguajes de programación.



Conceptos previos a la POO

También hemos visto la ventaja de descomponer (modularizar) un problema (sistema de software) en procedimientos y funciones.

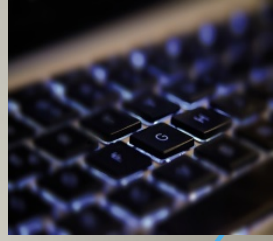
De este modo se abstraen las operaciones del sistema, de modo de descomponerlo en funcionalidades (procedimientos/funciones) relativamente independientes y más simples que el problema global.



Conceptos previos a la POO

En el desarrollo de sistemas de software es importante trabajar con especificaciones abstractas, de modo de mejorar la calidad de los programas.

Los tipos definidos por el usuario permiten realizar encapsulamiento o empaquetamiento de los datos: se define un nuevo tipo y se integran en un módulo todas las operaciones que se pueden hacer con él.

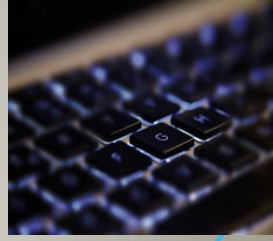


Conceptos previos a la POO

Si el lenguaje permite separar la parte visible (interfaz) de la implementación se tendrá ocultamiento de datos (data hiding).

Si se logra que en la solución del módulo los cambios en los tipos de datos internos NO afecten la parte visible del módulo, se tendrá independencia de la representación.

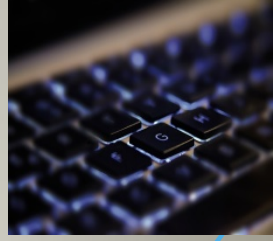
Conceptos previos a la POO



El concepto de tipo abstracto de dato es un tipo de dato definido por el programador que incluye:

- Especificación de la representación de los elementos del tipo.
- Especificación de las operaciones permitidas para el tipo. (visible)
- Desarrollo interno de cada una de las operaciones permitidas, con independencia para el usuario de cómo se implementan.
- Encapsulamiento de todo, de manera que el usuario no pueda manipular los datos del objeto excepto por el uso de las operaciones definidas en el punto anterior.

Conceptos previos a la POO



Tipos de datos abstractos (TAD)

Representación, valores y operaciones (programador)

Mayor abstracción

Estructuras de datos

Representación (lenguaje) , Operaciones y Valores (Programador)

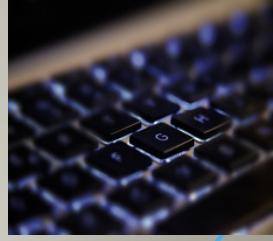
Tipos de datos definidos por el usuario

Representación y operaciones (lenguaje), Valores (Programador)

Menor abstracción

Tipos de datos definidos por el lenguaje

Representación y operaciones (lenguaje), Valores (Lenguaje)



¿De donde viene la POO?

Programación Estructurada

La idea principal de esta forma de programación es separar las partes complejas del programa en módulos, que sean ejecutados a medida que sean necesarios. Estos módulos son independientes entre sí, y además deben poder comunicarse.

Problemas de la Programación Estructurada

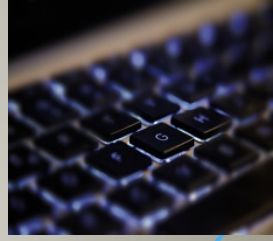
Varios programadores trabajan en equipo desarrollando una aplicación grande.

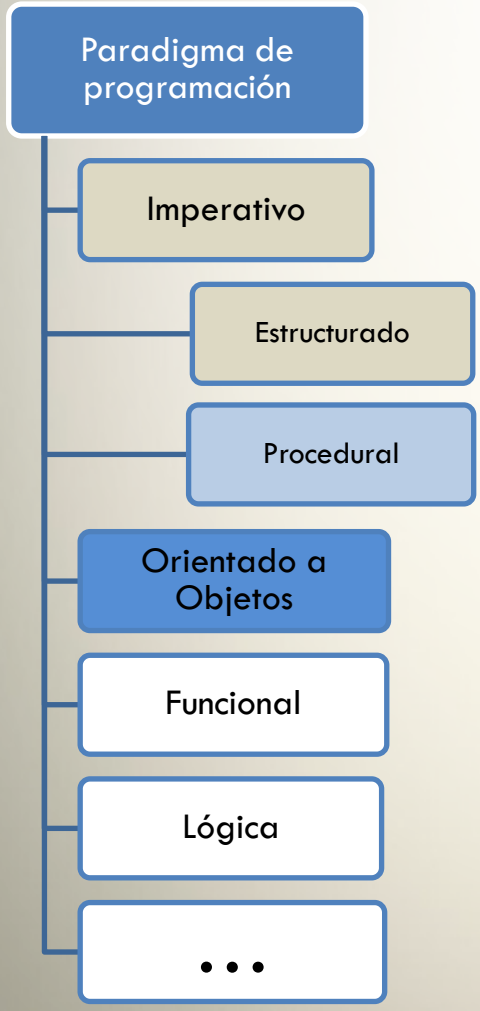
Mas de un programador manipula funciones separadas que pueden referirse a tipos de datos mutuamente compartidos, y los cambios de un programador se deben reflejar en el trabajo del resto del equipo.

Qué pasa si uno de los programadores decide que una estructura existente en el sistema en vez de representarse con una lista, ahora se representa con un arreglo?

Este es uno de los problemas de la programación estructurada, por lo cual se siguió investigando sobre diferentes metodologías de programación.

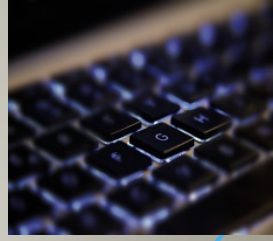
Paradigmas de programación



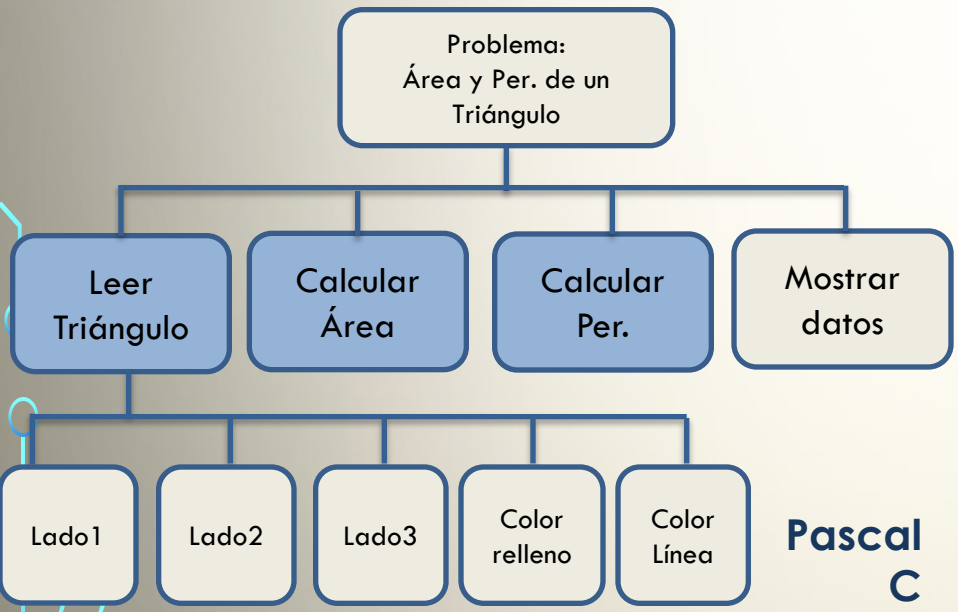


- Indica la manera de estructurar y organizar las tareas de nuestro programa.
- Los lenguajes de programación suelen ser multiparadigma.
- Hasta ahora: Imperativo + Estructurado + Procedural.
- POO

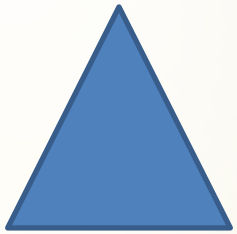
Paradigmas de programación

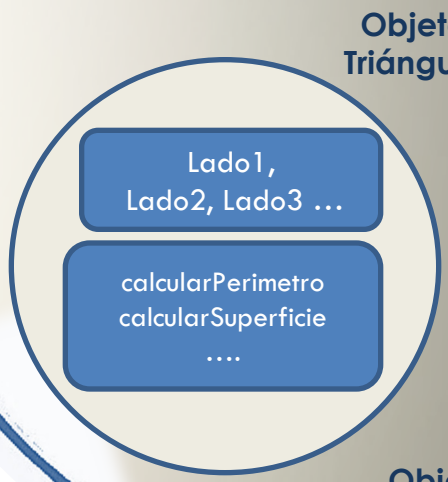


- Desarrollo estructurado

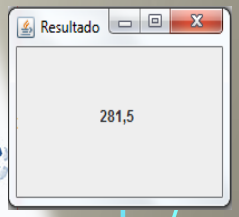


- Desarrollo Orientado a Objetos





Objeto Ventana



Pascal
C
Fortran
…

Smalltalk
Java
C++
…

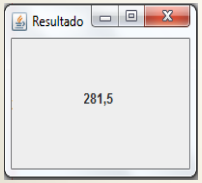
¿Qué paradigma utilizar?

OBJETO

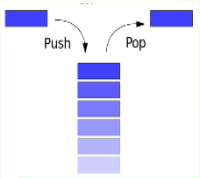
- Objeto: ***abstracción*** de un objeto del mundo real, definiendo qué lo caracteriza y qué acciones sabe realizar (*comportamiento*).
- ¿Qué cosas son objetos? ***"Todo es un objeto"***

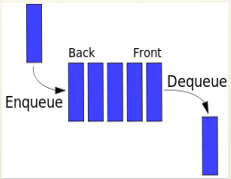
Objetos Físicos

Elementos de interfaces gráficas



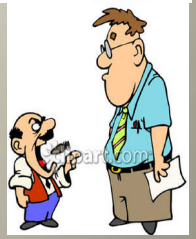
Estructuras de datos

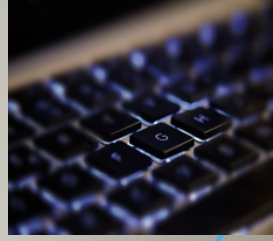




Seres vivos

Roles





OBJETO

Características:
Raza
Edad en años
Color pelaje

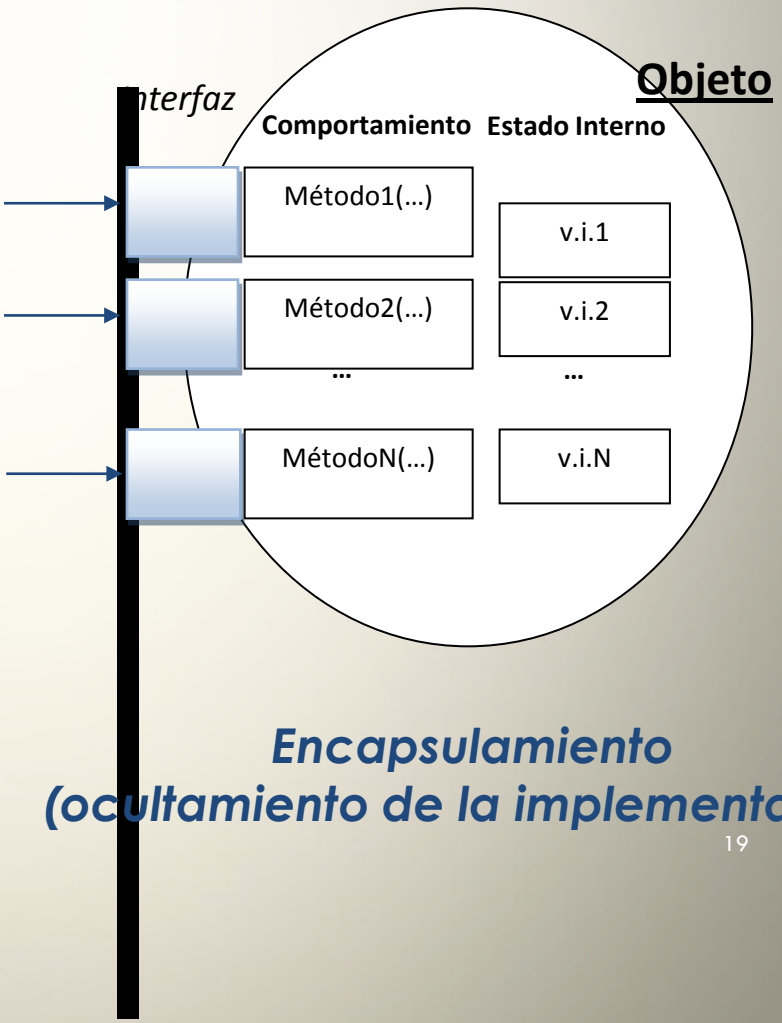
Comportamiento:
ladrar / gruñir / aullar
(entre otras)

Características:
Marca
Patente
Color
Caballos de fuerza
Comportamiento :
arrancar / frenar / acelerar
(entre otras)

Características:
Lado1
Lado2
Lado3
Color de línea
Color de relleno
Comportamiento :
calcular área / calcular perímetro /
(entre otras)

OBJETO

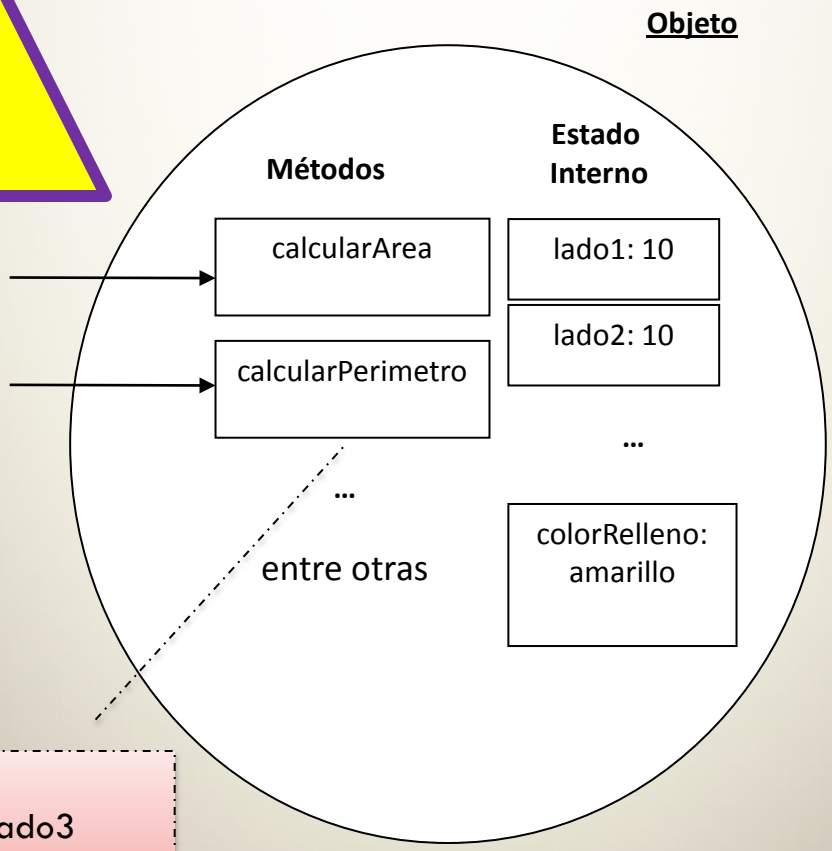
- *Objeto:* entidad que combina en una unidad
  - *Estado interno*: datos/atributos que caracterizan al objeto, serán implementados a través de **variables de instancia.**
  - *Comportamiento:* son las acciones/servicios a los que sabe responder el objeto. Se implementan a través de **métodos de instancia** que operan sobre el estado interno. Los servicios que ofrece al exterior constituyen la *interfaz*.



Encapsulamiento (ocultamiento de la implementación)

OBJETO

Todo cómputo en la aplicación es realizado por objetos

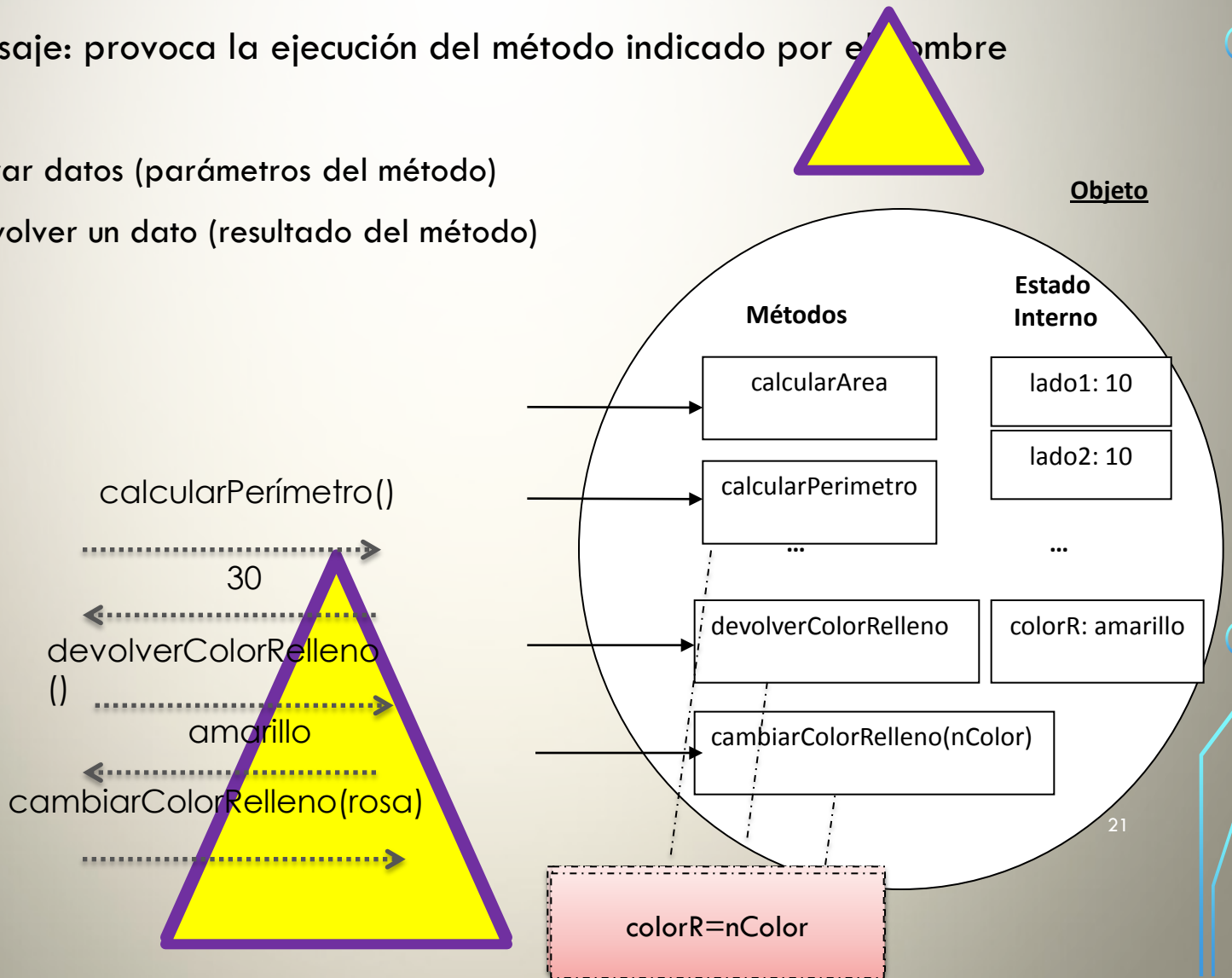


¿Cómo le pido al objeto que calcule el perímetro y me lo devuelva?

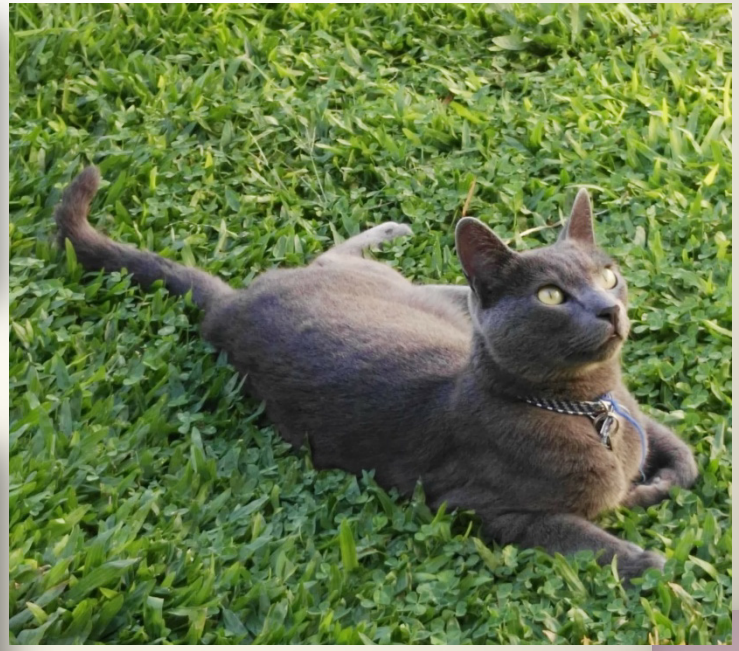
return lado1+lado2 + lado3

MENSAJE

- Envío de Mensaje: provoca la ejecución del método indicado por el nombre del mensaje.
  - Puede llevar datos (parámetros del método)
  - Puede devolver un dato (resultado del método)

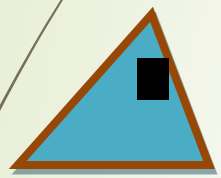


¿Que tenemos en esta imagen?



CLASE

- ¿Cuántos objetos ves?



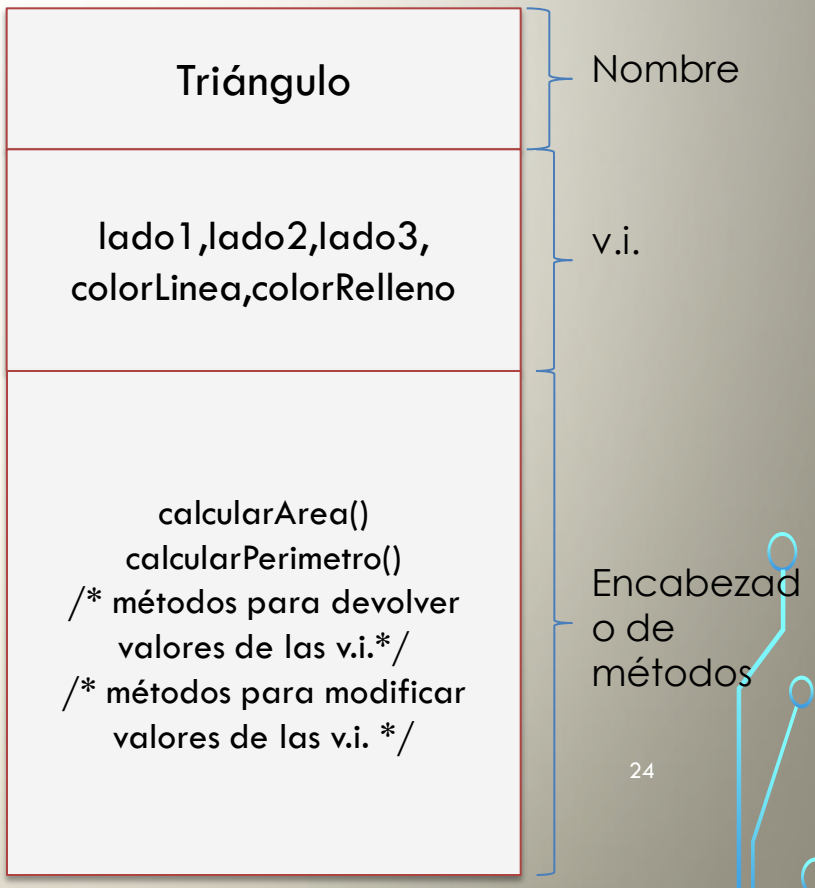
Métodos	Estado Interno
calcularArea	lado1: 10
calcularPerimetro	lado2: 20
...	...
devolverColorRelleno	colorR: azul
cambiarColorRelleno(nColor)	

Métodos	Estado Interno
calcularArea	lado1: 10
calcularPerimetro	lado2: 10
...	...
devolverColorRelleno	colorR: amarillo
cambiarColorRelleno(nColor)	

CLASE

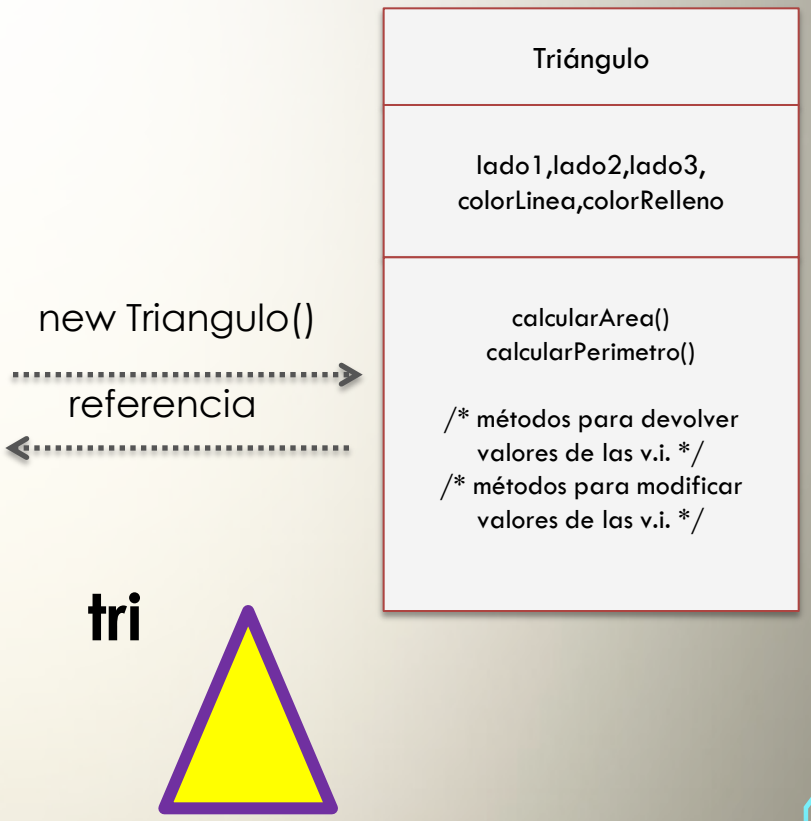
- Una *clase* describe un conjunto de objetos comunes. Consta de:
  - La declaración de las v.i. que implementan el estado del objeto.
  - La codificación de los métodos que implementan su comportamiento.
- Un objeto se crea a partir de una clase, decimos que el *objeto* es *instancia* de una clase.

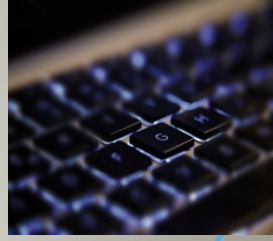
- Representación gráfica



INSTANCIACIÓN

- La *instanciación* se realiza enviando un mensaje de creación a la clase.
  - Alocación de espacio para el objeto.
  - Ejecución el código inicializador o *constructor* (se verá más adelante)
- Devuelve la referencia al objeto.
- Asociar la referencia a una variable (a través de ella podemos enviarle mensajes al objeto).





Programa orientado a objetos

- Los programas se organizan como una colección de ***objetos*** que cooperan entre sí enviándose mensajes.

- Cada objeto es instancia de una ***clase***.

- Los objetos se crean a medida que se necesitan.
- El usuario le envía un mensaje a un objeto, en caso de que un objeto conozca a otro puede enviarle un mensaje, así los mensajes fluyen por el sistema.
- Cuando los objetos ya no son necesarios se borran de la memoria.

-

¿Cómo desarrollar un programa orientado a objetos (POO)?

Pasos:

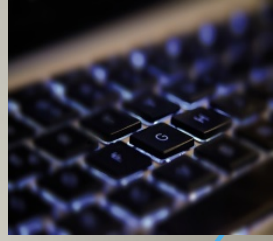
- Identificar los objetos a abstraer en nuestra aplicación.
  - Identificar las características relevantes de los objetos
  - Identificar las acciones relevantes que realizan los objetos
- Los objetos con características y comportamiento similar serán instancia de una misma *clase*.
- Encontrar relaciones entre objetos de distintas clases.
- Encontrar similitudes entre objetos de distintas clases y establecer relaciones de herencia.

¿Cómo desarrollar un programa orientado a objetos (POO)?

Grady Booch resume la diferencia de la siguiente forma:

"Lea las especificaciones del sistema que desea construir. Subraye los verbos si persigue un código procedimental, o los sustantivos si su objetivo es un programa orientado a objetos".

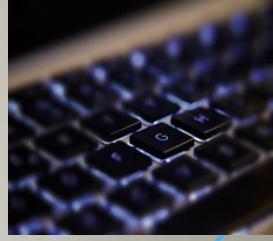
Programación Orientada a Objetos



Ejemplo: una computadora está compuesta por varios componentes (tarjeta madre, chip, disco duro y otros), el trabajo en conjunto de todos ellos hace operar a una computadora. El usuario no necesita saber como trabajan internamente cada uno de estos componentes, sino como es la interacción con ellos. Es decir, cuando se conoce como interaccionan los componentes entre sí, el usuario podrá armar la computadora.

Relación con la Programación Orientada a Objetos

Introducción - POO



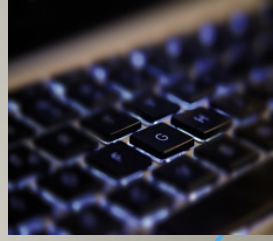
¿Qué es lo que tienen en común?

Se podría encontrar una forma de definir "algo" que encapsule las características y comportamiento comunes

Modelo
Marca
Color
Velocidad

Acelerar
Desacelerar
Apagar
Arrancar

Clase AUTO



¿Cómo deberíamos implementar la clase auto?

nombre

Auto

características

marca
modelo
color
velocidad
capacidad baúl

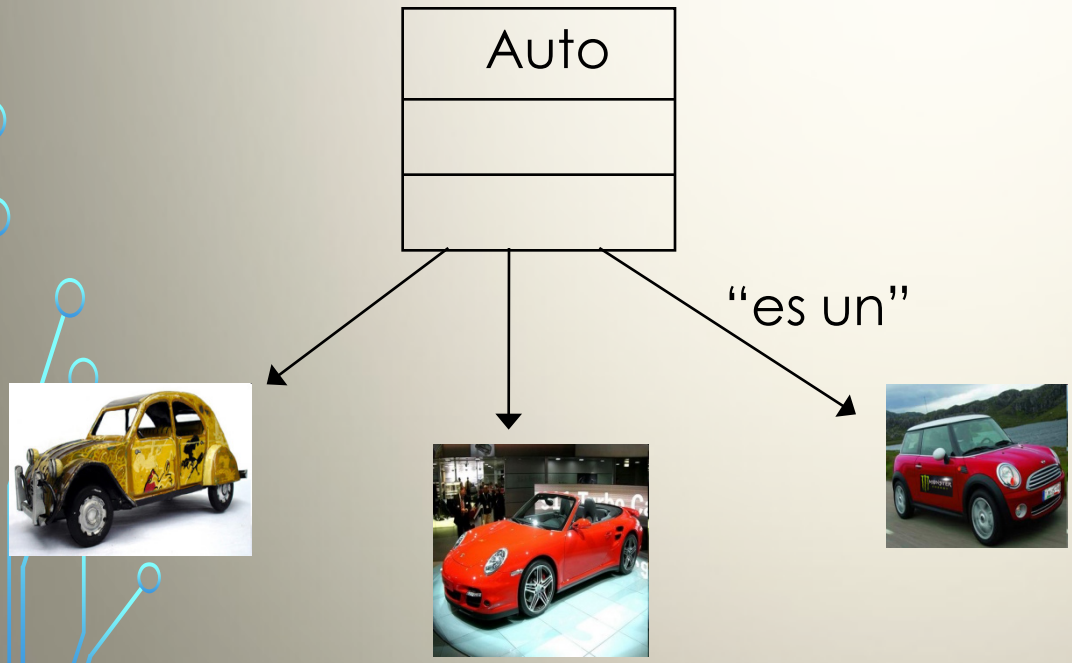
comportamiento

acelerar()
desacelerar()
arrancar()
apagar()
Todos los métodos para ver y modificar c/ característica.

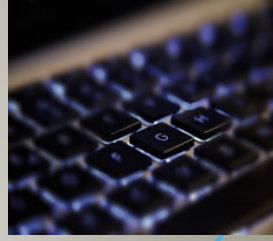
Objetos de la clase

Cada uno de los diferentes autos vistos anteriormente tienen características comunes pero con valores diferentes. Es decir los tres autos tienen color pero cada uno un color diferente.

Instancia de una clase = OBJETO

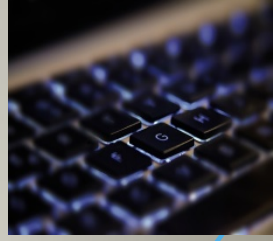


Programación Orientada a Objetos



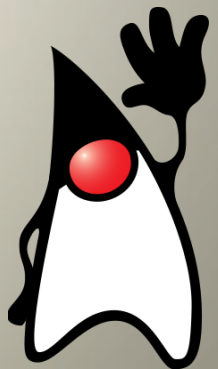
¿Cuales son las ventajas de la POO?

- Fomenta la reutilización y extensión de código
- Permite crear sistemas mas complejos
- Relacionar el sistema al mundo real
- Facilita la creación de programas visuales
- Construcción de prototipos
- Agiliza el desarrollo de software
- Facilita el trabajo en equipo
- Facilita el mantenimiento del software



- ¿Qué lenguaje de programación usaremos?

JAVA



Evolución de los lenguajes de programación

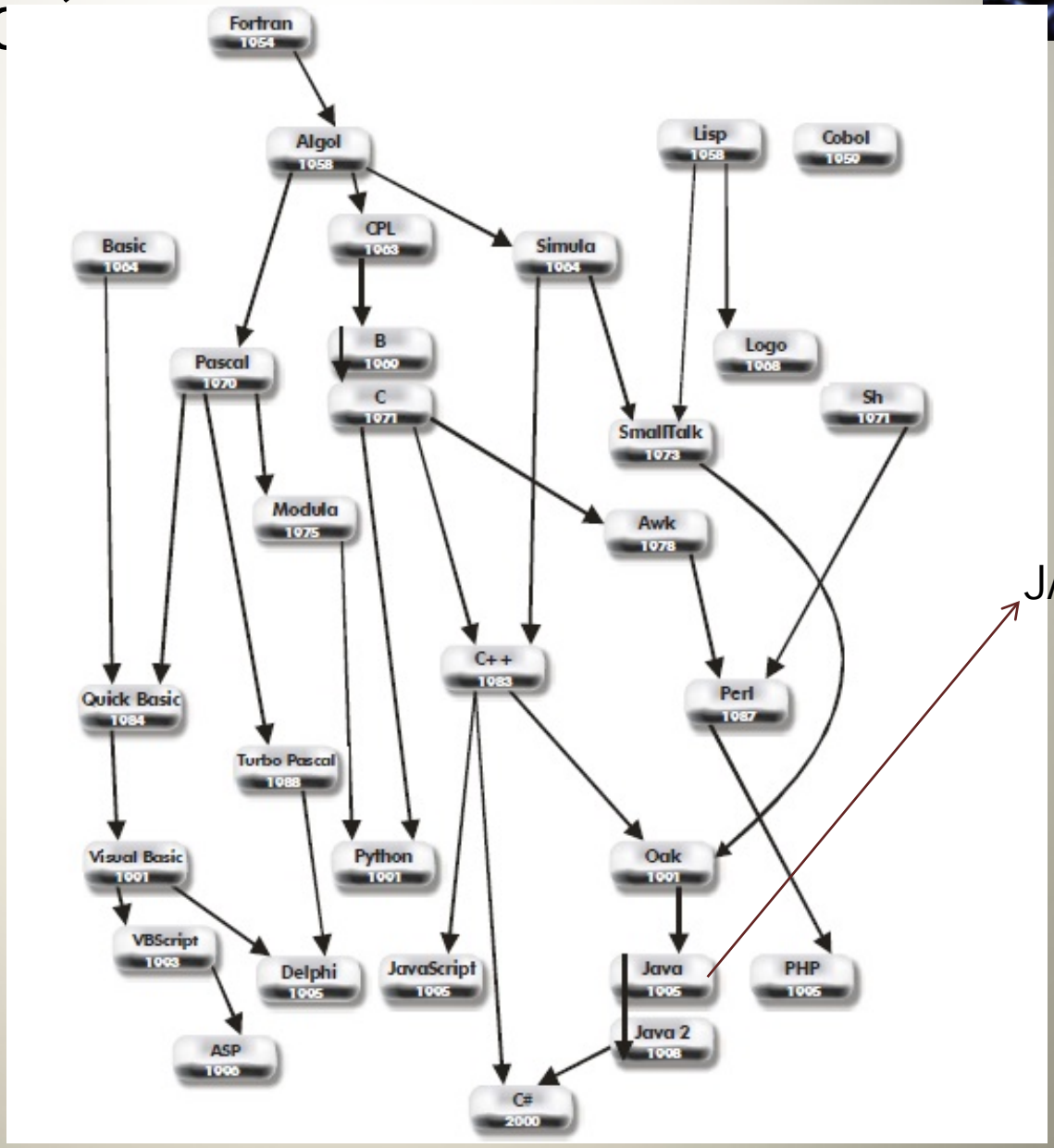
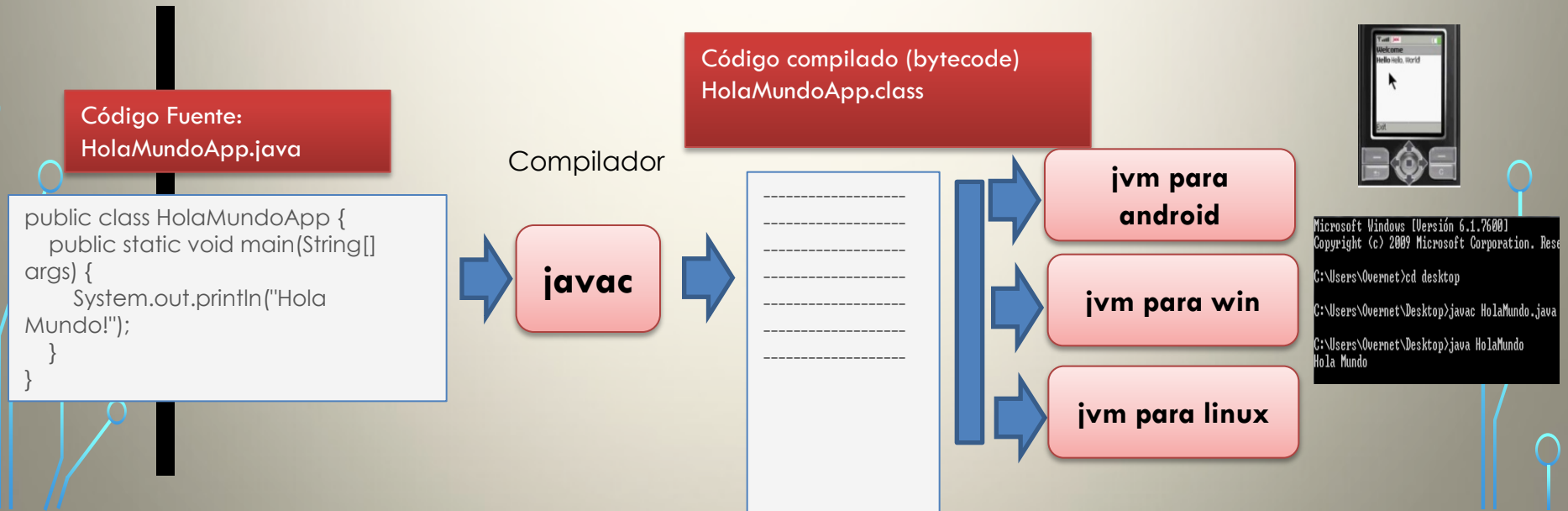


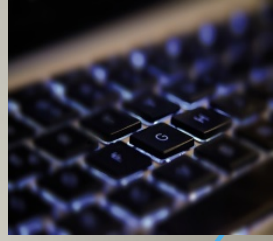
Imagen tomada del libro Java 2 de Jorge Sanchez, 2004

Java

- Lenguaje de propósito general. Paradigmas: Imperativo/OO
- Permite generar aplicaciones multiplataforma.
- Plataforma Java:
  - Plataforma de desarrollo (JDK): incluye compilador, depurador, generador de documentación,
  - Plataforma de ejecución (JRE): incluye componentes requeridas para ejecutar aplicaciones Java, entre ellas la JVM.
- Codificación y ejecución de app. java:



Interoperabilidad



WORA "write once, run anywhere"

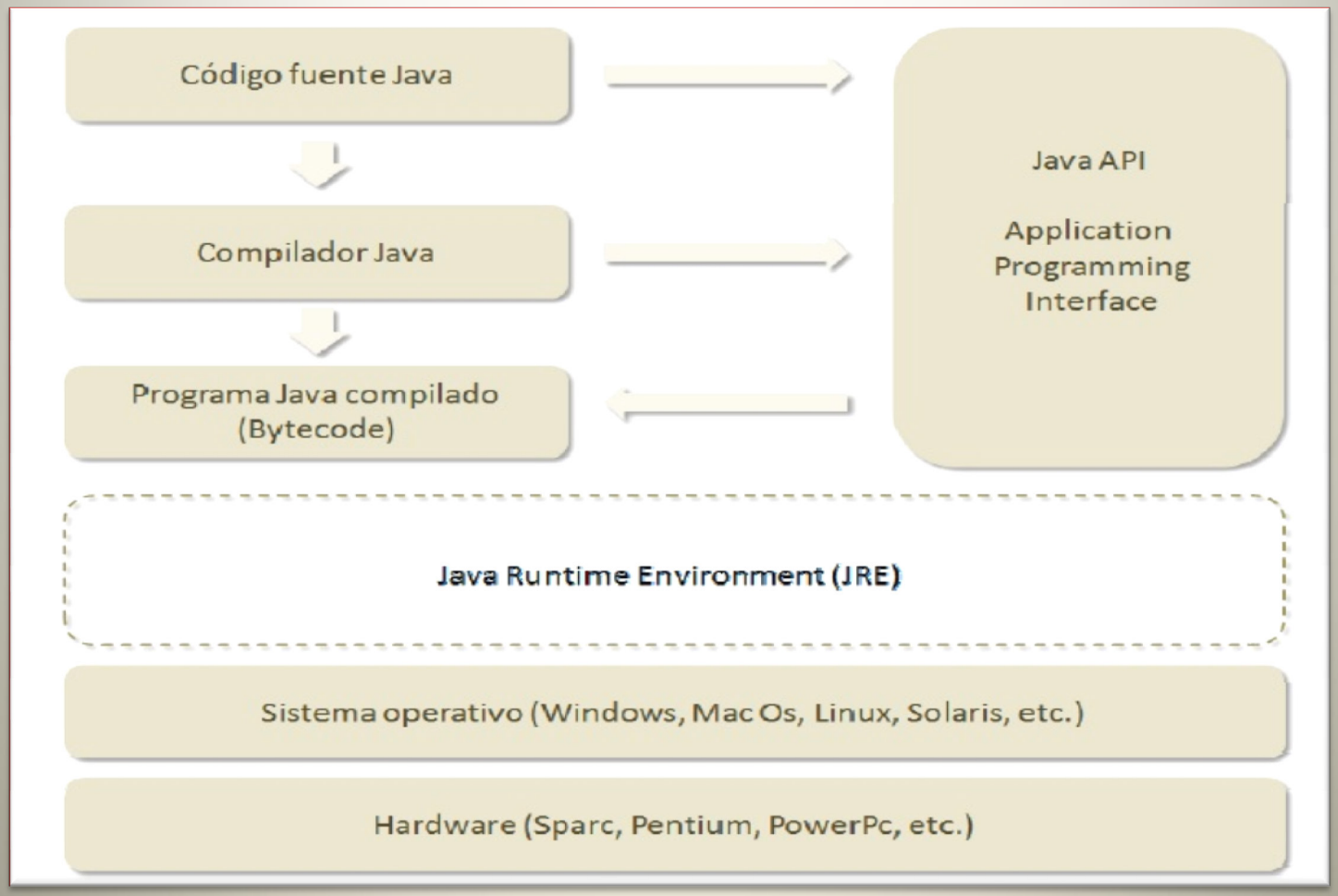
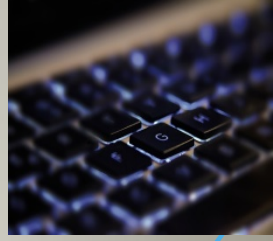


Imagen tomada del libro Java 2 de Jorge Sanchez, 2004



Proceso de Desarrollo

- Existen entornos de desarrollo integrado para JAVA, se los denomina IDE (Integrated Development Environment) como:

De distribución libre:

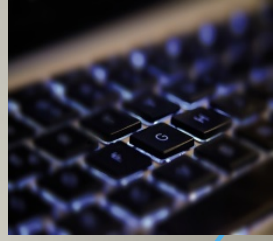
NetBEans, Eclips e o *BlueJ*

Comerciales:

Jbuilder o JCreatorPro

Nosotros utilizaremos:

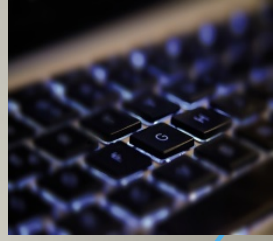
BLueJ



Programación en JAVA

- Estructura de un programa en JAVA
- Definiciones de comentarios
- Definiciones de clases
- Definiciones de metodos
- Definiciones de sentencias

El “programa principal”

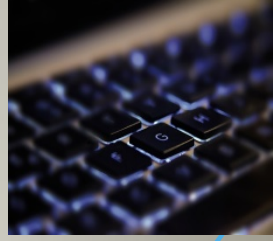


```
public class NombreAplicacion {
   public static void main(String[] args) {
        /* Código */
   }
}
```

Ejemplo

```
public class HolaMundoApp{
   public static void main(String[] args) {
      System.out.println("Hola Mundo!");
   }
}
```

- Main = “Programa principal”. { } delimita el cuerpo.
- Sentencias de código separadas por punto y coma (;).
- Se recomienda indentar el código para facilitar su lectura.
- Comentarios:
  - De líneas múltiples /* Esto es un comentario */.
  - De línea única // Este es un comentario
- Case-sensitive (sensible a las mayúsculas y minúsculas)



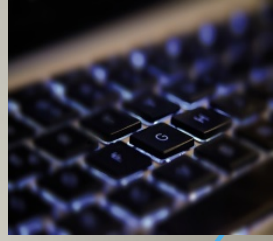
Sentencia. Dentro del método `main()` se incluye una sentencia para mostrar un texto por la consola. Los textos siempre se escriben entre comillas dobles para diferenciarlos de otros elementos del lenguaje. Todas las sentencias de un programa Java deben terminar con el símbolo punto y coma. Este símbolo indica al compilador que ha finalizado una sentencia.

```
/
* Este programa escribe el texto "Hola Mundo" en la consola
* utilizando el método System.out.println()
*/

public class HolaMundo {

   public static void main (String[] args) {
      System.out.println("Hola Mundo");


}
```

Definición de clase. La primera línea del programa, después del primer comentario. Define una clase que se llama `HolaMundo`. La definición de la clase comienza por el carácter { y termina con el carácter }. El nombre de la clase lo define el programador.

Definición de método. Después de la definición de clase se escribe la definición del método `main()`. Todos los programas Java deben incluir un método `main()`. Este método indica las sentencias a realizar cuando se ejecuta un programa. Un método es una secuencia de sentencias ejecutables. Las sentencias de un método quedan delimitadas por los caracteres { y } que indican el inicio y el fin del método, respectivamente.

Declaración variables locales a método



- Se declaran dentro del cuerpo del método (main / otro)

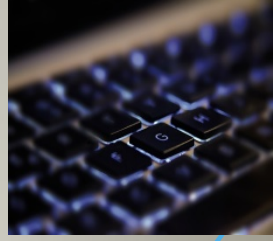
  Tipo nombreVariable; (Opcional: dar valor inicial)

- Convención para nombres: comenzar con minúscula, luego cada palabra en mayúscula.
- Asignación: nombreVariable = valor;
- Tipos primitivos: la variable almacena un valor

Tipo Primitivo	Tamaño	Mínimo	Máximo	Ejemplo
boolean	—	—	—	true false
char	16 bits	Unicode 0	Unicode 2^{16}-1	'a' '0' '*'
byte	8 bits	-128	+127	0b11
short	16 bits	-2^{15}	$+2^{15}-1$	102
int	32 bits	-2^{31}	$+2^{31}-1$	102
long	64 bits	-2^{63}	$+2^{63}-1$	102L
float	32 bits	IEEE754	IEEE754	123.4f
double	64 bits	IEEE754	IEEE754	123.4
void	—	—	—	

- **Clase *String*** para manipular cadenas. Ejemplo "esto es un string". *Son objetos.*

Manipulación de variables



Operadores

Operador de asignación (todos los tipos)

= Operador de asignación

Operadores aritméticos (tipos de datos numéricos)

+ operador suma

- operador resta

* operador multiplicación

/ operador división

% operador resto

Operadores relacionales (tipos de datos primitivos)

== Igual

!= Distinto

> Mayor

>= Mayor o igual

< Menor

<= Menor o igual

Operadores unarios aritméticos (tipos de datos numéricos)

++ operador de incremento; incrementa un valor en 1

-- operador de decremento; decrementa un valor en 1

Operadores Condicionales

&& AND

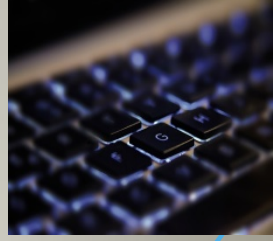
|| OR

! NOT

Operador de concatenación para String

+ Operador de concatenación de Strings

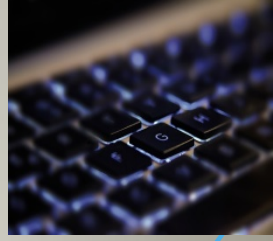
Declaración de variables. Ejemplos



```
public class DemoDeclaracionVariables {
public static void main(String[] args) {
    boolean encontre=false;                          //1
    int miDNI =11222333, tuDNI = 10555444;   //2
    char sexo;                                        //3
    sexo = 'F';                                       //4
    int uno, dos =2;                                 //5
    double miSueldo=1000.30;                         //6
    String miNombre="Pepe";                         //7
}
}
```

```
public class DemoOperadoresUnarios {
public static void main(String[] args) {
        int i,j;
        i = 3;   // i vale 3
        i++;     // i vale 4
        i--;     // i vale 3
}
}
```

```
public class DemoCalculoAritmetico {
public static void main (String[] args) {
        int result = 1 + 2;        // result es 3
        result = result - 1;       // result es 2
        result = result * 2;       // result es 4
        result = result / 2;       // result es 2
        result = result % 2;       // result es 0
}
}
```



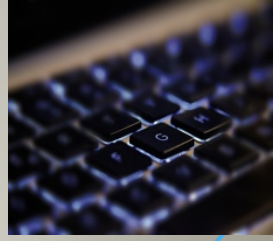
Void

Existe aun un tipo más que pertenece a los tipos primitivos del lenguaje.

Este es el tipo void (void es una palabra reservada).

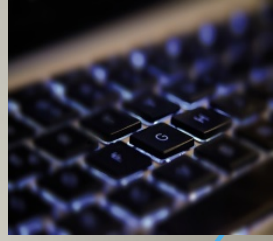
No existen valores de tipo void, así que no tiene representación.

El único uso de void es para la declaración de métodos que no retornan ningún resultado (o para ser mas precisos, retornan un valor de tipo void para el cual no hay representación, y por lo tanto, no requiere forma de almacenamiento en memoria).



Mostrar datos en la salida estándar

- Sentencias que permiten mostrar un string en consola:
  - System.out.print("Hola Mundo! ") NO realiza salto de línea
  - System.out.println("Hola Mundo! ") Realiza salto de línea
- Se pueden usar también con dato de tipo primitivo. Conversión automática a string.
  - System.out.println(1234)
  - System.out.println(true)
- Para mostrar varios datos, formar un string concatenando con +
  - int año=2015;
  - System.out.println ("Hola Mundo " + año + "!");



Ingreso de datos desde entrada estándar

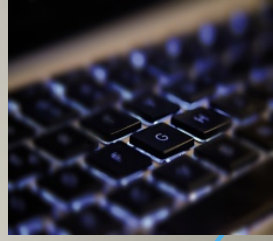
- Clase *Scanner* permite tomar datos desde una entrada (ej: teclado).
- Importarla en nuestra aplicación:

```
import java.util.Scanner;
```

- Indicar que se realizará lectura desde teclado (System.in)

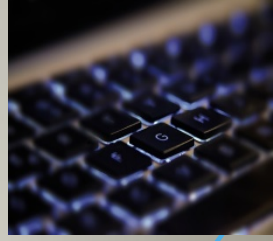
```
Scanner in = new Scanner(System.in);
```

- Nuestro scanner in tomará datos desde teclado.
  - Dato es una serie de caracteres terminada en blanco(b), tab(t), cr (cr).
  - Sintaxis: in.next####() #### según lo que queremos leer
    Retorna el dato leído
- Cuando no queremos leer más datos: in.close()



Ingreso de datos desde entrada estándar

Lectura de:	**Ejemplo**
Int	int dni = in.nextInt();
Double	double sueldo = in.nextDouble();
Boolean	boolean soltero = in.nextBoolean();
String (sec. caracteres hasta ~~b~~/ ~~t~~ / ~~cr~~)	String nombre= in.next();
String (sec. caracteres hasta ~~cr~~)	String descripcion = in.nextLine();



E/S de datos. Ejercitación.

- Escriba un programa para leer desde teclado los 3 lados de un triángulo (double), el color de relleno (string) y el color de línea (string).

  Luego, mostrar en consola lo siguiente:

  El triángulo tiene color de relleno *R* color de línea *L* y perímetro *P*

  - R y L son los colores de relleno y línea
  - P es el perímetro (resultado de calcular lado1+lado2 + lado3)