Programación I

TEMAS de la CLASE

1. Modularización
2. Procedimientos y Funciones
3. Alcance de una variable

1 Modularización

Actividad de Motivación

https://www.youtube.com/watch?v=FlUxUMCNwnA

Modularización

¿Qué observamos en los ejemplos del video?

- La tarea individual es tan importante como la grupal
- Cada uno hace una parte para resolver el problema
- El problema sólo se resuelve con el aporte de todos

Etapas de resolución de un problema por computadora

Problema del Mundo Real

Análisis

Complejos
Extensos
Modificables

Modelo

Diseño

Solución Modularizada

Implementación

Verificación

Modularización

Los problemas del mundo real implican:

- Complejidad
- Extensión
- Modificaciones/Situaciones de cambio

Los tratamos de resolver con:

- Abstracción.
- Descomposición funcional.

Modularización

ABSTRACCIÓN

SALUDAR

- LEVANTAR BRAZO DERECHO
- SACUDIR MANO DERECHA
- PRENDER LUZ

Modularización

ABSTRACCIÓN

AVANZAR

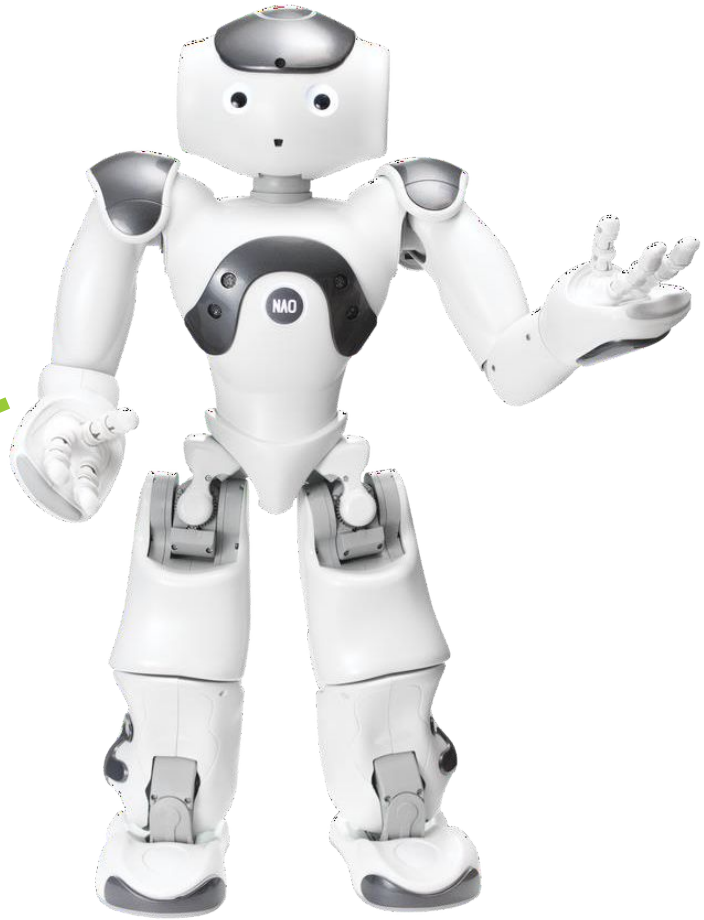
- MOVER PIERNA DERECHA
- MOVER PIERNA IZQUIERDA

Modularización

ABSTRACCIÓN

HABLAR

- REPRODUCIR SONIDO 1 - HOLA
- REPRODUCIR SONIDO 2 - SOY DOBBY



DESCOMPOSICIÓN FUNCIONAL

HABLAR

AVANZAR

SALUDAR

Modularización

Modularizar significa dividir un problema en partes funcionalmente independientes, que encapsulen operaciones y datos.

X

No se trata simplemente de subdividir el código de un sistema de software en bloques con un número de instrucciones dado.

Separar en funciones lógicas con datos propios y datos de comunicación perfectamente especificados.

Modularización - Abstracción

La descomposición tiene siempre un objetivo.

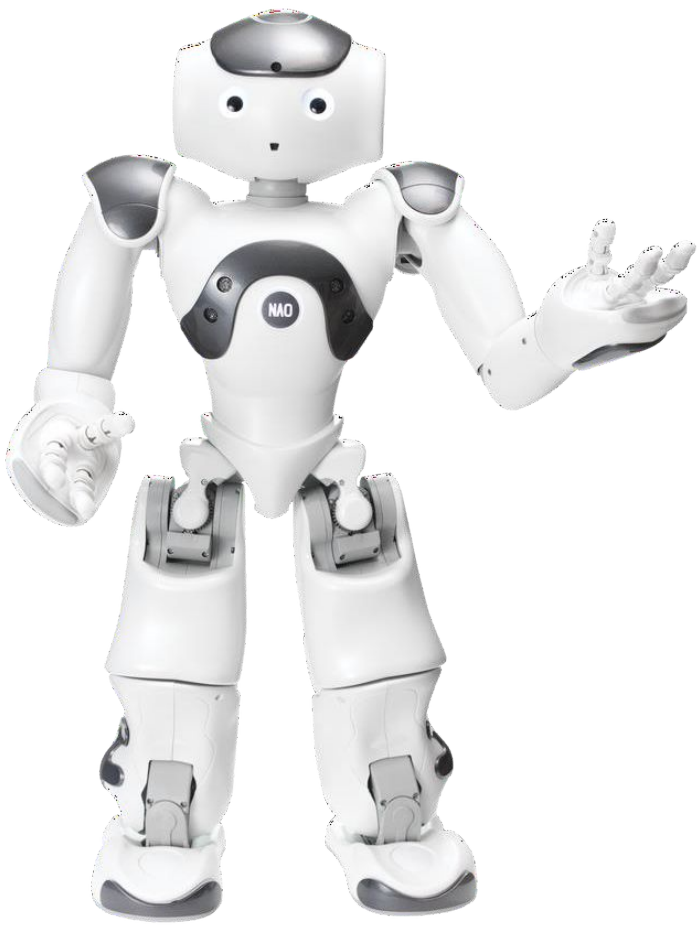
Se busca obtener:

Alta Cohesión: medida del grado de identificación de un módulo con una función concreta.

Bajo Acoplamiento: medida de la interacción de los módulos que constituyen un programa.

DOBBY PRESENTARSE

- HABLAR
- SALUDAR



Al diseñar con bajo acoplamiento y alta cohesión se puede reutilizar para crear nuevas funcionalidades.

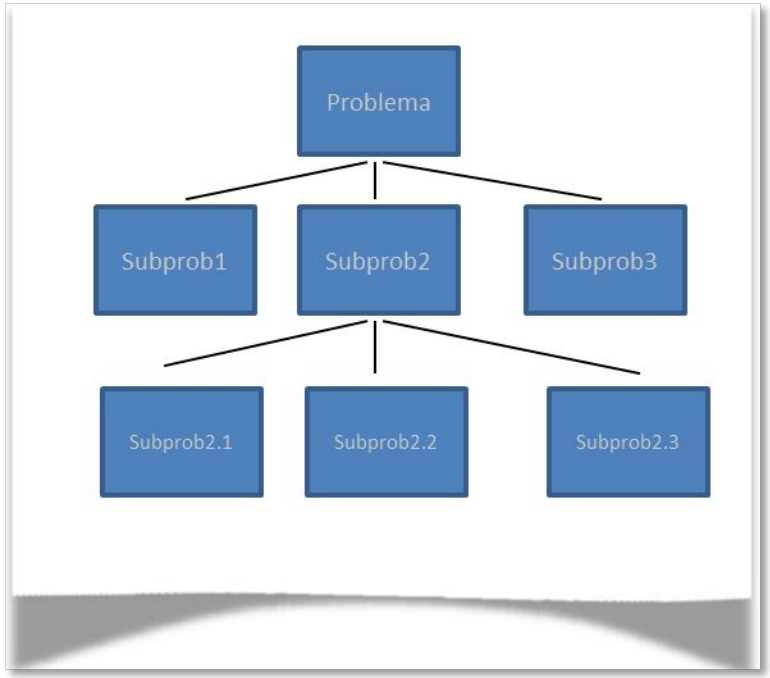
Modularización - Abstracción

Principio de "Divide y vencerás"

Descomponer el problema en partes (subproblemas) mas simples

Al descomponer un problema se debe tener en cuenta:

- Que cada subproblema resuelva una parte "bien" simple.
- Que cada subproblema pueda resolverse independientemente
- Que las soluciones a los subproblemas deben combinarse para resolver el problema original



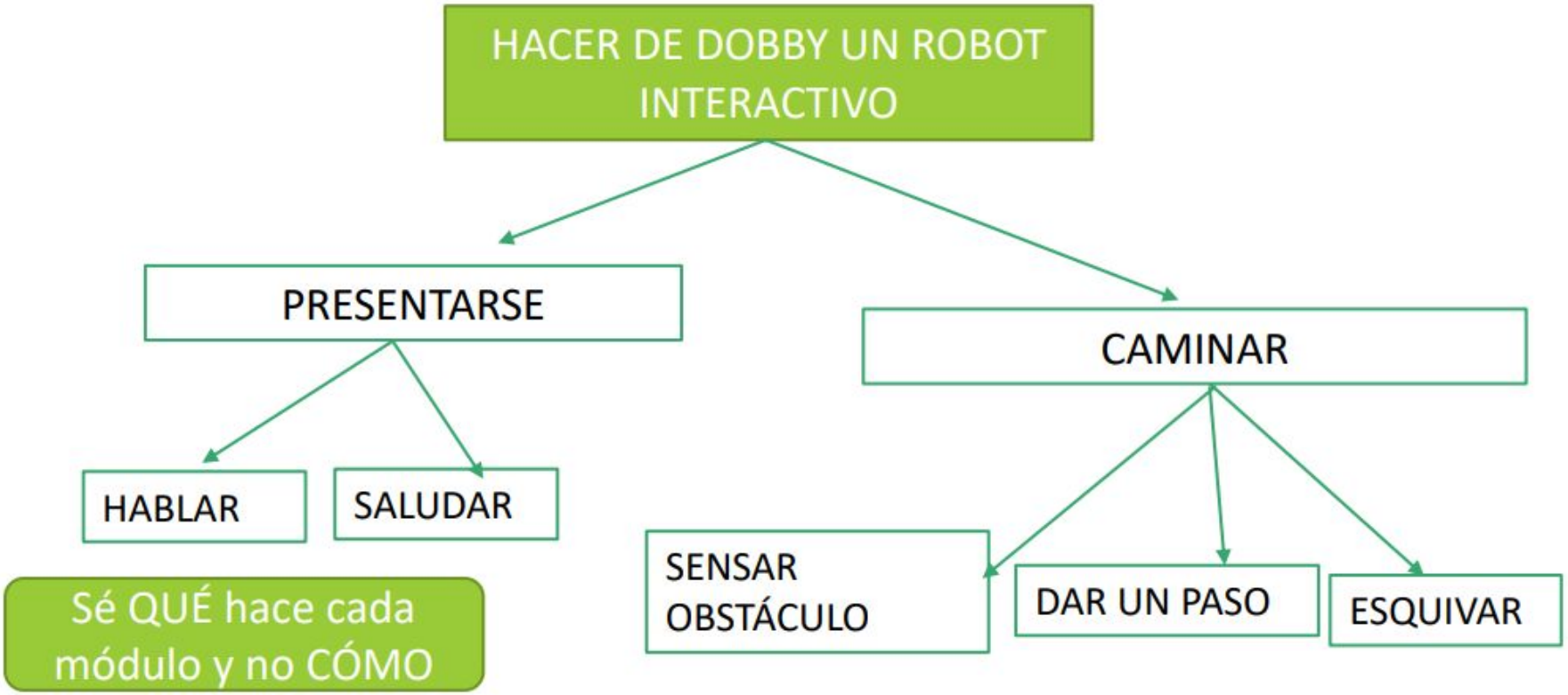
Modularización - Descomposición

¿Qué son los Módulos?

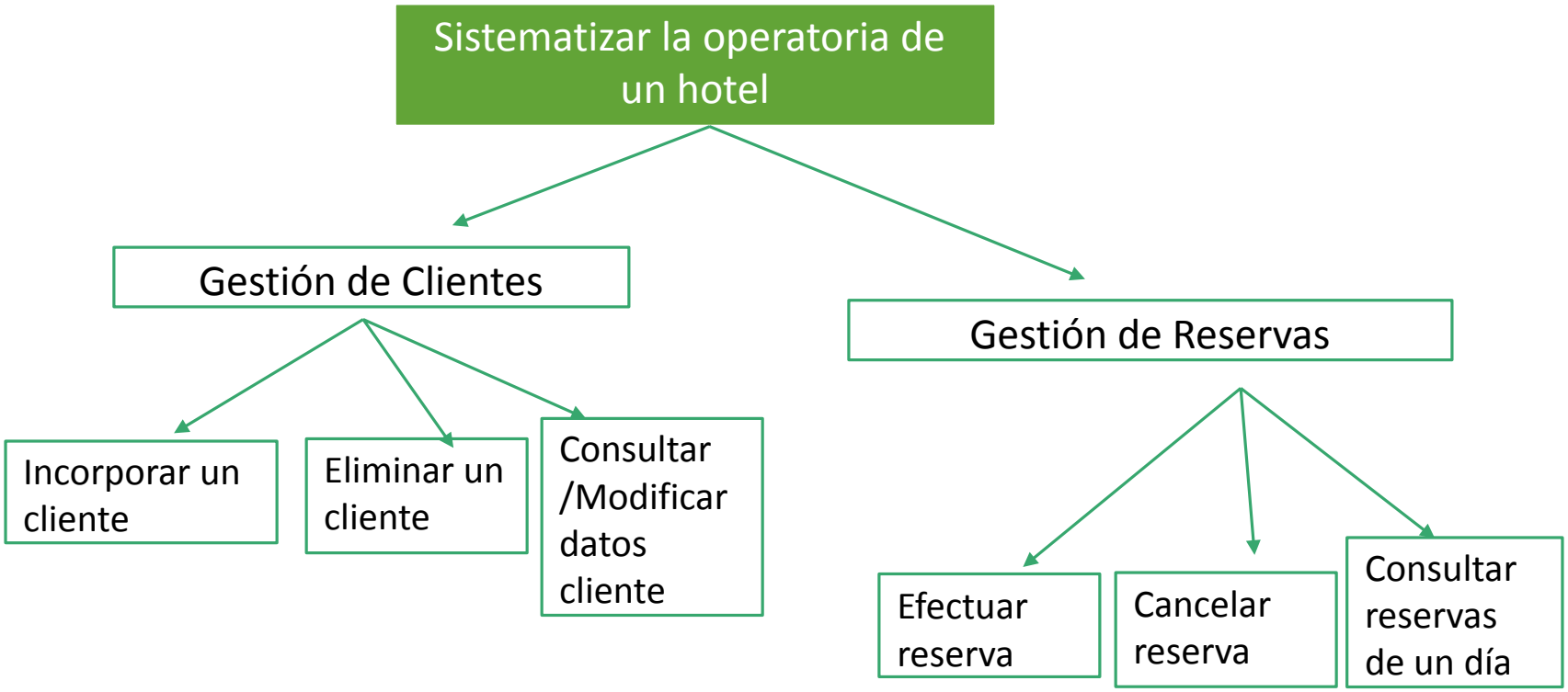
Es un conjunto de instrucciones que cumplen una tarea específica bien definida, se comunican entre sí adecuadamente y cooperan para conseguir un objetivo común.

- Cada módulo encapsula, acciones tareas o funciones
- Hay que representar los objetos relevantes del problema a resolver.

Modularización - Ejemplo 1



Modularización - Ejemplo 2



VENTAJAS DE LA MODULARIZACIÓN

Ventajas de la modularización

- Permite distribuir el trabajo
- Favorece el mantenimiento correctivo
- Facilita la reutilización dentro del mismo problema o en otro similar
- Facilita el crecimiento de los sistemas
- Aumenta la legibilidad

Modularización - Ventajas

Mayor productividad

Al dividir un sistema de software en módulos funcionalmente independientes, un equipo de desarrollo puede trabajar simultáneamente en varios módulos, incrementando la productividad (es decir reduciendo el tiempo de desarrollo global del sistema). Ejemplo.

Modularización - Ventajas

Sistematizar la operatoria de un hotel

Gestión de Clientes

- Incorporar un cliente
- Eliminar un cliente
- Consultar /Modificar datos cliente

Gestión de Reservas

- Efectuar reserva
- Cancelar reserva
- Consultar reservas de un día

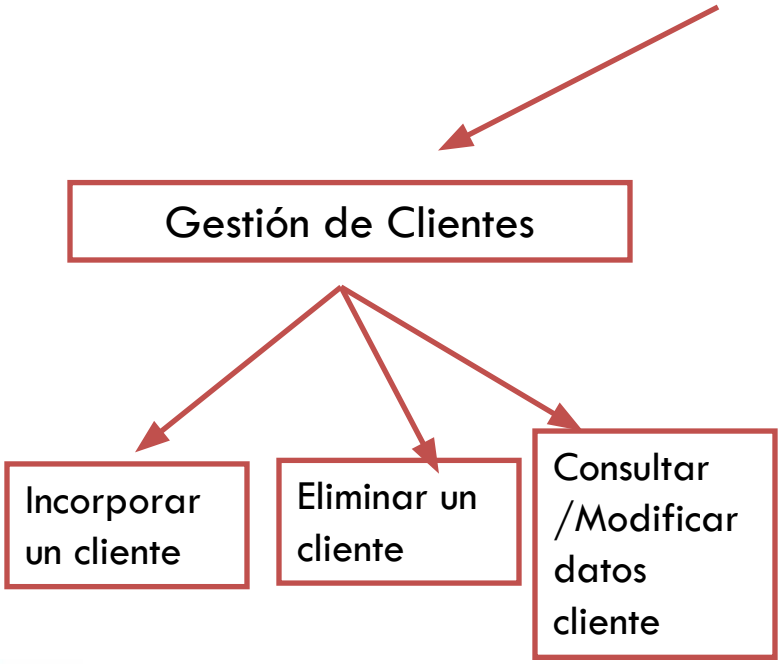
Modularización - Ventajas

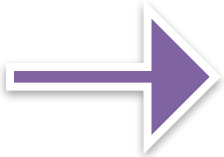
Reusabilidad

Un objetivo fundamental de la Ingeniería de Software es la *reusabilidad*, es decir la posibilidad de utilizar repetidamente el producto de software desarrollado.

Naturalmente la **descomposición funcional** que ofrece la **modularización *favorece el reuso***. Ejemplo.

Modularización - Ventajas





Se puede utilizar para cualquier otro sistema

Modularización - Ventajas

Facilidad de Mantenimiento Correctivo

La división lógica de un sistema en módulos permite **aislar los errores** que se producen con mayor facilidad. Esto significa poder **corregir los errores** en menor tiempo y disminuye los costos de mantenimiento de los sistemas. Ejemplo

Modularización – Ejemplo

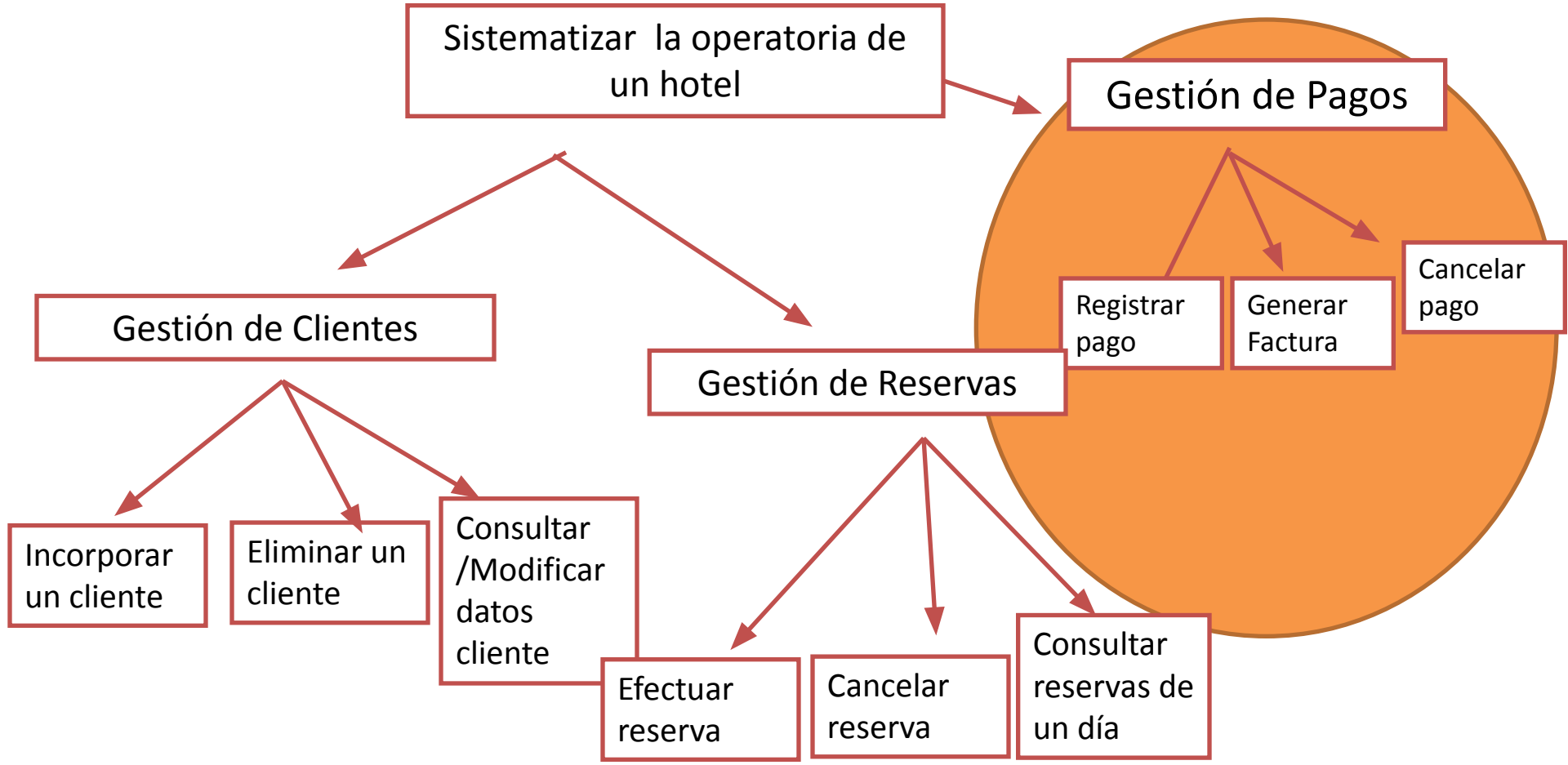
Eliminar un usuario

Modularización - Ventajas

Facilidad para el crecimiento del sistema

Los sistemas de software reales crecen (es decir aparecen con el tiempo nuevos requerimientos del usuario). La modularización permite **disminuir los riesgos y costos de incorporar nuevas prestaciones** a un sistema en funcionamiento. Ejemplo.

Modularización - Ventajas



Modularización - Ventajas

Mayor Legibilidad

Un efecto de la modularización es una **mayor claridad para leer y comprender el código fuente.**

El ser humano maneja y comprende con mayor facilidad un número limitado de instrucciones directamente relacionadas.

Ejemplo.

2 Procedimientos y Funciones

- Se debe elegir el lenguaje de programación para escribir los algoritmos de cada módulo y la declaración de sus datos

- Los lenguajes de programación ofrecen diversas opciones para implementar la modularización.

Definición del módulo

¿Qué hace el módulo cuando se ejecuta?

- Encabezamiento (Interface)
  - Tipo de módulo
  - Identificación
  - Datos de comunicación
- Declaración de tipos
- Declaración de variables
- Sección de instrucciones ejecutables

Invocación del módulo

¿Qué se hace cuando se quiere usar el módulo?

- Se debe conocer de qué manera se invoca al módulo para que ejecute sus acciones

La invocación puede hacerse mas de una vez

¿Qué ocurre con el flujo de control del programa?

¿Qué tipos de módulos ofrece Pascal?

PROCEDIMIENTOS (PROCEDURE)

FUNCIONES (FUNCTION)

Tienen características comunes, pero ciertas particularidades determinan cual es el mas adecuado para implementar un módulo particular

- ¿El módulo devuelve datos?
  - ¿Cuántos datos devuelve?
    - ¿De qué tipo son los datos que devuelve?
      - ¿Qué tipo de acciones ejecuta el módulo?

¿PROCEDURE?

¿FUNCTION?

¿Cuáles son los aspectos que los diferencian?

- Encabezamiento del módulo
- Invocación
- Lugar donde retorna el flujo de control una vez ejecutado el módulo

PROCEDURE

Conjunto de instrucciones que realiza una tarea específica y como resultado puede retornar 0, 1 o más valores.

¿Cómo se define el módulo?

```
Procedure nombre (lista de parámetros formales);
```

Encabezamiento

```
Type
   ……
```

Declaración de tipos internos del módulo (opcional)

```
Var
   ……
```

Declaración de variables internas del módulo(opcional)

```
begin
     ……..
     ……..
end;
```

Sección de instrucciones

PROCEDURE

Conjunto de instrucciones que realiza una tarea específica y como resultado puede retornar 0, 1 o más valores.

¿Cómo se invoca el módulo?

```
Program uno;
      .........

procedure Calculo (Parámetros Formales);
Type
      ....
Var
      ......
Begin

......
.....

End;


Begin
   ......
    Calculo (parámetros actuales);
   ......
End.
```

PROCEDURE

Conjunto de instrucciones que realiza una tarea específica y como resultado puede retornar 0, 1 o más valores.

Luego de ejecutado el módulo ¿Qué instrucción se ejecuta?

¿Qué ocurre con el flujo de control del programa?

Luego de ejecutado el módulo, el flujo de control retorna a la instrucción siguiente a la invocación del módulo

```
Program uno;
      .........

   procedure Calculo (Parámetros Formales);
   Type
         ....
   Var
        ......
   Begin

   ......
   ....

   End;

Begin
   ......
   Calculo (parámetros actuales);
   ......
End.
```

PROCEDURE

Puede ocurrir que un módulo Procedure contenga además, de la declaración de tipos y variables propias, la definición de otros módulos. Se dice que el procedimiento contiene módulos anidados. Por ejemplo:

```
Procedure principal (lista de parametros formales);
Type
   ……
Var
   ……

Procedure uno (lista de parametros formales);
  begin

  end;

Procedure dos (lista de parametros formales);
  begin

  end;

Var ……

Begin {instrucciones ejecutables del procedure principal}
  …
  uno (parametros actuales);
  dos (parametros actuales);
  ……
end;
```

FUNCTION

Conjunto de instrucciones que realiza una tarea específica y como resultado retorna un único valor de tipo simple.

¿Cómo se define el módulo?

```
Function nombre (lista de parámetros formales): tipo;

Type   ……

Var    ……

begin
   ……..
   nombre := ….;
end;
```

Asignación (obligatoria)

Encabezamiento

Declaración de tipos (opcional)

Declaración de variables (opcional)

Sección de instrucciones

FUNCTION

Conjunto de instrucciones que realiza una tarea específica y como resultado retorna un único valor de tipo simple.

¿Cómo se invoca el módulo?

```
Program otro;

Function cubo(Parámetros Formales):integer;
Type
      ….
Var
     ……
Begin
……
…..

End;

Begin
 …..
 …..
 write (cubo (parametro actual));
 ….
End.
```

```
Program otro;

Function cubo(Parámetros Formales):integer;
Type
      ….
Var
     ……
Begin
……
…..

End;

Var a: integer;
begin
 …..
 …..
 a := cubo (parametro actual));
 ….
End.
```

¿Qué ocurre con el flujo de control del programa?

Luego de ejecutado el módulo, el flujo de control retorna a la misma instrucción de invocación del módulo

FUNCTION

Conjunto de instrucciones que realiza una tarea específica y como resultado retorna un único valor de tipo simple.

¿Cómo se invoca el módulo?

```
Program otro;

Function cubo(Parámetros Formales):integer;
Type
      ….
Var
     ……
Begin
……
…..

End;


begin
…..
 …..
if (cubo (parametro actual) > 100) then
….;
End.
```

```
Program otro;

Function cubo(Parámetros Formales):integer;
Type
      ….
Var
     ……
Begin
……
…..

End;

Var a: integer;

begin…..
 …..
  ….
 while (cubo (parametro actual) > 50) do
     …….
End.
```

También puede ocurrir que un módulo function contenga además, de la declaración de tipos y variables propias, la definición de otros módulos procedimientos y/o funciones.

Esquema general de un programa que utiliza módulos

```
program uno;
```

```
Const
{Declaración de constantes del programa}
```

Zona de Declaración de constantes

```
Type
{Declaración de tipos definidos}
```

Zona de Declaración de tipos

```
Var
{variables}
```

Zona de Declaración de variables accedidas desde la sección del programa y los módulos

```
Procedure Calculo (parámetros formales);
……
Function EsPar (parámetros formales): Boolean;
...
```

Zona de Declaración de los módulos

```
Var
{variables}
```

Zona de Declaración de variables accedidas solo desde la sección del programa

```
Begin {Sección del programa principal}

……
End.
```

Zona de Instrucciones ejecutables

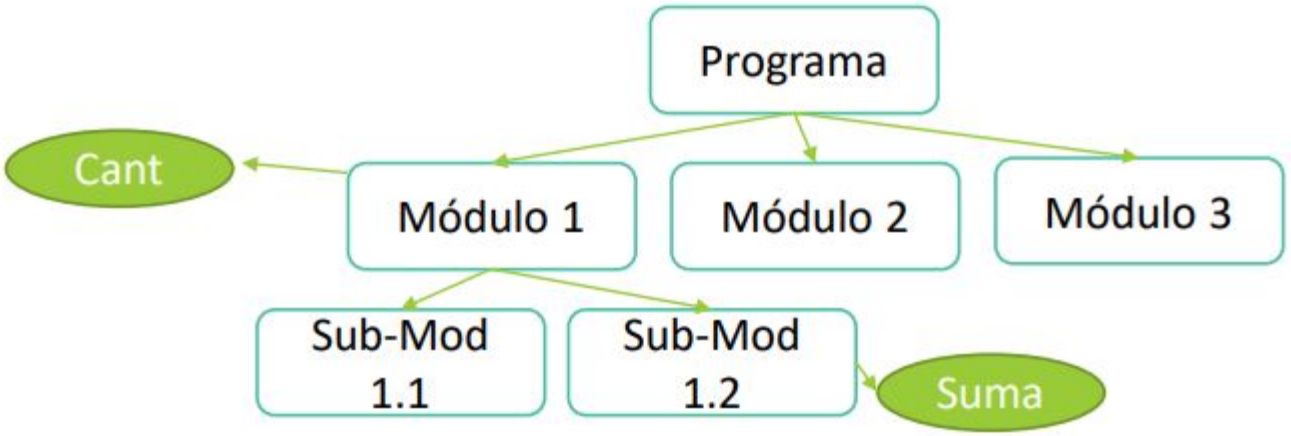
ASPECTOS IMPORTANTES PARA DIFERENCIAR UN PROCEDIMIENTO Y UNA FUNCIÓN

- ¿Dónde vuelve el flujo de control del programa ejecutado el módulo?
- ¿Cómo se invocan?
- ¿Qué tipos de parámetros aceptan?
- Cuántos valores devuelven como mínimo?
- Operaciones que se pueden realizar en cada uno.

3 Alcance de variables

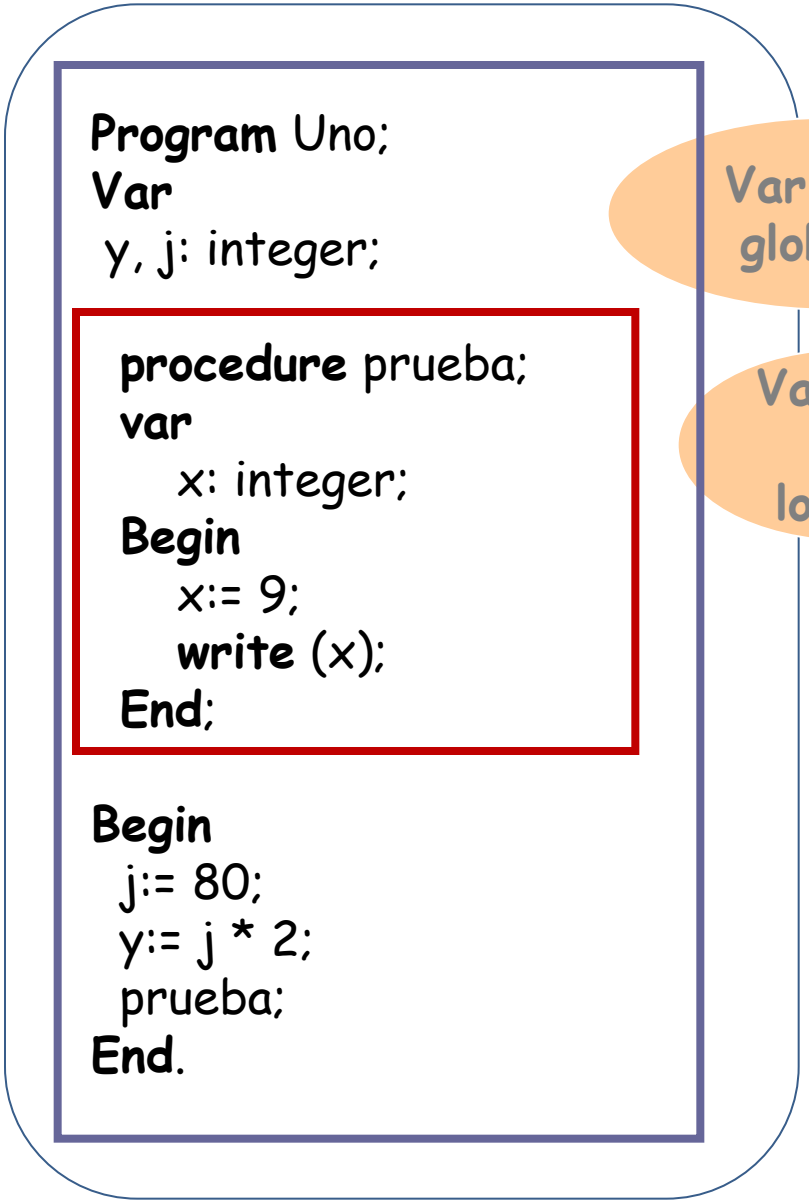
Alcance de variables

Alcance de una variable: Es el contexto donde una variable pueden ser referenciada o nombrada y ésta es reconocida.



¿Cuál es el alcance de la variable **Cant** y cuál es el de la varibale **Suma**?

Analicemos el alcance de Variables...



```
Program Uno;
Var
 y, j: integer;

  procedure prueba;
  var
     x: integer;
  Begin
     x:= 9;
     write (x);
  End;

Begin
  j:= 80;
  y:= j * 2;
  prueba;
End.
```

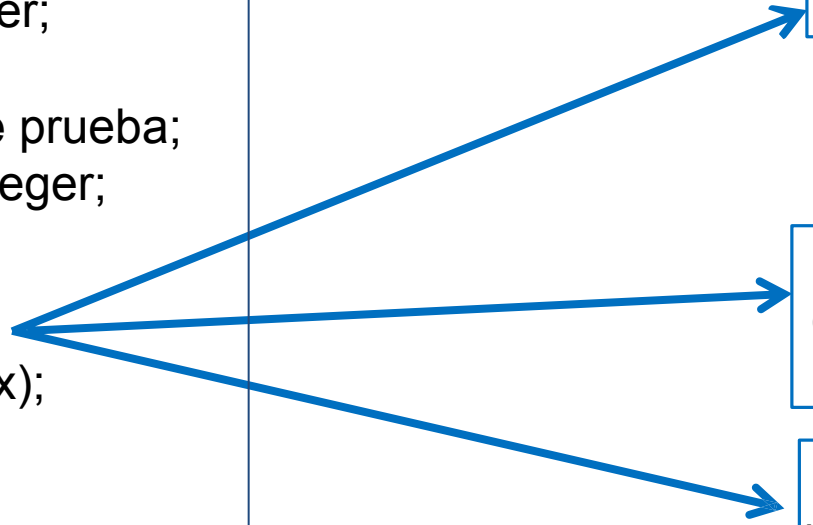

- ¿Dónde se pueden utilizar j e y?
- ¿Dónde se puede utilizar x?
- ¿Qué pasa si dentro de prueba se declara y: integer?
- ¿Qué pasa si dentro de prueba se declara y: char?

Alcance de variables

```
Program dos;
Var
 a, b: integer;

procedure prueba;
   var x: integer;
   Begin
      x:= 9;
      write (x);
   End;

Begin
  …..
End.
```



1. Se fija si es variable local

2. Si no es variable local, entonces se fija si es un parámetro.

3. Si no es variable local y no es parámetro, entonces se fija si es variable global.

Variables globales y

- **Variable global:** su declaración se hace en la sección de declaración del programa principal, es decir fuera de todos los módulos del programa y podrá ser usada en el programa y en todos los módulos del mismo.
- **Variable local a un módulo:** su declaración se hace en un módulo particular y sólo podrá ser usada por ese módulo. Si este módulo contiene a su vez otros módulos, entonces esa variable puede ser también usada por todos los módulos interiores.
- **Variable local al programa:** su declaración se hace antes de la sección de instrucciones ejecutables del programa y después de la declaración de los módulos del programa. Su uso se limita a la sección de instrucciones ejecutables.

¿Cuáles son variables locales y cuáles globales?

```
Program dos;
Var
 y, j: integer;

 procedure prueba;
 var   x: integer;
   procedure otro;
   var   z: integer;
   begin
      z:= 50;
      z := z div10;
      write (z, x);
   end;

  Begin
   x:= 9;
   write (x);
   x := x * j;
   otro;
  End;

Var  z: integer;
Begin
  j:= 90; z:= 10
  y:= j*z;
  prueba;
End.
```

Ejemplo

¿Prueba tiene Variables locales?

¿Con que datos Trabaja el módulo?

¿Qué imprime?

```
Program  tres;
  Var  dato: Integer;

Procedure  prueba;
   begin
      dato := 25;
      write  ( dato );
       dato: = 30;
    end;
begin       {del programa principal}
      dato := 0;
      write ( dato );
      prueba;
      write ( dato );
end.
```

Ejemplo

¿Proceso tiene Variables locales?

¿Con que datos Trabaja el módulo?

¿Qué imprime?

```
Program  cuatro;
   Var  dato: Integer;

 Procedure  prueba;
 Var dato: integer;
    begin
        write  ( dato );
        dato: = 30;
        write  ( dato );
     end;
 begin       {del programa principal}
        dato:= 0;
        write ( dato);
        prueba;
        write ( dato);
 end.
```

Ejemplo

```
Program cinco;
  Var num: Integer;

 Procedure prueba;
 Var dato: integer;
    begin
       dato: = 30;
       num := num +1;
       write ( dato, num );
     end;

 Procedure otro;
 Var dato: integer;
    begin
        write ( dato );
        dato: = 15;
        write ( num, dato );
     end;
Var dato: integer;
begin      {del programa principal}
       num:= 0;
       write ( num);
       prueba;
       write ( num);
       otro;
       write ( dato);
 end.
```

¿Qué imprime en cada write?

Ejemplos

¿Qué imprimen?

```
Program dos;
var x: integer;
procedure prueba;
   var x: integer;
   Begin
      x:= 9;
      write (x);
   End;

Begin
   x:= 8;
  prueba;
  writeln(x);
End.
```

```
Program dos;
Var  x : integer;
procedure
prueba;
  Begin
      write (x);
   End;
Begin
  x:=8;
  prueba;
  writeln(x);
End.
```

```
Program dos;
Var x: integer;
procedure prueba;
  Begin
      x:= 9;
      write (x);
   End;
Begin
   x:=8;
   prueba;
   writeln(x);
End.
```