Programación I

Tema 4:Diseño de subprogramas. Procedimientos y Funciones

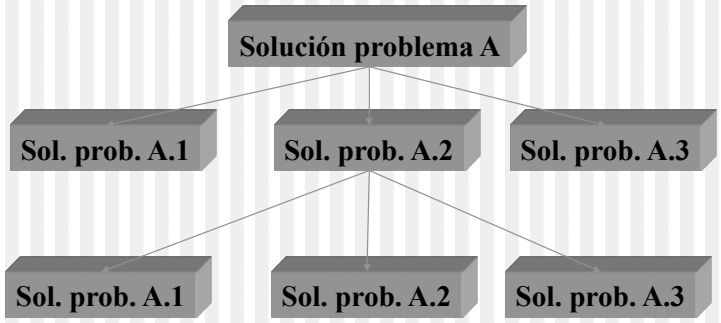
Índice

- Introducción
- Especificaciones de subprogramas: funciones y procedimientos
  - Parámetros
  - Parámetros formales y parámetros reales
  - Precondición y postcondición
- Diseño de subprogramas
- Ámbito de variables y visibilidad

Introducción

- **Diseño descendente (TOP-DOWN)**
  - Diseño de un algoritmo en fases
  - Dividir cada problema en subproblemas
  - Uniendo las diferentes partes de la solución se obtiene el algoritmo.

Ejemplo



Ejemplo

Escribe un algoritmo que, dado un año de los siglos XX ó XXI, escriba el calendario de ese año (sabiendo que el 1 de enero de 1900 fue lunes)

Enero

Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
		1	2	3	4	5
			…			
27	28	29	30	31		

Febrero

Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
					1	2

Ejemplo

- Especificación del problema
  - **Entrada:** un valor entero
  - **Precondición:** el valor corresponde a un año del siglo XX ó XXI
  - **Salida:** El calendario del año
  - **Postcondición:** Los meses aparecerán uno detrás del otro. Los días de los meses se repartirán en 7 columnas, empezando por el lunes y acabando en domingo.

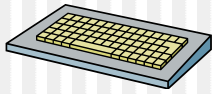
Ejemplo

- Diseño descendente

```
Leer Año
(1.1) Calcular el primer día del año indicado
Para cada Mes entre 1 y 12
        (1.2) Escribir cabecera correspondiente a Mes
        (1.3) Calcular número de días del Mes
        (1.4) Escribir "huecos" antes del primer día
        (1.5) Escribir los días del mes
Fin Para Cada
```

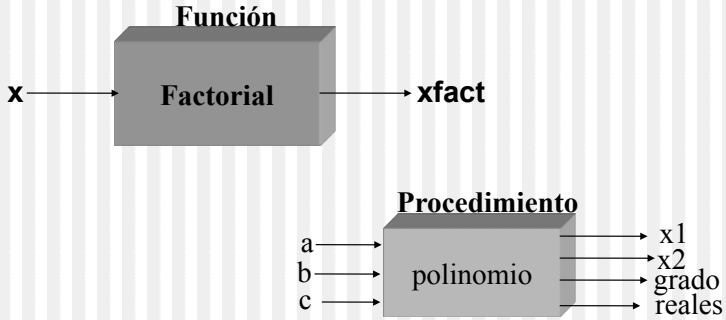
Motivación

- Facilita el diseño y la implementación
- Diseño = Programa
  - Legibilidad
    - Facilita la comprensión del código
  - Modularidad
    - Cada subprograma tiene una misión concreta
  - Abstracción
    - Se utilizan expresiones más ricas



Subprogramas

- Conjunto de acciones que permiten calcular un valor o valores a partir de unos datos de entrada



Subprogramas

Funciones

- Siempre devuelven un valor (**return**)
- Puede utilizarse en cualquier punto donde se pueda utilizar el valor devuelto

```
If (esBisiesto(fecha))
{
        desplazamiento = desplazamiento + 2;
}
```

Funciones

- Definición de una función
  modVisib* static *tipo nFunción (parámetros)
- modVisib
  - public, private, protected, package
- Tipo
  - Tipo de valor que devuelve la función
- Parámetros
  - Lista de datos de entrada junto con sus tipos
  - Ej: (int anno)
  - Ej: (int num1, int num2)

Funciones: Ejemplo

```
public static boolean esBisiesto (int fecha)
{
   if (fecha % 4 == 0)
   {
     if (fecha % 100)
      {
        if (fecha % 400)
                return true;
        else
      }         return false;
     else
          return true;
   }
   else
     return false;
}
```

Funciones

- Uso
  - Llamada a una función
    - [clase.]nombreFunción(valores)
  - Hay que asignar un valor a cada uno de los parámetros
  - Uso en expresiones o asignaciones
    - x = factorial(7);
    - if (factorial(y) > 1000)

Procedimientos

- A veces los problemas no se pueden resolver mediante funciones porque
  - Se debe obtener más de un resultado
  - No se obtiene ningún resultado (se procesan los datos y se muestran los datos en la salida)
- Ejemplo
  - Mostrar el contenido de un fichero
  - Visualizar en pantalla los números primos en el rango (1..100)

Procedimientos

- Definición de un procedimiento

  modVisib* static void *nFunción (parámetros)
- modVisib
  - public, private, protected, package
- VOID
  - Indica que el subprograma no devuelve un valor
  - No es una función
- Parámetros
  - Lista de datos de **entrada o salida** junto con sus tipos
  - Ej: (int anno)
  - Ej: (int num1, int num2)

Procedimientos

- Ejemplo

```
public static void escribeAsteriscos (int num)
{
   for (int cont = 1; cont <= num; cont++)
   {
     System.out.print("*");
   }
   System.out.println();
}
```

Procedimientos

- Uso
  - Llamada a un procedimiento
    - [clase.]nombreProcedimiento(valores)
  - Hay que asignar un valor a cada uno de los parámetros
  - No se pueden usar en expresiones o asignaciones
    - escribeResultado(result);

Consideraciones

- Funciones
  - Devuelven siempre un único valor
  - Expresiones y asignaciones
  - Cálculos aritméticos
  - Cuidado con las operaciones de entrada\salida
  - Representan un valor (resultado)
- Procedimientos
  - Pueden devolver varios resultados o ninguno
  - Representan una acción
  - Adecuados para visualizar resultados en pantalla

Parámetros

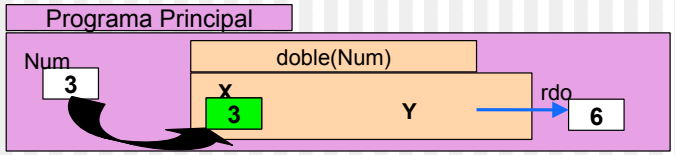
- Forma de pasar datos a un subprograma
  - Obtener resultados en procedimientos
- Paso por valor y Paso por referencia
  - Teoría
  - Java
- Parametros formales y Parámetros reales

Paso por valor y por referencia

- ¿Cómo se le pasa un dato a un subprograma?
  - Dato original
  - Copia del dato original
- ¿Qué puede pasar si se le pasa un dato original?
  - Modificar el dato original
  - Afecta al programa principal

Paso por valor y por referencia

- Paso por valor
  - Paso por copia
  - Copia el valor original y se lo pasa al subprograma

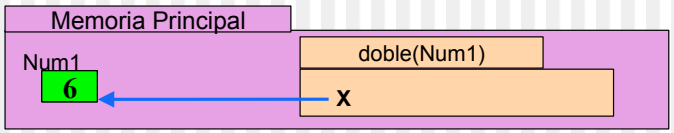


Paso por valor y por referencia

- Paso por referencia
  - Necesidad de modificar los datos proporcionados
  - Obtener varios resultados (procedimientos)
  - Estructuras complejas

Paso por valor y por referencia

- Paso por referencia
  - Se pasa la **dirección** (un apuntador) en la que se encuentra el dato necesario para el subprograma
  - Trabaja sobre el dato original
  - Permite que el subprograma devuelva resultados



Paso por valor y paso por referencia en Java

- Paso por valor para datos primitivos
  - int, char, …
- Paso por referencia
  - Estructuras complejas
    - Clases
    - Arrays

Parámetros formales y parámetros reales

- Los datos que se le pasan al subprograma constituyen los parámetros
  - Formal: variable en función de la cuál se especifica el subprograma
    - public static int suma(int **num1**, int **num2**)

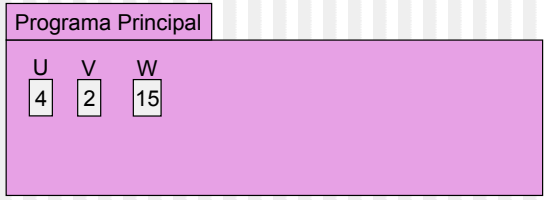
    Hay que indicar el tipo de los parámetros
  - Real: valor con el que se ejecuta el subprograma
    - x = suma(**5** , **3**)

    Se determinan los valores que va a utilizar la función. NO HAY QUE INDICAR EL TIPO

Ejecución de un subprograma

```
…
u = Integer.parseInt(ent.readLine());
v = Integer.parseInt(ent.readLine());
w = Integer.parseInt(ent.readLine());
w = factorial(u+v);
…
```

Las variables están en memoria

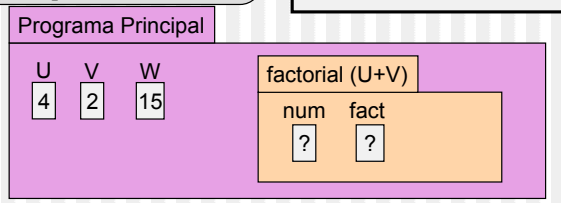


Ejecución de un subprograma

```
…
u = Integer.parseInt(ent.readLine());
v = Integer.parseInt(ent.readLine());
w = Integer.parseInt(ent.readLine());
w = factorial(u+v);
…
```

```
private static int factorial (int num)
{
  int fact = 1;
  for (int cont = 1; cont<= num;cont++)
  {
     fact *= cont;
  }
  return fact;
}
```

Se reserva espacio de memoria para las variables y los parámetros



Ejecución de un subprograma

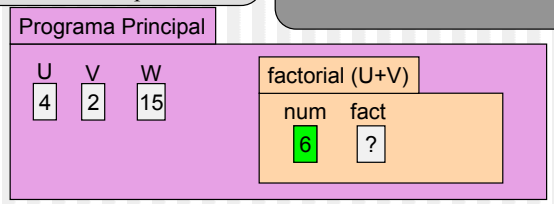
```
…
u = Integer.parseInt(ent.readLine());
v = Integer.parseInt(ent.readLine());
w = Integer.parseInt(ent.readLine());
w = factorial(u+v);
…
```

```
private static int factorial (int num)
{
  int fact = 1;
  for (int cont = 1; cont<= num;cont++)
  {
```

Se evalúan los parámetros reales asignando el valor a los parámetros reales correspondientes

u + v → 6

num ← 6

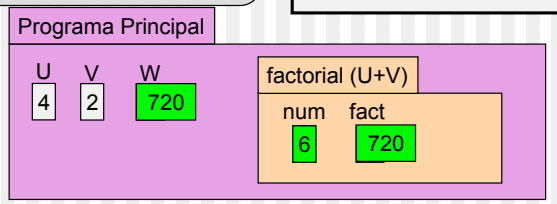


Ejecución de un subprograma

```
…
u = Integer.parseInt(ent.readLine());
v = Integer.parseInt(ent.readLine());
w = Integer.parseInt(ent.readLine());
w = factorial(u+v);
…
```

```
private static int factorial (int num)
{
  int fact = 1;
  for (int cont = 1; cont<= num;cont++)
  {
    fact *= cont;
  }
  return fact;
}
```

Calcula el resultado y lo devuelve al subprograma que lo ha invocado

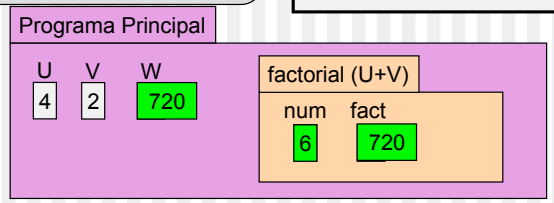


Ejecución de un subprograma

```
…
u = Integer.parseInt(ent.readLine());
v = Integer.parseInt(ent.readLine());
w = Integer.parseInt(ent.readLine());
w = factorial(u+v);
…
```

```
private static int factorial (int num)
{
  int fact = 1;
  for (int cont = 1; cont<= num;cont++)
  {
    fact *= cont;
  }
  return fact;
}
```

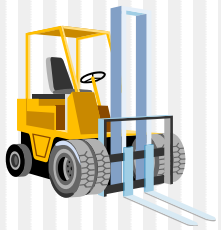
Libera el espacio ocupado por el subprograma



Especificación de subprogramas

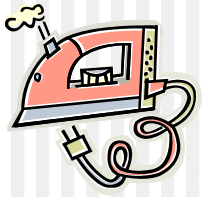
- Descripción del funcionamiento
- Sin detalles de implementación
- Imprescindible para la utilización de un subprograma
- Condiciones para el uso del subprograma

Especificación de objetos reales



Especificación:
- Funciona con gasolina
- max. 2000 kg.

Especificación:
- Llenar con agua antes de enchufar
- Corriente alterna 220-230 V



Especificación de subprogramas

- Utilizar Precondiciones y Postcondiciones
  - Precondición:
    - Condiciones que han de cumplir los datos de entrada para garantizar la corrección de los resultados
    - Valores obtenidos por el subprograma desde secuencias de entrada (teclado, ficheros,…)
  - Postcondición:
    - Resultados obtenidos por el subprograma
    - Valores (y formato) enviados por el subprograma a una secuencia de salida (pantalla, ficheros,…)

Especificación de subprogramas

- Ejemplo:

```
/**
  * Calcula el primer día del año
  * PRE: Un año del siglo XX o siglo XXI
  * POST: Devuelve el número de días que se desplaza el
  *           1 de enero dentro de la semana
  */

  private static int calcularPrimerDia (int pAño)
```

Diseño de un subprograma

- Especificación (Pre-Post como comentario)
- Acciones que manipulan los datos de entrada para obtener los resultados
  - Cabecera subprograma (identificador, parámetros)
  - Variables (locales)
  - Cuerpo del subprograma

Sobrecarga de subprogramas

- Se pueden definir subprogramas con el mismo nombre y distinta cabecera
- Identificador sobrecargado o subprograma sobrecargado
- Se ejecuta uno u otro subprograma en función de los parámetros
  - public static int suma (int num1, int num2)
  - public static float suma (float num1, float num2)

Ámbito y visibilidad de las variables

- Variables locales:
  - Se definen y utilizan en un subprograma (o un bloque)
  - Sólo visibles en ese bloque
  - Desaparecen al final del bloque
  - No conservan el valor de una ejecución a otra
- Ejemplos

Ámbito y visibilidad de las variables

- Variables globales:
  - Se definen fuera del subprograma (utilizando la palabra clave ***static***)
  - Visibles para todos los subprogramas
  - Desaparecen al finalizar el programa

Ámbito y visibilidad de las variables

- Ámbito de una variable:
  - desde la declaración hasta el final del bloque
- Reglas de visibilidad:
  - No se puede utilizar un identificador fuera de su ámbito
  - No se puede redefinir un identificador en un mismo bloque
  - Se puede redefinir un identificador en bloques distintos
  - Un identificador es potencialmente visible en todo su ámbito, pero si se redefine en un bloque anidado, entonces la declaración más externa queda oculta en el bloque anidado
    - V. Local - V. Global

Ámbito y visibilidad de las variables

```
public class Numeros {
    static int numero; // V. Global. Accesible para todos los subprgr.
    private static int suma(int num1, int num2) {
        int numero; // V.Local. Se accede a la local
        numero = num1 + num2; // Se modifica el valor de la V. Local
        return numero;
    }

    public static void main(String[] args) {
        int result;
        numero = 5; // Se accede a la V. global
        result = suma(6, 7);
      System.out.println(numero); // Se accede a la V. global
    }
}
```