Clase 6

PROGRAMACIÓN 1

Objetivos del tema

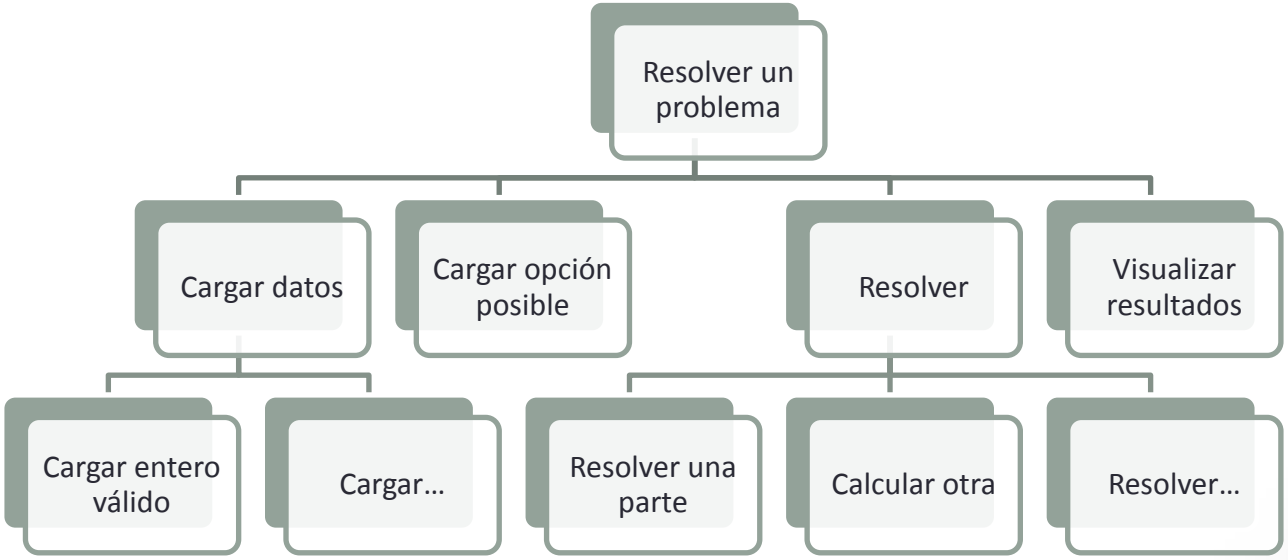
- Resolver problemas aplicando un diseño descendente o programación modular
- Identificar los dos tipos de métodos: procedimientos y funciones
- Utilización de la sentencia return

Diseño descendente

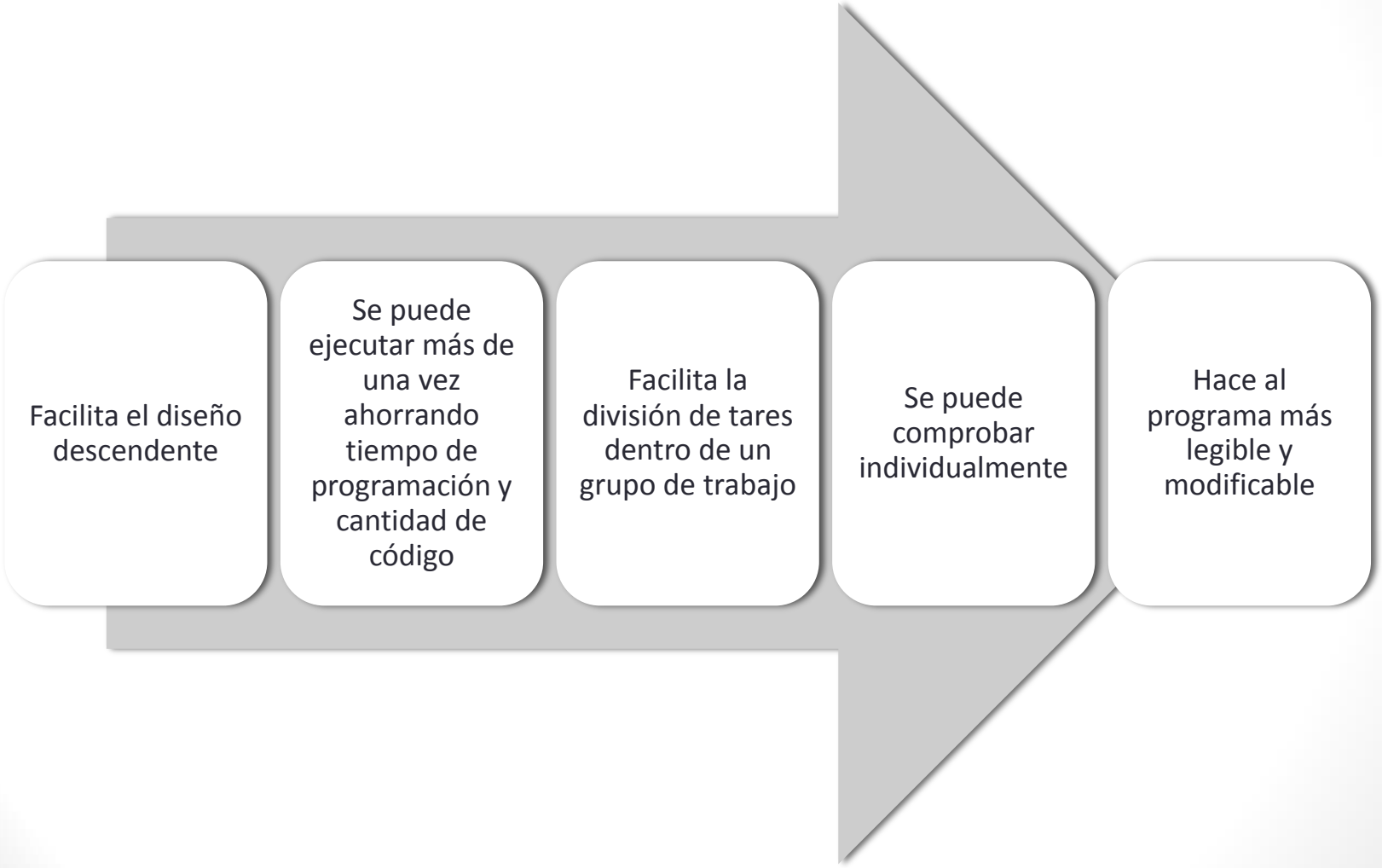
Uno de los métodos fundamentales para resolver un problema es dividirlo en problemas más chicos o subproblemas.

Está división se hace repetidamente hasta llegar a pequeños problemas fácilmente solucionables o ya solucionados.

Es deseable que cada subproblema sea independiente de los restantes, y se los denomina métodos.



Ventajas de modularizar la resolución



Ejemplo de diseño por pseudocódigo

- Escribir un diseño de programa que si el usuario ingresa un número natural imprima la tabla de multiplicar del 5.

…

```
public class Programa {
  public static void main(String[] args){
    definir numero natural
    obtener un numero natural por teclado
    imprimir tabla de multiplicar del numero 5
  }
}
```

Ejemplo de diseño

- Escribir un diseño de programa que mientras el usuario ingrese un número natural imprima la tabla de multiplicar del 5.

```
…
public class Programa {
  public static void main(String[] args){
   definir numero natural
   mientras sea numero natural{
     obtener un numero natural por teclado
     imprimir tabla de multiplicar del numero 5
   }
  }
}
```

Práctico

- Escribir un diseño de programa que mientras que el usuario ingrese un número distinto de 0, pida ingresar otro numero y lo imprima.

- Escribir un diseño de programa que mientras que el usuario ingrese un número distinto de 0, pida ingresar otros dos números e imprima el resultado de la multiplicación entre los dos últimos números ingresados.

- Escribir un diseño de programa que solicite desde teclado un número de mes válido y posteriormente notifique por pantalla la cantidad de días de ese mes. En el caso de que ingrese 2 como número de mes (febrero) deberá además solicitar ingresar un número de anio entre 2000 y 2019 inclusive, y dependiendo de si es bisiesto o no imprimir la cantidad de días correspondiente.

Diseño descendente con métodos

Un método es una sección de código que puede ser llamado por el programa principal u otros métodos para realizar alguna tarea específica.

El método es llamado por su nombre seguido por una secuencia de parámetros o argumentos (datos utilizados por el propio método para sus cálculos) entre paréntesis.

Cuando el método finaliza sus operaciones, puede o no devolver un valor simple al programa que lo llama.

Un método puede retornar un valor a través de la sentencia return y el tipo de dato debe coincidir con el tipo de dato declarado en la cabecera del método

Declaración de métodos

```
[modificadores] tipoDeDato identificadorMetodo (parametros formales) {
  declaraciones de variables locales;
  sentencia_1;
  ...
  sentencia_n; //se incluye al menos un return
 }
```

- La primera línea es la cabecera del método.
- Los modificadores especifican cómo puede llamarse al método.
- El tipo de dato indica el valor que devuelve o no la llamada al método.
- Los parámetros (entre paréntesis) introducen información para la ejecución del método (Se verán más adelante).
- La declaración de un método se denomina declaración formal.
- La invocación de un método se denomina declaración local o actual.

Ejemplo métodos

- Imprimir la tabla de multiplicar de 10.

```
public class Programa {

 public static void main (String [] args){
  imprimir_tabla10();//declaración local
 }

 public static void imprimir_tabla10(){//declaración
  formal
   final int multiplo = 10;
   for (int i = 1 ; i <= MAX; i++) {
      System.out.println(multiplo*i);
   }
 }
}
```

Acceso a métodos

- public y static son los modificadores que hacen al método accesible al resto. Los métodos estáticos son como los métodos de los lenguajes no orientados a objetos

```
public class Programa {
 public static void main (String [] args){
  imprimir_tabla10();//declaración local
 }
 public static void imprimir_tabla10(){//declaración
  formal
   final int multiplo = 10;
   for (int i = 1 ; i <= MAX; i++) {
       System.out.println(multiplo*i);
   }
 }
}
```

Procedimientos y funciones

- Las funciones devuelven un solo valor a la unidad de programa que lo referencia. Generalmente son cálculos.
- Los procedimientos se utilizan para resolver un problema concreto, que no corresponde con un calculo directo.

```
public class Programa {
 public static void main (String [] args){
  imprimir_tabla10();//declaración local
 }
 public static void imprimir_tabla10(){//declaración
  formal
   final int multiplo = 10;
   for (int i = 1 ; i <= MAX; i++) {
      System.out.println(multiplo*i);
   }
 }
}
```

Ejemplo con función

```
public class Programa {
  public static void main (String []
  args){//declaración formal
      double cubo;
      cubo = cubodetres(); //declaración actual
      System.out.println("El cubo de 3.0: " +cubo);
  }

  public static double cubodetres () {//declaración
  formal
      double x = 3.0;
      return x*x*x;
  }
}
```

Diferencias entre funciones y procedimientos

Funciones:

- Calcular… promedio, cantidad, operación compleja
- Obtener…mayor, menor, un índice, un valor
- Retornan un valor de tipo simple
- Puede ser parte de una expresión lógica o siempre estar a la derecha de una asignación

Procedimientos:

- Generar salida de resultados, imprimir
- Procesar o resolver un problema

Ejemplo de diseño

- Escribir un diseño de programa que si el usuario ingresa un número natural imprima la tabla de multiplicar del 5.

```
…
public class Programa {
  public static void main(String[] args){
   definir numero natural
   numero = obtener numero natural()
   imprimir tabla de multiplicar del numero 5
  }
}
```

Ejemplo de diseño

- Escribir un diseño de programa que mientras el usuario ingrese un número natural imprima la tabla de multiplicar del 5.

```
…
public class Programa {
  public static void main(String[] args){
    definir numero natural
    mientras sea numero natural{
      numero = obtener numero natural()
      imprimir tabla de multiplicar del numero 5
    }
  }
}
```

Tipo de variables

- Locales: declarada dentro de un método. Sólo está disponible para el mismo.
- Globales: pueden ser usadas por los distintos métodos.
- Ambas pueden tener el mismo nombre. En ese caso la global no actúa sobre el método.

```
public class Programa {
  public static final int a = 2;//constante global a todos
  public static int b = 2;//variable global a todos
  public static void main(String[] args) {
       int a = 3;//local a main
       System.out.println ("a = "+a);
       System.out.println ("b = "+b);
  }
}
```

Ámbito o alcance

El ámbito o alcance de una variable es la zona accesible.

Variables “globales”: disponibles a todos.	Variables locales • Están disponibles desde su declaración hasta el final del método. • No son visibles desde otros métodos. • Distintos métodos pueden contener variables con el mismo nombre. El nombre de una variable local debe ser único dentro de su ámbito.	Variables de bloque • Están disponibles desde su declaración hasta el final del bloque. • No son visibles desde otros bloques. • Distintos bloques pueden contener variables con el mismo nombre. Si un bloque contiene otro bloque, en el bloque interior no se puede declarar una variable con el mismo nombre.	En todos los casos las variables no tienen un valor inicial por defecto. El programador es el encargado de asignarles valores.

Ámbito o alcance

```
public class Programa {//BLOQUE programa
  int numero = 2;//variable global a todos
  public static void procesar() {//BLOQUE procesar
      int a = 3;//variable local a procesar
      {//BLOQUE A
            System.out.println (a + "," + numero);
            int b = 2; //variable local al BLOQUE A
            System.out.println (a + "," + b);
            {//BLOQUE B
                  int c = 3; //variable local al BLOQUE B
                  System.out.println (a + "," + b + "," + c);
            }//FIN BLOQUE B
            System.out.println (a + "," + b + "," + c);//ERROR
      }//FIN BLOQUE A
  }//FIN BLOQUE procesar
}//FIN BLOQUE programa
```

- Evitar definición de bloques internos
- Evitar variables globales
- Definir constantes globales
- Definir las variables al principio del bloque

Parámetros

- En java el pasaje de parámetros es por copia, se replica la variable y dentro del método se trabaja sobre la replica.
- Al finalizar el método se pierde la replica y la unidad invocadora no percibe cambios en la variable utilizada como parámetro de invocación.

Ejemplo de diseño

- Escribir un diseño de programa que dado un número natural ingresado por el usuario imprima la tabla de multiplicar de ese número.

…

```
public class Programa {
  public static void main(String[] args){
    definir numero natural
    numero = obtener numero natural()
    imprimir tabla de multiplicar (numero)
  }
}
```

Ejemplo de diseño

- Escribir un diseño de programa que mientras el usuario ingrese un número natural imprima la tabla de multiplicar de ese número.

```
…
public class Programa {
  public static void main(String[] args){
    definir numero natural
    mientras sea numero natural{
      numero = obtener numero natural()
      imprimir tabla de multiplicar (numero)
    }
  }
}
```

Ejemplo

```
public class Programa {
 …//COMPLETAR
 public static void main (String [] args){
   …//COMPLETAR
   imprimir_tabla_de_multiplicar(numero);//numero debe coincidir en tipo y lugar
 }
 public static int obtener_numero_natural() {
   int valori = 0;
   BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
   do {
          try {
                    System.out.println ("Ingrese valor valido: ");
                    valori = new Integer(entrada.readLine());
          }
          catch (Exception exc ) {
                    System.out.println( exc );
                    valori = 0;
          }
   }while (valori>0);
   return valori;
   }
 public static void imprimir_tabla_de_multiplicar(int multiplo){//declaración formal
    for (int i = 1 ; i <= MAX; i++) {
          System.out.println(multiplo*i);
    }
 }
}
```

Error típico de funciones

- Un método cuyo tipo de retorno no es void, **siempre necesita devolver algo**.
- Si el código de un método contiene varias sentencias if debe asegurarse de que cada una de las posibles opciones devuelve un valor.

```
public static boolean esPositivo(int x) {
   if (x<0)
         return false;
   if (x>0)
         return true;
  // Error: retorno perdido si x es igual a cero.
}

public static boolean esPositivo(int x) {
   if (x<0)
         return false;
   else
         return true;
}
```

Observaciones

- Comunicación entre métodos → Parámetros
- Evitar el uso de variables globales
- Una variable no declarada localmente → El compilador usa la declaración global
- Un parámetro es una variable local
- Variables locales → Almacenar información temporal

```
public static void … (int a, int b) {
  …
  a = 1;//pierde el valor que tenía como parámetro
  …
}
```

Ejemplo resolver

Realizar un diseño y el programa asociado que dado un carácter ingresado desde teclado (***a*** o ***b***) permita realizar dos operaciones entre dos enteros **N** y **M** menores a **10** ingresados desde teclado. Las operaciones son:

_ Si el usuario ingresa ***a*** obtener la suma entre **N** y **M**.

_ Si el usuario ingresa ***b*** obtener la resta entre **N** y **M**.

Ejemplo de diseño

```
…
public class Programa {
  definir MAX = 10
  public static void main(String[] args){
   definir opcion ingreso, N y M
   opcion = obtener caracter a o b()
   N = obtener numero < MAX()
   M = obtener numero < MAX()
   resolver operaciones (opcion, N, M)
  }
}
```

Ejemplo de diseño

```
…
public class Programa {
  definir MAX = 10
  public static void main(String[] args){
   definir opcion ingreso, N y M
   while (opcion != 'a') o (opcion != 'b')
     opcion = obtener caracter()
   while (N >= MAX) y (M >= MAX) {
     N = obtener numero()
     M = obtener numero()
   }
   resolver operaciones (opcion, N, M)
  }
}
```

Ejemplo de diseño

```
…
public class Programa {
  definir MAX = 10
  public static void main(String[] args){
   definir opcion ingreso, N y M
   while (opcion != 'a') o (opcion != 'b')
     opcion = obtener caracter()
     if(opcion == 'a') o (opcion == 'b'){
       while (N >= MAX) y (M >= MAX) {
         N = obtener numero()
         M = obtener numero()
       }
       resolver operaciones (opcion, N, M)
     }
  }
}
```

Práctico

- Escribir un método que retorne el mayor de dos números. Usar ese método para calcular el máximo de una serie de números ingresados por el usuario (20 números en total).

- Escribir un programa que simule el lanzamiento de un dado (1000 veces) y muestre por pantalla cuantas veces salió el valor del dado correspondiente al número entero N ingresado por el usuario. Considerar el valor N ingresado se corresponda a un valor posible para un dado. Usar la sentencia **Math.random()** que devuelve un valor aleatorio real entre 0 y 1.

  Ejemplo: para asignar un posible valor a la variable dado entero:

  dado = (int) (6*Math.random() + 1)

Práctico

Realizar un programa que dado un número entero ingresado desde teclado (***0***, ***1, 2, 3***) como opción, permita realizar operaciones entre tres floats positivos (***valor1, valor2, valor3***) ingresados desde teclado. Para la opción:

0 _Obtener la raíz cuadrada de la resta entre ***valor1*** y ***valor3***.

1 _Obtener el resultado del promedio entre los tres floats.

2 _Obtener el resultado de la función EquacionMat para los tres valores floats (y en el mismo orden), siendo que la misma está definida por $EquacionMat(v1, v2, v3) = \sqrt{(v1 - v3)}/v2$.

3 _Obtener el resultado del cociente entre el promedio entre los tres floats ingresados y $EquacionMat(valor1, valor1, valor2)$.

Observación: La raíz cuadrada de un numero se calcula con la sentencia: **Math.sqrt(numero)**,

raizcuadrada=Math.sqrt(valor1);

El resultado que se obtiene para cada opción deberá ser impreso por pantalla.