Modularización

Sitio: Agencia de Aprendizaje a lo largo de la Vida
Curso: Tecnicas de Programación 1° G
Libro: Modularización

Imprimido por: Eduardo Moreno
Día: martes, 22 de octubre de 2024, 19:11

Tabla de contenidos

1. Para modular, abstraer...

Durante años se han desarrollado diferentes técnicas para construir programas. El primer gran avance se conoce como programación modular conjuntamente con la programación estructurada.

La programación modular utiliza abstracción de procedimientos.

Ahora bien ¿Qué significa abstraer y por qué este concepto resulta tan importante en la ciencia informática y en específico para modular?

Abstraer significa reducir algunos aspectos irrelevantes, concepto o proceso para enfocarse en los aspectos más importantes y generales.

Esto resulta extremadamente útil en la ciencia informática, ya que abstraer un proceso complejo permite identificar los elementos básicos que se necesitan para construir un programa.

Esto hace que el proceso de creación de programas sea mucho más eficiente, ya que el programador puede comenzar con los elementos más simples y añadir detalles con el tiempo. Además, abstraer también permite que los/as programadores/as compartan sus conocimientos y trabajen en equipo para construir programas más complejos.

2. ¿Modular problemas? Un ejemplo

En el siguiente ejemplo vamos a dividir un problema llamado [mudanza], en sub-problemas más pequeños como:

1. Conseguir un departamento.
2. Firmar el contrato.
3. Conseguir una empresa de mudanza.
4. Realizar la mudanza.

De esta forma, nos podríamos ocupar de resolver cada uno de estos sub-problemas, realizando la mudanza al terminar con el último de ellos.

Esta técnica se usa mucho en programación ya que programar no es más que resolver problemas, y se le suele llamar diseño descendente, metodología del "*divide y vencerás*" o programación top-down que vimos en la semana 1.

Es evidente que si esta metodología nos lleva a tratar con sub-problemas, entonces también tenemos la necesidad de poder crear y trabajar con subprogramas para resolverlos.

A los sub-problemas que son sub-programas se les suele llamar módulos, de ahí viene el nombre de programación modular.

3. Entonces, la programación modular es...

La programación modular consiste en definir módulos, una especie de cajas negras que tienen una forma de comunicarse (interfaz) claramente definida.
Usando abstracción de procedimientos el/la programador/a puede separar el qué hace el módulo del cómo lo hace.

De esta manera es posible descomponer funcionalmente un programa en subprogramas. El propósito es facilitar la resolución de problemas, dividiendo el problema en sub-problemas más simples, basándose en la metodología «*divide y vencerás*».

Cuando adoptamos la técnica propuesta (identificar sub-problemas dentro de un problema), encontramos que existen un conjunto de estos ítems que están incluidos en más de un programa. Hasta podríamos pensar que algunos están en la mayoría de los programas.

Por lo tanto, si logramos identificar a los procesos comunes, y desarrollamos los sub-programas, obtendremos una solución que nos brinda algunas ventajas:

- La posibilidad de volver a utilizarlos cada vez que se necesite.
- Hacer pruebas para que funcione una sola vez y luego confiar en ella.
- Concentrarse en los subproblemas aún no resueltos.

Todas estas ventajas que provienen de la posibilidad de usar nuevamente una solución ya generada nos dan como resultado un mejor aprovechamiento del tiempo de programación.

También hemos podido observar en los ejemplos y la ejercitación propuesta en las prácticas anteriores, que determinadas acciones se repiten más de una vez en el mismo programa y a veces en más de uno.

Por ejemplo

- Pedir un número y validarlo
- Calcular el importe a abonar a cada empleado de una empresa

En los casos en que se repite el código varias veces en el programa, si detectamos un error en la ejecución (una vez desarrollado el mismo), corremos con la desventaja de tener que buscar en todo el código las múltiples repeticiones y corregirlo en cada una de éstas.

A su vez, es conveniente dividir un programa en módulos para una mayor facilidad de lectura del mismo. Lógicamente, verás que esto es así a medida que tus propios programas crezcan en cantidad de líneas y complejidad.

Entonces, para subsanar estos problemas es que debemos pensar en describir subprogramas que resuelvan las situaciones repetidas. Y por supuesto, utilizarlos toda vez que resulte necesario.

4. Principales ventajas de la programación modular

Una de las principales ventajas de la programación modular es la reutilización de código.

Los módulos pueden ponerse en una biblioteca y usarse en diferentes programas sin necesidad de escribir el código desde cero. Esto mejora la velocidad de desarrollo y reduce los errores de programación.

Otra ventaja es que permite al/la programador/a descomponer un gran problema en unidades pequeñas y manejables que se pueden probar y depurar fácilmente. Esto también ayuda a mantener el código limpio y fácil de entender.

También es útil para crear programas extensibles. Esto significa que un programa puede ser ampliado con módulos nuevos sin tener que reescribir todo el código y hace que sea mucho más fácil añadir nuevas características a programas existentes.

5. ¿Qué función cumplen los módulos?

¿Qué funciones cumple cada módulo?

Cada módulo cumple una función específica (qué hace) y, en algunos casos, comparte información con otras (interfaz).

En general, en las soluciones modularizadas, el programa principal es también un módulo, llamado programa principal (por ejemplo, como veremos a continuación [preparar tarta de manzanas]).

El módulo [programa principal] se encarga de controlar todo lo que sucede y es el responsable de transferir el control a los otros módulos de modo que ellos puedan llevar adelante su función o tarea y resolver el problema.

Cada uno de los otros módulos, cuando finaliza su trabajo, devuelve el control al módulo que lo invocó.

Cada módulo tiene un objetivo perfectamente definido y pueden ser desarrollados y resolverse en forma independiente uno de otro. La combinación de los distintos módulos deben resolver el problema original. Si la tarea de un módulo es demasiado compleja, se puede dividir en subtareas.

5.1. Cada programa contiene un módulo

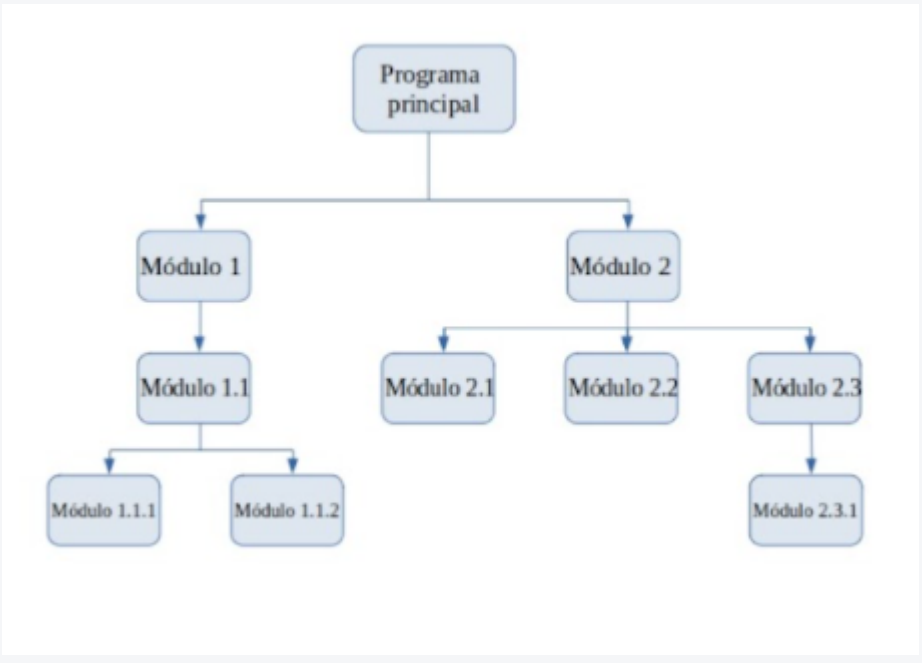
Cada programa:

- contiene un módulo denominado programa principal que controla todo lo que sucede
- se transfiere el control a submódulos (posteriormente se denominarán subprogramas), de modo que ellos puedan ejecutar sus funciones
- cada submódulo devuelve el control al módulo principal cuando se haya completado su tarea.
- Si la tarea asignada a cada submódulo es demasiado compleja, éste deberá romperse en otros módulos más pequeños.
- El proceso sucesivo de subdivisión de módulos continúa hasta que cada módulo tenga solamente una tarea específica que ejecutar.
- Esta tarea puede ser entrada, salida, manipulación de datos, control de otros módulos o alguna combinación de éstos.

Recordá:
Un módulo puede transferir temporalmente (bifurcar) el control a otro módulo; sin embargo, cada módulo debe eventualmente devolver el control al módulo del cual se recibe originalmente el control.

Los módulos son independientes en el sentido en que ningún módulo puede tener acceso directo a cualquier otro módulo excepto el módulo al que llama y sus propios submódulos.

Sin embargo, los resultados producidos por un módulo pueden ser utilizados por cualquier otro módulo cuando se transfiera a ellos el control. Dado que los módulos son independientes, diferentes programadores pueden trabajar simultáneamente en diferentes partes del mismo programa. Esto reducirá el tiempo del diseño del algoritmo y posterior codificación del programa.



Además, un módulo se puede modificar radicalmente sin afectar a otros módulos, incluso sin alterar su función principal. La descomposición de un programa en módulos independientes más simples se conoce también como el método de divide y vencerás (divide and conquer).

- Cada módulo se diseña con independencia de los demás, y siguiendo un método ascendente o descendente se llegará hasta la descomposición final del problema en módulos en forma jerárquica.
- Los módulos se conectan entre sí dando lugar a una estructura modular en árbol que permite resolver el problema de programación planteado.

¿Qué es un método o rutina?

Un método es un bloque de código que contiene una serie de instrucciones. Un programa hace que se ejecuten las instrucciones al llamar al método, en caso de ser necesario podremos especificar los argumentos necesarios para su funcionamiento. Haciendo analogía con diagramación lógica, podemos decir que:

- Un método que retorna un valor tiene el comportamiento de una función.
- Un método que no retorna un valor tiene el comportamiento de un procedimiento.

La definición del método especifica los nombres y tipos de todos los parámetros necesarios.

Si el código de llamada llama a métodos, éste le puede o no proporcionar valores concretos denominados argumentos para que el método pueda funcionar. La cantidad de argumentos deben coincidir con la cantidad de parámetros que tiene el método. Además deben ser compatibles con el tipo de parámetro, pero el nombre del argumento utilizado en el código de llamada no tiene porque ser el mismo que el nombre del parámetro definido en el método. Es decir, en la llamada se pasan argumentos y del lado del método se los recibe como parámetros.

Por ejemplo: si nuestro programa invoca al método llamado sumar2Numeros con los argumentos 3 y 8 almacenados en las variables enteras numero1 y numero2 respectivamente. Sería:

sumar2Numeros(numero1,numero2)

En este caso, numero1 y numero2 son argumentos de la llamada a la función sumar2Numeros

En el desarrollo de la función quedaría:

sumar2Numeros(n1, n2)

Las variables n1 y n2 del lado de la función son los parámetros que recibe la función al momento de ser invocada.

Notemos n1 tomará el valor 3 (pasado a través del argumento numero1) y que n2 tomará el valor 8 (pasado a través del argumento numero2).

De esta manera podrá realizar la suma entre los 2 números recibidos en sus parámetros y devolver el resultado.

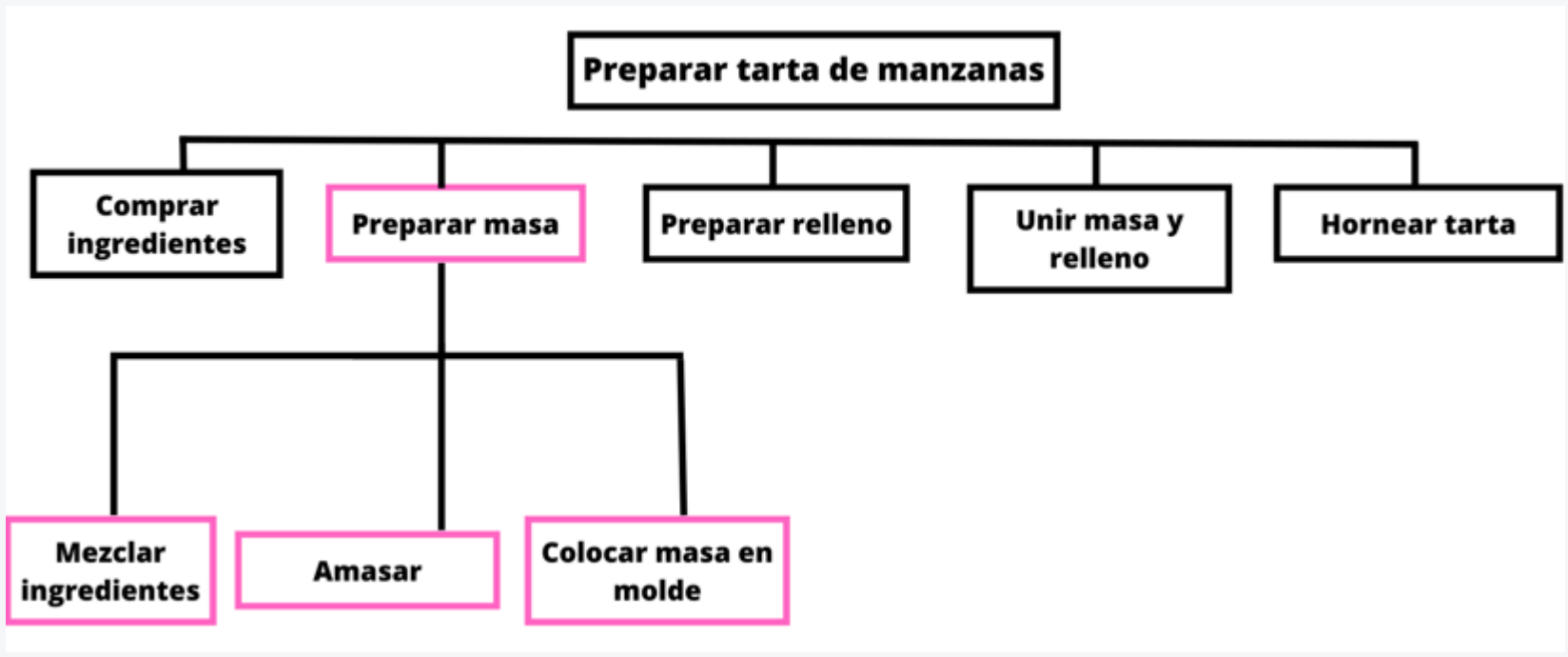
5.2. Un ejemplo: Preparar la tarta de manzanas

Analizá el siguiente problema: Preparar una tarta de manzanas

Observá la siguiente imagen: ¿Qué tareas se mencionan para preparar la tarta de manzanas?

Ahora, analizá la tarea: "Preparar masa" que implica hacer más de una tarea y sub-tareas. ¿Cuáles son?

Observá el siguiente esquema



Y se puede encontrar más sub tareas, por ejemplo para "preparar relleno" implica:

1. pelar las manzanas
2. cortar las manzanas.
3. cocinar las manzanas.

Dos aspectos claves:

- Cada módulo se analiza, se desarrolla y se pone punto por separado.
- Para los nombres de los módulos se utilizan palabras que son verbos en infinitivo, ya que indican una tarea a realizar por parte del módulo.

Pensá otra cosa

¿Será muy distinto el módulo “Preparar masa” para realizar una tarta de pera y una de frutos rojos? ¡Claro que no! Los pasos serán los mismos, con lo cual ya lo tenemos pensado y sabemos que funciona. Lo mismo ocurre con “Unir masa y relleno” y “Hornear tarta”. Sólo deberíamos repensar el método “Preparar relleno”.

6. Ejemplo: Construir una pared

Te proponemos comprender el mecanismo para *particionar* problemas en sub-problemas.

Seguramente conocemos de qué se trata cuando hablamos de construir una pared; ahora bien, indudablemente no todos sabemos cómo se realiza esta actividad. Por lo tanto, si aguardamos un poco, vamos a aprender los rudimentos básicos de la construcción.

- Enunciado del problema

Necesitamos levantar una pared de ladrillos a la vista cuyas medidas son 20 metros de longitud x 5 metros de altura.

Debe realizarse en el patio trasero de una casa. Para poder realizarlo se cuenta con la cantidad necesaria de ladrillos de 20cm de largo x 7 cm de alto.

- Hipótesis

El largo es lineal, es decir, un ladrillo al lado del otro sin hacer zigzag. La altura que utiliza la mezcla es de 3 cm sobre cada ladrillo. Esto servirá para calcular la altura de una hilera.

- Datos provistos

Largo, alto, ubicación, cantidad de ladrillos disponibles (nos especifican que tenemos suficientes), medidas del ladrillo, que servirán para realizar el cálculo de cuántos tendremos que utilizar.

Vamos a suponer además, que contamos con todos los materiales y herramientas de construcción necesarias.

- Materiales: ladrillos, cemento, cal, arena y agua.
- Herramientas: baldes, palas, cucharas, hilo, nivel, etc.

Si dividiéramos el problema en tareas: ¿Cuáles son las tareas que hay que resolver para levantar la pared?

6.1. ¿Cuáles son las tareas que hay que resolver para levantar la pared?

Si dividiéramos el problema en tareas: ¿cuáles son las tareas que hay que resolver para levantar la pared?

Para responder adecuadamente a esta pregunta primero deberíamos conocer cómo es el proceso de construcción y luego qué se debe hacer para construir la pared pedida.

Como podemos imaginar, para construir, además de conocer las técnicas adecuadas que permitan concluir la obra cuanto menos decorosamente, debemos saber qué necesitamos en cuanto a materiales, esto es: cal, arena, cemento y agua, mezclados en determinadas proporciones y hasta una consistencia determinada.

Primera tarea: reparar mezcla

Además sabemos que una pared consiste en un conjunto superpuesto de hileras de ladrillos, unidas entre sí mediante la mezcla.
Cada una de las hileras se conforman colocando la mezcla y sobre la misma un ladrillo junto al otro (separados por 2 centímetros aproximadamente).

Segunda tarea: construir hilera

Para poder comenzar a levantar la pared, debemos preparar adecuadamente el lugar (limpiar, alisar, mojar, etc.), obteniendo así la tercer tarea.

Tercera tarea: preparar el lugar

Definimos la tercer tarea: que no vamos a dejar el lugar y las herramientas sucias, por lo cual debemos realizar la cuarta tarea:

Cuarta tarea: limpiar herramientas y lugar

Quinta tarea: calcular cantidad de hileras

Tenemos que averiguar la cantidad de hileras que debemos construir para cumplir con la medida solicitada. Para esto debemos realizar algunos cálculos.

Recordá que debemos construir una pared con una longitud de 20 metros, y una altura de 5 metros, según las especificaciones.

6.2. ¿Cuántas hileras de ladrillos debemos construir?

¿Cuántas hileras de ladrillos debemos construir?

Para obtener la cantidad de hileras de ladrillos que necesitamos para levantar el muro, sabemos que la altura del ladrillo es de 7 centímetros, y la mezcla ocupa 3 centímetros más, por lo tanto una hilera mide 10 centímetros de altura.

Medidas

1 ladrillo	**7 cm**
mezcla de unión	1cm
altura de hilera	10 cm
1 hilera	10 cm
10 hileras	1 m

Por último, para completar la altura de 5 metros debemos levantar 50 hileras.

Cantidad

1m	10hileras
5 m	50 hileras

Este mismo proceso que estamos realizando para calcular la cantidad de hileras, debemos realizarlo dentro de la resolución del ejercicio.

> Aprovechando que ya conocemos el uso de variables y constantes, utilicemos algunas para realizar correctamente los cálculos.

```
Algoritmo preparandoLaPared
    //Declaración de constantes
    Definir LARGO_ESPERADO como Entero //utilizada para indicar el largo de la pared expresada en metros
    LARGO_ESPERADO = 20
    Definir ALTO_ESPERADO Como Entero //utilizada para indicar la altura de la pared expresada en metros
    ALTO_ESPERADO = 5
    Definir ALTO_LADRILLO Como Real //Alto estándar de un ladrillo expresado en metros.
    ALTO_LADRILLO = 0.07
    Definir ALTO_MEZCLA Como Real //Alto estándar de la mezcla sobre el ladrillo
    ALTO_MEZCLA = 0.03
    Definir LARGO_LADRILLO Como Real //Largo estándar de un ladrillo expresado en metros.
    LARGO_LADRILLO = 0.20

    //*Declaración de variables
    Definir cantLadrillosxHilera Como Entero // Almacenará la cantidad de ladrillos que se necesitan para una hilera
    Definir cantHileras Como Entero // Almacenará la cantidad de hileras que hay que construir.
FinAlgoritmo
```

6.3. En pseudocódigo

Si pensamos el proceso de construcción utilizando el pseudocódigo que ya conocemos, una posible solución podría ser:

1. **Preparar el lugar:**
   - 1.1 Limpiar el lugar.
   - 1.2 Alisar el lugar.
   - 1.3 Mojar el lugar.
2. **Realizar cálculos:**
   - Almacenar en cantLadrillosxHilera el resultado de: LARGO_ESPERADO \ LARGO_LADRILLO.
   - Almacenar en cantHileras el resultado de: ALTO_ESPERADO \ (ALTO_LADRILLO + ALTO_MEZCLA).
   - Repetir desde 1 a cantHileras: (inicio ciclo FOR de hileras).
3. **Preparar mezcla para una hilera:**
   - Tomar Balde Vacío.
   - Repetir.
   - Agregar Cal.
   - Agregar Arena.
   - Agregar Agua.
   - Agregar Cemento.
   - Mezclar mientras que la consistencia no sea la adecuada.
4. **Construir hilera:**
   - Repetir desde 1 hasta cantLadrillosxHilera: (inicio ciclo FOR de cada hilera).
   - Poner mezcla en el lugar correspondiente.
   - Colocar un ladrillo.
   - Fin-Repetir(fin ciclo FOR de cada hilera).
   - Fin-Repetir (fin ciclo FOR de hileras).
5. **Limpiar herramientas y lugar.**
6. **Poner un cartel que diga: "El material está fresco".**
7. **Guardar herramientas.**

Tené en cuenta que:

- Utilizar algoritmos en donde una tarea puede dividirse en otras más pequeñas, por ejemplo, preparar el lugar, implica realizar tres acciones numeradas desde el 1.1 al 1.3.
- Todas comienzan con 1 y están desplazadas unos espacios hacia la derecha, se entiende que es la tarea 1 la que está compuesta por estas 3 acciones.

Notemos que para entender mejor cómo está resuelto un pseudocódigo, podemos dividirlo y quitarle la mayor cantidad de líneas posibles, pero siempre realizando todas las tareas.

A continuación conocé el pseudocódigo para cada una de las tareas

6.4. Pseudocódigo por cada una de las tareas

Notemos que para entender mejor cómo está resuelto un pseudocódigo, podemos dividirlo y quitarle la mayor cantidad de líneas posibles, pero siempre realizando todas las tareas.

De esta forma, crearemos un pseudocódigo por cada una de las tareas que habíamos encontrado en la resolución. Así, la nueva representación de la solución quedaría de la siguiente forma:

Pseudocódigo principal

1. Preparar el lugar.
2. Realizar Cálculos.
3. Repetir desde 1 a cantHileras: (inicio ciclo FOR de hileras).
    - Preparar mezcla para una hilera.
    - Construir hilera.
4. Fin-Repetir (fin ciclo FOR de hileras).
5. Limpiar herramientas y lugar.
6. Poner un cartel que diga: “El material está fresco”.
7. Guardar herramientas.

Ahora, sólo expresamos el encabezado de las tareas, clarificando la lectura del pseudocódigo y de la resolución del problema.

Pseudocódigo: preparar el lugar

1. Limpiar el lugar.
2. Alisar el lugar.
3. Mojar el lugar.

Vemos cómo generamos un algoritmo para la tarea Preparar el lugar, detallando los pasos a seguir para cumplirla. No cabe duda que la tarea Limpiar el lugar es más simple que Preparar el lugar, que es compuesta.

Pseudocódigo: Realizar Cálculos.

1. Almacenar en cantLadrillosxHilera el resultado de: LARGO_ESPERADO \ LARGO_LADRILLO
2. Almacenar en cantHileras el resultado de: ALTO_ESPERADO \ (ALTO_LADRILLO + ALTO_MEZCLA)

Pseudocódigo: Preparar Mezcla para una Hilera.

1. Tomar balde vacío.
2. Repetir
    - Agregar Cal.
    - Agregar Arena.
    - Agregar Agua.
    - Agregar Cemento.
    - Mezclar.
3. Mientras que la consistencia no sea la adecuada.

Pseudocódigo: Construir Hilera

1. Repetir desde 1 hasta cantLadrillosxHilera: (inicio ciclo FOR de cada hilera)
    - Poner mezcla en el lugar correspondiente.

        - Colocar un ladrillo.

    2. Fin-Repetir(fin ciclo FOR de cada hilera)

La solución que proponemos no es ni más ni menos que el desarrollo mediante pseudocódigo de la descripción del proceso indicado en la consigna.

7. En resumen

Repasá los conceptos centrales de los contenidos por medio del siguiente video

Ten presente la diagramación de programas modularizados

La experiencia ha demostrado que un programa que presenta dificultades para ser modificado está condenado a la muerte informática. Debemos procurar, por lo tanto, que los programas sean a la vez flexibles y transportables:

- flexibles para que se adapten con facilidad a cualquier cambio
- transportables de forma que cualquier nuevo proceso pueda utilizar sus métodos (procedimientos y funciones) sin introducir grandes cambios.

Conviene para ello emplear técnicas de programación que faciliten el desarrollo de software fácilmente modificable. De esta forma el/la programador/a que utilice el software desarrollado anteriormente no tendrá que efectuar dos tareas muy tediosas:

- Escribir partes del programa ya escritas.
- Probar subprogramas ya probados.

La programación modular es uno de los métodos de diseño más flexibles y potentes para mejorar la productividad de un programa.