<codoacodo/>

Agencia de Aprendizaje a lo largo de la vida

Codo a Codo inicial
Clase 25

BA

Les damos la bienvenida

Vamos a comenzar a grabar la clase

Formulario de presentismo

Link:

Clase 24

Funciones return

- Repaso de contenidos.

Clase 25

Funciones Combinadas

- Funciones que llaman a Funciones.
- Funciones que se llaman a si misma.

Clase 26

Funciones Combinadas

- Repaso de contenidos.

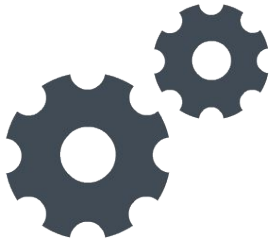
Write Once, Run Anywhere

(Escríbelo una vez, ejecútalo en cualquier lugar)

Funciones Anidadas

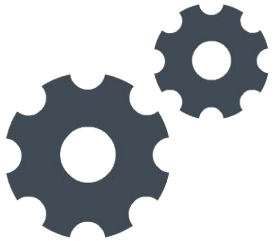
Método de anidamiento de funciones

- Lo más común en la programación es la **anidación de llamadas entre métodos**, porque en circunstancias normales, **un solo método es probable que no resuelva un problema.**
- Ya que **si el problema es grande** y hallamos la solución con **un solo método**, **la lógica del código y la lista de parámetros en él inevitablemente se volverán relativamente complicados**, lo que conduce a no usarlo.



Método de anidamiento de funciones

- Por lo tanto, como dice el dicho **cada método nos aportará su granito de arena.**
- Para resolver un problema específico, se puede utilizar una **función simple combinándola con otras o consigo misma y usándola de manera más compleja**.



Ejemplo de anidamiento de funciones

- Por ejemplo, si necesitamos calcular el área de superficie de un cilindro podemos:

    a. **Pensar en una única solución** que englobe todas las variables necesarias con su método lo cual se torna algo compleja de entender y utilizar.

    b. O bien **descomponer el problema en problemas más sencillos.**

Ejemplo de anidamiento de funciones

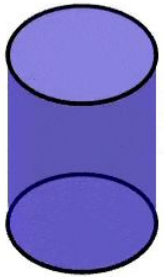
- Optando por la **segunda lógica** podríamos **reutilizar las funciones** programadas anteriormente:
    a. **Calcular el área de un círculo** y,
    b. **Calcular el área de un rectángulo**.

Nota: *Para calcular el área del rectángulo podemos valernos del perímetro del círculo para hallar uno de los lados del rectángulo.*

Ejemplo de anidamiento de funciones

De esta manera el cálculo del área de la superficie del cilindro se puede obtener:

- **Sumando** el **área de la tapa y el fondo** del cilindro (círculo) y el **área del lado lateral del cilindro** (rectángulo).



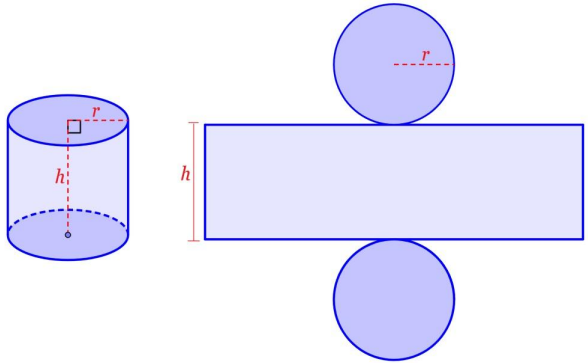
Solo necesitamos pasar los parámetros razonablemente y devolver el valor.

Lo vemos a continuación.

Ejemplo de anidamiento de funciones

La superficie del cilindro la obtenemos de anidar dentro de una función las funciones realizadas anteriormente

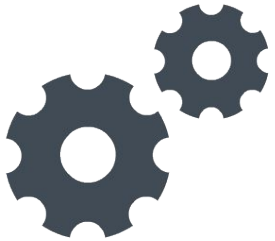
```
static double superficieCilindro (double radio, double altura){
   //Superficies de tapa y piso
   double supCirculos = 2*superficieCirculo(radio);
   //Base del rectángulo
   double base = perimetroCirculo(radio);
   //Superficie del rectangulo
   double supRect = superficieRectangulo(base, altura);

   return supRect+supCirculos;
}
```



Funciones Recursivas

Qué es Recursividad

- En general, **la recursividad es el proceso de definir algo en términos de sí mismo** y es algo **similar a una definición circular**, por ejemplo cuando veíamos él autoincremento de una **variable contadora**, que se sumaba a sí misma.
- El **componente clave** de un método recursivo es una **declaración que ejecuta una llamada a sí mismo.**



Qué es Recursividad

- La recursividad es un **mecanismo de control.**
- **La recursividad** es una técnica potente de programación que **puede utilizarse en lugar de la iteración para resolver determinados tipos de problemas.**
- **No es un ciclo sino más bien una caja de donde se saca otra caja cada vez mas pequeña.**

Ejemplos de función recursiva

```java
// Método Java recursivo para
calcular el factorial de un número

static double factorial(int n){

    if (n==0){

        return 1;

    }else{

        return n*(factorial(n-1));

    }

}
```

Explicación del factorial de 5

factorial de 5

```
factorial(n)= return n*(factorial(n-1));
// La función siempre devolverá está expresión mientras n no sea 0.
factorial(5)= return 5*( factorial(4));
factorial(5)= return 5*( 4*(factorial(3));
factorial(5)= return 5*(4*( 3*(factorial(2)));
factorial(5)= return 5*(4*(3* (2*(factorial(1))));
factorial(5)= return 5*(4*(3*(2*( 1*(factorial(0))))));
if (n==0) factorial (0) = 1;
factorial(5)= return 5*(4*(3*(2*(1* (1)))));
```

Cómo funciona un método recursivo

- La solución iterativa es fácil de entender. **Utiliza una variable para “acumular”** los productos y obtener la solución. En la solución recursiva se **realizan llamadas al propio método con valores de n cada vez más pequeños** para resolver el problema.
- Cada vez que se produce una **nueva llamada** al método **se crean en memoria de nuevo las variables y comienza la ejecución del nuevo método.**
- Para entender el funcionamiento de la recursividad, **podemos pensar que cada llamada supone hacerlo a un método diferente, copia del original,** que se ejecuta y devuelve el resultado a quien lo llamó.

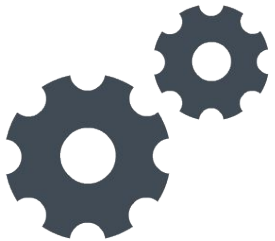
Un método recursivo debe contener:

1. **Caso base:** Siempre ha de existir uno o más casos en los que los valores de los parámetros de entrada **permitan al método devolver un resultado directo.** Estos casos también se conocen como **solución trivial del problema**.

- En el ejemplo del factorial el caso base es la condición:

```
if (n==0)
    return 1;
```

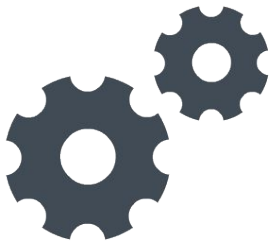
- si n=0 el resultado directo es 1. **No se produce llamada recursiva**



Un método recursivo debe contener:

2. **Llamada recursiva:** Si los valores de los parámetros de entrada **no cumplen la condición del caso base se llama recursivamente al método.** En las llamadas recursivas **el valor del parámetro** en la llamada **se ha de modificar** de forma que se **aproxime cada vez más** hasta alcanzar al valor del **caso base**.

- En el ejemplo del factorial en cada llamada recursiva se utiliza **n-1**
- `return n * ( factorial(n-1) );`
- por lo que en **cada llamada el valor de n se acerca más a 0 que es el caso base.**



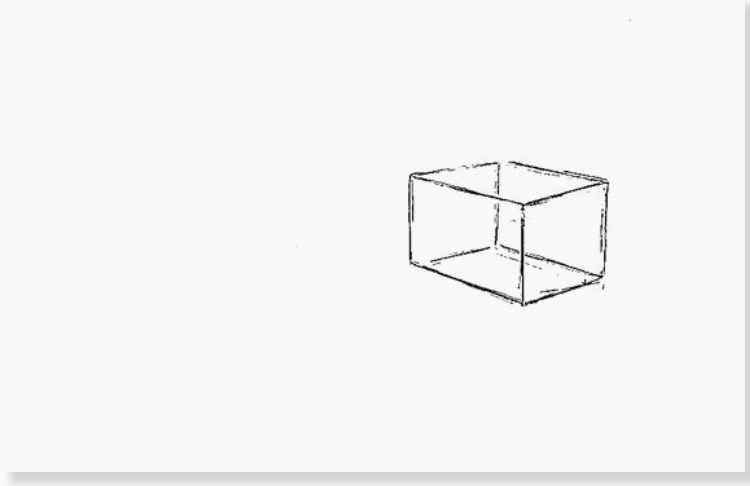
Repo de Clase

https://app.codingrooms.com/w/80onUUtmIBwl

Desafío de clase

Desafío I de clase

Hallar la superficie y volumen de un prisma, cuyos datos podamos ingresar por teclado.



Desafío II de clase

Estudiá la teoría y repasá lo dado en clases.
La clase que viene hacemos un práctico de repaso.

Desafío III - Pensamiento modular

Inspeccionando código
https://app.codingrooms.com/w/uahyqBz8QHSi

1. Dale fork dentro de coding para duplicar el trabajo y poder editarlo
2. Comenzá completando la clase Funcion
3. Luego completá la clase Main
4. Compilá el progama.

Herramientas que utilizamos en clases

IDE IntelliJ o VSCode+plugins

No te olvides de dar el presente

Recordá:

- Revisar la Cartelera de Novedades.
- Hacer tus consultas en el Foro.
- Realizá los ejercicios obligatorios.

Todo en el Aula Virtual.