1. ****Que es clase y que es objeto? Definición y ejemplos****

Una clase es un modelo, una plantilla en cuyo contenido se encuentran instrucciones que explican cómo construir objetos de esa clase, qué atributos tendrán esos objetos y qué propiedades.

Un objeto viene a ser la ejecución de las instrucciones dadas en el constructor de la clase. El objeto luego puede utilizarse de acuerdo a las especificaciones que están en la clase en cuanto a sus atributos y propiedades.

Ejemplo:

Clase Persona {

**// Atributos**

Int edad

String nombre

**// constructor**

Public Persona(int e, string n) {

Edad = e; nombre = n

}

**// propiedades**

Public void gritar() {

Print(“Mi nombre es ” + nombre + “!!!”)

}

Public void envejecer() {

Edad += 1

}

}

Y la forma de crear y utilizar un objeto:

Public static void main(String[] args) {

**// Crea el objeto obj que se llama Luis y tiene 25 años**

Persona obj = new Persona(25,”Luis”)

**// Llamadas a propiedades y atributos**

Obj.gritar()

Obj.envejecer()

Print(obj.getNombre() + “ tiene ” + obj.getEdad() + “ años”

}

1. ****Explique el concepto de recurrencia en programación. A modo de práctica resuelva los siguientes ejercicios con una función recurrente (utilice programación defensiva para los parámetros):****
   1. **La suma de los primeros n números impares**
   2. **Si n=1, imprimir ‘ab’, si n=2, imprimir ‘aabb’ y asi sucesivamente. Escribir la función que toma como argumento un entero n.**

En programación, recurrencia se refiere a un proceso o procedimiento (puede ser una función, un método) que está definido por sí mismo.

Por ejemplo:

f(n) = n + f(n)

Para realizar correctamente la recurrencia y no caer en un bucle infinito el proceso o procedimiento debe incluir una condición por la cual el método deja de ser recurrente. Por ejemplo:

f(n) = 1 si n = 1 y f(n-1) para n > 1

Los programas están resueltos a continuación, primero vamos a mostrar la suma de los primeros números:

public int suma(int n) {

if (n == 1)

return 1;

return n + suma(n-1);

}

Y el Segundo programa, que imprime aabb si n=2

public String ab(int n) {

if (n == 1)

return “ab”;

return “a” + ab(n-1) + “b”;

}

1. Explique cuál es la complejidad de un algoritmo de ordenamiento como el algoritmo de burbuja. Fundamente su respuesta para llegar a la expresión matemática de la complejidad de este tipo de algoritmos

El algoritmo de ordenamiento de burbuja es muy similar al algoritmo de ordenamiento simple que se ha visto en el curso. Específicamente el pseudo algoritmo es el siguiente:

func bubblesort( var a as array )

for i from 1 to N

for j from 0 to N - 1

if a[j] > a[j + 1]

swap( a[j], a[j + 1] )

end func

Como se puede ver en el algoritmo, básicamente se recorre el arreglo N x N veces.

En el peor de los casos entonces se tiene una complejidad de tipo n^2.

1. Escriba un pequeño programa donde se simule una carrera de caballos. El programa debe contar con 2 clases, Programa y Caballo. Ademas, Caballo debe implementar Runnable para que pueda ser utilizado como Thread. Se deja a criterio del usuario el código de run() para todos los caballos simulando una carrera.

Para realizar este código vamos a tratar de NO utilizar el sleep y math.random ya que nos traen muchos problemas para poder definir claramente que el proceso se realice realmente al azar.

Public class Caballo implements Runnable {

private int id;

public Caballo(int id) {

This.id = id;

}

public void run() {

Int max = 10000000;

For(int i=10000; i< max; i++) {

Double m1 = Math.sqrt(i);

Double m2 = Math.atan(m1 / max);

If (i % 50 == 0)

Thread.sleep(10);

}

System.out.println(“El caballo ” + id + “ terminó la carrera”;

}

}

Public class Principal {

Public static void main(String[] args) {

int nbCaballos = 4;

Caballo[] carrera = new Caballo[nbCaballos];

for(int i = 0; i < nbCaballos; i++) {

carrera[i] = new Caballo();

}

System.out.println(“Comenzar carrera con ” + nbCaballos);

for(int i = 0; i < nbCaballos; i++) {

Thread t = new Thread(carrera[i]);

t.start();

}

System.out.println(“Caballos corriendo ”);

}

}

De esta manera, una vez creados los caballos, por lo tanto los threads, se ejecutan estos y las operaciones de CPU son las que hacen, de acuerdo a lo que el CPU asigne a cada una, que los caballos corran y terminen de manera diferente cada vez.

1. Se dice que cuando el universo comenzó se inició un Hanoi con 64 discos. Y que, de la misma manera, cuando el Hanoi sea resuelto, el universo terminará. Si el universo comenzó hace 13.700.000.000 años[[1]](#footnote-1) y los monjes en Hanoi logran mover correctamente un anillo en 1 segundo. ¿Cuánto tiempo le queda de vida al universo?

La complejidad del algoritmo de Hanoi es 2^n-1. Eso significa que se necesitan 2^64 movimientos para resolver el Hanoi. Eso corresponde aproximadamente a 1.6E19, aquí el desarrollo:

Un año tiene

Con lo cual, para saber la cantidad de años dividimos 1.6E19 / 3.1E7 lo cual nos da un aproximado de 5E11 años. A este número le restamos 13.7E9 nos da 4.87E11 años. Falta todavía mucho tiempo.

1. Se puede ver fácilmente que Quicksort se comporta muy mal para arreglos pequeños (de 50 elementos o menos), adicionalmente, dependiendo cómo elegimos el pivote, el algoritmo mejora o no. ¿Qué alternativas se tienen para mejorar el algoritmo? Proponga una implementación con mejora

Un arreglo, cuando debe ser ordenado con Quicksort generalmente tiene una performance de O(n) = n log(n). si observamos los valores posibles para esta curva y los comparamos con los valores de la curva de algoritmos tradicionales de ordenamiento (burbuja, inserción, etc.) que tienen performance O(n) = n2 entonces podemos ver que:

|  |  |  |
| --- | --- | --- |
| N | N^2 | NLOG(N) |
| 4 | 16 | 8 |
| 8 | 64 | 24 |
| 16 | 256 | 48 |
| 32 | 1024 | 320 |
| 64 | 4096 | 384 |

Se puede ver que en algún lugar entre 32 y 64 el valor de complejidad de Quicksort se estanca y le saca gran ventaja al algoritmo de ordenamiento de n^2.

Normalmente no habría razón por la cual Quicksort pierda en ninguna circunstancia, pero para tamaños pequeños de arreglos (~50) se nota demasiado la mala distribución de menores y mayores que hace Quicksort alrededor del pivote, para arreglos que no tienen regla específica, es decir, para aquellos que están desordenados.

Entonces, cuando el

1. De la definición del patrón de diseño Observer. Indique cuál es su función principal, su diagrama de clases y de un ejemplo de su uso en Java.
2. Haga un método que tome como argumento una matriz m1 y una matriz m2. El método revisa que las matrices tengan las dimensiones adecuadas para hacer la multiplicación m1 x m2 y por supuesto, hace la multiplicación de matrices y devuelve la matriz resultante.

A continuación el código para la multiplicación de matrices en Java:

public static Double[][] multiplicar(Double[][] A, Double[][] B) {

int aRows = A.length;

int aColumns = A[0].length;

int bRows = B.length;

int bColumns = B[0].length;

if (aColumns != bRows) {

throw new IllegalArgumentException("A:Rows: " + aColumns +

" did not match B:Columns " + bRows + ".");

}

Double[][] C = new Double[aRows][bColumns];

for (int i = 0; i < aRows; i++) {

for (int j = 0; j < bColumns; j++) {

C[i][j] = 0.00000;

}

}

for (int i = 0; i < aRows; i++) { // aRow

for (int j = 0; j < bColumns; j++) { // bColumn

for (int k = 0; k < aColumns; k++) { // aColumn

C[i][j] += A[i][k] \* B[k][j];

}

}

}

return C;

}

1. Proponga el modelo de diagrama de clases para dibujar el histograma de una imagen. Indique el código central donde se calcula el histograma de una imagen.
2. En la clase Cadena del texto, escribir el código del método

public void unirCon(Cadena<E> otraCadena) {

…

}

* 1. Para que se puedan unir dos cadenas

1. De la definición del patrón de diseño Iterator. Indique cuál es su función principal, su diagrama de clases y de un ejemplo de su uso en Java.
2. Implementar eliminar(int n) n es la posición comenzando de 0.

Para eliminar en una posición se debe tener el cuidado de los extremos. Qué pasa cuando queremos eliminar la posición 0 o la ultima posición. Aquí el código:

public void eliminar(int n) throws Exception {

if (n < 0) throw new Exception(“Indice positivo”);

if (raiz == null)

throw new Exception(“Cadena vacia, no se puede eliminar nada”);

if (n == 0) {

raiz = raiz.getSiguiente();

return;

}

Nodo<T> actual = raiz;

int i = 1;

while(i < n && actual.getSiguiente() != null) {

actual = actual.getSiguiente();

i++;

}

if (i == n) {

actual.setSiguiente(actual.getSiguiente().getSiguiente());

return;

}

throw new Exception(“El indice ” + n + “ esta fuera de la cadena”);

}

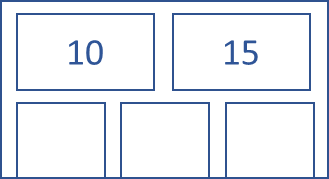
1. Escriba el código del método insertar(E obj) en una Cadena que sea ordenada. Es decir, al insertar el objeto, la Cadena lo coloca en el lugar que corresponde y entonces la Cadena siempre esta ordenada.
2. Utilice una estructura de datos árbol para representar una operación aritmética. EL árbol será binario. En cada nodo que no sea hoja se coloca un operador aritmético (+, -, \*, /) y en las hojas se colocara el número. En especial describa el código que debe realizar para evaluar el valor de la expresión siguiendo el diagrama de clases que proponga

Clases Arbol, Nodo, el nodo es binario. Los objetos son de tipo Aritmetico: Numero u Operación

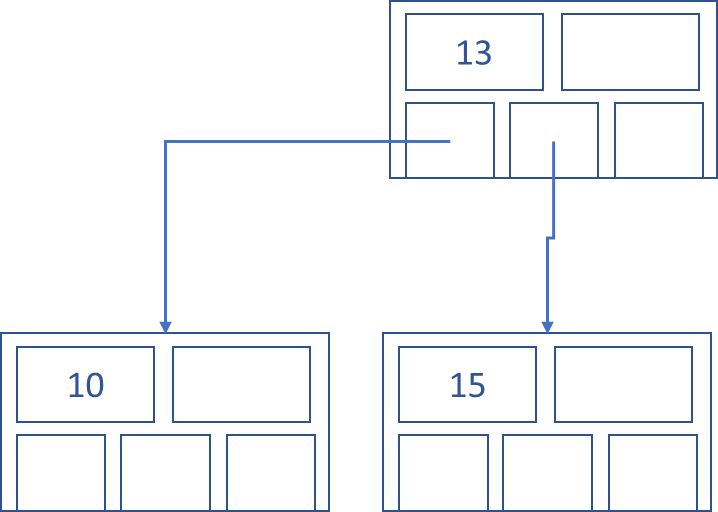
1. De la definición de Generics. Indique cuál es su función principal. Donde se ha utilizado en la materia.
2. Proponga una implementación de Anillo. Un Anillo es una cadena donde el último elemento no tiene un puntero a nulo sino al primer elemento.
3. Se ha visto un programa para evaluar el desarrollo de un árbol B. Siguiendo las reglas de árbol B dibuje el desarrollo de insertar los siguientes números en un árbol B de grado 1: 10, 15, 13, 23, 19, 8, 12, 16, 29.

Aquí el resultado de ir colocando los valores indicados en un árbol B de orden 1.

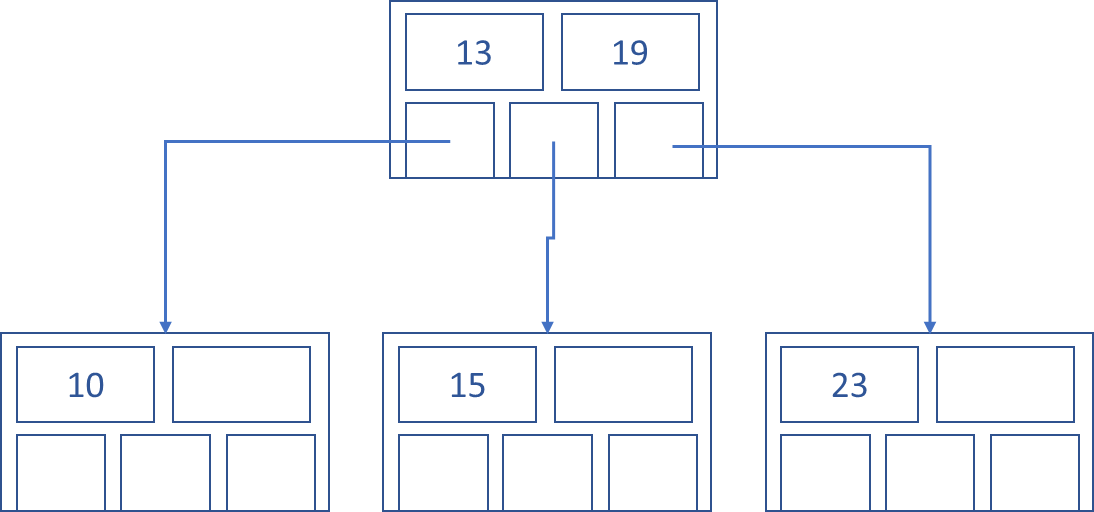
La primera figura toma los dos primeros valores cuyo desenlace es evidente:



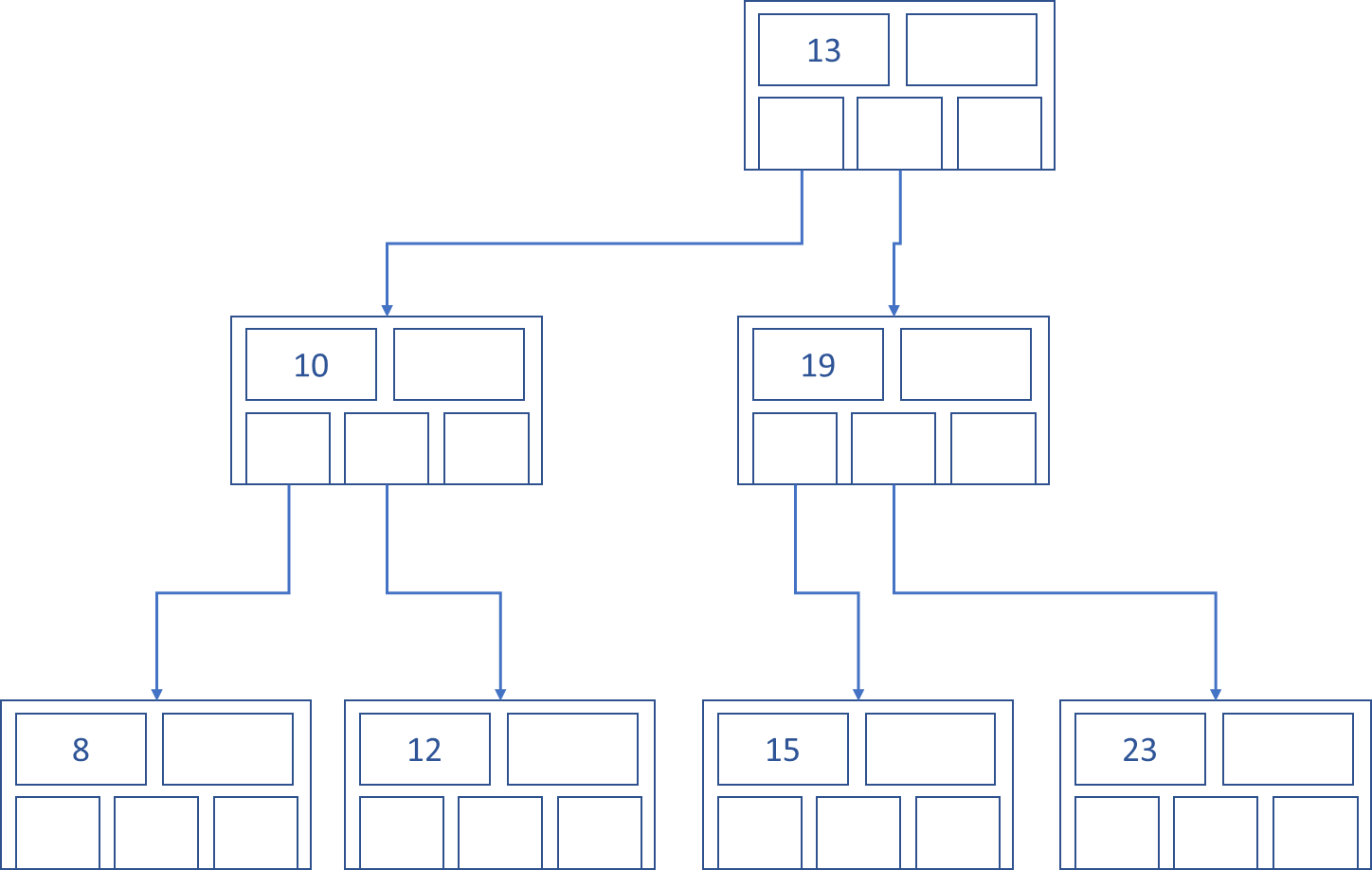
Al entrar el 13 se toman los 3 valores: 10, 13, 15 (en orden) y se toma el valor medio para ‘subirlo’ y tener:



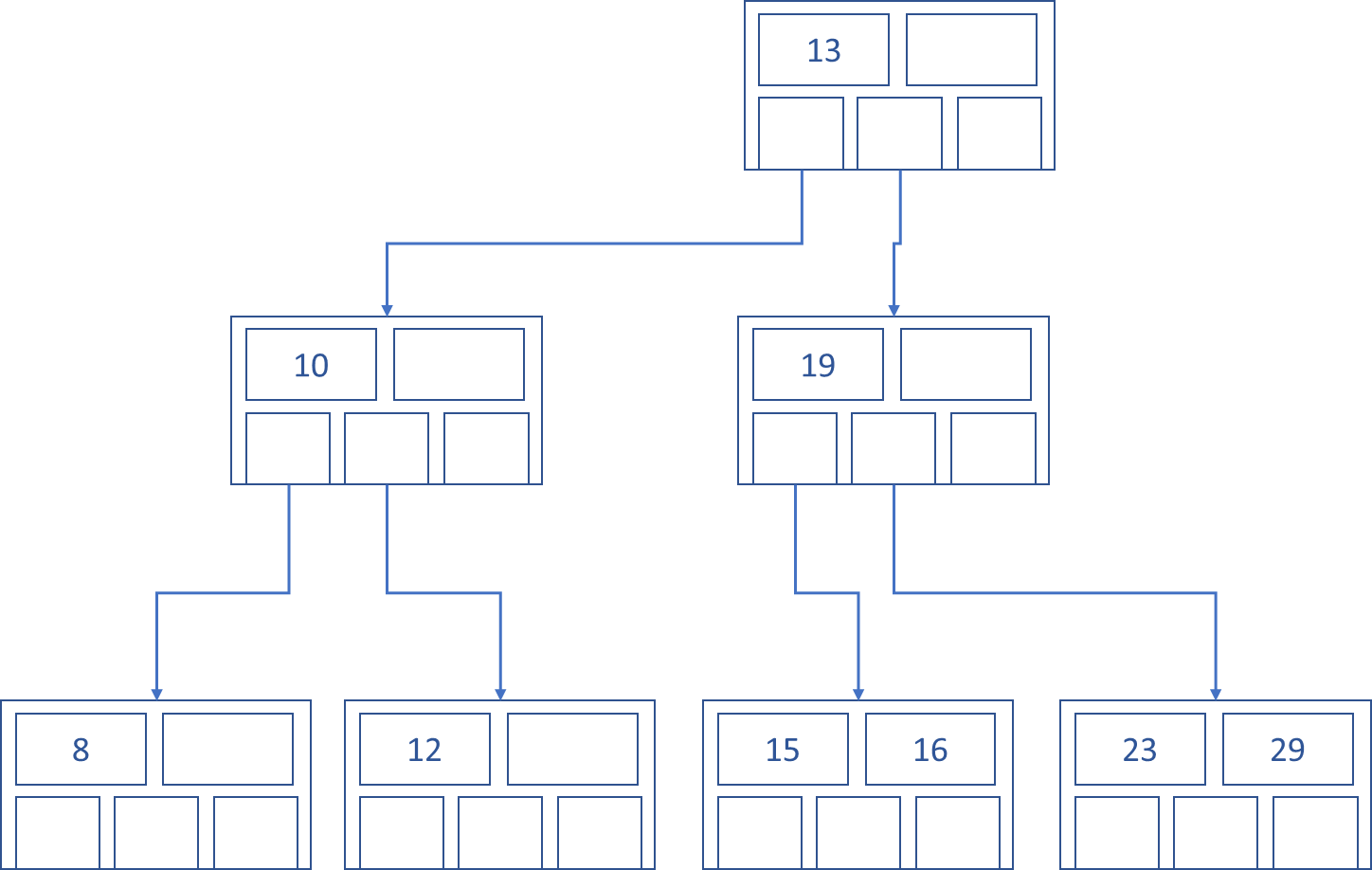
El 23 es trivial, aquí lo que ocurre cuando se coloca el 19, observe que 15, 19 y 23 se deben ordenar y ‘sube’ el del medio que es el 19.



EL 8 es igualmente trivial. El 12 es bastante complejo ya que 8, 10 y 12 se alinean y ‘sube’ 10. Pero al subir 10 se encuentra que está lleno y hay que alinear 10, 13, 19 y hay que ‘subir’ el 13



Y finalmente nos quedan los últimos dos que son fáciles el 16 y el 29:



1. Ver http://es.wikipedia.org/wiki/Edad\_del\_Universo [↑](#footnote-ref-1)