
CC10A

Sección 05 - 2004

Profesor: Andrés Muñoz O.

Auxiliares: Agustín Almonte

Marcelo Muñoz

Patricio Salles



**GUÍA DE EJERCICIOS
EXAMEN**

Tabla de Contenidos

TABLA DE CONTENIDOS	2
1. ARREGLOS IMPLEMENTADOS CON LISTAS	4
2. MAZO DE CARTAS	9
3. EL SORTEO	14
4. BALANCEO DE ÁRBOLES	16
5. TESEO Y EL LABERINTO	23
6. LA TRANSFORMADA DE BURROWS-WHEELER	27
7. BASE DE DATOS DE MP3 (I PARTE)	29
8. BASE DE DATOS DE MP3 (II PARTE)	32
9. PROCESADOR DE TEXTOS	34
10. ARBOLES	39
11. ABB (ÁRBOLES DE BÚSQUEDA BINARIA)	41
12. ANCESTROS Y PARIENTES	43
13. FUNCIONAMIENTO DE UN RESTAURANT	45
14. LA CLASE TABLA (PREGUNTA 1, EXAMEN 2001)	47
15. SELECCIÓN UNIVERSITARIA (PREGUNTA 2, EXAMEN 2001)	49
16. JUEGO (PREGUNTA 3, EXAMEN 2001)	50
17. OPERACIÓN OLITAS.	51
18. AGENDA ENLAZADA.	52
19. TABLA DE ALUMNOS.	54
20. SELECCIÓN UNIVERSITARIA.	55
21. PLANILLA ELECTRÓNICA	56

<u>22. CINEMA.</u>	<u>58</u>
<u>23. PUZZLE.</u>	<u>59</u>
<u>24. CLASE DE NÚMEROS.</u>	<u>62</u>
<u>25. SERIES GEOMÉTRICAS</u>	<u>64</u>
<u>26. ELECCIONES.</u>	<u>66</u>



1. Arreglos implementados con Listas

La restricción de los arreglos es muy clara en Java. El problema que poseen es que son de espacio limitado:

```
int[] a = new int[4];      // Solo tendremos 4 posiciones
```

Se desea implementar una Estructura de Dato más dinámica que los arreglos a partir de las Listas Enlazadas. Le llamaremos DArray (Dinamic Array).

Considere la siguiente representación de una lista enlazada:

```
public class Nodo {
    public String info;
    private Nodo sgte;

    // Crea un nodo con siguiente NULO
    public Nodo (String x) {
        // Llama al constructor de 2 parámetros
        this(x, null);
    }

    // Crea un nodo con siguiente s
    public Nodo (String x, Nodo s) {
        // Asgna los valores a las variables de instancia
        this.info = x;
        this.sgte = s;
    }

    // Asigna al nodo el siguiente s
    public void asignar (Nodo s) {
        this.asignar(this.info, s);
    }

    // Asigna al nodo el valor x
    public void asignar (String x) {
        this.asignar(x, this.sgte);
    }

    // Asigna al nodo el valor x con siguiente s
    public void asignar (String x, Nodo s) {
        this.info = x;
        this.sgte = s;
    }

    // Entrega el siguiente nodo
    public Nodo siguiente () {
        // Retorna el siguiente elemento del Nodo
        return this.sgte;
    }
}
```

Como podemos ver esta representación de lista no requiere una clase lista y casi todo el procesamiento lo realiza la misma clase.

(a) Programe la clase DArray que permita implementar los siguientes métodos:

Método	Significado
DArray()	Constructor que crea un Dinamic Array vacío.
void asignar(String x, int i)	Da el valor x a la posición i-ésima del arreglo (no inserta).
String obtener(int i)	Busca el valor del i-ésimo elemento y lo devuelve.
void borrar(int i)	Elimina el i-ésimo nodo del DArray.
int largo()	Retorna el largo del DArray.
void nuevo(String x)	Crea un nuevo elemento al final del DArray para agrandarlo.

Nota:

- Recuerde que las variables de instancia que debe utilizar son la cabeza de la lista y el largo de ella misma. Además, solo es posible asignar hasta uno más allá del largo del DArray.
- Si ocurre un error (como por ejemplo el índice fuera del rango entre 0 y el largo del arreglo) utilice **throw new Exception("[Mensaje de error]")** para interrumpir la ejecución del programa. Tendrá que eso si agregar **throws Exception** en cada uno de los métodos que use lo anterior.

```
public class DArray {
    private Nodo lista;
    private int cant;

    public DArray() {
        this.lista = null;
        this.cant = 0;
    }

    public void asignar(String x, int i) throws Exception {
        // Verificamos que esté dentro del rango
        if (i<0 || i >= this.cant)
            throw new Exception("Valor " + i +
                                " fuera del rango.");

        // Buscamos la posición
        Nodo p = this.lista;
        for (int n=0; n<i; n++)
            p = p.siguiete();

        // Asignamos el nuevo valor
        p.info = x;
    }

    public String obtener(int i) throws Exception {
        // Verificamos que esté dentro del rango
        if (i<0 || i >= this.cant)
            throw new Exception("Valor " + i +
                                " fuera del rango.");

        // Buscamos la posición
        Nodo p = this.lista;
        for (int n=0; n<i; n++)
```

```
        p = p.siguiente();

        // Retornamos el valor del elemento
        return p.info;
    }

    public void borrar(int i) throws Exception {
        // Verificamos que esté dentro del rango
        if (i<0 || i >= this.cant)
            throw new Exception("Valor " + i +
                                " fuera del rango.");

        // Verificamos que no se esté borrando el primero
        if (i == 0) {
            this.lista = this.lista.siguiente();
            this.cant--;
            return;
        }

        // Buscamos la posición
        Nodo p = this.lista;
        for (int n=0; n<i-1; n++)
            p = p.siguiente();

        // Se elimina el valor
        p.asignar(p.siguiente());

        // Decrementamos la cantidad de elementos
        this.cant--;
    }

    public int largo() {
        // Retornamos la cantidad de elementos
        return this.cant;
    }

    public void nuevo(String x) {
        // Verificamos de que la lista no esté vacía
        if (this.lista == null) {
            this.lista = new Nodo(x);
            this.cant++;
            return;
        }

        // Buscamos la ultima posición
        Nodo p = this.lista;
        while (p.siguiente() != null)
            p = p.siguiente();

        // Agregamos el nuevo valor
        Nodo q = new Nodo(x);
        p.asignar(q);

        // Incrementamos la cantidad de elementos
        this.cant++;
    }
}
```

Testing: Para probar este programa utilice el siguiente código:

```
public class Test {
```

```
static void dibujar(DArray a) throws Exception {
    System.out.print("Lista");
    for (int j=0; j<a.largo(); j++) {
        System.out.print(" -> " + a.obtener(j));
    }
    System.out.println(" -||");
}

static void main(String[] args) throws Exception {
    DArray a = new DArray();
    long old;

    dibujar(a);

    for (int i=0; i<10; i++) {
        int n = (int) Math.round(Math.random() * 1000);
        System.out.print("Valor(" + i + ") = " + n);
        a.nuevo(" " + n);

        dibujar(a);

        old = System.currentTimeMillis() / 500;
        while (old ==
            System.currentTimeMillis() / 1000);
    }

    System.out.println();

    while (true) {
        System.out.print("Eliminando...");
        a.borrar(a.largo()-1);
        System.out.println(" OK!");

        dibujar(a);

        old = System.currentTimeMillis() / 500;
        while (old ==
            System.currentTimeMillis() / 1000);
    }
}
```

- (b) **Propuesto.** Se desea manejar las notas del curso a través de este sistema de arreglos dinámicos (DArray). El problema radica en que el profesor no sabe a priori cuántos ejercicios va a realizar.

Escribe un programa que simule el siguiente diálogo:

```
Ingresa el número de alumnos? 100

Opciones:
1 = Agregar Nota
2 = Modificar Nota
3 = Promediar
Su opción? 1
    Ingresa el código del alumno (1-100)? 53
    Ingresa la nota número 1? 3.3

Opciones:
1 = Agregar Nota
```

```
2 = Modificar Nota
3 = Promediar
Su opción? 2
  Ingrese el código del alumno (1-100)? 23
  Tiene 3 notas. Cuál quiere modificar? 2
  La nota es un 4.5. Nueva nota? 6.5

Opciones:
1 = Agregar Nota
2 = Modificar Nota
3 = Promediar
Su opción? 3
  Ingrese el código del alumno (1-100)? 23
  El alumno tiene: 7.0, 2.3, 6.5, 3.3 y su promedio es: 4.775
```

Nota: Recuerde que puede crear un arreglo de clases para solucionar este problema y que el código del alumno es equivalente a la posición del alumno dentro de un arreglo.

2. Mazo de Cartas

Para un juego de cartas es necesario un mazo de cartas. Sin embargo el procedimiento para sacar cartas simula lo que es una pila de elementos.

Suponga la siguiente representación de una **Carta**:

```
public class Carta {
    public int valor;
    public String pinta;

    public Carta (int n, char p) {
        this.valor = n;
        this.pinta = "" + p;
    }
}
```

(a) Escriba una clase **Mazo** que implemente un mazo de cartas a través de una Pila. Los métodos que esta clase debe tener son:

Método	Significado
Mazo()	Constructor que crea un Mazo con las 52 cartas del mazo inglés.
void ponerCarta (Carta c)	Pone una carta sobre el mazo.
Carta sacarCarta ()	Saca una carta del mazo.
Mazo cortar (int n)	Corta el mazo en la carta n-ésima y devuelve la mitad superior (la inferior se queda en el mismo mazo).
Mazo copiar ()	Entrega una copia del mazo actual.

Nota: Utilice la clase **Pila** como si esta contenera cartas con los métodos **Pila(int n)** (constructor con n elementos la pila), **void poner(Carta c)** (pone c en la cúspide de la pila), **Carta sacar()** (saca una carta de la cúspide) y **boolean estaVacia()** (retorna true si está vacía y false si no).

```
public class Mazo {
    private Pila cartas;

    /* Las letras que representan las pintas:
       C = Corazón
       P = Pick
       D = Diamante
       T = Trébol */
    private final String pintas = "CPDT";

    public Mazo() {
        // Pone las 52 cartas en el mazo.
        cartas = new Pila(52);
        for (int p=0; p<4; p++)
            for (int i=1; i<=13; i++)
```

```
        cartas.poner(  
            new Carta(i, pintas.charAt(p)));  
    }  
  
    public void ponerCarta(Carta c) {  
        // Pone la carta en la pila  
        cartas.poner(c);  
    }  
  
    public Carta sacarCarta() {  
        // Saca la carta de la pila  
        if (this.cartas.estaVacia())  
            return null;  
        return cartas.sacar();  
    }  
  
    public Mazo cortar(int n) {  
        // Creamos un nuevo mazo  
        Mazo m = new Mazo();  
  
        // Eliminamos toda la información posible  
        m.cartas = new Pila(52);  
  
        // Saca el número de cartas indicados desde la pila  
        // y las pone en el nuevo mazo  
        for (int i=0; i<n; i++) {  
            m.ponerCarta(this.sacarCarta());  
        }  
  
        // Retornamos el mazo nuevo  
        return m;  
    }  
  
    public Mazo copiar() {  
        // Creamos un nuevo mazo  
        Mazo m = new Mazo();  
        Mazo t = new Mazo();  
  
        // Eliminamos toda la información posible  
        m.cartas = new Pila(52);  
        t.cartas = new Pila(52);  
  
        // Saca el resto de cartas desde la pila  
        // y las pone en el nuevo mazo  
        while (!this.cartas.estaVacia()) {  
            Carta c = this.sacarCarta();  
            m.ponerCarta(c);  
            t.ponerCarta(c);  
        }  
  
        // Deja el primer mazo tal cual  
        while (!t.cartas.estaVacia())  
            this.ponerCarta(t.sacarCarta());  
  
        // Retornamos el mazo nuevo  
        return m;  
    }  
}
```

- (b) Para revolver un mazo, una forma es cortar consecutivamente por la mitad el mazo original y mezclar sus cartas. Programe un método

public void mezclar()

perteneciente a la clase **Mazo** que realice la función de:

- Cortar en un número aleatorio entre 1 y 52.
- Mezclar las mitades del mazo intercalando cada uno de ellos en el otro.
- Asigne el nuevo mazo como el mazo que llamó el método.

```
static public int random(int min, int max) {
    // Calcula un número al azar entre min y max
    long azar = Math.round(Math.random() * (max - min));
    return min + (int) azar;
}

public void mezclar() {
    // Calculamos cuántas veces cortamos
    int veces = Mazo.random(1, 10);

    for (int i=0; i<veces; i++) {
        // Por cada vez, cortamos el mazo aleatoriamente
        Mazo m1 = this.cortar(Mazo.random(1, 52));
        Mazo m2 = this.copiar();
        this.cartas = new Pila(52);

        // Se mezclan ambos mazos
        while (!m1.cartas.estaVacia() ||
            !m2.cartas.estaVacia()) {
            if (!m1.cartas.estaVacia())
                this.ponerCarta(m1.sacarCarta());
            if (!m2.cartas.estaVacia())
                this.ponerCarta(m2.sacarCarta());
        }
    }
}
```

(c) Implemente la clase **Pila** (con arreglo) que soporte las características indicadas.

```
public class Pila {
    private Carta[] elems;
    private int tope;

    public Pila(int n) {
        // Se crea el arreglo (no las cartas)
        this.elems = new Carta[n];
        this.tope = -1;
    }

    public void poner(Carta c) {
        // Se desplaza el tope en uno y luego se pone c
        this.elems[++this.tope] = c;
    }

    public Carta sacar() {
        // Se saca el tope y luego se decrementa
        return this.elems[this.tope--];
    }
}
```

```
public boolean estaVacia() {  
    // Se verifica solo si el tope es menor que 0  
    return (this.tope < 0);  
}  
}
```

Testing: Para probar este programa, utiliza el siguiente juego de Black Jack.

```
class BlackJack {  
    static int sumar(Pila mano) {  
        Pila temp = new Pila(52);  
        int total = 0;  
        boolean as = false;  
  
        // Traspasamos la información mientras vamos contando  
        while (!mano.estaVacia()) {  
            Carta c = mano.sacar();  
            total += c.valor;  
            if (c.valor == 1 && !as) {  
                total += 10;  
                as = true;  
            }  
            if (c.valor > 10)  
                total -= c.valor - 10;  
            temp.poner(c);  
        }  
  
        // Volvemos las cartas a la mano  
        while (!temp.estaVacia()) {  
            mano.poner(temp.sacar());  
        }  
  
        // Retornamos la suma  
        return total;  
    }  
  
    static void main(String[] args) throws Exception {  
        BufferedReader in = new BufferedReader(new  
            InputStreamReader(System.in));  
  
        // Se crea el mazo de cartas  
        Mazo mazo = new Mazo();  
  
        // Ciclo del juego (2 jugadores)  
        // cada mano es representada por una pila de cartas  
        while (true) {  
            // Creamos las manos de ambos jugadores  
            Pila mano = new Pila(52);  
            Pila casa = new Pila(52);  
  
            // Revolvemos el mazo  
            mazo.mezclar();  
  
            // Ciclo del jugador  
            boolean gana = true;  
            while (true) {  
                // Saca la carta y la muestra  
                Carta c = mazo.sacarCarta();  
                mano.poner(c);
```

```
        System.out.println("Con " + c.valor +
            c.pinta + " sumas " + sumar(mano));

        // Si se pasa, perdió
        if (sumar(mano) > 21) {
            gana = false;
            break;
        }

        // Preguntamos si quiere más cartas
        System.out.print("Otra carta (s/n)?");
        String resp = in.readLine();
        if (resp.toUpperCase().equals("N"))
            break;
    }

    // Ciclo de la casa
    while (gana) {
        // Sacar carta y la muestra
        Carta c = mazo.sacarCarta();
        casa.poner(c);
        System.out.println("Con " + c.valor +
            c.pinta + " sumo " + sumar(casa));

        // Si se pasa, perdió
        if (sumar(casa) > 21)
            break;

        // Si saca igual o más gana
        if (sumar(casa) >= sumar(mano)) {
            gana = false;
            break;
        }
    }

    // Definicion
    if (gana)
        System.out.println("Está bien, ganaste!");
    else
        System.out.println("La casa gana!");

    // Vuelve cartas al mazo
    while (!mano.estaVacia() || !casa.estaVacia()) {
        if (!mano.estaVacia())
            mazo.ponerCarta(mano.sacar());
        if (!casa.estaVacia())
            mazo.ponerCarta(casa.sacar());
    }

    // Fin del juego
    System.out.print("Juagmos otra (s/n)?");
    String resp = in.readLine();
    if (resp.toUpperCase().equals("N"))
        break;
    }
}
```

3. El Sorteo

Te contratan desde la Lotería de Cachiyuyo para desarrollar un programa que les facilite la vida al momento de sacar los números premiados de su juego archiconocido NIKO (si... se parece a otro, pero es solo un alcance de nombre).

Las bases del juego son las siguientes:

- Cada jugador posee una cartilla con 15 números aleatorios elegidos de un conjunto de 24.
- Los números sorteados son sacados de una tómbola que está siempre girando con las 24 bolitas numeradas.
- Las 15 bolitas premiadas se van sacando de a una cada vez.
- La tómbola después de girar, pone las bolitas ordenadas en un tubo (por donde salen) de manera aleatoria.

Considere la siguiente clase:

```
class Tombola {
    Cola tubo;
    int[] bolitas;

    public Tombola(int n) {
        // Crea las n bolitas y las pone en la tómbola
        this.bolitas = new int[n];
        for (int i = 0; i < n; i++)
            this.bolitas[i] = i+1;
    }
}
```

- (a) Implemente un método **void girar(int veces)** que gire la tómbola con las bolitas una cantidad de **veces** indicada. Considere que un giro significa ordenar aleatoriamente el arreglo de bolitas.

```
public void girar(int veces) {
    for (int n = 0; n < veces; n++) {
        // Elegiremos un número al azar para
        // intercambiar esa posición con la actual.
        for (int i = 0; i < this.bolitas.length; i++) {
            int bola = this.bolitas[i];
            int cambio = 1 + (int) Math.round(
                Math.random() *
                this.bolitas.length);
            this.bolitas[i] = this.bolitas[cambio];
            this.bolitas[cambio] = bola;
        }
    }
}
```

- (b) Implemente el método **void preparar(int n)** que permite al animador hacer girar la tómbola un número aleatorio de veces entre 1 y 100, para luego dejar las n primeras bolitas en el tubo listas para ser extraídas.

```
public void preparar(int n) {  
    // Primero se revuelve la tómbola  
    int veces = 1 + (int) Math.round(Math.random() * 99);  
    this.girar(veces);  
  
    // Ahora se van poniendo en la cola las bolitas  
    for (int i=0; i<n; i++)  
        this.tubo.poner(this.bolitas[i]);  
}
```

- (c) Dado el siguiente método:

```
public int extraer() {  
    return this.tubo.sacar();  
}  
} // Fin de la Clase
```

El cual va sacando las bolitas del tubo una a una en orden FIFO. Implemente un programa que permita sacar de a una cada bolita y las vaya mostrando en la salida estándar utilizando la clase **Tombola**.

```
public class NIKO {  
    static public void main(String args[]) {  
        System.out.println("NIKO - Sorteo de Cachiyuyo");  
        System.out.println("Las bolitas a la tómbola...");  
        Tombola t = new Tombola(24);  
        System.out.println("La tómbola gira y gira...");  
        t.preparar(15);  
        System.out.println("Ahora, los premiados...");  
        for (int i=0; i<15; i++)  
            System.out.println("Bolita: " + t.extraer());  
        System.out.println("Felicidades a los afortunados!");  
    }  
}
```

4. Balanceo de Árboles

Los árboles de búsqueda binaria poseen una característica muy particular lo que los hace un tanto atractivos: Los elementos menores que X están a la izquierda de él y los mayores que X a la derecha de él.

El problema radica que, cuando se ingresan valores ordenados al árbol, éste se convierte en casi una lista enlazada que prácticamente no utiliza la característica de ABB cuando realiza una búsqueda.

Es por eso que se quiere implementar una clase árbol con la siguiente representación:

```
public class Nodo {
    public int valor;
    public Nodo izq, der;

    public Nodo (int valor) {
        this.valor = valor;
        this.izq = null;
        this.der = null;
    }
}

public class Arbol {
    public Nodo raiz;

    public Arbol() {
        this.raiz = null;
    }
}
```

(a) Escriba un método estático que calcula la altura de un árbol entregado en la raíz:

static int altura (Nodo raiz)

```
// Método de altura en versión recursiva
static int altura (Nodo raiz) {
    // Vemos si el nodo no es nulo y retornamos 0
    if (raiz == null)
        return 0;

    // La altura es 1 + la máxima altura entre sus hijos
    return ( 1 +
        Math.max(altura(raiz.izq), altura(raiz.der)) );
}
```

(b) Un árbol balanceado es aquél que posee una altura igual en ambos sub-arboles hijos. Programe un método estático que indica si el árbol entregado está o no balanceado:

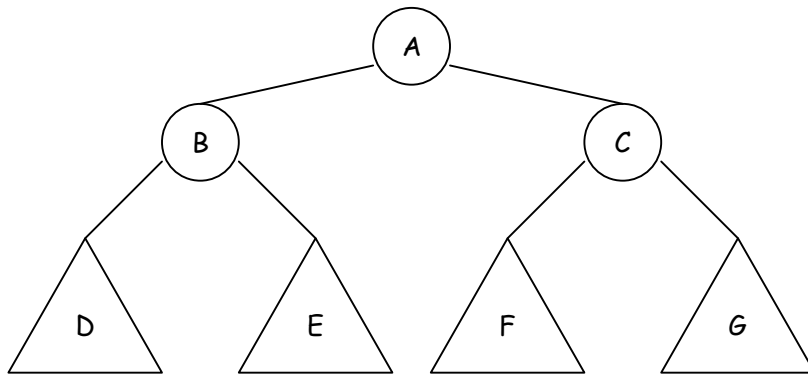
static boolean balanceado (Nodo raiz)


```
static boolean balanceado (Nodo raiz) {  
    if (raiz == null)  
        return true; // Un arbol por defecto balanceado  
    return ( altura(raiz.izq) == altura(raiz.der) );  
}
```

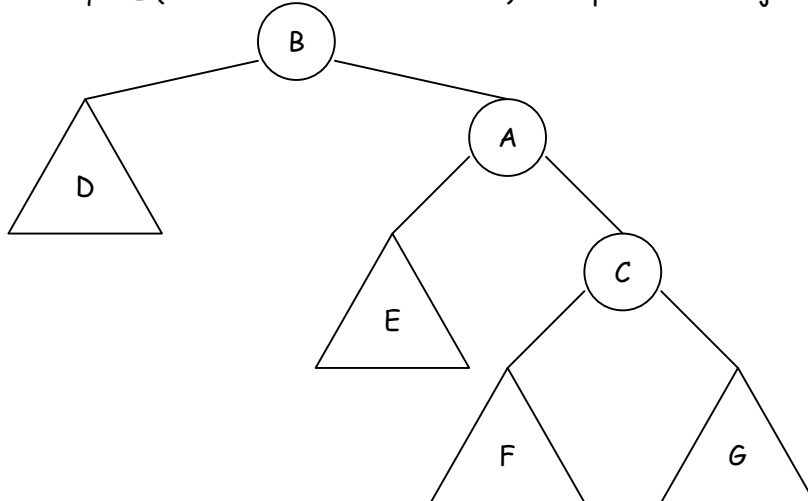
(c) Se requiere un método que rote un árbol.

static Nodo rotar (Nodo raiz, boolean derecha)

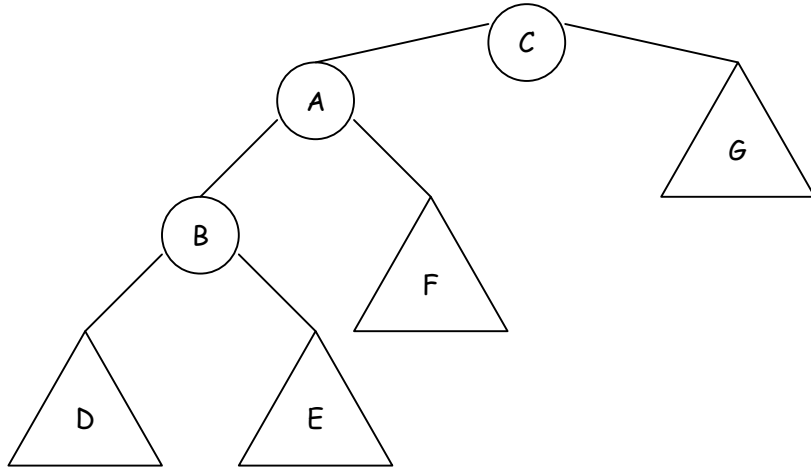
Considere que el proceso de rotación en el siguiente caso de árbol:



- Rotar A a la derecha significa poner B como nueva raíz y A como su hijo derecho. Sin embargo es claro que E (o el sub-árbol derecho de B) debe pasar a ser hijo izquierdo de A.



- Rotar A a la izquierda significa poner C como nueva raíz y A como su hijo izquierdo. Sin embargo es claro que F (o el sub-árbol izquierdo de C) debe pasar a ser hijo derecho de A.



```
static Nodo rotacion (Nodo raiz, boolean derecha) {
    Nodo p = raiz;
    if ( p == null ) return p;

    // Rotar a la derecha
    if ( derecha ) {
        // No se puede rotar si IZQ es nulo
        if ( p.izq == null )
            return null;

        // Se construyen los nuevos hijos
        Nodo izq = p.izq.izq;
        Nodo der = new Nodo(p.valor);
        der.izq = p.izq.der;
        der.der = p.der;
        // Se arma el árbol completo
        p = new Nodo(p.izq.valor);
        p.izq = izq;
        p.der = der;
    }
    else { // Rotar a la izquierda
        // No se puede rotar si DER es nulo
        if ( p.der == null )
            return null;

        // Se construyen los nuevos hijos
        Nodo izq = new Nodo(p.valor);
        izq.izq = p.izq;
        izq.der = p.der.izq;
        Nodo der = p.der.der;
        // Se arma el árbol completo
        p = new Nodo(p.der.valor);
        p.izq = izq;
        p.der = der;
    }
    return p;
}
```

- (d) Ahora que ya sabemos como verificar si un árbol está balanceado o no, se pide implementar un método de la clase árbol que permita balancear el árbol en forma recursiva.

Para esto se define el siguiente método:

```
public void balancear () {  
    // Balancea el árbol activo  
    this.raiz = Arbol.balanceo (this.raiz);  
}
```

Usted debe implementar:

static Nodo balanceo (Nodo raiz)

Que balancea el árbol de nodo raiz **raiz** y que retorna su resultado balanceado. La estrategia a seguir es:

- Verificar si está balanceado el árbol y retornar la misma raíz.
- Si no está balanceado, balancear los hijos (llamada recursiva).
- Una vez balanceados, rotar al lado correspondiente para balancear si aún está desbalanceado.
- Retornar el nuevo árbol.

```
static Nodo balanceo (Nodo raiz) {  
    // Se verifica si está balanceado.  
    if ( balanceado(raiz) )  
        return raiz;  
  
    Nodo p = new Nodo(raiz.valor);  
  
    // Se balancean los hijos  
    p.izq = balanceo(raiz.izq);  
    p.der = balanceo(raiz.der);  
  
    // Se rota para que quede balanceado  
    if ( !balanceado(p) )  
        p = rotacion(p,  
                      altura(p.izq)>altura(p.der));  
  
    // Se retorna el árbol balanceado  
    return p;  
}
```

Existen otras técnicas más eficientes de balanceo, pero la idea del problema era entender árboles y no aprender técnicas en árboles balanceados.

Ahora debes agregar los métodos insertar y eliminar a la clase árbol para que quede completamente listo:

```
// Inserta un elemento
public void insertar (int valor) {
    raiz = insertar(this.raiz, valor);
}

static Nodo insertar (Nodo raiz, int valor) {
    // Se verifica que no hayamos llegado a una hoja
    if ( raiz == null ) return new Nodo(valor);

    // Se inserta en el lado correspondiente
    if ( valor < raiz.valor )
        raiz.izq = insertar (raiz.izq, valor);
    else if ( valor > raiz.valor )
        raiz.der = insertar (raiz.der, valor);

    return raiz;
}

// Elimina un elemento
public Nodo eliminar (int valor) {
    return eliminar(this.raiz, valor);
}

static Nodo eliminar (Nodo raiz, int valor) {
    // Se verifica que no hayamos llegado a una hoja
    if ( raiz == null ) return null;

    // Se verifica si lo encontramos
    if ( raiz.valor == valor ) {
        Nodo p = raiz;
        raiz = mezclar(p.izq, p.der);
        return p;
    }

    // Elegimos el camino a seguir para encontrarlo
    if ( raiz.valor > valor )
        return eliminar(raiz.izq, valor);
    else
        return eliminar(raiz.der, valor);
}

// Mezcla dos árboles ordenándolos según ABB
static Nodo mezclar (Nodo a, Nodo b) {
    Nodo p;

    // Elegimos cuál promover (rama más larga para
    // mantener la idea de balanceo)
    if ( altura(a) > altura(b) ) {
        p = a;
        if (p != null) {
            Nodo q = p;
            while ( q.der != null ) q = q.der;
            q.der = b;
        }
        else p = b;
    }
    else {
        p = b;
        if (p != null) {
            Nodo q = p;
            while ( q.izq != null ) q = q.izq;
            q.izq = a;
        }
    }
}
```

```
        }  
        else p = a;  
    }  
  
    // Se retorna el nuevo árbol.  
    return p;  
}  
  
// Fin de la clase Arbol  
}
```

Testing: El siguiente programa realiza una prueba de este tipo de árboles (para ejecutar).

```
public class Test {  
    static BufferedReader in = new BufferedReader(  
        new InputStreamReader(System.in));  
  
    // Dibuja en pantalla el árbol  
    static void dibujar(Nodo raiz, String espacio) {  
        if (raiz != null) {  
            System.out.println(espacio + raiz.valor);  
            dibujar(raiz.izq, espacio + " (i) ");  
            dibujar(raiz.der, espacio + " (d) ");  
        }  
    }  
  
    static void main(String[] args) {  
        Arbol a = new Arbol();  
        Arbol b = new Arbol();  
        for (int i=1; i<10; i++) {  
            System.out.print("Insertando " + i + "...");  
            a.insertar(i);  
            b.insertar(i);  
            System.out.println(" OK!");  
            if ( !Arbol.balanceado(a.raiz) ) {  
                System.out.print("Balanceando...");  
                a.balancear();  
                System.out.println(" OK!");  
            }  
            long old = System.currentTimeMillis();  
            while (old / 1000 ==  
                System.currentTimeMillis() / 1000);  
        }  
        System.out.println("\nARBOL BALANCEADO\n");  
        dibujar(a.raiz, "");  
        System.out.println();  
        System.out.println("  Altura Balanceada = " +  
            Arbol.altura(a.raiz));  
        System.out.println("\nARBOL NO BALANCEADO\n");  
        dibujar(b.raiz, "");  
        System.out.println();  
        System.out.println("Altura No Balanceada = " +  
            Arbol.altura(b.raiz));  
    }  
}
```

Puedes realizar la prueba tu mismo cambiando la cantidad de elementos en el **for**. También cambia la forma de obtener valores por una forma aleatoria cambiando:

```
System.out.print("Insertando " + i + "...");  
a.insertar(i);  
b.insertar(i);  
System.out.println(" OK!");
```

por

```
int n = (int) Math.round(Math.random() * 1000);  
System.out.print("Insertando " + n + "...");  
a.insertar(n);  
b.insertar(n);  
System.out.println(" OK!");
```

para que analices si es mejor o no utilizar árboles balanceados.

5. Teseo y el Laberinto

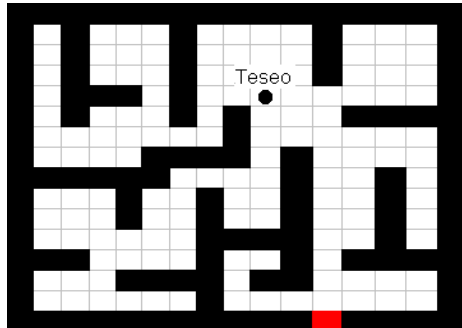


Figura1

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	1	0	0	0	1	0	0	0	0	1	0	0	0
1	0	1	0	0	0	1	0	0	0	0	1	0	0	0
1	0	1	0	0	0	1	0	0	0	0	1	0	0	0
1	0	1	1	1	0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	1	0	1	0	0	0	1	1	1
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1	0	1	0	0	0	0
1	1	1	1	1	1	0	0	0	0	1	0	0	1	0
1	0	0	0	1	0	0	1	0	0	1	0	0	1	0
1	0	0	0	1	0	0	1	0	0	1	0	0	1	0
1	0	0	0	0	0	0	1	1	1	1	0	0	1	0
1	1	1	0	0	0	0	1	0	0	1	0	1	1	1
1	0	0	0	1	1	1	1	0	1	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	2	1	1	1

Figura 2

En la Figura 1 puedes apreciar a nuestro distraído amigo Teseo sin poder encontrar la salida del Laberinto. En vista de ello, se te pide que lo ayudes generando un algoritmo que le permita escapar, ya que pronto tendrá que dar un Examen de Cálculo. Para ello, puedes modelar el Laberinto como un Arreglo de dos dimensiones, en el cual las paredes están representadas por 1, los espacios vacíos (por donde puede caminar Teseo) por 0 y la salida por un 2 (como se ve en la Figura2). Por suerte, un computín amigo tuyo te ayudó con parte del trabajo, pero no lo alcanzó a terminar.

```
class posicion{
    int x,y;
    public posicion (int a, int b) {
        x=a;
        y=b;
    }
}

class Laberinto{
    int A [][];           //guarda el Laberinto
    int n,m;              //dimensiones del Laberinto
    posicion pos;         //posicion actual de Teseo
    posicion salida;      //guarda la posicion de la salida

    public Laberinto (p,q){
        /* Este método crea un Laberinto de dimensiones pxq y
        lo guarda en el arreglo A. Se preocupa de poner las
        paredes (poner los 1), poner la salida (poner el 2 en
        un borde del laberinto y guardar la posición en salida)
        y llenar el resto con ceros. */
    }

    public posicion iniciar (){
        // Este método entrega la posición original de Teseo
        int a,b;
        while(true){
            a=(int)Math.random()*n;

```

```
        b=(int)Math.random()*m;
        if (A[a][b]==0)
            break;
    }
    posicion aux= new posicion (a,b);    return aux;
}

public String mover(posicion z ){
    /* Este método toma la posición actual de Teseo (z)
    y entrega un String de la forma x1y1 x2y2 x3y3
    x4y4... xjyj que representa la trayectoria de
    Teseo desde la posición z (x1y1) hasta la salida
    (xjyj).*/
}

public boolean ganaste(posicion w){
    return (w.x==salida.x &&w.y==salida.y);
}
```

Completa el método **public String mover(posicion z)**

Obs: En el laberinto no se forman ciclos, es decir, todas las paredes llegan al borde

Propuesto: Resolverlo con ciclos

Obs:

El algoritmo es:

- Me guío siempre por la pared que está a mi izquierda
- Lo primero que hago es subir hasta la pared más cercana y girar a la derecha (así la pared queda a mi izquierda)
- Si la celda de adelante esta desocupada avanzo
- Si la pared de mi izquierda no continúa, giro con ella, sino sigo derecho
- Si la celda de adelante está ocupada, la ocupo como pared de apoyo (giro hacia mi derecha)

```
public String mover(posicion z ){
    // reviso si no quedé justo con una pared "sobre" mi.
    boolean encuentre_pared = false;
    if (A[z.x-1][z.y]==1) encuentre_pared==true;
    int direccion=0 //1 arriba, 2 abajo, 3 derecha, 4 izquierda
    posicion po=new posicion(z.x,z.y) //posición de Teseo
    String s = "" //Guarda el trayecto

    while(!ganaste(po)){

        // parto hacia arriba hasta encontrar una pared, y
        // caminar hacia la derecha
        while(!encontre_pared){
            s+=po.x ;
            s+=po.y;
            po.x--;
            if(A[po.x-1][po.y]==1) { //si arriba hay una pared
                direccion = 3;
            }
        }
    }
}
```



```
        encuentre_pared==true;
        break;
    }

    //Ahora viene el algoritmo general
    // si voy hacia la derecha
    if(direccion==3){
        //si no hay pared al frente
        if(A[po.x][po.y+1]==0){
            s+=po.x;
            s+=po.y;
            po.y++;
            //si se "abrió" la pared de apoyo
            if(A[po.x-1][po.y]==0)
                //debo girar con ella
                direccion = 1;
        }
        //si al frente estaba cerrado solo puedo
        //girar en 90°
        direccion = 2;
    }
    //si voy hacia abajo
    if(direccion==2){
        //si no hay pared al frente
        if(A[po.x+1][po.y]==0){
            //primero entrego mi posicion actual
            s+=po.x;
            s+=po.y;
            po.x++; //doy un paso hacia adelante
            //si "abrió" la pared de apoyo
            if(A[po.x+1][po.y+1]==0)
                //debo girar con ella
                direccion = 3;
        }
        //si al frente estaba cerrado solo puedo
        //girar en 90°
        direccion = 4;
    }
    //si voy hacia arriba
    if(direccion==1){
        //si no hay pared al frente
        if(A[po.x-1][po.y]==0){
            //primero entrego mi posicion actual
            s+=po.x;
            s+=po.y;
            po.x--; //doy un paso hacia adelante
            //si "abrió" la pared de apoyo
            if(A[po.x][po.y-1]==0)
                //debo girar con ella
                direccion = 4;
        }
        //si al frente estaba cerrado solo puedo
        //girar en 90°
        direccion = 3;
    }
    //si voy hacia la izquierda
    if(direccion==4){
        //si no hay pared al frente
        if(A[po.x][po.y-1]==0){
            //primero entrego mi posicion actual
            s+=po.x;
            s+=po.y;
            po.y--; //doy un paso hacia adelante
```

```
        //si "abrió" la pared de apoyo
        if(A[po.x+1][po.y]==0)
            //debo girar con ella
            direccion =2;
        }
        //si al frente estaba cerrado solo puedo
        //girar en 90°
        direccion = 1;
    }

    }
    return s;//retorno el trayecto
} //fin método
} //fin clase
```

6. La Transformada de Burrows-Wheeler

Esta transformada es muy utilizada en el área de la compresión. Lo que hace es convertir un String x en un resultado compuesto por un String y un número:



En este caso, convirtió "BANANA" en el par ordenado ("NNBAAA", 4). Se calcula de la siguiente manera. Primero toma el String, y lo "rota circularmente" hacia la izquierda, escribiendo así todas las palabras resultantes dentro de una matriz cuadrada:

B	A	N	A	N	A
A	N	A	N	A	B
N	A	N	A	B	A
A	N	A	B	A	N
N	A	B	A	N	A
A	B	A	N	A	N

Luego ordenamos las palabras (las filas) en orden lexicográfico. La transformada se obtiene observando, dentro de la matriz resultante, viendo la última columna y la posición en que quedó la palabra original:

A	B	A	N	A	N
A	N	A	B	A	N
A	N	A	N	A	B
B	A	N	A	N	A
N	A	B	A	N	A
N	A	N	A	B	A

4

Se pide escribir:

- el método `public static ordenarArregloChars(char[][] c, int n, int m)`, que ordene las filas de la matriz c , dejándolas en orden lexicográfico en el mismo arreglo, y
- el método `public static String BW(String x)`, que retorne la transformada de x (un String como "NNBAAA 4", colocando el número tras un espacio).

```
// solución con método de la burbuja
public void ordenarArregloChars(char[][] c, int n, int m)
{
```

```
// vamos agrandando el conjunto de los "listos"
for (int i = n-1; i>=0; i--) {
    // subimos la burbuja hasta que queda "lista"
    for(int j = 0 ; j < i ; j++) {
        // sacamos palabras de las fila j y (j+1)
        String pal_j = "", pal_j1 = "" ;
        for(int k = 0 ; k < m ; k++) {
            pal_j += c[j][k] ;
            pal_j1 += c[j+1][k] ;
        }
        // si la palabra en j va después de
        // la de (j+1), las intercambiamos
        if(pal_j.compareTo(pal_j1) > 0)
            for (int k=0; k<m; k++)
            {
                char aux = c[j][k] ;
                c[j][k] = c[j+1][k] ;
                c[j+1][k] = aux ;
            }
    }
}

// parte (b)
public String BW(String x)
{
    int Tam = x.length() ;
    char[][] arr = new char[Tam][Tam] ;

    // llenamos primera fila del arreglo
    for (int j = 0 ; j < Tam ; j++)
        arr[0][j] = x.charAt(j) ;

    // llenamos las siguientes con los "shifteos"
    for (int i=0; i<Tam; i++)
        for (int j=1; j<Tam; j++)
            arr[j][(i+Tam-j)%Tam] = arr[0][i];

    // ordenamos
    ordenarArregloChars(arr,Tam,Tam) ;

    // obtenemos ultima columna
    String pal = "" ;
    for (int i=0; i<Tam; i++)
        pal += arr[i][Tam-1] ;

    // buscamos donde quedó la palabra original
    int i = 0 ;
    for (; i < Tam ; i++) {
        String aux = "" ;
        for (int j = 0 ; j < Tam ; j++)
            aux += arr[i][j] ;
        if (aux.equals(x))
            break ;
        else
            aux = "" ;
    }

    return pal + " " + (i+1);
}
```

7. Base de Datos de Mp3 (I Parte)

Los mp3 son archivos que contienen música codificada. Para efectos computines se pueden ver como un arreglo con bytes ("char") que tienen significado especial para los reproductores como winamp. Al final del arreglo de bytes, los últimos 128, suelen utilizarse para guardar información sobre la canción: Título, Artista, Album, Año, Comentario, Genero.

...mp3	3 bytes TAG	30 bytes TITULO	30 bytes ARTISTA	30b ALBUM	4b AÑO	30b COMEN	1b GENERO
--------	----------------	--------------------	---------------------	--------------	-----------	--------------	--------------

Se te pide programar la siguiente clase que permite escribir esta información al final de un mp3.

public ID3TAG(String file){}	Constructor
Public String Format(String s, int n){...}	Le da largo n al string s. Rellena con blancos " ", o trunca el largo para alcanzar n.
Public guardar(){ ...}	Escribe en el archivo, el TAG en memoria.
Public boolean tieneTAG(){ ...}	Retorna verdadero si el archivo tiene un TAG
Public boolean cargarTAG(){...}	Carga el TAG del archivo en el objeto. Retorna verdadero si lo logra, falso si no existe el TAG

Dato: writeBytes(String s), escribe el string s en un archivo RAF.
readByte(), lee un byte desde el archivo RAF y avanza el cabezal en un byte.

```
class mp3TAG{

    public String Titulo;
    public String Artista;
    public String Album;
    public String Comenario;
    public String Year;
    public String Genero;

    String archivo;

    public ID3TAG(String file){
        archivo=file;
    }

    // Le da largo n al string s.
    // Rellena con blancos " ", o truncan el largo para alcnasar n
    public String Format(String s, int n){
        if(s.length>n)
            return s.substring(0,n);
        for(int i= s.length(); i<=n; i++)
            s+=" ";
        return s;
    }

    // Escribe en el archivo el TAG en memoria
    public guardar(){
```

```
RandomAccesFile A =
    new RandomAccessFile("agenda.bin", "w");

/* si ya existe lo pisamos */
if(tieneTAG)
    A.seek(length()-128);
else /*sino nos colocamos al final */
    A.seek(length());

A.writeBytes(Format(titulo, 30));
A.writeBytes(Format(artista,30));
A.writeBytes(Format(album,30));
A.writeBytes(Format(year,4));
A.writeBytes(Format(comment,30));
A.writeBytes(Format(genero,1));

A.close();
}

// retorna verdadero si el archivo tiene tag
public boolean tieneTAG(){
    String s="";
    RandomAccesFile A =
        new RandomAccessFile("agenda.bin", "r");

    if(A.length==128) return false;

    A.seek(A.length-128);
    s+=(String)A.readByte();
    s+=(String)A.readByte();
    s+=(String)A.readByte();

    if(s.equals("TAG"))
        return true;
    A.close();
}

// Carga el TAG del archivo en el objeto
// Retorna verdadero si lo logra, falso si no existe el TAG
public boolean cargarTAG(){
    if(!tieneTAG)
        return false;

    RandomAccesFile A= new RandomAccessFile(archivo,"r");
    A.seek(A.length-128);

    titulo="";
    for(int i=0; i<30;i++) titulo+=A.readByte();

    artista="";
    for(int i=0; i<30;i++) titulo+=A.readByte();

    album="";
    for(int i=0; i<30;i++) titulo+=A.readByte();

    year=""
    for(int i=0; i<4;i++) titulo+=A.readByte();

    comentario="";
    for(int i=0; i<30;i++) titulo+=A.readByte();

    genero =A.readByte();
}
```

```
        A.close();  
        return true;  
    }  
}
```

8. Base de Datos de Mp3 (II Parte)

Suponga que existen las siguientes tablas:

ARTISTA

Cod_artista	Nombre	Nacionalidad	Fec_nacimiento(ddmmyyyy)
000007	Magdalena	Argentina	12071980

ALBUM

Cod_album	Album	Productora
654321	Sublime	Virgin Records

CANCION

cod_cancion	Archivo	Titulo	Fecha	letra	Genero(Rock,...)
123456	Musica.mp3	Something	2001	...	Rock Clasico

ARTISTA_ALBUM:

Cod_artista	Cod_album
000007	654321

CANCION_ALBUM:

Cod_cancion	Cod_album
123456	654321

- a) Genere un programa usando JDBC y la clase **mp3TAG** que actualice los TAGs de los mp3s que aparecen en la tabla **CANCION**.

```
Statement s= con.crearSatement();
Result set r;

r=s.executeQuery("SELECT C.nombre_archivo, "+
                  "C.titulo, "+
                  "AR.nombre, "+
                  "AL.album, "+
                  "CA.fecha, "+
                  "GENERO.numero "+
                  "FROM CANCION C, "+
                  "ARTISTA-ALBUM AA, "+
                  "CANCION-ALBUM CA, "+
                  "ARTISTA AR, "+
                  "ALBUM AL "+
                  "WHERE C.cod_cancion = CA.cod_cancion AND "+
                  "CA.cod_album=AA.cod_album AND "+
                  "AA.cod_artista= AR.cod_artista AND "+
                  "AA.cod_album =AL.cod_album AND "+
                  "GENERO.numero=C.genero ");

while(r.next()){

    mp3TAG mp3 = new mp3TAG(r.getString(nombre_archivo));
    mp3.titulo = r.getString("titulo");
    mp3.artista = r.getString("nombre");
    mp3.album = r.getString("album");
```



```
mp3.comentario = "";  
mp3.year = r.getString("fecha");  
mp3.genero = r.getString("numero");  
  
mp3.guardar();  
}
```

- b) **Propuesto:** Suponga las tablas están en TablasRAF.
- c) Imprima un listado de todos los artistas que publicaron canciones el año 2001 y tenían 18 o 19 años de edad. Suponga que las canciones son creadas el primero de enero del año "fecha".

```
Statement s= con.crearSatement();  
Result set r, r1;  
  
//extraemos el listado de todos los artistas  
r=s.executeQuery("SELECT nombre, cod_artista,fec_nacim "+  
                "FROM ARTISTA AR);  
  
while(r.next()){  
  
    //sacamos el agno que nacion  
    int agno= Integer.parseInt(  
        r.getString("fec_nacim").substring(4));  
    // ahora agno es el agno en que el artista tubo 18  
    int agno+=18;  
  
    r1=s.executeQuery("SELECT C.cod_cancion+  
        "FROM CANCION C, "+  
        "ARTSISTA AR, "+  
        "ARTISTA_ALBUM, "+  
        "CANCION_ALBUM CA "+  
        "WHERE C.cod_cancion=CA.cod_cancion AND"+  
        "CA.cod_album=AA.cod_album AND"+  
        "AA.cod_artista="+r.getString("cod_artista") AND +  
        "(C.fech="+agno+" OR C.fecha="+agno+1));  
  
    if(r1.next())  
        System.out.println(r.getString(nombre));  
}
```

9. Procesador de Textos

Para implementar un procesador de texto "serio" puede aprovecharse la Orientación a Objetos que proveen lenguajes como Java. En este caso, nos basaremos en el modelo siguiente:

Un **Texto** es básicamente una lista de **Bloques**. Estos bloques pueden básicamente ser de tres tipos:

- **Título:** Una línea de texto correspondientemente subrayada.

Ej:

```
Informe de Práctica Profesional
-----
```

- **Párrafo:** Una o más líneas de texto, con sangría en la primera línea.

Ej:

```
    La lógica difusa es una rama de la Inteligencia Artificial que
    pretende imitar en el computador la arquitectura lógica humana.
    Aunque la informática sólo conoce un sistema de información binario
    (de ceros y unos), en el mundo real las cosas no son tan fáciles:
    existen diferentes grados de verdad en una frase; se puede estar un
    poco dormido o muy concentrado...
```

- **Enumeración:** Un listado de párrafos, cada uno precedido del símbolo "*".

Ej:

```
* Popular Condimento (3:02)
* Buenos Días a Todos (5:15)
* Eligiendo a una reina (4:05)
```

Ésta es la implementación de la clase abstracta **Bloque**:

```
abstract class Bloque {
    public Bloque() {}
    abstract public String representacion() ;
}
```

(a) Escribe la implementación de **Título** y **Párrafo**, considerando que:

- Ambos heredan de la clase **Bloque**
- Guardan el texto interno en un String corriente
- El método **representacion()** (que están obligados a definir) realiza el "formateo" del contenido, en el momento de retornarlo (no lo modifica).

(b) La clase **Texto**, como se dijo, guarda los **Bloques** en una *lista enlazada*, cuyos eslabones pertenecen a a clase **Elemento** definida a continuación:

```
class Elemento {
    Bloque bq ;
    Elemento sgte ;
    public Elemento (Bloque x, Elemento y) {
        bq = x ;
        sgte = y ;
    }
}
```

Un resumen de las componentes de la clase **Texto** es el siguiente:

Elemento prim;	cabeza de la lista
void agregar(Bloque b)	agrega el Bloque b al final del Texto
void inicializar()	deja la lista de Bloques vacía
String representacion()	retorna un String con la representacion de todo el Texto (concatenacion de las representaciones de los bloques individuales)

Se define además la interfaz **Guardable** como la que obliga a la clase que a implemente a definir un método con el cual guardar sus contenidos en un archivo.

```
interface Guardable {
    public void guardar(String nombreArchivo) ;
}
```

Escribe la clase **Documento** (que hereda de **Texto** e implementa **Guardable**), que incorpora además el método

```
void exportarHTML(String nomArch)
```

Este método guarda todos los **Párrafos** entre `<p>` y `</p>`, los **Títulos** entre `<h1>` y `</h1>` y las enumeraciones entre `` y `` (cada elemento de la **Enumeración** debe ir entre `` y ``).

Hint: escribe una función auxiliar que decida de qué clase es cada bloque y aplique el formato respectivo; y aplícala sobre todos los bloques de la lista.

PROPUESTOS:

- Implementar las enumeraciones (hint: usar la lista de bloques ya definida)
- Permitir párrafos alineados a la izquierda, derecha, centrados (en particular los títulos) y justificados (márgenes rectos)
- Crear una clase Plantilla, derivada de Documento, y que predefina una estructura para cartas, informes, currículums, etc
- Agregar la definición del bloque Tabla.

```
import java.io.* ;

interface Guardable {
    public void guardar(String nombreArchivo) ;
}

abstract class Bloque {

    public Bloque() {
    }
    abstract public String representacion() ;
}

// Parte (a)
class Parrafo extends Bloque {
    private String contenido ;
    public Parrafo(String x) {
        contenido = x ;
    }
    public String representacion() {
        return "    "+contenido+"\n" ;
        // con indentacion y salto de linea final
    }
}

class Titulo extends Bloque {
    private String contenido ;
    public Titulo (String x) {
        contenido = x ;
    }
    public String representacion() {
        return contenido+subrayado(contenido)+"\n" ;
    }
    // el subrayado es del mismo largo que el contenido
    private static String subrayado(String x) {
        String sub = "\n" ;
        for (int i = 0 ; i < x.length() ; i++)
            sub += "-" ;
        return sub ;
    }
}

class Enumeracion extends Bloque {
    Texto contenidos;
    // la enumeracion es una lista de bloques :)

    public Enumeracion () {
        contenidos = new Texto() ;
    }

    public void agregarItem(String x) {
        contenidos.agregar(new Parrafo("*    "+x)) ;
        // se agregan con "*" " al principio
    }
    public String representacion() {
        return contenidos.representacion()+"\n" ;
        // chutea la pelota a Texto
    }
}

// Parte (b)
```

```
class Texto {
    class Elemento {
        Bloque bq ;
        Elemento sgte ;
        public Elemento (Bloque x, Elemento y) {
            bq = x ;
            sgte = y ;
        }
    }
    Elemento prim, ult ;
    public Texto() {
        inicializar() ;
    }
    public void agregar(Bloque b){
        if(prim == null)
            prim = ult = new Elemento(b,null) ;
        else
            ult = ult.sgte = new Elemento(b,null) ;
    }
    public String representacion() {
        String rep = "" ;
        for (Elemento p = prim ; p != null ; p = p.sgte)
            rep += p.bq.representacion() ;
        // concatenacion de todas las representaciones
        return rep ;
    }
    public void inicializar() {
        prim = ult = null ;
    }
}

class Documento extends Texto implements Guardable {
    public Documento() {super();}
    public void guardar(String nombreArchivo) {
        try {
            PrintWriter pw = new PrintWriter(
                new FileWriter(nombreArchivo)) ;
            for (Elemento p = prim; p != null ; p = p.sgte )
                pw.println(p.bq.representacion()) ;
            pw.close() ;
        }
        catch(Exception e) {
            System.err.println("Error: No se pudo guardar "+
                "el archivo");
        }
    }
    public void exportarHTML(String nombreArchivo) {
        try {
            PrintWriter pw = new PrintWriter(
                new FileWriter(nombreArchivo+".html")) ;
            for (Elemento p = prim; p != null ; p = p.sgte )
                pw.println(traducirHTML(p.bq) ;
                // imprime la traduccion de cada bloque
            pw.close() ;
        }
        catch(Exception e) {
            System.err.println("Error: No se pudo guardar "+
                "el archivo");
        }
    }
    private String traducirHTML(Bloque b) {
        String cont = "" ;
        // primero se averigua la clase original,
```

```
// y luego se ponen los tags
if (b instanceof Parrafo)
    cont = "<p>" + ((Parrafo)b).contenido + "</p>";
else if (b instanceof Titulo)
    cont = "<h1>" + ((Titulo)b).contenido + "</h1>";
else if (b instanceof Enumeracion) {
    Enumeracion e = (Enumeracion)b ;

    // en este caso hay que traducir los bloques
    // de la enumeracion, agregarles
    // a cada uno los <li> y </li>, y
    // finalmente agregarles los <ul> y </ul> a todo
    String aux = "" ;
    for (Bloque v = e.contenidos.prim ;
        v != null ;
        v = v.sgte)
        aux += "<li>" + traducirHTML(v) + "</li>";
    cont = "<ul>" + aux "</ul>" ;
}
return cont ;
}
```

10. Árboles

Para los siguientes problemas considere las declaraciones:

```
class Nodo {
    public String valor;
    public Nodo izq, der,
    public Nodo(String x, Nodo y, Nodo z){
        valor=x; izq=y; der=z;
    }
}

Nodo raiz = null;
raiz=new Nodo("C",
             new Nodo("B",null,null),
             new Nodo("D",null, new Nodo("F",null,null)));
```

(a) Escriba un método recursivo que entregue el número de hojas de un árbol.

Encabezamiento: **int hojas(Nodo x)**

Ejemplo:

```
int a = hojas(raiz)           // a toma el valor 2
```

```
private int hojas(Nodo x){
    if( x == null )
        return 0;
    else if( x.izq==null && x.der==null )
        return 1;
    else
        return hojas(x.izq) + hojas(x.der);
}
```

(b) Escriba un método recursivo que determine la altura de un árbol, es decir el número de niveles de nodos.

Encabezamiento: **int altura(Nodo x)**

Ejemplo:

```
int a = altura(raiz);         // a toma el valor 3
```

```
private int altura(Nodo x){
    if( x == null )
        return 0;
    else
        return 1 + Math.max(altura(x.izq),altura(x.der));
}
```

- (c) Escriba un método (no recursivo) que entregue el valor del sucesor de un nodo, es decir, el menor descendiente del subárbol derecho.

Encabezamiento: **String sucesor(Nodo x)**

Ejemplo:

```
String x = sucesor(raiz);           // x toma el valor D
```

```
private String sucesor(Nodo x){  
    if( x==null || x.der==null) return null;  
    Nodo p = x.der;  
    while(p.izq != null )  
        p = p.izq;  
    return p.valor;  
}
```

- (d) Escriba un método recursivo que determine si un árbol es o no un árbol binario de búsqueda.

Encabezamiento: **boolean esABB(Nodo x)**

Ejemplo:

```
boolean b = esABB(raiz);           // b toma el valor TRUE
```

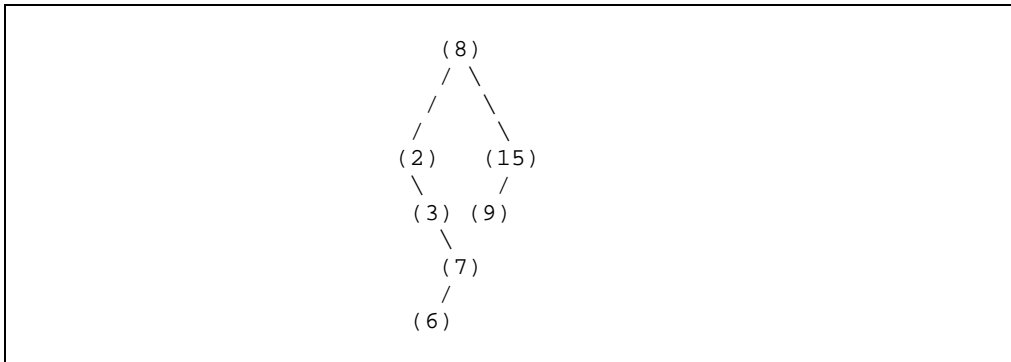
```
private boolean esABB(Nodo x){  
    if( x == null )  
        return true;  
    else if( x.izq == null && x.der == null )  
        return true;  
    else if( x.izq == null )  
        return (x.valor).compareTo(sucesor(x))<0  
            && esABB(x.der);  
    else if( x.der == null )  
        return pred(x).compareTo(x.valor)<0  
            && esABB(x.izq);  
    else  
        return pred(x).compareTo(x.valor)<0  
            && (x.valor).compareTo(sucesor(x))<0  
            && esABB(x.izq) && esABB(x.der);  
}
```


11. ABB (Árboles de Búsqueda Binaria)

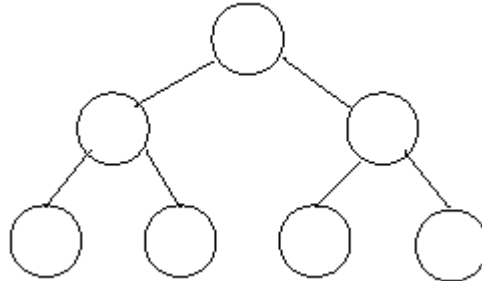
(a) Inserta los siguientes números en un árbol binario inicialmente vacío:

8, 2, 15, 3, 9, 7 y 6

Dibuje el árbol resultante.



(b) ¿En qué orden habría que insertar los elementos para que el árbol resultante tuviera la forma?



Lo primero que hay que hacer es ordenar los números que nos dan, de menor a mayor:

2, 3, 6, 7, 8, 9, 15

Luego aplicamos un procedimiento similar a búsqueda binaria y nos queda:

2, 3, 6, 7, 8, 9, 15

 ^

2, 3, 6, 7, 8, 9, 15

 ^ ^

2, 3, 6, 7, 8, 9, 15

 ^ ^ ^

Entonces:

1ro. el 7.

2do. el 3 y el 9 (en cualquier orden).

3ro. el 2, el 6, el 8 y el 15 (en cualquier orden).

- (c) Escriba un método que reciba un árbol binario y responda si éste es de búsqueda binaria o no. Recuerde que para que un árbol binario sea de búsqueda binaria es necesario que la raíz sea mayor que TODOS los elementos del subárbol izquierdo y menor que TODOS los elementos del subárbol derecho, y que sus subárboles sean también de búsqueda binaria.

```
class Arbol {  
    int info;  
    Arbol der;  
    Arbol izq;  
  
class esABB {  
    static public boolean esABB (Arbol tree) {  
        if (tree == null)  
            return false;  
        if ((tree.izq != null && tree.izq.info > tree.info) ||  
            (tree.der != null && tree.der.info < tree.info))  
            return false;  
        return esABB (tree.izq) && esABB (tree.der);  
    }  
}
```

12. Ancestros y Parientes

La Familia González lo ha contratado a usted para que escriba un programa que les permita determinar cuáles de sus miembros son parientes sanguíneos. Los miembros de la familia serán representados mediante objetos de la clase **Persona**:

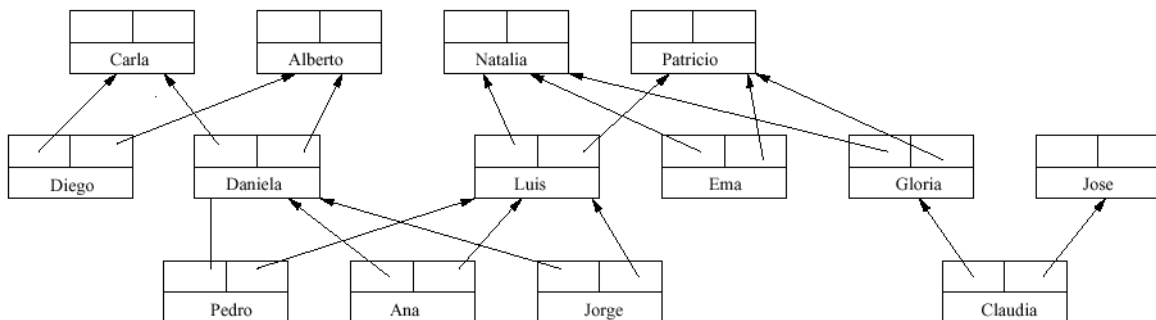
```
class Persona {  
    String nombre;  
    Persona madre, padre;  
    Persona (String nombre, Persona madre, Persona padre) {  
        this.nombre = nombre;  
        this.madre = madre;  
        this.padre = padre;  
    }  
}
```

Como se muestra en la clase, en cada objeto se almacena el nombre completo, quién es su madre y quién es su padre.

De esta forma el árbol genealógico de los González se construiría como:

```
Persona carla = new Persona ("Carla", null, null);  
Persona alberto = new Persona ("Alberto", null, null);  
Persona diego = new Persona ("Diego", carla, alberto);  
Persona daniela = new Persona ("Daniela", carla, alberto);  
...  
Persona jorge = new Persona ("Jorge", daniela, luis);  
... etc.
```

Básicamente, el árbol quedaría representado como (se suprimieron apellidos solo por espacio):



Aquí se aprecia que los padres de Jorge son Daniela y Luis. Como no se tiene información de la madre de Patricio, entonces *patricio.madre* es null. Lo mismo ocurre con el padre.

Nota: Observe que *pedro.padre.madre* es la abuela paterna de Pedro. Considere un caso general, es decir que funcione para cualquier árbol genealógico.

(a) Programe el método:

boolean esAncestroDe (Persona a, Persona b)

que determina si a es un ancestro de b. Por ejemplo, en el diagrama se observa que **esAncestroDe (carla, pedro)** es verdadero y que **esAncestroDe (daniela, alberto)** es falso.

```
boolean esAncestroDe (Persona a, Persona b) {  
    if (b == null)  
        return false;  
    if (a.nombre.equals(b.nombre))  
        return true;  
    return (esAncestroDe (a, b.madre) ||  
            esAncestroDe (a, b.padre));  
}
```

(b) Implemente el método:

boolean sonParientesSanguineos (Persona a, Persona b)

en donde se determina si a está sanguíneamente relacionado con b (es decir que tienen algún ancestro en común). Por ejemplo, **sonParientesSanguineos (pedro, claudia)** es verdadero porque Natalia es un ancestro común (también lo es Patricio), sin embargo **sonParientesSanguineos (diego, luis)** es falso. Por simplicidad considere que una persona es pariente sanguínea de si misma.

```
boolean sonParientesSanguineos (Persona a, Persona b) {  
    if (a == null)  
        return false;  
    if (esAncestro (a, b) || esAncestro(b, a));  
        return true;  
    return sonParientesSanguineos (a.madre, b) ||  
            sonParientesSanguineos (a.padre, b);  
}
```

13. Funcionamiento de un Restaurant

El restaurant de comida china "SHI WA" quiere implementar un sistema que administre su funcionamiento diario. Para esto, en la reunión con el cliente, éste nos cuenta como funciona el restaurant:

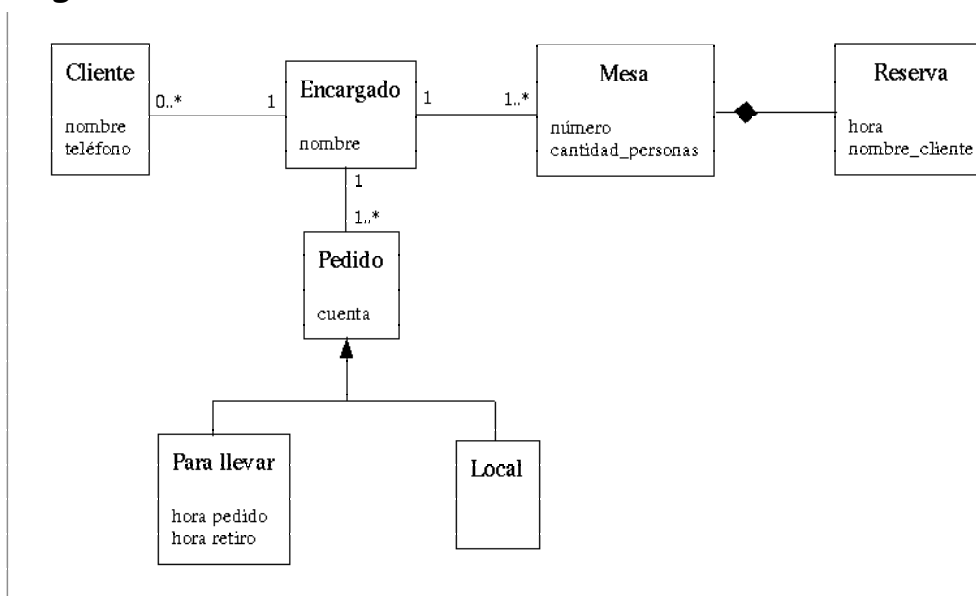
"En el local existe un cajero que se preocupa de administrar las reservas que los clientes hagan, además de tener las cuentas de las mesas. Para hacer las reservas, los clientes llaman por teléfono y avisan la cantidad de gente que va a venir, y, como tenemos mucha clientela, tenemos que pedirle una hora aproximada de reserva. Si después de 20 minutos no llega el cliente, entonces la mesa puede ser asignada a cualquier persona. Como este es un restaurant muy fino, no permitimos que nos desordenen el local, juntando mesas o colocándolas en distintos lugares."

Este restaurant, así como muchos de los restaurants de comida china, permite hacer pedidos para llevar, además de los pedidos en el local. También le preguntamos un poco de este sistema al dueño del local:

"Nuestro local, también tiene el servicio de comida para llevar. El pedido lo puede hacer el cliente por teléfono y venirlo a buscar al rato, o viniendo al local para hacer el pedido y esperando aquí. Como se habrán dado cuenta, nuestro restaurant es muy moderno, y tiene ideas revolucionarias... si el pedido no está listo en menos de 45 minutos, el cliente tiene un 10 % de descuento en la compra."

A usted se le pide que haga el diagrama de clases involucrado, y el diagrama de los casos de uso del sistema. También es necesario que haga un diagrama de estado, mostrando los distintos estados del sistema, cuando hace la compra para llevar.

Diagrama de Clases



Casos de Uso

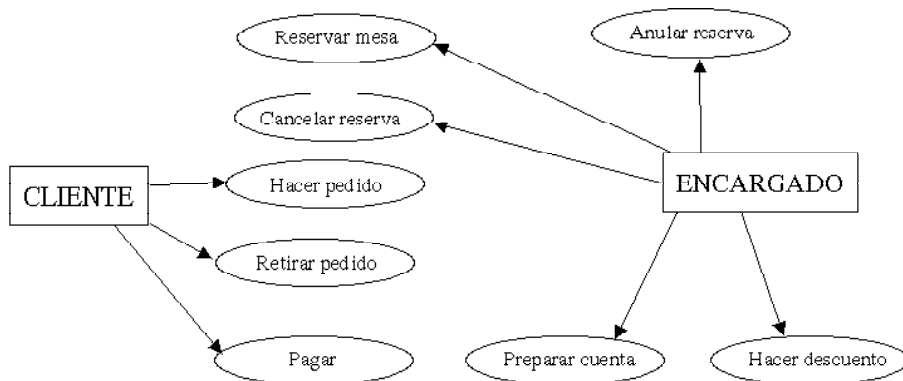
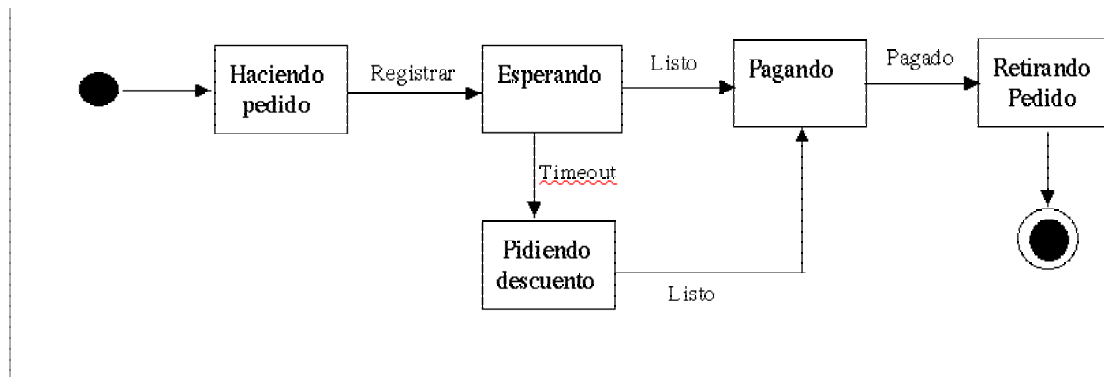


Diagrama de Estado



14. La Clase Tabla (Pregunta 1, Examen 2001)

La clase **Tabla** permite operar con una lista de pares (String X, int Y) a través de los siguientes métodos:

Ejemplo	Resultado	Significado
T = new Tabla()	-	constructor que inicializa tabla vacía (sin pares)
T.agregar("rosa",11)	void	agrega el par ("rosa",11) a la tabla T
T.contar("ros")	int	Entrega la cantidad de pares cuyos valores de X comienzan con el String "ros"
T.sumar("ros")	int	Entrega la suma de los valores Y de los pares cuyos valores de X comienzan con el String "ros"

- a) Escriba un programa que, utilizando la clase anterior, muestre la cantidad promedio de alumnos de los cursos impartidos por el Departamento de Ciencias de la Computación (cuyos códigos comienzan por "CC").

La información de los cursos se encuentra en el archivo "cursos.txt". Cada línea contiene la siguiente información de un curso:

- Columnas 1 a 5: Código de curso (ejemplo: MA10A)
- Columnas 6 a 8: número de alumnos del curso (ejemplo: 100)

```
...
Tabla cursos = new Tabla();
BufferedReader fd = new BufferedReader(
    new FileReader("cursos.txt"));
while(true) {
    String line = fd.readLine();
    cursos.agregar(line.substring(0, 6),
        Integer.parseInt(line.substring(6)));
}
fd.close();
System.out.println(cursos.sumar("CC") / cursos.contar("CC") +
    " alumnos por curso");
...
```

- b) Escriba el método sumar, suponiendo que cada objeto de la clase **Tabla** está representado con un árbol binario de búsqueda (o árbol de búsqueda binaria) de elementos o nodos de la clase:

```
class Nodo{
    public String X;
    public int Y;
    public Nodo izq, der;
}
```

```
public int sumar(String X) {
    Nodo p = this.raiz;
    return sumarRec(X, p);
}
```

```
public int sumarRec(String X, Nodo p) {  
    int suma = 0;  
    if (p == null)  
        return 0;  
    if (p.X.indexOf(X) == 0)  
        suma += p.Y;  
    suma += sumarRec(X, p.izq) + sumarRec(X, p.der);  
    return suma;  
}
```


15. Selección Universitaria (Pregunta 2, Examen 2001)

Para el proceso de selección de alumnos a las universidades se dispone de una base de datos compuesta por las siguientes tablas:

Carreras		Alumnos		Postulaciones	
Columna	Tipo	Columna	Tipo	Columna	tipo
Codigo	String	Cedula	String	CedulaAlumno	String
Nombre	String	Nombre	String	CodigoCarrera	String
Ponderador1	Int	Puntaje1	Int		
...		
Ponderador7	Int	Puntaje7	int		

Escriba un programa en Java que muestre el nombre del postulante con el mejor puntaje ponderado para ingresar al Plan Común de Ingeniería (código 1520).

Notas

- Las tres tablas se encuentran ordenadas por sus primeras columnas
- El puntaje ponderado se calcula como:

$$\text{Puntaje1} * \text{Ponderador1} + \dots + \text{Puntaje7} * \text{Ponderador7}$$

- El programa puede escribirse utilizando JDBC (Java y SQL) o archivos (de texto o acceso directo) según la convención utilizada por su profesor

17. Operación Olitas.

La operación OLITAS consiste en maniobras navales en las que participan naves pertenecientes a 9 países. La superficie marítima se cuadricula para representarla en una matriz de 100 por 80 caracteres. Consecuentemente, cada barco se representa por una secuencia ininterrumpida horizontal o vertical del dígito (1 a 9) que representa al país. Los lugares no ocupados por un barco se representan con un dígito cero.

Escriba un programa que escriba el número del país y el tamaño (cantidad de espacios que ocupa) del barco más grande que participa en la operación. La información se encuentra grabada en las 100 líneas del archivo "barcos.txt". Cada línea contiene 80 caracteres (del '0' al '9').

18. Agenda enlazada.

R.B. es una persona que viaja mucho y tiene amigos repartidos por todo el mundo. R.B. ha ideado una estructura muy eficiente para guardar los números de teléfono de sus amigos por medio de un árbol cuyos nodos responden a la siguiente declaración:

```
public class Nodo {  
    public class int numero;  
    public String nombre;  
    //para la raíz esta información es irrelevante  
    public Nodo[] hijo=new Nodo[10];  
    //cantidad máxima de hijos  
    public int hijos=0;  
    //cantidad efectiva de hijos  
}
```

De acuerdo a lo anterior:

- * La raíz del árbol (nivel 1) contiene como información el número que debe marcar para iniciar una llamada internacional (ejemplo 00)
- * Los hijos de la raíz (nivel 2) tienen los números correspondientes a los códigos de cada país donde R.B. tiene registrado el teléfono de al menos un amigo. Estos nodos también contienen el nombre del país. Por ejemplo, 56, "Chile".
- * Los hijos de cada nodo del nivel anterior (nivel 3) contienen los códigos de las ciudades del país representado por el padre, donde R.B. tiene por lo menos un número registrado. Estos nodos contienen además el nombre de la ciudad. Por ejemplo, 2, "Santiago".
- * Finalmente los nodos del nivel 4 contienen la parte final del número telefónico para personas. Estos nodos contienen además el nombre de la persona que tiene ese número. Por ejemplo, 1234567, "Pato".

Por lo anterior, se puede ver que para obtener el número telefónico internacional de una persona dada, basta concatenar los números desde la raíz hasta una de las hojas. Suponga que la clase que implementa esta agenda en forma de árbol como:

```
public class Agenda {  
    private Nodo raiz=new Nodo();  
    public String nombre(int numero_pais, int numero_ciudad, int  
        numero_persona)  
        //devuelve nombre de la persona con ese número de teléfono, null si no  
        está  
    public void agregar (int numero_pais, String nombre_pais, int  
        numero_ciudad,  
        String nombre_ciudad, int numero_persona, String nombre_persona)  
        //agrega a la persona de nombre dado con el telefono dado a la  
        estructura  
        //sin duplicar nodos de país ni ciudad  
    public String fono (String x);  
        //busca el nombre x y entrega el fono en la forma pais-ciudad-numero  
        //ejemplo : 56-2-1234567  
}
```

- * (2 puntos) Escriba un método public static Nodo Hijo (Nodo x, int y) que retorna el nodo x que contiene en su componente numero un valor igual a y.

- * (2 puntos) Usando a) escriba el método nombre.
- * (2 puntos) Escriba el método fono.

19. Tabla de alumnos.

La clase **Tabla** permite operar con una lista de pares (**String X, int Y**) a través de los siguientes métodos:

Ejemplo	Resultado	Significado
T = new Tabla()	-	Constructor que inicializa tabla vacía (sin pares)
T.agregar("rosa",11)	void	agrega el par ("rosa",11) a la tabla T
T.contar("ros")	int	Entrega la cantidad de pares cuyos valores de X comienzan con el String "ros"
T.sumar("ros")	int	Entrega la suma de los valores Y de los pares cuyos valores de X comienzan con el String "ros"

- a. Escriba un programa que, utilizando la clase anterior, muestre la cantidad promedio de alumnos de los cursos impartidos por el Departamento de Ciencias de la Computación (cuyos códigos comienzan por "CC").

La información de los cursos se encuentra en el archivo "cursos.txt". Cada línea contiene la siguiente información de un curso:

- Columnas 1 a 5: Código de curso (ejemplo: MA10A)
- Columnas 6 a 8: número de alumnos del curso (ejemplo: 100)

- b. Escriba el método sumar, suponiendo que cada objeto de la clase **Tabla** está representado con un árbol binario de búsqueda (o árbol de búsqueda binaria) de elementos o nodos de la clase:

```
class Nodo
{
    public String X;
    public int Y;
    public Nodo izq, der;
}
```

20. Selección Universitaria.

Para el proceso de selección de alumnos a las universidades se dispone de una base de datos compuesta por las siguientes tablas:

Carreras		Alumnos		Postulaciones	
Columna	Tipo	Columna	Tipo	Columna	Tipo
codigo	String	cedula	String	cedulaAlumno	String
nombre	String	nombre	String	codigoCarrera	String
ponderador1	String	puntaje1	String		
ponderador2	String	puntaje2	String		
...		
ponderador7	String	puntaje7	String		

Escriba un programa en Java que muestre el nombre del postulante con el mejor puntaje ponderado para ingresar al Plan Común de Ingeniería (código 1520).

Notas :

Las tres tablas se encuentran ordenadas por sus primeras columnas

El puntaje ponderado se calcula como:

$\text{Puntaje1} * \text{Ponderador1} + \dots + \text{Puntaje7} * \text{Ponderador7}$

El programa puede escribirse utilizando JDBC (Java y SQL) o archivos (de texto o acceso directo) según la convención utilizada por su profesor

21. Planilla electrónica

La clase **Planilla** permite realizar las operaciones básicas para manejar una planilla electrónica (hoja de cálculo) de 26 columnas (A a la Z) y 100 filas (1 a 100) a través de los siguientes métodos:

Ejemplo	Significado
p = new Planilla("notas")	constructor que inicializa la planilla p con los valores contenidos en el archivo "notas"
p.asignar("B37", "hola")	Guarda el string "hola" en la celda ubicada en la columna B y la fila 37 de la planilla p
p.obtener("B37")	Entrega el string contenido en la celda ubicada en la columna B y la fila 37 de la planilla p (o null si la celda está vacía)
p.grabar("aux")	Graba la planilla p en el archivo "aux"

- a. Escriba un programa (frame) que, utilizando la clase anterior, actualice la planilla grabada en un archivo (cuyo nombre se recibe como parámetro del programa principal) utilizando la siguiente interfaz:

quit	A	B	...	Z
1				
2				
...				
100				

b. *Notas*

- El usuario puede cambiar el valor de cualquier celda (utilizando componentes de clase `TextField`)
 - La planilla debe ser regrabada en el mismo archivo al oprimirse el botón **quit**
- c. Escriba el método `obtener`, suponiendo que cada objeto de la clase `Planilla` está representado con un árbol binario de búsqueda (o árbol de búsqueda binaria) de elementos o nodos de la clase:

```
class Nodo
{
    // ubicación (ej "B37") y contenido (ej "hola") de un casillero
    public String celda, valor;

    public Nodo izq, der;
}
```


Nota En el árbol se guardan sólo las celdas que contienen algún valor, ordenadas por ubicación.

22. Cinema.

Una base de datos cinematográfica contiene las siguientes cuatro tablas (todas ordenadas por el identificador ubicado en su primera columna):

Películas		Artistas		Dirige		Actua	
Columna	Tipo	Columna	Tipo	Columna	Tipo	Columna	Tipo
id	String	id	String	idArtista	String	idArtista	String
titulo	String	nombre	String	idPelicula	String	idPelicula	String
pais	String	pais	String				

- Escriba un programa que, utilizando JDBC (Java+SQL), muestre el nombre del director chileno que ha dirigido más películas.
- Escriba un programa que, utilizando las clases y convenciones utilizadas por el profesor de su sección, muestre los nombres de las películas en que actúan conjuntamente Penélope Cruz y Antonio Banderas.

23. Puzzle.

Escriba una Applet (o Frame) que permita controlar un juego en que inicialmente aparece un cuadrículado de 9 botones en 3 filas y 3 columnas, en que 8 de ellos contienen números enteros entre 1 y 8 (que se ubican al azar). Por ejemplo, la configuración inicial en un juego puede ser la siguiente:

4	7	1
3		5
2	8	6

El juego consiste en usar la posición vacía para mover alguno de los números contiguos (en el ejemplo, 3, 7, 5 u 8). El movimiento se debe activar haciendo un click en el botón que contiene el número. El programa debe terminar cuando se alcance la configuración:

1	2	3
4	5	6
7	8	

Nota. Recuerde que los métodos `setLabel(x)` y `getLabel()` permiten establecer y recuperar el texto asociado a un botón.

Solucion:

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Puzzle extends Frame implements ActionListener {

    Button[][] botones;
    Button vacio;

    public Puzzle(){
        init();
    }

    public void init(){
        botones = new Button[3][3];
        this.setLayout(new GridLayout(3, 3));

        String[][] labels = getLabels();
        for(int i = 0; i < botones.length; i++){
            for(int j = 0; j < botones[i].length; j++){
                botones[i][j] = new Button(labels[i][j]);
                add(botones[i][j]);
                botones[i][j].addActionListener(this);

                if(botones[i][j].getLabel().equals("")){
                    vacio = botones[i][j];
                }
            }
        }
    }
}
```

```
/**
 * Aquí manejamos los eventos
 */
public void actionPerformed(ActionEvent e){
    Button origen = (Button) e.getSource();
    if(!estaCerca(origen)){
        return;
    }

    //intercambio los valores
    String label = origen.getLabel();
    origen.setLabel("");
    vacio.setLabel(label);
    vacio = origen;

    //SI esta listo, desactivo los botones
    if(estaListo()){
        for(int i = 0; i < botones.length; i++){
            for(int j = 0; j < botones[i].length; j++){
                botones[i][j].setEnabled(false);
            }
        }
    }
}

private boolean estaCerca(Button origen){
    int pos_origen_x = 0;
    int pos_origen_y = 0;

    for(int i = 0; i < botones.length; i++){
        for(int j = 0; j < botones[i].length; j++){
            if(botones[i][j] == origen){
                pos_origen_x = i;
                pos_origen_y = j;
            }
        }
    }

    int pos_vacio_x = 0;
    int pos_vacio_y = 0;

    for(int i = 0; i < botones.length; i++){
        for(int j = 0; j < botones[i].length; j++){
            if(botones[i][j] == vacio){
                pos_vacio_x = i;
                pos_vacio_y = j;
            }
        }
    }

    int dx = Math.abs(pos_origen_x - pos_vacio_x);
    int dy = Math.abs(pos_origen_y - pos_vacio_y);

    if(dx == 0 && dy == 1){
        return true;
    }

    if(dy == 0 && dx == 1){
        return true;
    }

    return false;
}
```

```
private boolean estaListo(){
    int valor = 1;
    for(int i = 0; i < botones.length; i++){
        for(int j = 0; j < botones[i].length; j++){
            if(!botones[i][j].getLabel().equals(valor + ")){
                return false;
            }

            ++valor;
            if(valor == 9){
                return true;
            }
        }
    }

    return true;
}

private String[][] getLabels(){
    String[] numeros = new String[9];
    for(int i = 1; i < numeros.length; i++){
        numeros[i] = i + "";
    }
    numeros[0] = "";

    String[][] labels = new String[3][3];
    for(int i = 0; i < labels.length; i++){
        for(int j = 0; j < labels[i].length; j++){

            int azar;
            while(numeros[azar = azarozo(0, 9)] == null){
            }

            labels[i][j] = numeros[azar];
            //lo elimino de la lista de numeros posibles
            numeros[azar] = null;
        }
    }

    return labels;
}

/**
 * Metodo auxiliar que genera un numero al azar entre [n, m[
 */
public static int azarozo(int n, int m){
    return (int) Math.floor(Math.random() * (m - n) + n);
}

public static void main(String[] args){
    Frame puzzle = new Puzzle();
    puzzle.setSize(300, 300);
    puzzle.show();
}
}
```

24. Clase de números.

La clase `Tabla` permite mantener una lista de números reales a través de los siguientes métodos:

ejemplo	resultado	significado
<code>T = new Tabla()</code>	-	constructor que inicializa tabla con cero elementos
<code>T.agregar(x)</code>	void	agregar el N° real <code>x</code> a la Tabla <code>T</code>
<code>T.estaVacia()</code>	boolean	Devuelve true si <code>T</code> está vacía (y false si no)
<code>T.extraerMayor()</code>	double	Devuelve (y elimina) el mayor N° de la Tabla <code>T</code>
<code>T.extraerMenor()</code>	double	Devuelve (y elimina) el menor N° de la Tabla <code>T</code>

a) Escriba el método **extraerMenor** suponiendo la siguiente declaración de la clase `Tabla`:

```
class Tabla{
    private double[] numeros;
    private int n; //cantidad efectiva de elementos
    public Tabla(){ numeros=new double[100]; n=0; }
    ...
}
```

b) Utilice la clase anterior en un programa que genere un número entero *n* al azar entre 10 y 100, y a continuación genere *n* números reales al azar. Finalmente, escriba los 10 mayores y los 10 menores.

Nota. Recuerde que `Math.random()` genera un número al azar de tipo `double` en el intervalo `[0,1[`

Solución:

```
public class Tabla {
    private double[] numeros;
    private int n;

    public Tabla() {
        numeros = new double[100];
        n = 0;
    }

    public void agregar(double x) {
        numeros[n] = x;
        ++n;
    }

    public boolean estaVacia() {
        return n == 0;
    }

    public double extraerMayor() {
```

```
        int indice_mayor = 0;
        for (int i = 0; i < n; i++) {
            if (numeros[indice_mayor] < numeros[i]) {
                indice_mayor = i;
            }
        }
        double mayor = numeros[indice_mayor];
        --n;
        numeros[indice_mayor] = numeros[n];
        return mayor;
    }

    public double extraerMenor() {
        int indice_menor = 0;
        for (int i = 0; i < n; i++) {
            if (numeros[indice_menor] > numeros[i]) {
                indice_menor = i;
            }
        }

        double menor = numeros[indice_menor];
        --n;
        numeros[indice_menor] = numeros[n];

        return menor;
    }

    /**
     * Parte B
     */
    public static void main(String[] args) {
        int n = azarozo(10, 100);
        Tabla t1 = new Tabla();
        Tabla t2 = new Tabla();

        for (int i = 1; i <= n; ++i) {
            double azar = Math.random();
            t1.agregar(azar);
            t2.agregar(azar);
        }

        System.out.println("10 Mayores :");
        for (int i = 1; i <= 10; ++i) {
            System.out.println(t1.extraerMayor());
        }

        System.out.println("");
        System.out.println("10 Menores :");
        for (int i = 1; i <= 10; ++i) {
            System.out.println(t2.extraerMenor());
        }
    }

    /**
     * Metodo auxiliar que genera un numero al azar entre [n, m[
     */
    public static int azarozo(int n, int m) {
        return (int) Math.floor(Math.random() * (m - n) + n);
    }
}
```



```
double cos = Math.sqrt(1 - sen * sen);
String tangente;
if (cos == 0) {
    tangente = "Infinito";
}
else {
    tangente = sen / cos + "";
}
System.out.println(angulo + "\t\t\t" + tangente);
angulo = angulo + Math.PI / 8;
} } }
```

26. Elecciones.

El servicio electoral le pide a Ud. Que escriba un programa que agilice el recuento de votos para una posible segunda vuelta en las próximas elecciones. El programa debe desplegar una ventana (Frame) con dos botones (uno para cada candidato de la segunda vuelta) y 4 campos de texto que muestren la cantidad de votos que ha percibido cada candidato y el porcentaje relativo de votos. La siguiente figura muestra la apariencia que debe tener la ventana:



Cada vez que el usuario presione algunos de los botones, su programa debe incrementar el número de votos del candidato respectivo y recalcular los porcentajes relativos. En este recuento no se consideran los votos nulos y blancos. No se preocupe de cómo termina su programa.