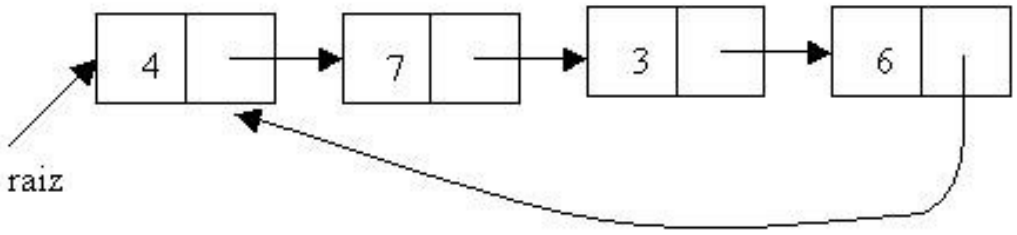
Curso de Java a distancia

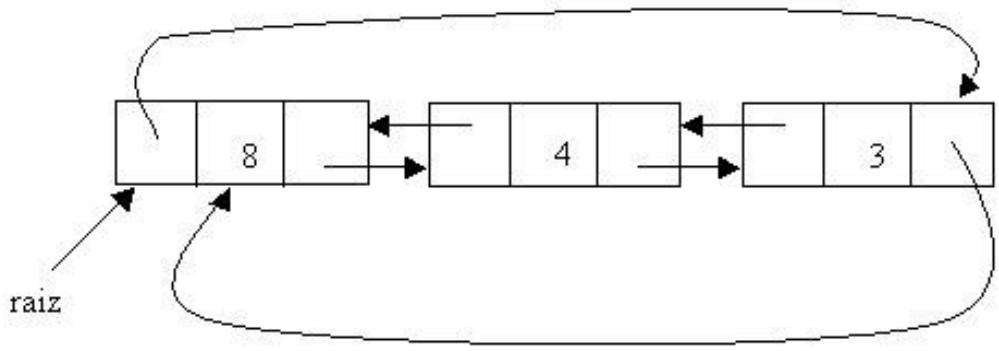
Clase 17: Estructuras dinámicas: Listas genéricas circulares

Una lista circular simplemente encadenada la podemos representar gráficamente:



Observemos que el puntero sig del último nodo apunta al primer nodo. En este tipo de listas si avanzamos raiz no perdemos la referencia al nodo anterior ya que es un círculo.

Una lista circular puede también ser doblemente encadenada:



El puntero ant del primer nodo apunta al último nodo de la lista y el puntero sig del último nodo de la lista apunta al primero.

Resolveremos algunos métodos para administrar listas genéricas circulares doblemente encadenadas para analizar la mecánica de enlace de nodos.

Programa:

```
public class ListaCircular {

   class Nodo {
      int info;
      Nodo ant,sig;
   }

   private Nodo raiz;

   public ListaCircular () {
      raiz=null;
   }

   public void insertarPrimero(int x) {
      Nodo nuevo=new Nodo();
      nuevo.info=x;
      if (raiz==null) {
         nuevo.sig=nuevo;
         nuevo.ant=nuevo;
         raiz=nuevo;
      } else {
         Nodo ultimo=raiz.ant;
         nuevo.sig=raiz;
         nuevo.ant=ultimo;
         raiz.ant=nuevo;
         ultimo.sig=nuevo;
         raiz=nuevo;
      }
   }

   public void insertarUltimo(int x) {
      Nodo nuevo=new Nodo();
      nuevo.info=x;
      if (raiz==null) {
         nuevo.sig=nuevo;
         nuevo.ant=nuevo;
         raiz=nuevo;
      } else {
         Nodo ultimo=raiz.ant;
         nuevo.sig=raiz;
         nuevo.ant=ultimo;
         raiz.ant=nuevo;
         ultimo.sig=nuevo;
      }
   }

   public boolean vacia ()
```

```
{
   if (raiz == null)
      return true;
   else
      return false;
}

public void imprimir ()
{
   if (!vacia()) {
      Nodo reco=raiz;
      do {
         System.out.print (reco.info + "-");
         reco = reco.sig;
      } while (reco!=raiz);
      System.out.println();
   }
}

public int cantidad ()
{
   int cant = 0;
   if (!vacia()) {
      Nodo reco=raiz;
      do {
         cant++;
         reco = reco.sig;
      } while (reco!=raiz);
   }
   return cant;
}

public void borrar (int pos)
{
   if (pos <= cantidad ())    {
      if (pos == 1) {
         if (cantidad()==1) {
            raiz=null;
         } else {
            Nodo ultimo=raiz.ant;
            raiz = raiz.sig;
            ultimo.sig=raiz;
            raiz.ant=ultimo;
         }
      } else {
         Nodo reco = raiz;
         for (int f = 1 ; f <= pos - 1 ; f++)
            reco = reco.sig;
```

```
            Nodo anterior = reco.ant;
            reco=reco.sig;
            anterior.sig=reco;
            reco.ant=anterior;
          }
        }
    }

    public static void main(String[] ar) {
        ListaCircular lc=new ListaCircular();
        lc.insertarPrimero(100);
        lc.insertarPrimero(45);
        lc.insertarPrimero(12);
        lc.insertarPrimero(4);
        System.out.println("Luego de insertar 4 nodos al principio");
        lc.imprimir();
        lc.insertarUltimo(250);
        lc.insertarUltimo(7);
        System.out.println("Luego de insertar 2 nodos al final");
        lc.imprimir();
        System.out.println("Cantidad de nodos:"+lc.cantidad());
        System.out.println("Luego de borrar el de la primer posición:");
        lc.borrar(1);
        lc.imprimir();
        System.out.println("Luego de borrar el de la cuarta posición:");
        lc.borrar(4);
        lc.imprimir();
    }
}
```

Para insertar al principio de una lista circular doblemente encadenada:

```
    public void insertarPrimero(int x) {
```

Creamos un nodo y guardamos la información:

```
        Nodo nuevo=new Nodo();
        nuevo.info=x;
```

Si la lista está vacía luego tanto el puntero sig y ant apuntan a si mismo ya que debe ser circular (y raiz apunta al nodo creado):

```
        if (raiz==null) {
            nuevo.sig=nuevo;
            nuevo.ant=nuevo;
            raiz=nuevo;
```

En caso que la lista no esté vacía disponemos un puntero al final de la lista (el puntero ant del primer nodo tiene dicha dirección):

```
        } else {
            Nodo ultimo=raiz.ant;
```

El nodo a insertar lo enlazamos previo al nodo apuntado por raiz:

```
nuevo.sig=raiz;
nuevo.ant=ultimo;
raiz.ant=nuevo;
ultimo.sig=nuevo;
```

Finalmente hacemos que raiz apunte al nodo creado luego de haber hecho todos los enlaces:

```
raiz=nuevo;
```

Para insertar un nodo al final de la lista:

```
public void insertarUltimo(int x) {
```

El algoritmo es idéntico al método que inserta al principio con la salvedad que no desplazamos raiz con la dirección del nodo creado (es decir al insertar en la posición anterior del primer nodo lo que estamos haciendo realmente es insertar al final de la lista):

```
    Nodo nuevo=new Nodo();
    nuevo.info=x;
    if (raiz==null) {
       nuevo.sig=nuevo;
       nuevo.ant=nuevo;
       raiz=nuevo;
    } else {
       Nodo ultimo=raiz.ant;
       nuevo.sig=raiz;
       nuevo.ant=ultimo;
       raiz.ant=nuevo;
       ultimo.sig=nuevo;
    }
}
```

Para imprimir la lista ya no podemos disponer un puntero reco que apunte al primer nodo y que se detenga cuando encuentre un nodo que el atributo sig almacene null.

```
public void imprimir ()
```

Si la lista no está vacía disponemos un puntero en el primer nodo y utilizamos un do/while para recorrer la lista. La condición del do/while es que se repita mientras el puntero reco sea distinto a raiz (es decir que no haya dado toda la vuelta a la lista):

```
    if (!vacia()) {
       Nodo reco=raiz;
       do {
          System.out.print (reco.info + "-");
          reco = reco.sig;
       } while (reco!=raiz);
       System.out.println();
    }
}
```

Para borrar el nodo de una determinada posición:

public void borrar (int pos)

Debemos primero identificar si es el primero de la lista (ya que en este caso se modifica el puntero externo raiz):

```
if (pos <= cantidad ())    {
   if (pos == 1) {
```

Si es el primero y el único de la lista hacemos que raiz apunte a null:

```
if (cantidad()==1) {
   raiz=null;
```

Si no disponemos un puntero al final de la lista, avanzamos raiz y enlazamos el último nodo con el segundo de la lista:

```
} else {
   Nodo ultimo=raiz.ant;
   raiz = raiz.sig;
   ultimo.sig=raiz;
   raiz.ant=ultimo;
}
```

En caso que queremos borrar un nodo que se encuentra en medio de la lista o inclusive al final debemos recorrer con un for hasta el nodo que queremos borrar y luego disponemos un puntero en el nodo anterior y otro puntero en el nodo siguiente. Seguidamente procedemos a enlazar los nodos:

```
Nodo reco = raiz;
for (int f = 1 ; f <= pos - 1 ; f++)
   reco = reco.sig;
Nodo anterior = reco.ant;
reco=reco.sig;
anterior.sig=reco;
reco.ant=anterior;
```

Recursividad: Conceptos básicos

Primero debemos decir que la recursividad no es una estructura de datos, sino que es una técnica de programación que nos permite que un bloque de instrucciones se ejecute n veces. Remplaza en ocasiones a estructuras repetitivas.

Este concepto será de gran utilidad para el capítulo de la estructura de datos tipo árbol.

La recursividad es un concepto difícil de entender en principio, pero luego de analizar diferentes problemas aparecen puntos comunes.

En Java los métodos pueden llamarse a sí mismos. Si dentro de un método existe la llamada a sí mismo decimos que el método es recursivo.

Cuando un método se llama a sí mismo, se asigna espacio en la pila para las nuevas variables locales y parámetros.

Al volver de una llamada recursiva, se recuperan de la pila las variables locales y los parámetros antiguos y la ejecución se reanuda en el punto de la llamada al método.

Problema 1:

Implementación de un método recursivo.

Programa:

```
public class Recursividad {

   void repetir() {
      repetir();
   }

   public static void main(String[] ar) {
      Recursividad re=new Recursividad();
      re.repetir();
   }
}
```

La función repetir es recursiva porque dentro de la función se llama a sí misma.
Cuando ejecuta este programa se bloqueará y generará una excepción: "Exception in thread "main" java.lang.StackOverflowError"

Analicemos como funciona:
Primero se ejecuta la función main, luego de crear un objeto llamamos a la función repetir.
Hay que tener en cuenta que cada vez que se llama a una función se reservan 4 bytes de la memoria que se liberarán cuando finalice su ejecución.
La primera línea de la función llama a la función repetir, es decir que se reservan 4 bytes nuevamente. Se ejecuta nuevamente una instancia de la función repetir y así sucesivamente hasta que la pila estática se colme y se cuelgue el programa.

Problema 2:

Implementación de un método recursivo que reciba un parámetro de tipo entero y luego llame en forma recursiva con el valor del parámetro menos 1.

Programa:

```
public class Recursividad {

   void imprimir(int x) {
      System.out.println(x);
      imprimir(x-1);
   }

   public static void main(String[] ar) {
      Recursividad re=new Recursividad();
      re.imprimir(5);
   }
}
```

Desde la main se llama a la función imprimir y se le envía el valor 5. El parámetro x recibe el valor 5. Se ejecuta el algoritmo de la función, imprime el contenido del parámetro (5) y seguidamente se llama a una función, en este caso a sí misma (por eso decimos que es una función recursiva), enviándole el valor 4.
El parámetro x recibe el valor 4 y se imprime en pantalla el cuatro, llamando nuevamente a la función imprimir enviándole el valor 3.
Si continuamos este algoritmo podremos observar que en pantalla se imprime:

5 4 3 2 1 0 ?1 ?2 ?3 . . . . . .

hasta que se bloquee el programa.
Tener en cuenta que cada llamada a una función consume 4 bytes por la llamada y en este caso 4 bytes por el parámetro x. Como nunca finaliza la ejecución completa de las funciones se desborda la pila estática por las sucesivas llamadas.

Problema 3:

Implementar un método recursivo que imprima en forma descendente de 5 a 1 de uno en uno.

Programa:

```
public class Recursividad {

   void imprimir(int x) {
      if (x>0) {
         System.out.println(x);
         imprimir(x-1);
      }
   }

   public static void main(String[] ar) {
      Recursividad re=new Recursividad();
      re.imprimir(5);
   }
}
```

Ahora si podemos ejecutar este programa y observar los resultados en pantalla. Se imprimen los números 5 4 3 2 1 y no se bloquea el programa.
Analice qué sucede cada vez que el if (x>0) se evalúa como falso, ¿a qué línea del programa retorna?

Problema 4:

Imprimir los números de 1 a 5 en pantalla utilizando recursividad.

Programa:

```
public class Recursividad {

   void imprimir(int x) {
      if (x>0) {
         imprimir(x-1);
         System.out.println(x);
      }
```

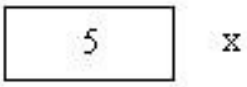
```
    }

    public static void main(String[] ar) {
        Recursividad re=new Recursividad();
        re.imprimir(5);
    }
}
```

Con este ejemplo se presenta una situación donde debe analizarse línea a línea la ejecución del programa y el porque de estos resultados.

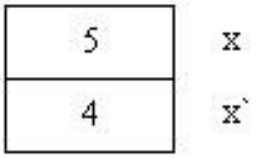
¿Por qué se imprime en pantalla 1 2 3 4 5 ?

Veamos como se apilan las llamadas recursivas:

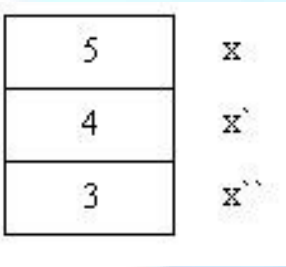
En la primera llamada desde la función main el parámetro x recibe el valor 5.



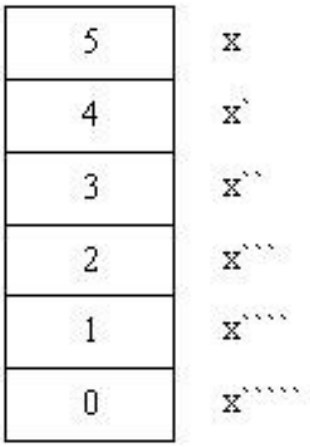
Cuando llamamos desde la misma función le enviamos el valor de x menos 1 y la memoria queda de la siguiente forma:



Debemos entender que el parámetro x en la nueva llamada está en otra parte de la memoria y que almacena un 4, nosotros le llamaremos x prima.
Comienza a ejecutarse la función, la condición del if se valúa como verdadero por lo que entra al bloque y llama recursivamente a la función imprimir pasándole el valor 3 al parámetro.



Nuevamente la condición se valúa como verdadero y llama a la función enviándole un 2, lo mismo ocurre cuando le envía un 1 y un 0.



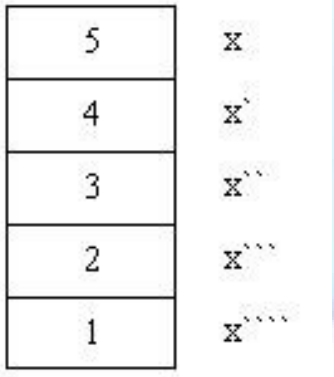

```
void imprimir(int x) {
   if (x>0) {
      imprimir(x-1);
      System.out.println(x);
   }
}
```

Cuando x vale 0 la condición del if se valúa como falsa y sale de la función imprimir.
¿Qué línea ahora se ejecuta ?
Vuelve a la función main ? NO.

Recordemos que la última llamada de la función imprimir se había hecho desde la misma función imprimir por lo que vuelve a la línea:

```
System.out.println(x);
```

Ahora si analicemos que valor tiene el parámetro x. Observemos la pila de llamadas del gráfico:



x cuarta tiene el valor 1. Por lo que se imprime dicho valor en pantalla.
Luego de imprimir el 1 finaliza la ejecución de la función, se libera el espacio ocupado por el parámetro x y pasa a ejecutarse la siguiente línea donde se había llamado la función:

```
System.out.println(x);
```

Ahora x en esta instancia de la función tiene el valor 2.
Así sucesivamente hasta liberar todas las llamadas recursivas.

Es importante tener en cuenta que siempre en una función recursiva debe haber un if para finalizar la recursividad (en caso contrario la función recursiva será infinita y provocará que el programa se bloquee)

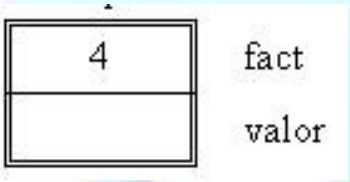
Problema 5:

Otro problema típico que se presenta para analizar la recursividad es el obtener el factorial de un número.
Recordar que el factorial de un número es el resultado que se obtiene de multiplicar dicho número por el anterior y así sucesivamente hasta llegar a uno.
Ej. el factorial de 4 es 4 * 3 * 2 * 1 es decir 24.

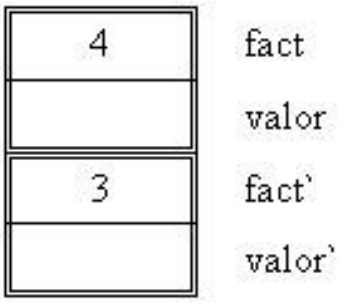
Programa:

```
public class Recursividad {

   int factorial(int fact) {
      if (fact>0) {
         int valor=fact * factorial(fact-1);
         return valor;
      } else
         return 1;
   }

   public static void main(String[] ar) {
      Recursividad re=new Recursividad();
      int f=re.factorial(4);
      System.out.println("El factorial de 4 es "+f);
   }
}
```

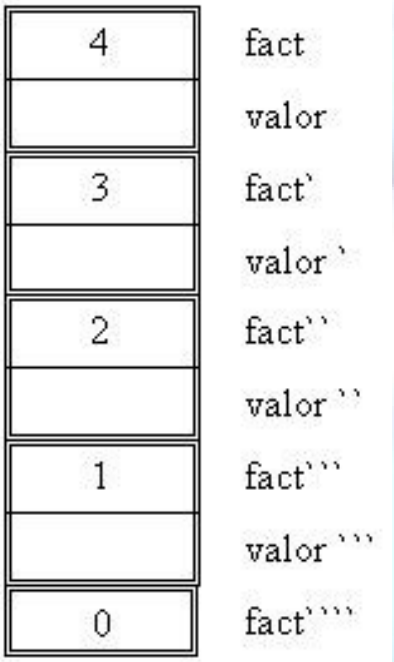
La función factorial es recursiva porque desde la misma función llamamos a la función factorial.
Debemos hacer el seguimiento del problema para analizar como se calcula.
La memoria en la primera llamada:



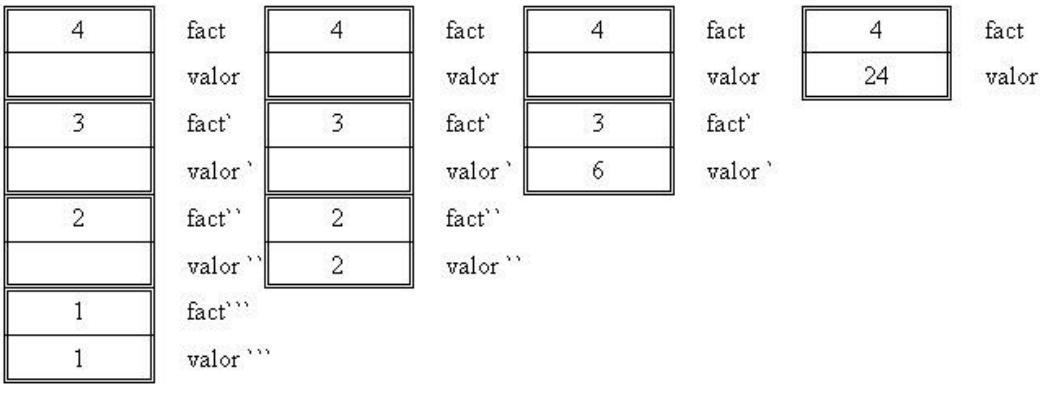
fact recibe el valor 4 y valor se cargará con el valor que se obtenga con el producto de fact por el valor devuelto por la función factorial (llamada recursiva)



Nuevamente se llama recursivamente hasta que el parámetro fact reciba el valor 0.



Cuando fact recibe un cero la condición del if se valúa como falsa y ejecuta el else retornando un 1, la variable local de la llamada anterior a la función queda de la siguiente manera:



Es importantísimo entender la liberación del espacio de las variables locales y los parámetros en las sucesivas llamadas recursivas.
Por último la función main recibe "valor", en este caso el valor 24.

Problema 6:

Implementar un método recursivo para ordenar los elementos de un vector.

Programa:

```
class Recursivdad {
   static int [] vec = {312, 614, 88, 22, 54};

   void ordenar (int [] v, int cant) {
      if (cant > 1) {
         for (int f = 0 ; f < cant - 1 ; f++)
            if (v [f] > v [f + 1]) {
               int aux = v [f];
               v [f] = v [f + 1];
               v [f + 1] = aux;
            }
         ordenar (v, cant - 1);
      }
   }

   void imprimir () {
      for (int f = 0 ; f < vec.length ; f++)
         System.out.print (vec [f] + "  ");
      System.out.println("\n");
   }


   public static void main (String [] ar)  {
      Recursivdad r = new Recursivdad();
      r.imprimir ();
      r.ordenar (vec, vec.length);
      r.imprimir ();
   }
}
```

Hasta ahora hemos visto problemas que se pueden resolver tanto con recursividad como con estructuras repetitivas.
Es muy importante tener en cuenta que siempre que podamos emplear un algoritmo no recursivo será mejor (ocupa menos memoria de ram y se ejecuta más rápidamente)
Pero hay casos donde el empleo de recursividad hace mucho más sencillo el algoritmo (tener en cuenta que no es el caso de los tres problemas vistos previamente)

Recursividad: Problemas donde conviene aplicar la recursividad

En el concepto anterior se vieron pequeños problemas para entender como funciona la recursividad, pero no se desarrollaron problemas donde conviene utilizar la recursividad.

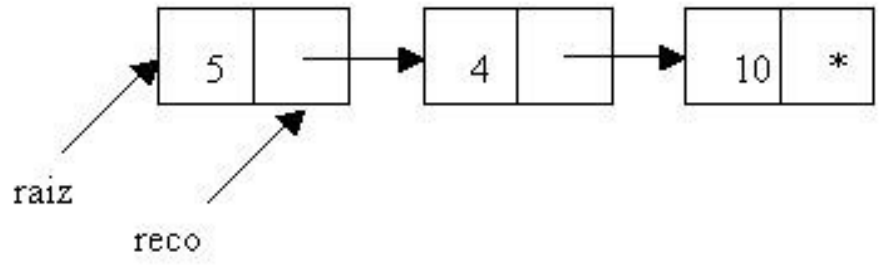
Problema 1:

Imprimir la información de una lista simplemente encadenada de atrás para adelante.
El empleo de estructuras repetitivas para resolver este problema es bastante engorroso y lento (debemos avanzar hasta el último nodo e imprimir, luego avanzar desde el principio hasta el anteúltimo nodo y así sucesivamente)
El empleo de la recursividad para este problema hace más sencillo su solución.

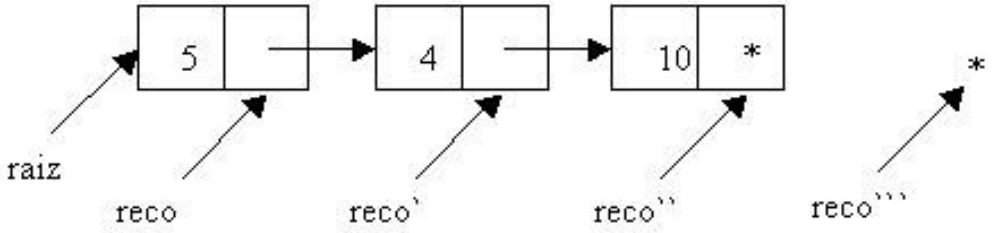
Programa:

```
public class Recursividad {

   class Nodo {
      int info;
      Nodo sig;
   }

   private Nodo raiz;


   void insertarPrimero(int x)
   {
        Nodo nuevo = new Nodo ();
      nuevo.info = x;
      nuevo.sig=raiz;
      raiz=nuevo;
   }


   public void imprimirInversa(Nodo reco) {
      if (reco!=null) {
         imprimirInversa(reco.sig);
         System.out.print(reco.info+"-");
      }
   }

   public void imprimirInversa () {
      imprimirInversa(raiz);
   }

   public static void main(String[] ar) {
      Recursividad r=new Recursividad();
      r.insertarPrimero (10);
      r.insertarPrimero(4);
```

```
        r.insertarPrimero(5);
        r.imprimirInversa();
    }
}
```



Cuando llamamos al método recursivo le enviamos raiz y el parámetro reco recibe esta dirección. Si reco es distinto a null llamamos recursivamente al método enviándole la dirección del puntero sig del nodo.
Por lo que el parámetro reco recibe la dirección del segundo nodo.



Podemos observar como en las distintas llamadas recursivas el parámetro reco apunta a un nodo. Cuando se van desapilando las llamadas recursivas se imprime primeramente el 10 luego el 4 y por último el 5.

Problema 2:

Recorrer un árbol de directorios en forma recursiva.

Programa:

```
import java.io.File;
public class Recursividad{

   public void leer(String inicio,String altura)
   {
      File ar=new File(inicio);
      String[] dir=ar.list();
      for(int f=0;f<dir.length;f++){
         File ar2=new File(inicio+dir[f]);
         if (ar2.isFile())
            System.out.println(altura+dir[f]);
         if (ar2.isDirectory()) {
           System.out.println(altura + "Directorio:"+dir[f].toUpperCase());
           leer(inicio+dir[f]+"\\",altura+"  ");
         }
```

```
    }
  }

  public static void main(String[] arguments)
  {
    Recursividad rec=new Recursividad();
    rec.leer("d:\\windows\\","");
  }
}
```

Para recorrer y visitar todos los directorios y archivos de un directorio debemos implementar un algoritmo recursivo que reciba como parámetro el directorio inicial donde comenzaremos a recorrer:

```
  public void leer(String inicio,String altura)
```

Creamos un objeto de la clase File con el directorio que llega como parámetro y mediante el método list obtenemos todos los archivos y directorios de dicho directorio:

```
    File ar=new File(inicio);
    String[] dir=ar.list();
```

Mediante un for recorremos todo el vector que contiene la lista de archivos y directorios:

```
    for(int f=0;f<dir.length;f++){
```

Creamos un objeto de la clase File para cada directorio y archivo:

```
      File ar2=new File(inicio+dir[f]);
```

Luego de crear un objeto de la clase file podemos verificar si se trata de un archivo o directorio:

```
      if (ar2.isFile())
         System.out.println(altura+dir[f]);
      if (ar2.isDirectory()) {
        System.out.println(altura + "Directorio:"+dir[f].toUpperCase());
        leer(inicio+dir[f]+"\\",altura+"  ");
      }
```

Si es un archivo lo mostramos y si es un directorio además de mostrarlo llamamos recursivamente al método leer con el directorios nuevo a procesar.

Problema 3:

Desarrollar un programa que permita recorrer un laberinto e indique si tiene salida o no.
Para resolver este problema al laberinto lo representaremos con una matriz de 10 x 10 JLabel.

El valor:

"0"	Representa pasillo
"1"	Representa pared
"9"	Persona
"s"	Salida

A la salida ubicarla en la componente de la fila 9 y columna 9 de la matriz. La persona comienza a recorrer el laberinto en la fila 0 y columna 0. Los ceros y unos disponerlos en forma aleatoria (con la función random)

Programa:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class Laberinto extends JFrame implements ActionListener {
   JLabel[][] l;
   JButton b1;
   JButton b2;
   boolean salida;
   Laberinto()
   {
      setLayout(null);
      l=new JLabel[10][10];
      for(int f=0;f<10;f++) {
         for(int c=0;c<10;c++) {
            l[f][c]=new JLabel();
            l[f][c].setBounds(20+c*20,50+f*20,20,20);
            add(l[f][c]);
         }
      }
      b1=new JButton("Recorrer");
      b1.setBounds(10,300,100,25);
      add(b1);
      b1.addActionListener(this);
      b2=new JButton("Crear");
      b2.setBounds(120,300,100,25);
      add(b2);
      b2.addActionListener(this);
      crear();
   }

   public void crear()
   {
      for(int f=0;f<10;f++) {
         for(int c=0;c<10;c++) {
            int a=(int)(Math.random()*4);
            l[f][c].setForeground(Color.black);
            if (a==0)
               l[f][c].setText("1");
            else
               l[f][c].setText("0");
         }
      }
      l[9][9].setText("s");
```

```
      l[0][0].setText("0");
   }

   public void recorrer(int fil,int col)
   {
      if (fil>=0 && fil<10 && col>=0 && col<10 && salida==false) {
         if (l[fil][col].getText().equals("s"))
            salida=true;
         else
            if (l[fil][col].getText().equals("0")) {
               l[fil][col].setText("9");
               l[fil][col].setForeground(Color.red);
               recorrer(fil,col+1);
               recorrer(fil+1,col);
               recorrer(fil-1,col);
               recorrer(fil,col-1);
            }
      }
   }

   public void actionPerformed(ActionEvent e)
   {
      if (e.getSource()==b1) {
         salida=false;
         recorrer(0,0);
         if (salida)
            setTitle("tiene salida");
         else
            setTitle("no tiene salida");
      }
      if (e.getSource()==b2)
         crear();
   }

   public static void main(String[] ar)
   {
      Laberinto l=new Laberinto();
      l.setBounds(0,0,300,400);
      l.setVisible(true);
   }
}
```

El método más importante es el recorrer:

```
   public void recorrer(int fil,int col)
```

Primero verificamos si la coordenada a procesar del laberinto se encuentra dentro de los límites correctos y además no hayamos encontrado la salida hasta el momento:

```
      if (fil>=0 && fil<10 && col>=0 && col<10 && salida==false)
```

Si entra al if anterior verificamos si estamos en la salida:

```
if (l[fil][col].getText().equals("s"))
   salida=true;
```

En el caso que no estemos en la salida verificamos si estamos en pasillo:

```
if (l[fil][col].getText().equals("0")) {
```

En caso de estar en el pasillo procedemos a fijar dicha JLabel con el caracter "9" e intentamos desplazarnos en las cuatro direcciones (arriba, abajo, derecha e izquierda), este desplazamiento lo logramos llamando recursivamente:

```
l[fil][col].setText("9");
l[fil][col].setForeground(Color.red);
recorrer(fil,col+1);
recorrer(fil+1,col);
recorrer(fil-1,col);
recorrer(fil,col-1);
```

Problemas propuestos

1. Desarrollar el juego del Buscaminas. Definir una matriz de 10*10 de JButton y disponer una 'b' para las bombas (10 diez) un cero en los botones que no tienen bombas en su perímetro, un 1 si tiene una bomba en su perímetro y así sucesivamente. Cuando se presiona un botón si hay un cero proceder en forma recursiva a destapar los botones que se encuentran a sus lados. Disponer el mismo color de frente y fondo de los botones para que el jugador no pueda ver si hay bombas o no.

Solución

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class Buscaminas extends JFrame implements ActionListener
{
   JButton [] [] bot;
   JButton b1;
   Buscaminas ()
   {
      setLayout (null);
      bot = new JButton [10] [10];
      for (int f = 0 ; f < 10 ; f++)
      {
         for (int c = 0 ; c < 10 ; c++)
         {
            bot [f] [c] = new JButton ("0");
            bot [f] [c].setBounds (20 + c * 41, 50 + f * 41, 41, 41);
            bot [f] [c].setBackground (Color.lightGray);
            bot [f] [c].setForeground (Color.lightGray);
            bot [f] [c].addActionListener (this);
            add (bot [f] [c]);
         }
      }
      b1 = new JButton ("Reiniciar");
      b1.setBounds (20, 470, 100, 30);
      add (b1);
      b1.addActionListener (this);
      disponerBombas ();
      contarBombasPerimetro ();
   }


   void disponerBombas ()
   {
      int cantidad = 10;
      do
      {
         int fila = (int) (Math.random () * 10);
         int columna = (int) (Math.random () * 10);
         if (bot [fila] [columna].getText ().equals ("b") == false)
         {
            bot [fila] [columna].setText ("b");
            cantidad--;
         }
      }
      while (cantidad != 0);
```

```
}


void contarBombasPerimetro ()
{
   for (int f = 0 ; f < 10 ; f++)
   {
      for (int c = 0 ; c < 10 ; c++)
      {
         if (bot [f] [c].getText ().equals ("0") == true)
         {
            int cant = contarCoordenada (f, c);
            bot [f] [c].setText (String.valueOf (cant));
         }
      }
   }
}


int contarCoordenada (int fila, int columna)
{
   int total = 0;
   if (fila - 1 >= 0 && columna - 1 >= 0)
   {
      if (bot [fila - 1] [columna - 1].getText ().equals ("b") == true)
      total++;
   }
   if (fila - 1 >= 0)
   {
      if (bot [fila - 1] [columna].getText ().equals ("b") == true)
      total++;
   }
   if (fila - 1 >= 0 && columna + 1 < 10)
   {
      if (bot [fila - 1] [columna + 1].getText ().equals ("b") == true)
      total++;
   }

   if (columna + 1 < 10)
   {
      if (bot [fila] [columna + 1].getText ().equals ("b") == true)
      total++;
   }
   if (fila + 1 < 10 && columna + 1 < 10)
   {
      if (bot [fila + 1] [columna + 1].getText ().equals ("b") == true)
      total++;
   }
```

```
   if (fila + 1 < 10)
   {
      if (bot [fila + 1] [columna].getText ().equals ("b") == true)
      total++;
   }
   if (fila + 1 < 10 && columna - 1 >= 0)
   {
      if (bot [fila + 1] [columna - 1].getText ().equals ("b") == true)
      total++;
   }
   if (columna - 1 >= 0)
   {
      if (bot [fila] [columna - 1].getText ().equals ("b") == true)
      total++;
   }
   return total;

}


void desactivarJuego ()
{
   for (int f = 0 ; f < 10 ; f++)
   {
      for (int c = 0 ; c < 10 ; c++)
      {
         bot [f] [c].setEnabled (false);
      }
   }
}


void reiniciar ()
{
   setTitle ("");
   for (int f = 0 ; f < 10 ; f++)
   {
      for (int c = 0 ; c < 10 ; c++)
      {
         bot [f] [c].setText ("0");
         bot [f] [c].setEnabled (true);
         bot [f] [c].setBackground (Color.lightGray);
         bot [f] [c].setForeground (Color.lightGray);
      }
   }
   disponerBombas ();
   contarBombasPerimetro ();
```

```
}


public void actionPerformed (ActionEvent e)
{
   if (e.getSource () == b1)
   {
      reiniciar ();
   }
   for (int f = 0 ; f < 10 ; f++)
   {
      for (int c = 0 ; c < 10 ; c++)
      {
      if (e.getSource () == bot [f] [c])
      {
         if (bot [f] [c].getText ().equals ("b") == true)
         {
            setTitle ("Boooooooooooooomm");
            desactivarJuego ();
         }
         else
         if (bot [f] [c].getText ().equals ("0") == true)
         {
            recorrer (f, c);
         }
         else
            if (bot [f] [c].getText ().equals ("1") == true ||
               bot [f] [c].getText ().equals ("2") == true ||
               bot [f] [c].getText ().equals ("3") == true ||
               bot [f] [c].getText ().equals ("4") == true ||
               bot [f] [c].getText ().equals ("5") == true ||
               bot [f] [c].getText ().equals ("6") == true ||
               bot [f] [c].getText ().equals ("7") == true ||
               bot [f] [c].getText ().equals ("8") == true)
            {
               bot [f] [c].setBackground (Color.yellow);
               bot [f] [c].setForeground (Color.black);
            }
      }
      }
   }
   verificarTriunfo ();
}


void verificarTriunfo ()
{
   int cant = 0;
```

```
   for (int f = 0 ; f < 10 ; f++)
   {
      for (int c = 0 ; c < 10 ; c++)
      {
         Color col = bot [f] [c].getBackground ();
         if (col == Color.yellow)
            cant++;
      }
   }
   if (cant == 90)
   {
      setTitle ("Ganooooooooo");
      desactivarJuego ();
   }
}


void recorrer (int fil, int col)
{
   if (fil >= 0 && fil < 10 && col >= 0 && col < 10)
   {
      if (bot [fil] [col].getText ().equals ("0"))
      {
         bot [fil] [col].setText (" ");
         bot [fil] [col].setBackground (Color.yellow);
         recorrer (fil, col + 1);
         recorrer (fil, col - 1);
         recorrer (fil + 1, col);
         recorrer (fil - 1, col);
         recorrer (fil - 1, col - 1);
         recorrer (fil - 1, col + 1);
         recorrer (fil + 1, col + 1);
         recorrer (fil + 1, col - 1);
      }
      else
         if (bot [fil] [col].getText ().equals ("1") == true ||
            bot [fil] [col].getText ().equals ("2") == true ||
            bot [fil] [col].getText ().equals ("3") == true ||
            bot [fil] [col].getText ().equals ("4") == true ||
            bot [fil] [col].getText ().equals ("5") == true ||
            bot [fil] [col].getText ().equals ("6") == true ||
            bot [fil] [col].getText ().equals ("7") == true ||
            bot [fil] [col].getText ().equals ("8") == true)
         {
            bot [fil] [col].setBackground (Color.yellow);
            bot [fil] [col].setForeground (Color.black);
         }
   }
```

```
    }

    public static void main (String [] ar)
    {
        Buscaminas m = new Buscaminas ();
        m.setBounds (0, 0, 470, 600);
        m.setVisible(true);
    }
}
```

Muchas gracias hasta la próxima clase.

Alsina 16 [B1642FNB] San Isidro | Pcia. De Buenos Aires |Argentina |

TEL.: [011] 4742-1532 o [011] 4742-1665 |

www.institutosanisidro.com.ar info@institutosanisidro.com.ar