INSTITUTO SAN ISIDRO

Curso de Java a distancia

Clase 14: Estructuras dinámicas

Conocemos algunas estructuras de datos como son los vectores y matrices. No son las únicas. Hay muchas situaciones donde utilizar alguna de estas estructuras nos proporcionará una solución muy ineficiente (cantidad de espacio que ocupa en memoria, velocidad de acceso a la información, etc.)

Ejemplo 1. Imaginemos que debemos realizar un procesador de texto, debemos elegir la estructura de datos para almacenar en memoria las distintas líneas que el operador irá tipeando. Una solución factible es utilizar una matriz de caracteres. Pero como sabemos debemos especificar la cantidad de filas y columnas que ocupará de antemano. Podría ser por ejemplo 2000 filas y 200 columnas. Con esta definición estamos reservando de antemano 800000 bytes de la memoria, no importa si el operador después carga una línea con 20 caracteres, igualmente ya se ha reservado una cantidad de espacio que permanecerá ociosa.

Tiene que existir alguna estructura de datos que pueda hacer más eficiente la solución del problema anterior.

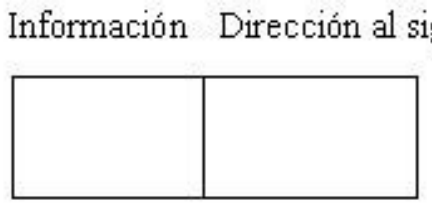
Ejemplo 2. ¿Cómo estarán codificadas las planillas de cálculo? ¿Reservarán espacio para cada casilla de la planilla al principio? Si no la lleno, ¿lo mismo se habrá reservado espacio? Utilizar una matriz para almacenar todas las casillas de una planilla de cálculo seguro será ineficiente.

Bien, todos estos problemas y muchos más podrán ser resueltos en forma eficiente cuando conozcamos estas nuevas estructuras de datos (Listas, árboles)

Estructuras dinámicas: Listas

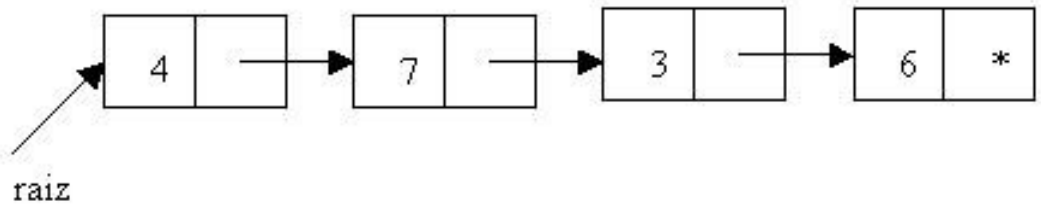
Una lista es un conjunto de nodos, cada uno de los cuales tiene dos campos: uno de información y un apuntador al siguiente nodo de la lista. Además un apuntador externo señala el primer nodo de la lista.

Representación gráfica de un nodo:



La información puede ser cualquier tipo de dato simple, estructura de datos o inclusive uno o más objetos.
La dirección al siguiente nodo es un puntero.

Representación gráfica de una lista:

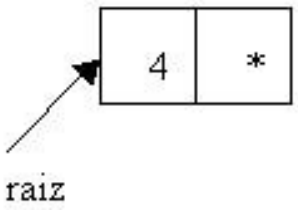


Como decíamos, una lista es una secuencia de nodos (en este caso cuatro nodos). La información de los nodos en este caso es un entero y siempre contiene un puntero que guarda la dirección del siguiente nodo.
raiz es otro puntero externo a la lista que contiene la dirección del primer nodo.

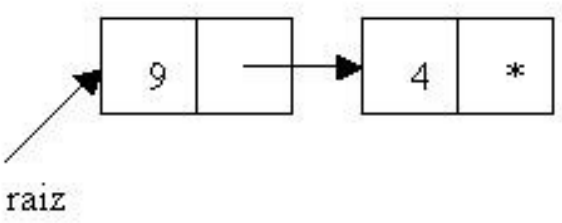
El estado de una lista varía durante la ejecución del programa:

De esta forma representamos gráficamente una lista vacía.

Si insertamos un nodo en la lista quedaría luego:



Si insertamos otro nodo al principio con el valor 9 tenemos:



Lo mismo podemos borrar nodos de cualquier parte de la lista.
Esto nos trae a la mente el primer problema planteado: el desarrollo del procesador de texto. Podríamos utilizar una lista que inicialmente estuviera vacía e introdujéramos un nuevo nodo con cada línea que tipea el operador. Con esta estructura haremos un uso muy eficiente de la memoria.

TIPOS DE LISTAS.

Según el mecanismo de inserción y extracción de nodos en la lista tenemos los siguientes tipos:

- Listas tipo pila.
- Listas tipo cola.
- Listas genéricas.

Una lista se comporta como una pila si las inserciones y extracciones las hacemos por un mismo lado de la lista. También se las llama listas LIFO (Last In First Out - último en entrar primero en salir)

Una lista se comporta como una cola si las inserciones las hacemos al final y las extracciones las hacemos por el frente de la lista. También se las llama listas FIFO (First In First Out - primero en entrar primero en salir)

Una lista se comporta como genérica cuando las inserciones y extracciones se realizan en cualquier parte de la lista.
Podemos en algún momento insertar un nodo en medio de la lista, en otro momento al final, borrar uno del frente, borrar uno del fondo o uno interior, etc.

Estructuras dinámicas: Listas tipo Pila

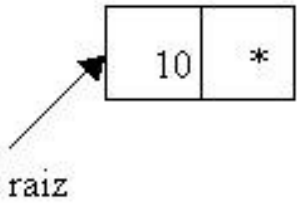
Una lista se comporta como una pila si las inserciones y extracciones las hacemos por un mismo lado de la lista. También se las llama listas LIFO (Last In First Out - último en entrar primero en salir)

Importante: Una pila al ser una lista puede almacenar en el campo de información cualquier tipo de valor (int, char, float, vector de caracteres, un objeto, etc)

Para estudiar el mecanismo de utilización de una pila supondremos que en el campo de información almacena un entero (para una fácil interpretación y codificación)

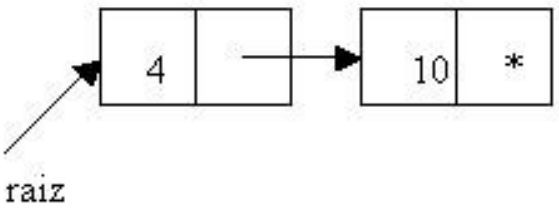
Inicialmente la PILA está vacía y decimos que el puntero raiz apunta a null (Si apunta a null decimos que no tiene una dirección de memoria):

Insertamos un valor entero en la pila: insertar(10)



Luego de realizar la inserción la lista tipo pila queda de esta manera: un nodo con el valor 10 y raiz apunta a dicho nodo. El puntero del nodo apunta a null ya que no hay otro nodo después de este.

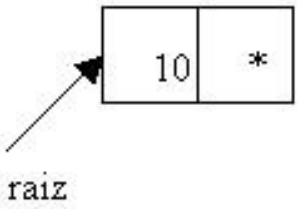
Insertamos luego el valor 4: insertar(4)



Ahora el primer nodo de la pila es el que almacena el valor cuatro. raiz apunta a dicho nodo. Recordemos que raiz es el puntero externo a la lista que almacena la dirección del primer nodo. El nodo que acabamos de insertar en el campo puntero guarda la dirección del nodo que almacena el valor 10.

Ahora qué sucede si extraemos un nodo de la pila. ¿Cuál se extrae? Como sabemos en una pila se extrae el último en entrar.

Al extraer de la pila tenemos: extraer()



La pila ha quedado con un nodo.
Hay que tener cuidado que si se extrae un nuevo nodo la pila quedará vacía y no se podrá extraer otros valores (avisar que la pila está vacía)

PROBLEMA 1:

Confeccionar una clase que administre una lista tipo pila (se debe poder insertar, extraer e imprimir los datos de la pila)

PROGRAMA:

```
public class Pila {


  class Nodo {
     int info;
     Nodo sig;
  }


  private Nodo raiz;


  public Pila () {
     raiz=null;
  }


  public void insertar(int x) {
        Nodo nuevo;
     nuevo = new Nodo();
```

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

public int extraer ()
{
   if (raiz!=null)
   {
      int informacion = raiz.info;
      raiz = raiz.sig;
      return informacion;
   }
   else
   {
      return Integer.MAX_VALUE;
   }
```

```
    }

    public void imprimir() {
        Nodo reco=raiz;
        System.out.println("Listado de todos los elementos de la pila.");
        while (reco!=null) {
            System.out.print(reco.info+"-");
            reco=reco.sig;
        }
        System.out.println();
    }

    public static void main(String[] ar) {
        Pila pila1=new Pila();
        pila1.insertar(10);
        pila1.insertar(40);
        pila1.insertar(3);
        pila1.imprimir();
        System.out.println("Extraemos de la pila:"+pila1.extraer());
        pila1.imprimir();
    }
}
```

Analicemos las distintas partes de este programa:

```
    class Nodo {
        int info;
```

```
    Nodo sig;
  }


  private Nodo raiz;
```

Para declarar un nodo debemos utilizar una clase. En este caso la información del nodo (info) es un entero y siempre el nodo tendrá una referencia de tipo Nodo, que le llamamos sig.
El puntero sig apunta al siguiente nodo o a null en caso que no exista otro nodo. Este puntero es interno a la lista.
También definimos un puntero de tipo Nodo llamado raiz. Este puntero tiene la dirección del primer nodo de la lista. En caso de estar vacía la lista, raiz apunta a null (es decir no tiene dirección)

El puntero raiz es fundamental porque al tener la dirección del primer nodo de la lista nos permite acceder a los demás nodos.

```
  public Pila () {
    raiz=null;
  }
```

En el constructor de la clase hacemos que raiz guarde el valor null. Tengamos en cuenta que si raiz tiene almacenado null la lista está vacía, en caso contrario tiene la dirección del primer nodo de la lista.

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

```
            nuevo.sig = raiz;

            raiz = nuevo;

        }

    }
```

Uno de los métodos más importantes que debemos entender en una pila es el de insertar un elemento en la pila.
Al método llega la información a insertar, en este caso en particular es un valor entero.

La creación de un nodo requiere dos pasos:

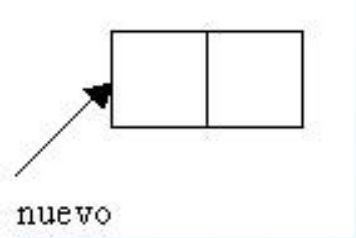
- Definición de un puntero o referencia a un tipo de dato Nodo:

```
Nodo nuevo;
```

- Creación del nodo (creación de un objeto):

```
nuevo = new Nodo();
```

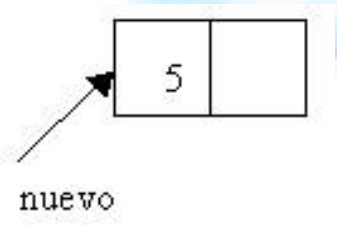
Cuando se ejecuta el operador new se reserva espacio para el nodo. Realmente se crea el nodo cuando se ejecuta el new.



Paso seguido debemos guardar la información del nodo:

```
nuevo.info = x;
```

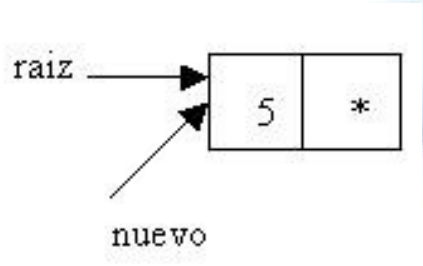
En el campo info almacenamos lo que llega en el parámetro x. Por ejemplo si llega un 5 el nodo queda:



Por último queda enlazar el nodo que acabamos de crear al principio de la lista.
Si la lista está vacía debemos guardar en el campo sig del nodo el valor null para indicar que

no hay otro nodo después de este, y hacer que raiz apunte al nodo creado (sabemos si una lista esta vacía si raiz almacena un null)

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



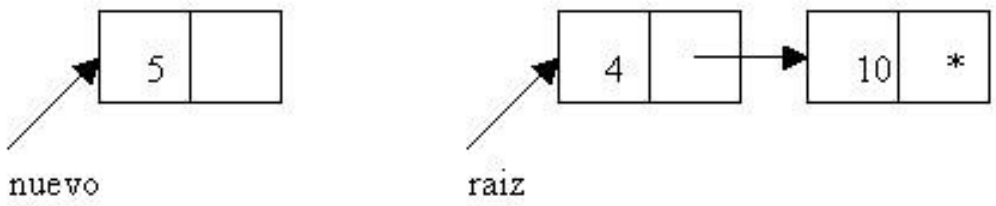
Gráficamente podemos observar que cuando indicamos raiz=nuevo, el puntero raiz guarda la dirección del nodo apuntado por nuevo.
Tener en cuenta que cuando finaliza la ejecución del método el puntero nuevo desaparece, pero no el nodo creado con el operador new.

En caso que la lista no esté vacía, el puntero sig del nodo que acabamos de crear debe apuntar al que es hasta este momento el primer nodo, es decir al nodo que apunta raiz actualmente.

```
else
{
   nuevo.sig = raiz;
   raiz = nuevo;
}
```

Como primera actividad cargamos en el puntero sig del nodo apuntado por nuevo la dirección de raiz, y posteriormente raiz apunta al nodo que acabamos de crear, que será ahora el primero de la lista.

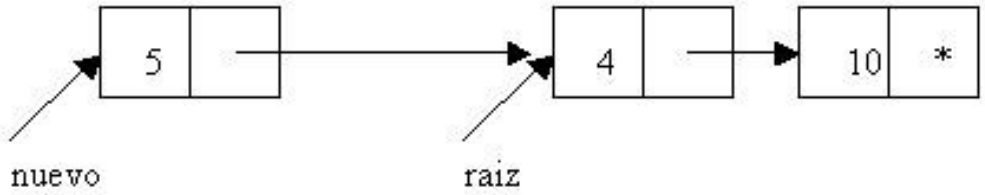
Antes de los enlaces tenemos:



Luego de ejecutar la línea:

```
nuevo.sig = raiz;
```

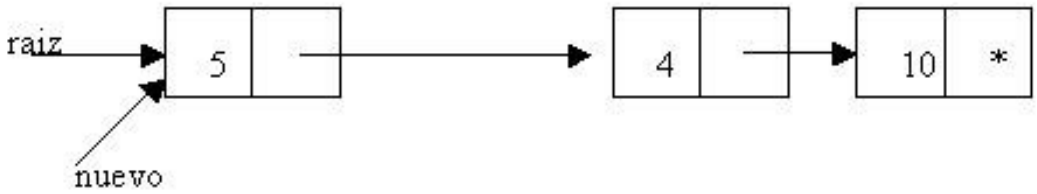
Ahora tenemos:



Por último asignamos a raiz la dirección que almacena el puntero nuevo.

```
raiz = nuevo;
```

La lista queda:



El método extraer:

```
public int extraer ()
{
   if (raiz!=null)
   {
      int informacion = raiz.info;
```

```
            raiz = raiz.sig;
            return informacion;
        }
        else
        {
            return Integer.MAX_VALUE;
        }
    }
```

El objetivo del método extraer es retornar la información del primer nodo y además borrarlo de la lista.

Si la lista no está vacía guardamos en una variable local la información del primer nodo:

```
int informacion = raiz.info;
```

Avanzamos raiz al segundo nodo de la lista, ya que borraremos el primero:

```
raiz = raiz.sig;
```

el nodo que previamente estaba apuntado por raiz es eliminado automáticamente por la máquina virtual de Java, al no tener ninguna referencia.

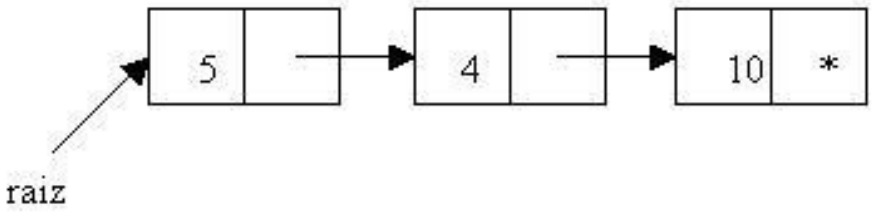
Retornamos la información:

```
return informacion;
```

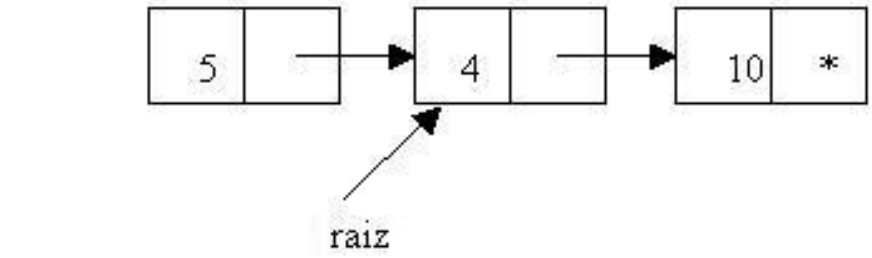
En caso de estar vacía la pila retornamos el número entero máximo y lo tomamos como código de error (es decir nunca debemos guardar el entero mayor en la pila)

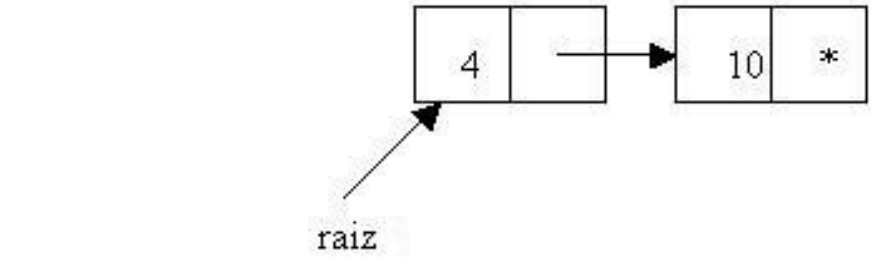
```
return Integer.MAX_VALUE;
```

Es muy importante entender gráficamente el manejo de las listas. La interpretación gráfica nos permitirá plantear inicialmente las soluciones para el manejo de listas.




```
raiz=raiz.sig;
```





Por último expliquemos el método para recorrer una lista en forma completa e imprimir la información de cada nodo:

```
public void imprimir() {
   Nodo reco=raiz;
   System.out.println("Listado de todos los elementos de la pila.");
   while (reco!=null) {
      System.out.print(reco.info+"-");
      reco=reco.sig;
   }
   System.out.println();
}
```

Definimos un puntero auxiliar reco y hacemos que apunte al primer nodo de la lista:

```
Nodo reco=raiz;
```

Disponemos una estructura repetitiva que se repetirá mientras reco sea distinto a null. Dentro de la estructura repetitiva hacemos que reco avance al siguiente nodo:

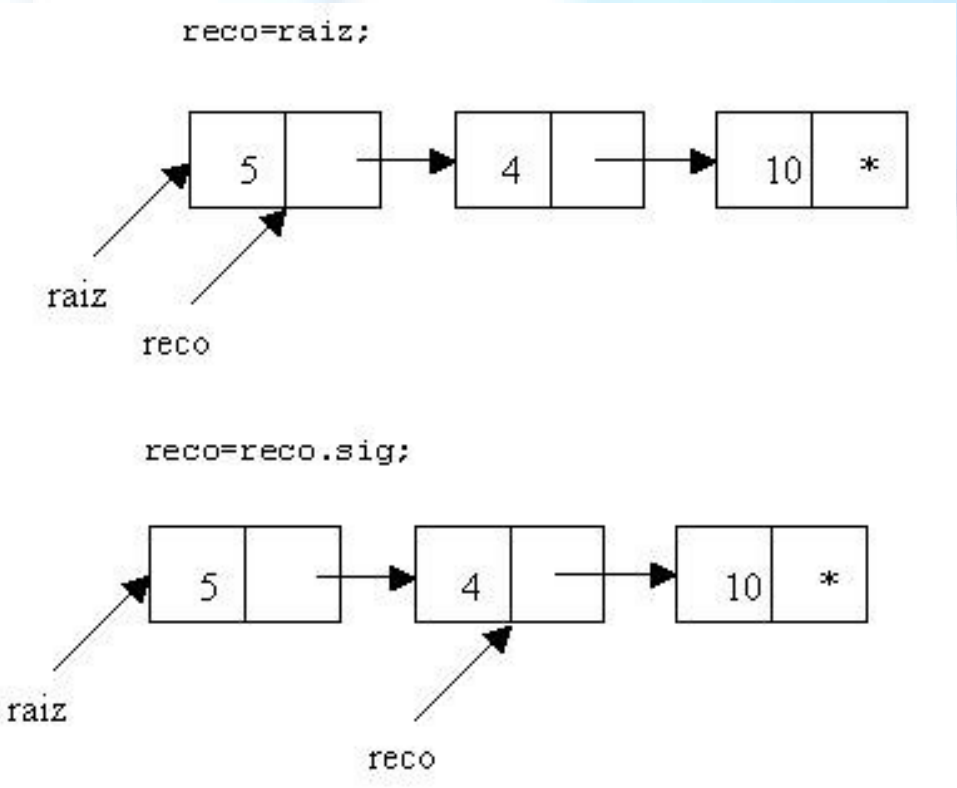
```
while (reco!=null) {
  System.out.print(reco.info+"-");
  reco=reco.sig;
}
```

Es muy importante entender la línea:

```
reco=reco.sig;
```

Estamos diciendo que reco almacena la dirección que tiene el puntero sig del nodo apuntado actualmente por reco.

Gráficamente:



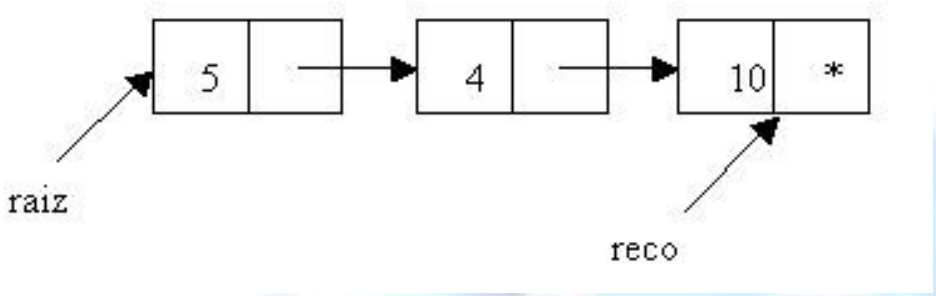
Al analizarse la condición:

```
while (reco!=null) {
```

se valúa en verdadero ya que reco apunta a un nodo y se vuelve a ejecutar la línea:

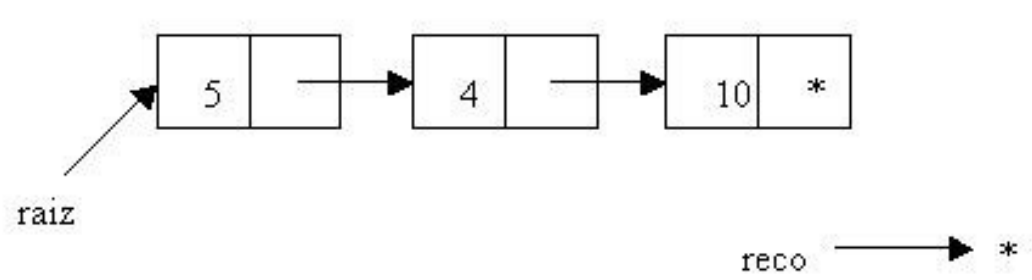
```
reco=reco.sig;
```

Ahora reco apunta al siguiente nodo:



La condición del while nuevamente se valúa en verdadera y avanza el puntero reco al siguiente nodo:

```
reco=reco.sig;
```



Ahora sí reco apunta a null y ha llegado el final de la lista (Recordar que el último nodo de la lista tiene almacenado en el puntero sig el valor null, con el objetivo de saber que es el último nodo)

Para poder probar esta clase recordemos que debemos definir un objeto de la misma y llamar a sus métodos:

```
public static void main(String[] ar) {

    Pila pila1=new Pila();

    pila1.insertar(10);

    pila1.insertar(40);

    pila1.insertar(3);

    pila1.imprimir();
```

```
    System.out.println("Extraemos de la pila:"+pila1.extraer());
    pila1.imprimir();
  }
```

Insertamos 3 enteros, luego imprimimos la pila, extraemos uno de la pila y finalmente imprimimos nuevamente la pila.

PROBLEMA 2:

Agregar a la clase Pila un método que retorne la cantidad de nodos y otro que indique si esta vacía.

PROGRAMA:

```
public class Pila {

        class Nodo {
          int info;
          Nodo sig;
        }

  private Nodo raiz;

  Pila () {
    raiz=null;
  }

  public void insertar(int x) {
        Nodo nuevo;
    nuevo = new Nodo();
    nuevo.info = x;
```

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

public int extraer ()
{
    if (raiz!=null)
    {
        int informacion = raiz.info;
        raiz = raiz.sig;
        return informacion;
    }
    else
    {
        return Integer.MAX_VALUE;
    }
}
```

```
public void imprimir() {
   Nodo reco=raiz;
   System.out.println("Listado de todos los elementos de la pila.");
   while (reco!=null) {
      System.out.print(reco.info+"-");
      reco=reco.sig;
   }
   System.out.println();
}

public boolean vacia() {
   if (raiz==null) {
      return true;
   } else {
         return false;
   }
}

public int cantidad() {
   int cant=0;
   Nodo reco=raiz;
   while (reco!=null) {
      cant++;
      reco=reco.sig;
```

```
        }
        return cant;
    }

    public static void main(String[] ar) {
        Pila pila1=new Pila();
        pila1.insertar(10);
        pila1.insertar(40);
        pila1.insertar(3);
        pila1.imprimir();
        System.out.println("La cantidad de nodos de la lista es:"+pila1.cantidad());
        while (pila1.vacia()==false) {
            System.out.println(pila1.extraer());
        }
    }
}
```

Para verificar si la pila esta vacía verificamos el contenido de la variable raiz, si tiene null luego la lista esta vacía y por lo tanto retornamos un true:

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

El algoritmo para saber la cantidad de nodos es similar al imprimir, pero en lugar de mostrar la información del nodo procedemos a incrementar un contador:

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

Para probar esta clase en la main creamos un objeto de la clase Pila insertamos tres enteros:

```
Pila pila1=new Pila();
pila1.insertar(10);
pila1.insertar(40);
pila1.insertar(3);
```

Imprimimos la pila (nos muestra los tres datos):

```
pila1.imprimir();
```

Llamamos al método cantidad (nos retorna un 3):

```
System.out.println("La cantidad de nodos de la lista es:"+pila1.cantidad());
```

Luego mientras el método vacía nos retorne un false (lista no vacía) procedemos a llamar al método extraer:

```
while (pila1.vacia()==false) {
   System.out.println(pila1.extraer());
}
```

PROBLEMAS PROPUESTOS

1. Agregar un método a la clase Pila que retorne la información del primer nodo de la Pila sin borrarlo.

```
public class Pila {


        class Nodo {

           int info;

           Nodo sig;

        }


  private Nodo raiz;


  Pila () {

     raiz=null;

  }


  public void insertar(int x) {

         Nodo nuevo;

     nuevo = new Nodo();

     nuevo.info = x;

     if (raiz==null)

     {

        nuevo.sig = null;

        raiz = nuevo;
```

```
    }
    else
    {
        nuevo.sig = raiz;
        raiz = nuevo;
    }
}

public int extraer ()
{
    if (raiz!=null)
    {
        int informacion = raiz.info;
        raiz = raiz.sig;
        return informacion;
    }
    else
    {
        return Integer.MAX_VALUE;
    }
}

public int retornar ()
{
    if (raiz!=null)
```

```
    {
        int informacion = raiz.info;
        return informacion;
    }
    else
    {
        return Integer.MAX_VALUE;
    }
}

public void imprimir() {
    Nodo reco=raiz;
    System.out.println("Listado de todos los elementos de la pila.");
    while (reco!=null) {
        System.out.print(reco.info+"-");
        reco=reco.sig;
    }
    System.out.println();
}

public static void main(String[] ar) {
    Pila pila1=new Pila();
    pila1.insertar(10);
    pila1.insertar(40);
    pila1.insertar(3);
```

```
        pila1.imprimir();
        System.out.println("Extraemos de la pila:"+pila1.extraer());
        pila1.imprimir();
        System.out.println("Retornamos primero de la pila:"+pila1.retornar());
        pila1.imprimir();
    }
}
```

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