

Laboratorio Nro. 4

Implementación de Listas Enlazadas

Objetivos

1. Implementar listas enlazadas
2. Usar herramientas para probar software
3. Utilizar listas, colas y pilas para solucionar problemas

Consideraciones Iniciales

Leer la Guía



Antes de comenzar a resolver el presente laboratorio, leer la **“Guía Metodológica para la realización y entrega de laboratorios de Estructura de Datos y Algoritmos”** que les orientará sobre los requisitos de entrega para este y todos los laboratorios, las rúbricas de calificación, el desarrollo de procedimientos, entre otros aspectos importantes.

Registrar Reclamos



En caso de tener **algún comentario** sobre la nota recibida en este u otro laboratorio, pueden **enviarlo** a través de <http://bit.ly/2q4TTKf>, el cual será atendido en la menor brevedad posible.

Traducción de Ejercicios

En el GitHub del docente, encontrarán la traducción al español de los enunciados de los Ejercicios en Línea.

**Visualización de
Calificaciones**

A través de **Eafit Interactiva** encontrarán **un enlace** que les permitirá **ver un registro de las calificaciones** que **emite el docente** para cada taller de laboratorio y según las rubricas expuestas. **Véase sección 3, numeral 3.8.**

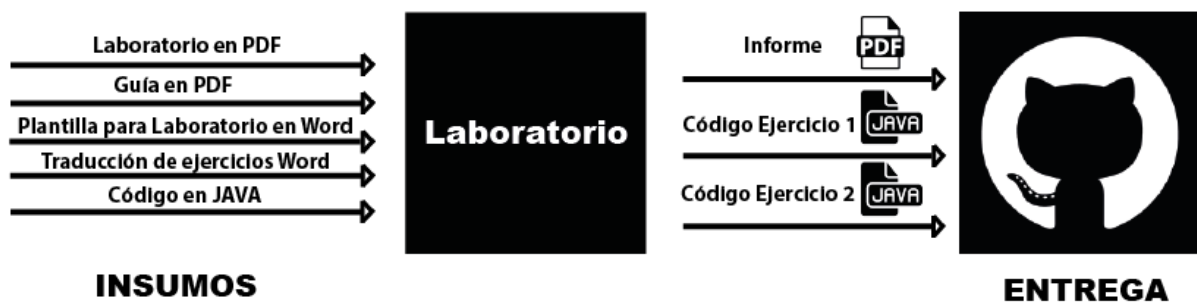
GitHub

Crear un repositorio en su cuenta de GitHub con el nombre `st0245-suCodigoAqui`. **2.** Crear una carpeta dentro de ese repositorio con el nombre `laboratorios`. **3.** Dentro de la carpeta `laboratorios`, crear una carpeta con nombre `lab04`. **4.** Dentro de la carpeta `lab04`, crear tres carpetas: `informe`, `codigo` y `ejercicioEnLinea`. **5.** Subir el informe pdf a la carpeta `infome`, el código del ejercicio 1 a la carpeta `codigo` y el código del ejercicio en línea a la carpeta `ejercicioEnLinea`. Así:

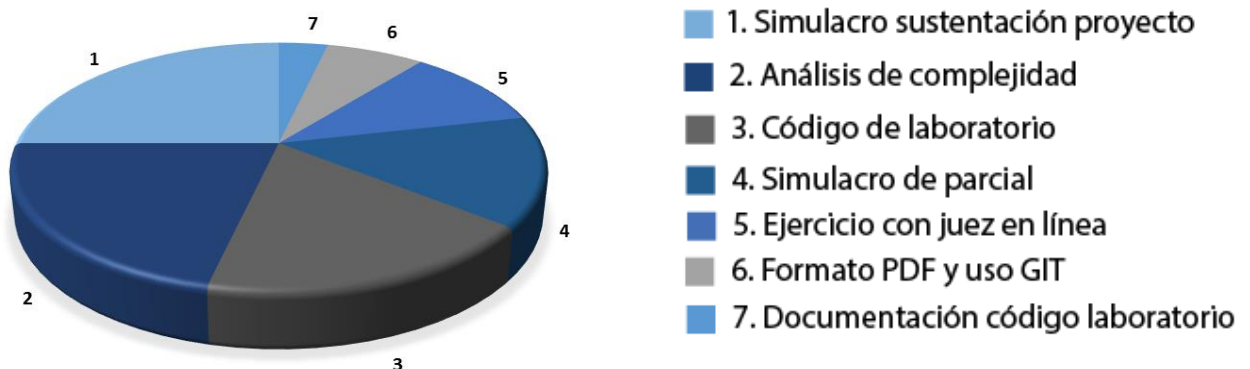
```
st0245-suCodigoAqui
  laboratorios
    lab01
      informe
      codigo
      ejercicioEnLinea
    lab02
    ...
```

Intercambio de archivos

Los archivos que **ustedes deben entregar** al docente son: **un archivo PDF** con el informe de laboratorio usando la plantilla definida, y **dos códigos**, uno con la solución al numeral 1 y otro al numeral 2 del presente. Todo lo anterior se entrega en **GitHub**.



Porcentajes y criterios de evaluación para el laboratorio



Resolver Ejercicios

1. Códigos para entregar en GitHub:



En la vida real, la documentación del software hace parte de muchos estándares de calidad como CMMI e ISO/IEC 9126



Véase Guía **en Sección 3, numeral 3.4**



Código de laboratorio en **GitHub**. Véase Guía en Sección 4, numeral 4.24



Documentación en **HTML**



No se reciben archivos en **.RAR** ni en **.ZIP**



En la vida real, las listas enlazadas se usan para representar objetos en videojuegos (Ver más en <http://bit.ly/2mcGa5w>) y para modelar pistas en juegos de rol (Ver más en <http://bit.ly/2IPyXGC>)

1.1 Implementen los métodos insertar un elemento en una posición determinada (conocido como *insert* o *add*), borrar un dato en una posición determinada (conocido como *remove*) y verificar si un dato está en la lista (conocido como *contains*) en la clase *LinkedListMauricio*, teniendo en cuenta que:

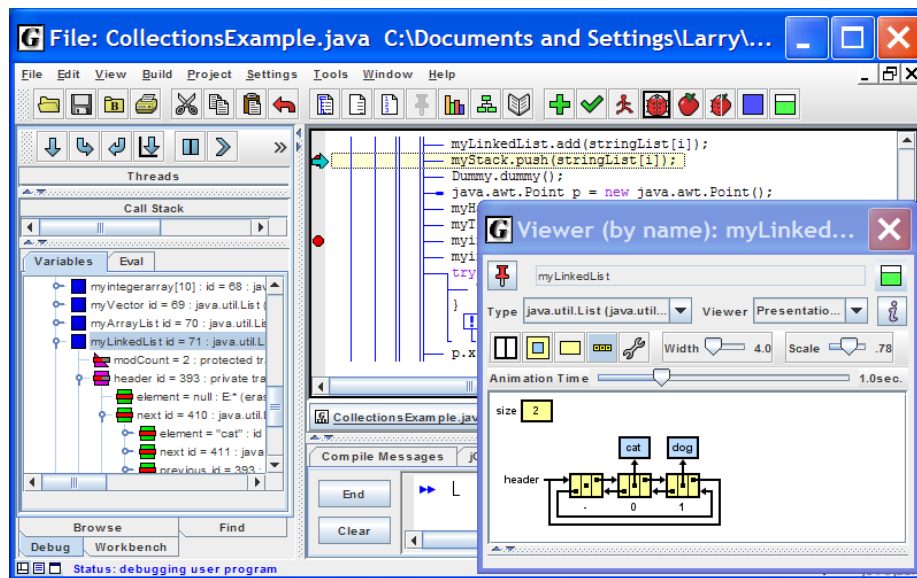
- a) Si desea un reto moderado, implemente una lista simplemente enlazada
- b) Si desea un reto mayor, implemente una lista doblemente enlazada
- c) Si desea un gran reto, implemente una lista circular doblemente enlazada



NOTA 1: Utilice el IDE Jgrasp (<http://www.jgrasp.org/>) porque tiene un depurador gráfico excelente para estructuras de datos.



NOTA 2: Véase a continuación gráfica de Jgrasp para efectos de ejemplificación



PISTA 1: Diferencien *LinkedListMauricio* de *LinkedList* del API de Java. Un error común es creer que todo se soluciona llamando los métodos existentes en *LinkedList* y, no es así, la idea es implementar una lista enlazada nosotros mismos. A continuación, un ejemplo del error:

DOCENTE MAURICIO TORO BERMÚDEZ

Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627

Correo: mtorobe@eafit.edu.co

```
// Retorna el tamaño actual de la lista
public int size()
{
    return size();
}

// Retorna el elemento en la posición index
public int get(int index)
{
    return get(index);
}

// Inserta un dato en la posición index
public void insert(int data, int index)
{
    if (index <= size())
    {
        insert(data, index);
    }
}

// Borra el dato en la posición index
public void remove(int index)
{
    remove(index);
}

// Verifica si está un dato en la lista
public boolean contains(int data)
{
    return contains(data);
}
```



PISTA 2: Otro error común es pensar que todo se soluciona usando `getNode`. Sí, `getNode` es de gran utilidad, pero ¿qué pasa si `index = 0` o si la lista está vacía? A continuación, un ejemplo de cómo no usar `getNode`.

```
// Borra el dato en la posición index
public void remove(int index)
{
    getNode(index-1).next=getNode(index+1);
}
```

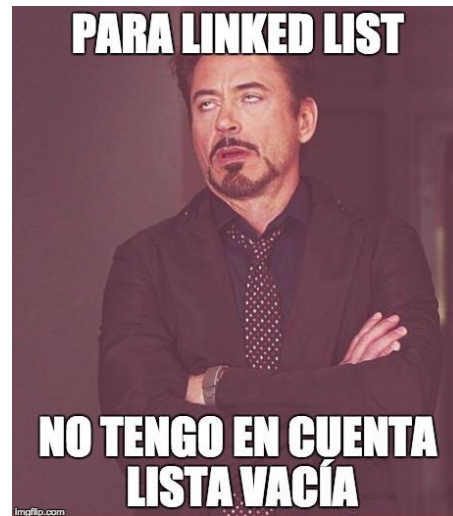


PISTA 3: Otro problema común es destruir la lista sin culpa cuando uno desea consultarla. Como un ejemplo, estos métodos de `size` calculan el tamaño pero dañan la lista, dejando la referencia `first` en `null`

```
// Malo porque daña la lista
public int size1() {
    int i = 0;
    while (first != null)
    {
        first = first.next;
        i++;
    }
    return i;
}
```

```
// Malo porque daña la lista
public int size2() {
    if (first == null)
        return 0;
    else
        first = first.next;
    return 1 + size();
}
```

Errores Comunes



Errores Comunes



En la vida real, la optimización por colas se utiliza para modelar filas de espera de bancos y otras entidades; además, para resolver problemas de logística y sistemas dinámicos

1.2 En el código entregado por el profesor está el método *get* y sus pruebas. Teniendo en cuenta esto, realicen **3 tests** unitarios para cada método. La idea es probar que su implementación de los métodos *insert* y *remove* funcionan correctamente por los menos en los siguientes casos:

- ☒ Cuando vamos a insertar/borrar y la lista está vacía,
- ☒ Cuando vamos *insertar/borrar* el primer el elemento,
- ☒ Cuando vamos a *insertar/borrar* el último elemento.



PISTA: Véase Guía en **Sección 4, numeral 4.14** “Cómo hacer pruebas unitarias en BlueJ usando JUnit” y **numeral 4.15** “Cómo compilar pruebas unitarias en Eclipse

1.3 Teniendo en cuenta lo anterior, resuelvan lo siguiente:

En un banco hay 4 filas de clientes y solamente 2 cajeros. **Se necesita simular cómo son atendidos los clientes por los cajeros.** Si lo desean, usen su implementación de listas enlazadas; de lo contrario, use la del API de Java

- Escriban un método que reciba las 4 filas de personas. No se sabe la longitud máxima que puede tener una fila de clientes. Representen las filas de clientes como mejor corresponda.
- El método debe hacer una simulación y mostrar en qué cajero (1 o 2) se atiende a cada cliente de cada fila. Coloquen el método dentro de una clase *Banco*. Impriman los datos de la simulación en pantalla.
- El cajero uno se identifica con el número 1, el cajero dos con el 2. Los clientes se identifican con su nombre.
- El orden en que se atienden los clientes en cada ronda es el siguiente: primero el de la fila 1, luego el de la fila 2, luego el de la fila 3, y finalmente el de la fila 4.

Los cajeros funcionan de la siguiente forma en cada ronda: primero el cajero 1 atiende un cliente, luego el cajero 2 atiende un cliente. Se hacen rondas hasta que no queden más clientes. Si no hay clientes en una fila, se pasa a la siguiente.

PISTAS:







- ☒ Realicen primero un dibujo de cada fila del banco y cada cajero
- ☒ Identifiquen si la fila es una lista, cola, pila o arreglo
- ☒ Identifiquen si el cajero es un número, una lista, una pila, una cola o un arreglo
- ☒ Identifiquen si va a guardar las 4 filas en una lista, pila, cola o arreglo.

```
major class Laboratory4{
    public static void simular(??? filas)  {
        ...
    }
}
```



NOTA: Todos los ejercicios del numeral 1 deben ser documentados en formato HTML. Véase *Guía en Sección 4, numeral 4.1 “Cómo escribir la documentación HTML de un código usando Javadoc”*

2) Ejercicios en línea sin documentación HTML en en GitHub

	Véase Guía en Sección 3, numeral 3.3		No entregar documentación HTML
	<i>Entregar un archivo en .JAVA</i>		No se reciben archivos en .PDF
	Resolver los problemas de CodingBat usando Recursión		Código del ejercicio en línea en GitHub . Véase Guía en Sección 4, numeral 4.24



NOTA: Recuerden que, si toman la respuesta de alguna fuente, deben referenciar según el tipo de cita correspondiente. Véase *Guía en Sección 4, numerales 4.16 y 4.17*

2.1 Resuelvan el siguiente problema usando pilas:

Antecedentes

En muchas áreas de la ingeniería de sistemas se usan dominios simples, abstractos para tanto estudios analíticos como estudios empíricos. Por ejemplo, un estudio de IA (inteligencia artificial) en plantación y robótica (STRIPS) usó un mundo de bloques en donde un brazo robótico realizaba tareas que involucraban manipulación de bloques.

En este problema usted va a modelar un mundo de bloques simple bajo ciertas reglas y restricciones. En vez de determinar cómo alcanzar un cierto estado, usted va a “programar” un brazo robótico para que responda a un cierto conjunto de comandos.

El Problema

El problema es interpretar una serie de comandos que dan instrucciones a un brazo robótico sobre como manipular bloques que están sobre una mesa plana. Inicialmente hay n bloques en la mesa (enumerados de 0 a $n-1$) con el bloque b_i adyacente al bloque b_{i+1} para todo $0 \leq i < n-1$ tal y como se muestra en el siguiente diagrama:



Figura 1: Configuración inicial de los bloques de mesa

Los comandos válidos para el brazo robótico que manipula los bloques son:

1. **move a onto b:** donde a y b son números de bloques, pone el bloque a encima del bloque b luego de devolver cualquier bloque que estén apilados sobre los bloques a y b a sus posiciones iniciales.

2. **move a over b:** donde a y b son números de bloques, pone el bloque a encima de la pila que contiene al bloque b , luego de retornar cualquier bloque que está apilado sobre el bloque a a su posición inicial.
3. **pile a onto b:** donde a y b son números de bloques, mueve la pila de bloques que consiste en el bloque a y todos los bloques apilados sobre este, encima de b . Todos los bloques encima del bloque b son movidos a su posición inicial antes de que se dé el apilamiento. Los bloques apilados sobre el bloque a conservan su orden original luego de ser movidos.
4. **pile a over b:** donde a y b son números de bloques, pone la pila de bloques que consiste en el bloque a y todos los bloques que están apilados sobre este, encima de la pila que contiene al bloque b . Los bloques apilados sobre el bloque a conservan su orden original luego de ser movidos.
5. **quit:** termina las manipulaciones en el mundo de bloques.

Cualquier comando en donde $a = b$ o en donde a y b están en la misma pila de bloques es un comando ilegal. Todo comando ilegal debe ser ignorado y no debe afectar la configuración de los bloques.

La entrada

La entrada inicia con un entero n sólo en una línea representando el número de bloques en el mundo de bloques. Asuma que $0 < n < 25$.

Este número es seguido por una secuencia de comandos de bloques, un comando por línea. Su programa debe procesar todos los comandos hasta que encuentre el comando quit.

Asuma que todos los comandos tendrán la forma especificada arriba. No se le darán comandos sintácticamente incorrectos.

La salida

La salida deberá consistir en el estado final del mundo de bloques. Cada posición original de los bloques enumerada $0 \leq i < n-1$ (donde n es el número de bloques) deberá aparecer seguida inmediatamente de dos puntos (:). Si hay por lo menos un bloque en esta posición, los dos puntos deberán estar seguidos de un espacio, seguido de una lista de bloques que aparece apilada en esa posición con el número de cada bloque separado de los demás por un espacio. No ponga espacios delante de las líneas.

Deberá imprimir una línea por cada posición (es decir, habrá n líneas de salida donde n es el entero en la primera línea de la salida)

Ejemplo de entrada

```
10
move 9 onto 1
move 8 over 1
move 7 over 1
move 6 over 1
pile 8 over 6
pile 8 over 5
move 2 over 1
move 4 over 9
quit
```

Ejemplo de salida

```
0: 0
1: 1 9 2 4
2:
3: 3
4:
5: 5 8 7 6
6:
7:
8:
9:
```



PISTA 1: Utilicen *pilas*



PISTA 2: Véase *Guía en Sección 4, numeral 4.13 “Cómo usar Scanner o BufferedReader”*



PISTA 3: Lo mejor es utilizar la técnica de diseño de modularización. Entonces, escribir un método para resolver cada uno de los siguientes problemas:

- ☒ move a onto b
- ☒ move a over b
- ☒ pile a onto b
- ☒ pile a over b

Posteriormente, se crea un mundo de bloques usando una pila para cada columna. Finalmente, se lee la entrada y se ejecuta la acción que corresponda de las 4 definidas anteriormente.

3) Simulacro de preguntas de sustentación de Proyectos



Véase Guía en **Sección 3, Numeral 3.5**



Entregar informe de laboratorio en **PDF**



Usen la **plantilla** para responder laboratorios



No apliquen Normas Icontec para esto



En la vida real, el trabajo de testing es uno de los mejor remunerados y corresponde a un Ingeniero de Sistemas

3.1 Teniendo en cuenta lo anterior, verifiquen, utilizando *JUnit*, que todos los *tests* escritos en el numeral 1.2 pasan. Muestren, en su informe de PDF, que los tests se pasan correctamente; por ejemplo, incluyendo una imagen de los resultados de los *tests*.

3.2 Expliquen con sus propias palabras cómo funciona el ejercicio en línea del numeral 2.1



NOTA: Recuerden que debe explicar su implementación en el informe PDF

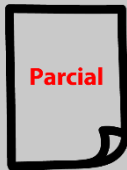
3.3 Calculen la complejidad del ejercicio en línea del numeral 2.1 y agréguela al informe PDF



PISTA: Véase *Guía en Sección 4, numeral 4.11 “Cómo escribir la complejidad de un ejercicio en línea”*

3.4 Expliquen con sus palabras las variables (qué es ‘n’, qué es ‘m’, etc.) del cálculo de complejidad del numeral 3.4

4) Simulacro de Parcial en el informe PDF



Para este simulacro, agreguen ***sus respuestas*** en el informe PDF.



El día del Parcial no tendrán computador, JAVA o acceso a internet.



PISTA: Véase ***Guía en Sección 4, Numeral 4.18*** “Respuestas del Quiz”



PISTA 2: Lea las diapositivas tituladas ***“Data Structures I: List Implementation”*** y ***“Data Structures I: Stacks and Queues”***, y encontrará la mayoría de las respuestas

1. El siguiente es un algoritmo invierte una lista usando como estructura auxiliar una pila:

```
01 public static LinkedList <String> invertir
    (LinkedList <String> lista){
02     Stack <String> auxiliar = new Stack <String>();
03     while(..... > 0){
04         auxiliar.push(lista.removeFirst());
05     }
06     while(auxiliar.size() > 0){
07         .....
08     }
09     return lista;
10 }
```

De acuerdo a lo anterior, responda las siguientes preguntas:

- a) ¿Qué condición colocaría en el ciclo while de la línea 3? (10%)

.....

- b) Complete la línea 7 de forma que el algoritmo tenga sentido (10%)

.....

2. En un banco, se desea dar prioridad a las personas de edad avanzada. El ingeniero ha propuesto un algoritmo para hacer que la persona de mayor edad en la fila quede en primer lugar.

Para implementar su algoritmo, el ingeniero utiliza colas. Las colas en Java se representan mediante la interfaz `Queue`. Una implementación de `Queue` es la clase `LinkedList`.

El método `offer` inserta en una cola y el método `poll` retira un elemento de una cola.

```
01 public Queue<Integer> organizar(  
    Queue<Integer> personas){  
02     int mayorEdad = 0;  
03     int edad;  
04     Queue<Integer> auxiliar1 =  
        new LinkedList<Integer>();  
05     Queue<Integer> auxiliar2 =  
        new LinkedList<Integer>();  
06     while(personas.size() > 0){  
07         edad = personas.poll();  
08         if(edad > mayorEdad) mayorEdad = edad;  
09         auxiliar1.offer(edad);  
10         auxiliar2.offer(edad);  
11     }  
12     while(.....){  
13         edad = auxiliar1.poll();  
14         if(edad == mayorEdad) personas.offer(edad);  
15     }
```

```
16 while(.....){
17     edad = auxiliar2.poll();
18     if(edad != mayorEdad) .....;
19 }
20 return personas;
21 }
```

a) ¿Que condiciones colocaría en los 2 ciclos `while` de líneas 12 y 16, respectivamente?

.....

b) Complete la línea 18 con el objeto y el llamado a un método, de forma que el algoritmo tenga sentido

.....

3. ¿Cuál es la complejidad asintótica, para el peor de los casos, de la función `procesarCola(q, n)`?

```
public void procesarCola(Queue q, int n)
    for (int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            q.add(j);
            //Hacer algo en O(1)
```

- a)** $O(n)$
- b)** $O(|q|)$, donde $|q|$ es el número de elementos de q
- c)** $O(n^2)$
- d)** $O(2^n)$

En Java, el método `add` agrega un elemento al comienzo de una cola.

5. [Ejercicio Opcional] Lectura recomendada



"Quienes se preparan para el ejercicio de una profesión requieren la adquisición de competencias que necesariamente se sustentan en procesos comunicativos. Así cuando se entrevista a un ingeniero recién egresado para un empleo, una buena parte de sus posibilidades radica en su capacidad de comunicación; pero se ha observado que esta es una de sus principales debilidades..."

Tomado de <http://bit.ly/2gJKzJD>



Véase Guía en **Sección 3, numeral 3.6 y 4.20** de la Guía Metodológica, "Lectura recomendada" y "Ejemplo para realización de actividades de las Lecturas Recomendadas", respectivamente

Posterior a la lectura del texto "**Robert Lafore, Data Structures and Algorithms in Java (2nd edition), 2002. Chapter 4: Stacks and Queues.**", realicen las siguientes actividades que les permitirán sumar puntos adicionales:

- a) Escriban un resumen de la lectura que tenga una longitud de 100 a 150 palabras



PISTA 1: En el siguiente enlace, unos consejos de cómo hacer un buen resumen <http://bit.ly/2knU3Pv>



PISTA 2: [Aquí](#) le explican cómo contar el número de palabras en Microsoft Word

- b) Hagan un mapa conceptual que destaque los principales elementos teóricos.



PISTA 1: Para que hagan el mapa conceptual se recomiendan herramientas como las que encuentran en <https://cacoo.com/> o <https://www.mindmup.com/#m:new-a-1437527273469>



NOTA 1: Si desean saber más sobre un tipo de cola de mucha utilidad, *las colas de prioridad*, consideren la siguiente lectura: “**Narasimha Karumanchi, Data Structures and Algorithms made easy in Java, (2nd edition)**” que pueden encontrarla en biblioteca



NOTA 2: Estas respuestas también deben incluirlas en el informe PDF

6. [Ejercicio Opcional] Trabajo en Equipo y Progreso Gradual



El trabajo en equipo es una exigencia actual del mercado. "Mientras algunos medios retratan la programación como un trabajo solitario, la realidad es que requiere de mucha comunicación y trabajo con otros. Si trabajas para una compañía, serás parte de un equipo de desarrollo y esperarán que te comuniques y trabajes bien con otras personas"

Tomado de <http://bit.ly/1B6hUDp>



Véase Guía en **Sección 3, numeral 3.7** y **Sección 4, numerales 4.21, 4.22 y 4.23** de la Guía Metodológica

- a) Entreguen copia de todas las actas de reunión usando el tablero Kanban, con fecha, hora e integrantes que participaron



PISTA: Véase *Guía en Sección 4, Numeral 4.21* “Ejemplo de cómo hacer actas de trabajo en equipo usando Tablero Kanban”

- b) Entreguen el reporte de *git*, *svn* o *mercurial* con los cambios en el código y quién hizo cada cambio, con fecha, hora e integrantes que participaron



PISTA: Véase *Guía en Sección 4, Numeral 4.23* “Cómo generar el historial de cambios en el código de un repositorio que está en *svn*”

- c) Entreguen el reporte de cambios del informe de laboratorio que se genera *Google docs* o herramientas similares



PISTA: Véase *Guía en Sección 4, Numeral 4.22* “Cómo ver el historial de revisión de un archivo en *Google Docs*”



NOTA: Estas respuestas también deben incluirlas en el informe PDF

Resumen de Ejercicios a Resolver

1.1 Implementen los métodos insertar un elemento en una posición determinada (conocido como *insert* o *add*), borrar un dato en una posición determinada (conocido como *remove*) y verificar si un dato está en la lista (conocido como *contains*) en la clase *LinkedListMauricio*, teniendo en cuenta que:

1.2. En el código entregado por el profesor está el método *get* y sus pruebas. Teniendo en cuenta esto, realicen 3 *tests* unitarios para cada método. La idea es probar que su implementación de los métodos *insert* y *remove* funcionan correctamente por los menos en los siguientes casos:

1.3 En un banco hay 4 filas de clientes y solamente 2 cajeros. **Se necesita simular cómo son atendidos los clientes por los cajeros.** Si lo desean, usen su implementación de listas enlazadas; de lo contrario, use la del API de Java

Escriban un método que reciba las 4 filas de personas

2.1 Resuelvan el siguiente problema usando pilas: <http://bit.ly/2hvOMEj>

3.1 Verifiquen, utilizando *JUnit*, que todos los *tests* escritos en el numeral 1.2 pasan. Muestren, en su informe de PDF, que los tests se pasan correctamente; por ejemplo, incluyendo una imagen de los resultados de los *tests*.

3.2 Expliquen con sus propias palabras cómo funciona el ejercicio en línea del numeral 2.1

3.3 Calculen la complejidad del ejercicio en línea del numeral 2.1 y agréguenla al informe PDF

3.4 Expliquen con sus palabras las variables (qué es 'n', qué es 'm', etc.) del cálculo de complejidad del numeral 3.4

4. Simulacro Parcial

5. Lectura recomendada **[Ejercicio Opcional]**

6. Trabajo en Equipo y Progreso Gradual **[Ejercicio Opcional]**