

Laboratorio #3

Grupo: 13

Integrantes:

- Añez Vladimirovna Leonardo Henry
- Caricari Torrejon Pedro Luis
- Mercado Oudalova Danilo Anatoli
- Mollinedo Franco Milena
- Oliva Rojas Gerson

Materia: Interacción Hombre-Computador

Fecha: 27 de enero de 2020

Porcentaje Completado: 100 %

Comentario(s): En esta práctica aprendimos a solucionar los problemas asignados de una manera más eficiente usando la interfaz List sin interacción utilizando Objects, aprendimos a usar interfaces y poder realizar los ejercicios aún más rápido, trabajamos en conjunto y nos organizamos como grupo. Realizamos programas utilizando el paradigma Orientado a Objetos (OOP).

a) Utilizando la interfaz List

Proponer e implementar al menos 10 ejercicios cualesquiera posibles, implementar utilizando únicamente los métodos de la interface List.

Ejercicio 1:

```
public static void intersection(List L1,List L2,List L3){
    for(int i=0;i<L1.size();++i){
        for(int j=0;j<L2.size();++j){
            if( L1.get(i).equals(L2.get(j))  && !L3.contains(L1.get(i))){
                L3.add(L1.get(i));
            }
        }
    }
}
```

Ejercicio 2:

```
public static boolean palindrome(List L){
    for(int i=0;i<L.size();++i){
        if(!L.get(i).equals(L.get(L.size()-1-i))){
            return false;
        }
    }
    return true;
}
```

Ejercicio 3:

```
public static void difference(List L1,List L2,List L3){
    for(int i=0;i<L1.size();++i){
        if(!L2.contains(L1.get(i))){
            L3.add(L1.get(i));
        }
    }
}
```

Ejercicio 4:

```
public static boolean complement(List U,List S,List LC){
    for(int i=0;i<U.size();++i){
        if(!S.contains(U.get(i))){
            LC.add(U.get(i));
        }
    }
}
```

```

        LC.add(U.get(i));
    }
}
}

```

Ejercicio 5:

```

public static void intercalar(List l1, List l2, List l3) {
    int i = 0 ;
    while(i< l1.size() && i< l2.size()){
        l3.add(l1.get(i));
        l3.add(l2.get(i));
        i++;
    }
    while(i<l1.size()){
        l3.add(l1.get(i));
        i++;
    }
    while(i<l2.size()){
        l3.add(l2.get(i));
        i++;
    }
}

```

Ejercicio 6:

```

public static void unirListas(List l1, List l2, List l3){

    for(int i = 0 ; i < l1.size(); i ++){
        l3.add(l1.get(i));
    }

    for(int i = 0 ; i < l2.size(); i ++){
        l3.add(l2.get(i));
    }
}

```

Ejercicio 7:

```

public static void invertir(List l1, List l2){
    while(!l1.isEmpty()){
        l2.add(l1.get(l1.size()-1));
        l1.remove(l1.size()-1);
    }
}

```

Ejercicio 8:

```
public static void subListas(List l1){
    for(int i = 0 ; i < l1.size() ; i ++){
        for(int j=i+1; j <=l1.size(); j ++){
            System.out.println(l1.subList(i, j));
        }
    }
}
```

Ejercicio 9:

```
public static boolean esInvertidoDe(List l1, List l2){
    if(l1.size() != l2.size()){
        return false;
    }
    else{
        boolean b = true;
        int pos = l1.size()-1;
        for(int i = 0 ; i < l1.size(); i ++){
            if(!l1.get(i).equals(l2.get(pos))){
                return false;
            }
            pos--;
        }
        return b;
    }
}
```

Ejercicio 10:

```
public static void productoPotencia(List l1, List l2){
    for(int i = 0 ; i < l1.size() ; i ++){
        for(int j=i+1; j <=l1.size(); j ++){
            l2.add(l1.subList(i, j));
        }
    }
}
```

b) Listas Parametrizadas

Implementar al menos 10 ejercicios cualesquiera, de Listas Parametrizadas con elementos homogeneos de un Objeto en particular.

Ejercicio 1:

```

public void combiSR(ArrayList<Integer> L1, LinkedList<Integer> L2, int r, int
    i) {
    if (L2.size() == r) {
        System.out.println(L2);
        return;
    }
    int k = i;
    while (k < L1.size()) {
        L2.addLast(L1.get(k));
        combiSR(L1, L2, r, k + 1);
        L2.removeLast();
        k = k + 1;
    }
}

```

Ejercicio 2:

```

public void combiCR(ArrayList<Integer> L1, LinkedList<Integer> L2, int r, int i) {
    if (L2.size() == r) {
        System.out.println(L2);
        return;
    }
    int k = i;
    while (k < L1.size()) {
        L2.addLast(L1.get(k));
        combiCR(L1, L2, r, k);
        L2.removeLast();
        k = k + 1;
    }
}

```

Ejercicio 3:

```

public void permuSR(ArrayList<Integer> L1, LinkedList<Integer> L2, int r) {
    if (L2.size() == r) {
        System.out.println(L2);
        return;
    }
    int i = 0;
    while (i < L1.size()) {
        if (!L2.contains(L1.get(i))) {
            L2.add(L1.get(i));
            permuSR(L1, L2, r);
            L2.removeLast();
        }
        i = i + 1;
    }
}

```

```
}  
}
```

Ejercicio 4:

```
public void permuCR(ArrayList<Integer> L1, LinkedList<Integer> L2, int r) {  
    if (L2.size() == r) {  
        System.out.println(L2);  
        return;  
    }  
    int i = 0;  
    while (i < L1.size()) {  
        L2.add(L1.get(i));  
        permuCR(L1, L2, r);  
        L2.removeLast();  
        i = i + 1;  
    }  
}
```

Ejercicio 5:

```
@Override  
public void sumandos(LinkedList<Integer> L1, int n, int i) {  
    int sum = suma(L1);  
    if (sum > n) {  
        return;  
    }  
    if (sum == n) {  
        System.out.println(L1);  
        return;  
    }  
    int k = i;  
    while (k <= n) {  
        L1.addLast(k);  
        sumandos(L1, n, k);  
        L1.removeLast();  
        k++;  
    }  
}  
  
private static int suma(LinkedList<Integer> L1) {  
    int sum = 0;  
    for (Integer j : L1) {  
        sum = j + sum;  
    }  
}
```

```

        return sum;
    }

```

Ejercicio 6:

```

public void factoresa(LinkedList<Integer> L1, int n, int i) {
    int mul = mult(L1);
    if (mul > n) {
        return;
    }
    if (mul == n) {
        System.out.println(L1);
        return;
    }
    int k = i;
    while (k < n) {
        L1.addLast(k);
        factoresa(L1, n, k);
        L1.removeLast();
        k++;
    }
}

```

```

private static int mult(LinkedList<Integer> L1) {
    int mul = 1;
    for (Integer j : L1) {
        mul = mul * j;
    }
    return mul;
}

```

Ejercicio 7:

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package b;

import java.util.LinkedList;

/**
 *
 * @author cartory
 */

```

```

public class Caballo implements lista {

    int m[][];

    public Caballo(int[][] m) {
        this.m = m;
    }

    @Override
    public LinkedList<Regla> reglasAplicables(int[][] m, int i, int j) {
        LinkedList<Regla> L1 = new LinkedList();
        if (posValida(m, i - 2, j - 1)) {
            L1.add(new Regla(i - 2, j - 1));
        }
        if (posValida(m, i - 2, j + 1)) {
            L1.add(new Regla(i - 2, j + 1));
        }
        if (posValida(m, i - 1, j + 2)) {
            L1.add(new Regla(i - 1, j + 2));
        }
        if (posValida(m, i + 1, j + 2)) {
            L1.add(new Regla(i + 1, j + 2));
        }
        if (posValida(m, i + 2, j + 1)) {
            L1.add(new Regla(i + 2, j + 1));
        }
        if (posValida(m, i + 2, j - 1)) {
            L1.add(new Regla(i + 2, j - 1));
        }
        if (posValida(m, i + 1, j - 2)) {
            L1.add(new Regla(i + 1, j - 2));
        }
        if (posValida(m, i - 1, j - 2)) {
            L1.add(new Regla(i - 1, j - 2));
        }
        return L1;
    }

    @Override
    public void mostrar(int[][] m) {
        for (int i = 0; i < m.length; i++) {
            for (int j = 0; j < m[i].length; j++) {
                System.out.print(" " + m[i][j] + '\t');
            }
            System.out.println(" ");
        }
    }
}

```



```

    }
    System.out.println(" ");
}

@Override
public boolean posValida(int[][] m, int i, int j) {
    return i >= 0 && i < m.length && j >= 0 && j < m[i].length && m[i][j] == 0;
}

@Override
public Regla mejorRegla(LinkedList<Regla> L1, int i1, int j1) {
    double menorDist = Double.MAX_VALUE;
    int i = 0;
    int posMen = 0;
    while (i < L1.size()) {
        double a = L1.get(i).fil - i1;
        double b = L1.get(i).col - j1;
        double dist = Math.sqrt(a * a + b * b);
        if (dist < menorDist) {
            menorDist = dist;
            posMen = i;
        }
        i = i + 1;
    }
    return L1.remove(posMen);
}

@Override
public void laberintoHeuristica(int[][] m, int i, int j, int i1, int j1, int
    paso) {
    m[i][j] = paso;
    if ((i == i1 && j == j1)) {
        mostrar(m);
    }
    LinkedList<Regla> L1 = reglasAplicables(m, i, j);
    while (!L1.isEmpty()) {
        Regla R = mejorRegla(L1, i1, j1);
        m[R.fil][R.col] = paso;
        laberinto(m, R.fil, R.col, i1, j1, paso + 1);
        m[R.fil][R.col] = 0;
    }
}

@Override
public void laberinto(int[][] m, int i, int j, int i1, int j1, int paso) {

```

```

        m[i][j] = paso;
        if ((i == i1 && j == j1)) {
            mostrar(m);
        }
        LinkedList<Regla> L1 = reglasAplicables(m, i, j);
        while (!L1.isEmpty()) {
            Regla R = L1.poll();
            m[R.fil][R.col] = paso;
            laberinto(m, R.fil, R.col, i1, j1, paso + 1);
            m[R.fil][R.col] = 0;
        }
    }

    public static void main(String[] args) {
        int m[][] = {
            {0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 1, 0},
            {0, 1, 0, 1, 0, 1, 0},
            {0, 1, 0, 1, 0, 1, 0},
            {0, 1, 0, 1, 0, 1, 1},
            {0, 1, 1, 1, 0, 1, 0},
            {0, 0, 0, 1, 0, 0, 0}
        };
        Caballo lab = new Caballo(m);
        lab.laberinto(m, 0, 0, 6, 6, 1);
        lab.laberintoHeuristica(m, 0, 0, 0, 0, 0);
    }
}

```

Ejercicio 8:

```

package b;

import java.util.LinkedList;

public class Laberinto implements lista {

    public int m[][];

    public Laberinto(int m[][]) {
        this.m = m;
    }

    @Override
    public LinkedList<Regla> reglasAplicables(int[][] m, int i, int j) {
        LinkedList<Regla> L1 = new LinkedList();
    }
}

```

```

        if (posValida(m, i, j - 1)) {
            L1.add(new Regla(i, j - 1));
        }
        if (posValida(m, i - 1, j)) {
            L1.add(new Regla(i - 1, j));
        }
        if (posValida(m, i, j + 1)) {
            L1.add(new Regla(i, j + 1));
        }
        if (posValida(m, i + 1, j)) {
            L1.add(new Regla(i + 1, j));
        }
        return L1;
    }
}

```

```

@Override
public void mostrar(int[][] m) {
    for (int i = 0; i < m.length; i++) {
        for (int j = 0; j < m[i].length; j++) {
            System.out.print(" " + m[i][j] + '\t');
        }
        System.out.println(" ");
    }
    System.out.println(" ");
}

```

```

@Override
public boolean posValida(int[][] m, int i, int j) {
    return i >= 0 && i < m.length && j >= 0 && j < m[i].length && m[i][j] == 0;
}

```

```

@Override
public Regla mejorRegla(LinkedList<Regla> L1, int i1, int j1) {
    double menorDist = Double.MAX_VALUE;
    int i = 0;
    int posMen = 0;
    while (i < L1.size()) {
        double a = L1.get(i).fil - i1;
        double b = L1.get(i).col - j1;
        double dist = Math.sqrt(a * a + b * b);
        if (dist < menorDist) {
            menorDist = dist;
            posMen = i;
        }
        i = i + 1;
    }
}

```

```

    }
    return L1.remove(posMen);
}

@Override
public void laberinto(int[][] m, int i, int j, int i1, int j1, int paso) {
    m[i][j] = paso;
    if ((i == i1 && j == j1)) {
        mostrar(m);
    }
    LinkedList<Regla> L1 = reglasAplicables(m, i, j);
    while (!L1.isEmpty()) {
        Regla R = mejorRegla(L1, i1, j1);
        m[R.fil][R.col] = paso;
        laberinto(m, R.fil, R.col, i1, j1, paso + 1);
        m[R.fil][R.col] = 0;
    }
}

@Override
public void laberintoHeuristica(int[][] m, int i, int j, int i1, int j1, int
paso) {
    m[i][j] = paso;
    if ((i == i1 && j == j1)) {
        mostrar(m);
    }
    LinkedList<Regla> L1 = reglasAplicables(m, i, j);
    while (!L1.isEmpty()) {
        Regla R = L1.poll();
        m[R.fil][R.col] = paso;
        laberinto(m, R.fil, R.col, i1, j1, paso + 1);
        m[R.fil][R.col] = 0;
    }
}

public static void main(String[] args) {
    int m[][] = {
        {0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 1, 0},
        {0, 1, 0, 1, 0, 1, 0},
        {0, 1, 0, 1, 0, 1, 0},
        {0, 1, 0, 1, 0, 1, 1},
        {0, 1, 1, 1, 0, 1, 0},
        {0, 0, 0, 1, 0, 0, 0}
    };
};

```

```
Laberinto lab = new Laberinto(m);  
lab.laberinto(m, 0, 0, 6, 6, 1);  
lab.laberintoHeuristica(m, 0, 0, 0, 0, 0);  
}  
}
```