2. Ejercicio (3 puntos)

Clases necesarias

```
public class Empresa  {
   LinkedList empleadosPorNss;
   LinkedListItr itrEmpleadosPorNss;
}

public class RepartoAsignado {
   private String nssEmpleado;
   private String empresa;
}

public class Empleado {
   private String nss;
   private String nombre;
   private int numDiasTrabajados;
}
```

```
public class LinkedListItr {
      LinkedList theList;
      Node current;
      Node preceding;
}

class Node {
      Object element;
      Node next;
}
public class LinkedList{
      Node top;
}
```

a) implementación:

```
class Empresa {
   ...
  public Queue crearCola RepartosAsignados(){
    Queue quAsignados = new Queue();
    RepartoAsignado a1 = new RepartoAsignado("1111", "empresa1");
    RepartoAsignado a2 = new RepartoAsignado ("3333", "empresa1");
    RepartoAsignado a3 = new RepartoAsignado ("5555", "empresa2");
    RepartoAsignado a4 = new RepartoAsignado ("7777", "empresa3");
    quAsignados.enqueue(a1);
    quAsignados.enqueue(a2);
    quAsignados.enqueue(a3);
    quAsignados.enqueue(a4);
    return quAsignados;
  }
}

Class RepartoAsignado {
   ...
   public RepartoAsignado(String nssEmpleado, String empresa){
     this.nssEmpleado = nssEmpleado;
     this.empresa = empresa;
   }
}
```

b) implementación

```
public class Empleado {
  ...
  public Empleado(String nss){
    this.Empleado(nss, "", 0);
  }
  public Empleado(String nss, String nombre, int numDiasTrabajados){
    this.nss = nss;
    this.nombre = nombre;
    this.numDiasTrabajados = numDiasTrabajados;
  }
```

```
  public String getNss(){
   return nss;
  }
  public void setNombre(String nombre){
    this.nombre = nombre;
  }
  public bolean equals(Object obj){
    if ((obj != null) && (obj instanceof Empleado)) {
      Empleado emp = (Empleado) obj;
      return this.nss.equals(emp.nss);
    }
    else return false;
  }
  public void incrementarDiaTrabajado(){
    this.numDiasTrabajados++;
  }
}

public class Empresa {
  ...
  public void actualizar(Queue quAsignados){
    Empleado emp;
    while (!quAsignados.isEmpty()) {
      RepartoAsignado reparto = (RepartoAsignado) quAsignados.dequeue();
      emp = new Empleado(reparto.getNssEmpleado());
      if (itrEmpleadosPorNss.find(emp)) { //requiere implementar equals()
         emp = (Empleado) itrEmpleadosPorNss.getCurrent();
      }
      else {
         String nombre = System.in.read();
         emp.setNombre(nombre);
         insertarOrdenado(emp);
      }
      emp.incrementarDiaTrabajado();
    }
   }
   public void insertarOrdenado(Empleado nuevo){
     Empleado empAnt = null;
     boolean enc = false;
     Empleado emp = null;
    //buscar posición en la lista por nss, NO puede haber repetidos
    itrEmpleadosPorNss.goFirst();
    while ( (itrEmpleadosPorNss.isOnList()) && !enc ) {
      emp = (Empleado) itrEmpleadosPorNss.getCurrent();
      if (emp.getNss().compareTo(nuevo.getNss())<0) {
        empAnt = emp;
        itrEmpleadosPorNss.advance();
      }
      else enc = true;
    }
    if ((enc) && (emp.getNss().compareTo(nuevo.getNss())==0))
      System.out.println("Error,el empleado con el NSS "+emp.getNss()+" ya esta");
    else {
      // insertar el nuevo en la lista
      if (empAnt == null)
        itrEmpleadosPorNss.insertFirst(nuevo);
      else  {
        itrEmpleadosPorNss.find(empAnt);
        itrEmpleadosPorNss.insert(nuevo);
      }
    }
  }
}
```

3. Ejercicio. (2,5 puntos)

Estrategia o diseño:

Controlar para cada paréntesis de cierre, que el último paréntesis de apertura fue un paréntesis del mismo tipo. De esta forma se controlan dos cosas: las parejas de paréntesis y el anidamiento correcto.

Se necesita llevar un histórico de paréntesis de apertura. Para ello la estructura más adecuada y eficiente es utilizar una pila, con la siguiente forma de utilizarlo: por cada paréntesis de apertura se almacena el carácter en la pila y por cada paréntesis de cierre se extrae de la pila el carácter de la cima para comprobar si corresponden al mismo tipo de paréntesis. Los demás caracteres no se almacenan. En caso de que todo vaya bien, el proceso finalizará cuando se haya procesado toda la secuencia de caracteres y en el caso de que no sea valido, se finalizará en cuanto se detecte el error.

Implementación

```
public static boolean cadenaCorrecta(String cadena) {
  boolean error = false;
  Stack parentesis = new Stack();
  int numCar = cadena.length;
  int i = 0;

  while (i<numcar-1 && !error)
  {
    String car = cadena.substring(i, i+1);
    If (esParentesisApertura(car))
       parentesis.push(car);
    else if (esParentesisCierre(car)){
       try{
         String apertura = (String) parentesis.pop();
         error = !coincidePareja(apertura, car);
       }
       catch(Exception e){
         error = true;
       }
    }
    i++;
  }
  if (error || !parentesis.isEmpty())
       return false;
  else return true;
}
public static bolean esParentesisApertura(String car){
   if (car.equals("(") || car.equals("[") || car.equals("{"))
       return true;
   else return false;
}
public static bolean esParentesisCierre(String car){
   if (car.equals(")") || car.equals("]") || car.equals("}"))
       return true;
   else return false;
}
public static boolean coincidePareja(String apertura, String cierre){
   if (apertura.equals("(") && cierre.equals(")"))
       return true;
   else if (apertura.equals("[") && cierre.equals("]"))
       return true;
   else if (apertura.equals("{") && cierre.equals("}"))
       return true;
   else return false;
}
```

4. Ejercicio (3,5 puntos)

a) ***Diseñar e implementar: public static void imprimir(Hashtable polinomio);***

Diseño

```
Algoritmo imprimir(Hashtable polinomio)

1. numeroTerminos = recuperar_numero_de_terminos(polinomio);
2. exponentes = recuperar_exponentes(polinomio);
3. arrayExponentes = pasarAarray(exponentes, numeroTerminos);
4. ordenarArray(arrayExponentes);
5. Para cada exponente en arrayExponentes
      5.1 exponente = recuperarExponente(arrayExponentes);
      5.2 coeficiente = recuperarCoeficiente(polinomio, exponente);
      5.3 imprimir_en_pantalla(coeficiente+"x"+exponente);
finPara
```

Implementación:

```
public static void imprimir(Hashtable polinomio){
    int numeroTerminos = polinomio.numpos();
    Enumeration exponentes = polinomio.keys();
    int[] arrayExponentes = pasarAarray(exponentes, numeroTerminos);
    ordernarArray(arrayExponentes);
    for(int i=0;i < arrayExponentes.length; i++){
      int exp = arrayExponentes[i];
      int coef = ((Integer) polinomio.find(new Integer(exp))).intValue();
      System.out.print(coef + "x" + exp);
      if ((i+1) < arrayExponents.length)
        System.out.print("+");
    }
}
public static int[] pasarAarray(Enumeration enum, int num){
  int[] enteros = new int[num];
  int i = 0;
  while ((i < num) && enum.hasMoreElements())
    enteros[i] = ((Integer) enum.nextElement()).intValue();
  return enteros;
}
// implementar un algoritmo de ordenación vista en clase: insertionSort,
// selectionSort, MergeSort, QuickSort
```

b) El coste de ejecución del algoritmo 'imprimir' será la suma del coste de ejecución de cada acción, numeradas del 1 al 5. A continuación se detalla el coste de cada acción:

1. *O*(1), constante: Operación sobre la tabla hash, cuyos métodos son de *O*(1)
2. *O*(1), constante: Operación sobre la tabla hash, cuyos métodos son de *O*(1)
3. *O*(n), lineal: depende del número de términos (n) guardados en la tabla hash
4. *O*(n^2), cuadrática: es el coste de ejecutar los siguientes algoritmos de ordenación: insertionSort, selectionSort o Quicksort (aunque en la mayoría de veces será O(nlogn), siempre hay que tener en cuenta el peor de los casos)

   O(nlog n)*:* es el coste de ejecutar el algoritmo de ordenación MergeSort
5. *O*(n), lineal: depende del número de términos (n) guardados en el array 'arrayExponentes', ya que el coste de todas sus subacciones 5.1, 5.2 y 5.3 es de *O*(1), es decir, constante.

Por lo tanto, el coste total se ve determinado por la acción 4, que será bien O(n^2) o O(nlog n), según implementación del algoritmo de ordenación.

c) No aparece contigua porque las tablas hash aplican una función hash sobre la clave de la información que se va a almacenar, para posicionarlo en el array interno**.**