```
'''
Created on 05/12/2013

@author: Flor
'''
import time # para horarios
from datetime import datetime  # para fechas y horarios

class Fabrica(object):
    '''
    classdocs
    '''


    def __init__(self, idFabrica, entrada, salida):

        '''
        Constructor
        '''
        self.juguetes = dict()
        self.id = idFabrica
        self.horarioEntrada = int(entrada)
        self.horarioSalida = int(salida)

    # Agrega un juguete al diccionario de juguetes
    def addJuguete(self,juguete):
        self.juguetes[juguete.getId()] = juguete

    # Toma el horario de salida
    def getHorarioSalida(self):
        return self.horarioSalida

    # Devuelve el diccionario de juguetes
    def getJuguetes(self):
        return self.juguetes

    # Devuelve el id
    def getId(self):
        return self.id

    # Compara horarios de salida, entrada o id
    def compare(self,fabrica):
        if (self.horarioSalida > fabrica.horarioSalida):
            return -1
        elif(self.horarioSalida < fabrica.horarioSalida):
            return 1
        else:
            if (self.horarioEntrada > fabrica.horarioEntrada):
                return -1
            elif(self.horarioEntrada < fabrica.horarioEntrada):
                return 1
            else:
                if (int(self.id) > int(fabrica.id)):
                    return -1
                elif (int(self.id) < int(fabrica.id)):
                    return 1
                return 0

    # Devuelve el horario de entrada
    def getHorarioEntrada(self):
        return self.horarioEntrada

    # Transforma la hora para que tenga el formato adecuado
    def getHorarioConId(self):
        # seteo una fecha cualquiera solo para tomar el formato horario del timestamp
        return self.id + "," + datetime.fromtimestamp(time.mktime(datetime(2013, 12, 24, 0, 0,
0).timetuple())+ 60*self.horarioEntrada).strftime("%H:%M") + "," + datetime.fromtimestamp(time.mktime
(datetime(2013, 12, 24, 0, 0, 0).timetuple())+ 60*self.horarioSalida).strftime("%H:%M") # revisar

    # Dado un limite, utiliza el problema de la mochila para devolver la suma de los valores
    # y una lista con los juguetes que corresponden a dichos valores
    def getSonrisasMaximas(self, limite):
```

```
        i = 1
        jug = dict()
        for idJuguete in self.juguetes:
            jug[i] = self.juguetes[idJuguete]
            i+=1
        suma = 0
        juguetes = sorted(self.bolsoParaSanta(jug, limite))
        for idR in juguetes:
            suma += idR.getValor()

        return suma, juguetes


    # Problema de la mochila para el trineo y los juguetes
    def bolsoParaSanta(self,items, limit):
        table = [[0 for w in range(limit + 1)] for j in xrange(len(items) + 1)]
        for j in xrange(1, len(items) + 1):
            juguete = items[j]
            for w in xrange(1, limit + 1):
                if juguete.peso > w:
                    table[j][w] = table[j-1][w]
                else:
                    table[j][w] = max(table[j-1][w],
                                      table[j-1][w-juguete.peso] + juguete.valor)
        result = list()
        w = limit
        for j in range(len(items), 0, -1):
            was_added = table[j][w] != table[j-1][w]

            if was_added:
                jug = items[j]
                result.append(items[j])
                w -= jug.peso

        return result
```