308-304B – Object-Oriented Design

Due date: Sunday February 3, 2002 before midnight

Practical information

- 1 <= team size <= 2
- Each team submits only once. Include name(s) and ID(s) in *every* submitted file.
- Your submission must be in the form of a simple HTML file (`index.html`) with explicit references to all submitted files as well as inline inclusion of images.
- The submission medium (handin, WebCT, ...) will be anounced shortly.
- Python (www.python.org) as well as dia (www.lysator.liu.se/∼alla/dia/dia.html) are installed on all SOCS FreeBSD machines.

Requirements

In this assignment you will implement and test three classes for a Spreadsheet application (no GUI at this stage). You will also have to make a design decision, based on performance, and draw the class diagram. The classes are:

- CellData:
  Instances of this class are stored in the spreadsheet's cells and contain data (currently only integer numbers, later also formulas , *etc.*)
- CellCoordinate:
  The spreadsheet uses instances of this class to index the cells.
- SpreadsheetData:
  The basic spreadsheet.

Application Programmer's Interface (API)

Your design/tests/implementation *must* use *exactly* the following externally visible, public API (API.py). This implies that *all other* attributes and methods used in the design must be private (internal to the class). Note: private attributes/methods names start with __ in Python. API documentation in HTML can be automatically generated with `pydoc -w`. For the assignment, this leads to api.html.

```
'''
### ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ###

      CellData, CellCoordinate, SpreadsheetData classes API.

### ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ###
'''


# Types of objects that can be stored in a CellData object
dataTypes = (types.IntType,)

### ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ###
### class CellData
```

```
class CellData:
  """
  Encapsulated Integer data
  """

  def __init__(self, value = 0):
    ''' value:Integer ->    --  CellData constructor.

    A TypeError is raised on bad argument.
    '''

  def getValue(self):
    ''' -> :Integer'''

  def setValue(self, value = 0):
    ''' value:Integer ->

    A TypeError is raised on bad argument.
    '''

  def __str__(self):
    ''' -> :String

    Return the string representation of the object value (Integer).
    '''

### ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ###
### class CellCoordinate

class CellCoordinate:
  """
  Encapsulated coordinates of cells in a spreadsheet.

  Note: might in the future want to implement "coordinate arithmetic"
  by means of __add__ etc.
  """

  def __init__(self, row = 1, column = 1):
    ''' row:PositiveInteger, column:PositiveInteger ->

    CellCoordinate constructor.
    A KeyError is raised on bad arguments.
    '''

  def getRow(self):
    ''' -> :Integer'''

  def getColumn(self):
    ''' -> :Integer'''

  def setRow(self, row = 1):
    '''row:PositiveInteger ->

    A KeyError is raised on bad argument.
    '''

  def setColumn(self, column = 1):
```

```
    '''column:PositiveInteger ->

    A KeyError is raised on bad argument.
    '''

  def __str__(self):
    ''' -> :String

    Return the string representation of the cell coordinate.
    Example: with row=2, column=88, __str__ will return <CellCoordinate:2,88>
    '''

### ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ## ###
### class SpreadsheetData

class SpreadsheetData:
  """
  Encapsulates a dynamically sized spreadsheet structure
  containing CellData data and indexed by CellCoordinate coordinates.
  """
  def __init__(self):
    ''' __init__() ->  -- SpreadsheetData constructor.'''

  def __setitem__(self, coord, data):
    '''coord:CellCoordinate, data:CellData ->

    Update the content of cell indexed by ``coord'' with ``data''.
    A KeyError is raised on bad coordinate,
    A TypeError is raised on bad value.
    Example use: sd[CellCoordinate(3,4)] = CellData(33)
    '''

  def __getitem__(self, coord):
    '''coord:CellCoordinate -> :CellData | None

    Return the content of a cell indexed by ``coord''
    (return None if the cell is empty).
    A KeyError is raised on bad coordinate.
    Example use: sd[CellCoordinate(3,4)]
    '''

  def __delitem__(self, coord):
    '''coord:CellCoordinate ->

    Empty the cell indexed by ``coord''.
    A KeyError is raised on bad coordinate.
    Example use: del sd[CellCoordinate(3,4)]
    '''

  def getLU(self):
    ''' -> :CellCoordinate | None

    Return a CellCoordinate containing the
    Left-most non-empty column, and Upper-most non-emtpy row.
    Return None in case of an empty spreadsheet
    '''
```

```
def getRB(self):
  ''' -> :CellCoordinate | None

  Return a CellCoordinate containing the Right-most non-empty column,
  and Bottom-most non-emtpy row.
  Return None in case of an empty spreadsheet
  '''

def __str__(self):
  ''' -> :String

  Return the string representation of the SpreadSheet.
  This looks like a table of values with spaces for empty cells.
  The row and column indexes are also shown.
  '''
```

1. Class Diagram

Use "dia" to draw the class diagram. Show all attributes and methods of the classes. produce a GIF or JPEG image. Both dia file and image file must be submitted.

2. Testing and implementation

Test scripts

For each class, write a `unittest` test script. Each script shall test the class for

- success
- failure
- identity (sanity)

Use `PyUnit` (aka `unittest.py`) to write your scripts.

To test the test scripts, make a "dummy" implementation of all classes (just implement all class methods with `pass`) and run all tests. They should all fail.

Prototype 0

Implement the classes `CellData` and `CellCoordinate`. Both should pass their respective tests.

Prototype 1

Implement the `SpreadsheetData` class, using Python lists (*i.e.,* an array is a list of lists) as an internal data structure. Empty cells are represented by `None` list entries. Internally, cells will be indexed like

```
self.__data[row][column]
```

This prototype should pass all tests.

Prototype 2

Implement the `SpreadsheetData` class, this time using a Python dictionary as an internal data structure. The dictionary keys should be tuples containing the row and column. Internally, cells will be indexed like

```
self.__data[ (row, column) ]
```

This prototype should pass all tests.

3. Performance testing

Write a script (you do not have to use `PyUnit`) to evaluate the performance of both prototypes 1 and 2. For each prototype, we expect the following experiments:

1. Full System: for a spreadsheet with $n * n$ cells, measure the time it takes to set, then get all cells. To measure the time, you might want to use the `time` module.

2. Sparse System: for a spreadsheet with $n * n$ cells, measure the time it takes to set, then get 10% of the cells (evenly distributed).

Timing must include the time to instantiate the `SpreadsheetData` class. Perform experiments with different spreadsheet sizes n, and plot the results (time as a function of n). This should result in 4 plots (Full and Sparse combined with Prototypes 1 and 2). You must submit 4 GIF or JPEG files. For each plot, try a sufficient number of n values to obtain a curve. n_{max} for each plot should be determined by the additional *requirement* that response time should not be more than 15 seconds. n_{max} will obviously depend on the power and load of the machine you run this test on. Thus, to make meaningful conclusions, all performance tests *must* be run on the same machine.

Comment on the results:

- Compare n_{max} in the different cases.
- Which prototype shall be used from now on, and why?