**BCPR301 – Advanced Programming**

**Assessment#1: Interpreter**

**This assessment is worth 10% of the marks out of the total of 100 for the whole course grade for BCPR301.**

##### Deadline

* You dropbox all required material by ***3:00pm*** on Monday 26 August 2019.

This assessment relates to the following learning objective 2

**“To ensure students have the knowledge and experience to effectively learn a new programming language.”**

##### Task:

* Extend the provided the Tiny Interpreted Graphics language package (TIGr) to give it more flexibility, functionality, robustness and portability. NOTE: These qualities are listed in priority order for this assessment. You MUST use the provided Abstract Base Classes.
* You MUST work in pairs on any part of this assessment, as a pair programming team. You must work in at least two different pairs. You should clearly identify the component you worked on. Equal marks will be given to each member of the pair.
* You can also freely incorporate the work of others as long as it is acknowledged. Note that only your OWN work will be awarded marks. When marking the assignment you will be asked to explain in front of the class how your code works.
* You can select which features you work on from the list provided below. You will be graded on a 0-2 scale for each component for which you present original source code. Your total grade will be the sum of the grade for each component.
* Interactive front-end
* Supports piping and scripting
* Command line switches
* Parsed from configurable lookup table
* Uses regular expressions in parser
* Uses generic parsing engine
* Outputs with Tkinter
* Outputs with turtle.py
* Outputs with another python graphics package
* Provide doctests
* Provide unittests
* Breadth of test coverage
* Amount of error trapping & handling

Grading scale:

1 = adequate quality (= How is this more flexible, functional, robust or portable?)

2 = complete and well implemented (=What is clever about this?)

**from** abc **import** ABC, abstractmethod  
  
**""" Tiny Interpreted GRaphic = TIGR  
Keep the interfaces defined below in your work.  
 """  
  
  
class** AbstractDrawer(ABC):  
 *""" Responsible for defining an interface for drawing """* @abstractmethod  
 **def** select\_pen(self, pen\_num):  
 **pass** @abstractmethod  
 **def** pen\_down(self):  
 **pass** @abstractmethod  
 **def** pen\_up(self):  
 **pass** @abstractmethod  
 **def** go\_along(self, along):  
 **pass** @abstractmethod  
 **def** go\_down(self, down):  
 **pass** @abstractmethod  
 **def** draw\_line(self, direction, distance):  
 **pass  
  
  
class** AbstractParser(ABC):  
 **def** \_\_init\_\_(self, drawer):  
 self.drawer = drawer  
 self.source = []  
 self.command = **''** self.data = 0  
  
 @abstractmethod  
 **def** parse(self, raw\_source):  
 **pass  
  
  
class** AbstractSourceReader(ABC):  
 *""" responsible for providing source text for parsing and drawing  
 Initiates the Draw use-case.  
 Links to a parser and passes the source text onwards  
 """* **def** \_\_init\_\_(self, parser, optional\_file\_name=**None**):  
 self.parser = parser  
 self.file\_name = optional\_file\_name  
 self.source = []  
  
 @abstractmethod  
 **def** go(self):  
 **pass**

**from** TIGr **import** AbstractDrawer, AbstractParser, AbstractSourceReader  
  
**"""These implementations should be replaced,  
by more flexible, portable and extensible solutions.  
"""  
  
  
class** Drawer(AbstractDrawer):  
 *""" Responsible for printing as text what the drawing commands are"""* **def** select\_pen(self, pen\_num):  
 print(**f'Selected pen {**pen\_num**}'**)  
  
 **def** pen\_down(self):  
 print(**'pen down'**)  
  
 **def** pen\_up(self):  
 print(**'pen up'**)  
  
 **def** go\_along(self, along):  
 print(**f'GOTO X={**along**}'**)  
  
 **def** go\_down(self, down):  
 print(**f'GOTO Y={**down**}'**)  
  
 **def** draw\_line(self, direction, distance):  
 print(**f'drawing line of length {**distance**} at {**direction**} degrees'**)  
  
  
**class** Parser(AbstractParser):  
  
 **def** parse(self, raw\_source):  
 *"""hard coded parsing like this is a Bad Thing!  
 It is inflexible and has no error checking  
 """* self.source = raw\_source  
 **for** line **in** self.source:  
 self.command = line[0]  
 **try**:  
 self.data = int(line[2])  
 **except**:  
 self.data = 0  
 **if** self.command == **'P'**:  
 self.drawer.select\_pen(self.data)  
 **if** self.command == **'D'**:  
 self.drawer.pen\_down()  
 **if** self.command == **'G'**:  
 self.drawer.goto(self.data)  
 **if** self.command == **'N'**:  
 self.drawer.draw\_line(0, self.data)  
 **if** self.command == **'E'**:  
 self.drawer.draw\_line(90, self.data)  
 **if** self.command == **'S'**:  
 self.drawer.draw\_line(180, self.data)  
 **if** self.command == **'W'**:  
 self.drawer.draw\_line(270, self.data)  
 **if** self.command == **'X'**:  
 self.drawer.go\_along(self.data)  
 **if** self.command == **'Y'**:  
 self.drawer.go\_down(self.data)  
 **if** self.command == **'U'**:  
 self.drawer.pen\_up()  
  
  
**class** SourceReader(AbstractSourceReader):  
 *""" responsible for providing source text for parsing and drawing  
 Initiates the Draw use-case.  
 Links to a parser and passes the source text onwards  
 """* **def** go(self):  
 self.source.append(**'P 2 # select pen 2'**)  
 self.source.append(**'X 5 # go to 5 along'**)  
 self.source.append(**'Y 15 # go to 15 down'**)  
 self.source.append(**'D # pen down'**)  
 self.source.append(**'W 2 # draw west 2cm'**)  
 self.source.append(**'N 1 # then north 1'**)  
 self.source.append(**'E 2 # then east 2'**)  
 self.source.append(**' S 12.7 '**)  
 self.source.append(**' U # pen up'**)  
 self.parser.parse(self.source)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 s = SourceReader(Parser(Drawer()))  
 s.go()