COMP1531

Week 6 Tutorial!

Housekeeping

- Project deliverable 2:
    - UML class diagram - submit this Sunday
        - feedback - next lab
    - back-end implementation - submit wk07 Sunday
    - demo - week 08 lab
- Lab 06 due this Sunday

Abstract Class

Abstract Class

```python
from abc import ABC, abstractmethod
class Shape(ABC):

    def __init__(self, color):
        self._color = color
    def get_color(self):
        return self._color
    def __str__(self):
        return self._color

    @abstractmethod
    def area(self):
        pass

    @abstractmethod
    def scale(self, ratio):
        pass
```

Inheriting the Abstract class (i)

```
class Rectangle(Shape):

    def __init__(self, color, width,height):
        super().__init__(color)
        self._width = width
        self._height = height
```

Inheriting Abstract Class (ii)

```
@property
      def width():
            return self._width;

      @width.setter
      def width(width):
            self._width = width;

      @property
      def height():
            return self._height;

      @property.setter
      def height(height):
            self._height = height;
```

Inheriting Abstract Class (iii)

```
    def area(self):
        return self._height * self._width

    def scale(self, ratio):
        self._width *= ratio
        self._height *= ratio
```

Inheriting Abstract Class (iv)

```
class Circle(Shape):
    def __init__(self, color, radius):
        Shape.__init__(self,color)
        self._radius = radius

    @property
    def radius(self):
        return self._radius

    @radius.setter
    def radius(self, radius):
        self._radius = radius

    def area(self):
        return 3.14 * self._radius * self._radius

    def scale(self, ratio):
        self._radius *= ratio
```

TDD - Test Driven Dev

TDD (i)

- goal of TDD is deliver a specification that delivers the customer’s goal
- think through your requirements/design before your write your functional code to implement them.

TDD - Steps

- Write a test that fails
- Build just enough code to make the test succeed
- Test the code and make necessary changes until test succeeds
- Refactor and add code for next piece of requirement

Let’s get coding

Refactoring

1. Got a new requirements at the start of each iteration
2. analyse our code-base, refactor existing code
3. Does the new requirement require duplication of some existing code
   a. yes - “extract” the code that is likely to be duplicated into an independent reusable method.

Exceptions Handling

Exceptions vs bugs

- bugs are errors that causes unexpected behaviour or behave in an unintended way
- exceptions are caused by unavoidable circumstances such as providing wrong input, or a file to be read not found. These are anticipated.

Bug - history

In 1946, Grace Hopper joined the Harvard Faculty at the Computation Laboratory where she worked on Mark II and Mark III.
Operators traced an error in the Mark II to a moth trapped in a relay, coining the term **bug**.