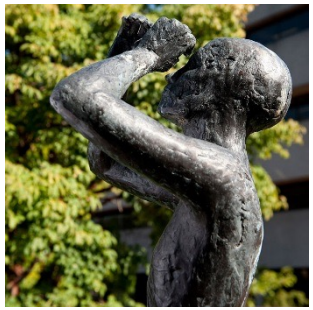


Lecture 20 – TkInter OO, UML, Unit Testing

CS2513

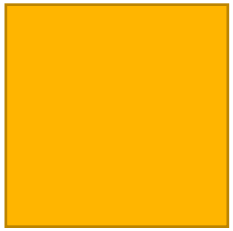
Cathal Hoare

A TRADITION OF INDEPENDENT THINKING

UCC

University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

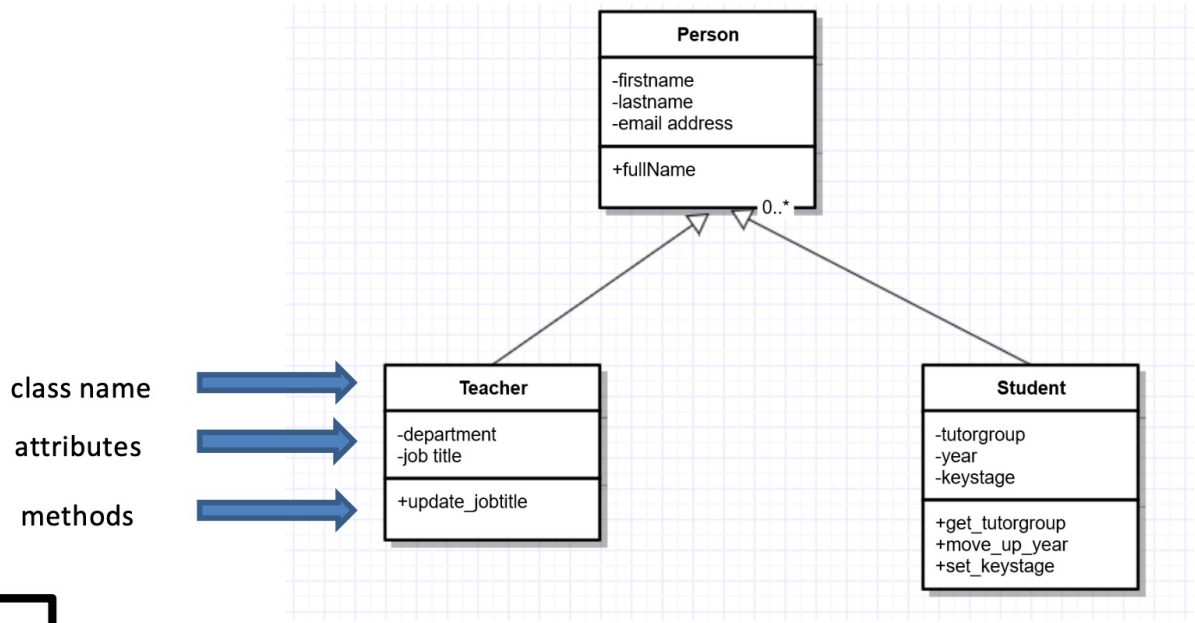
Reviewed Circle Game as a OO Implementation



- See example and Video

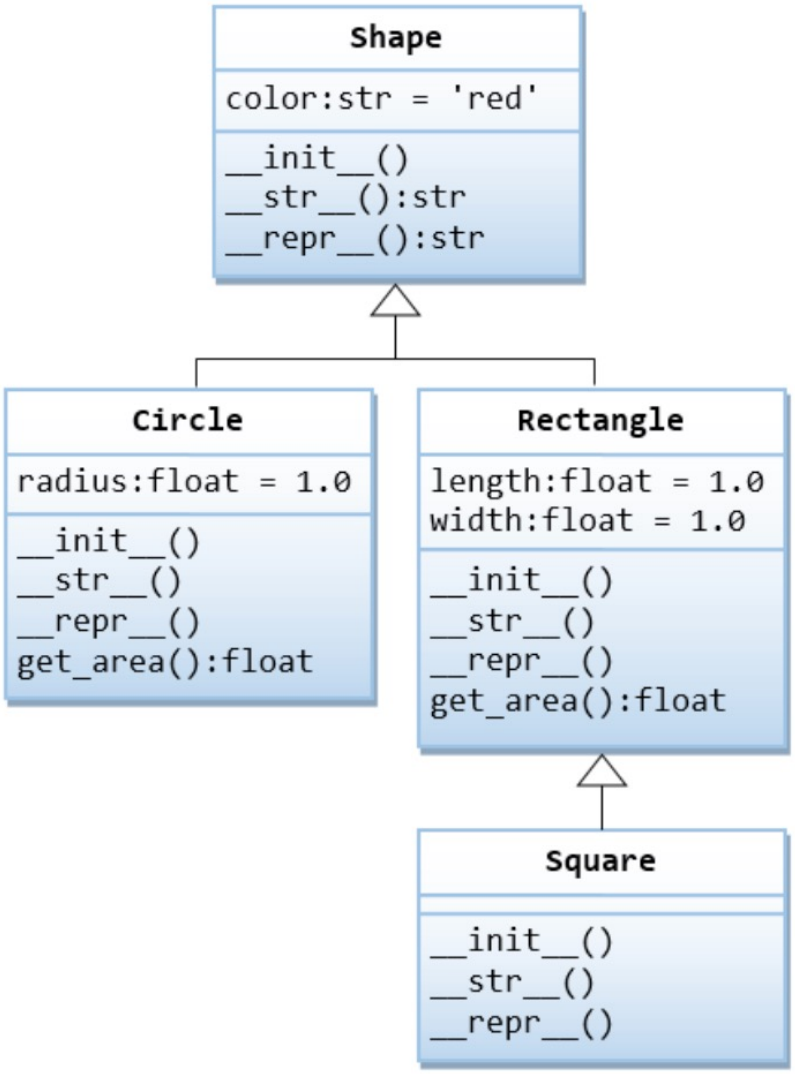
UML Diagrams

Aside: UML



- private
protected
+ public

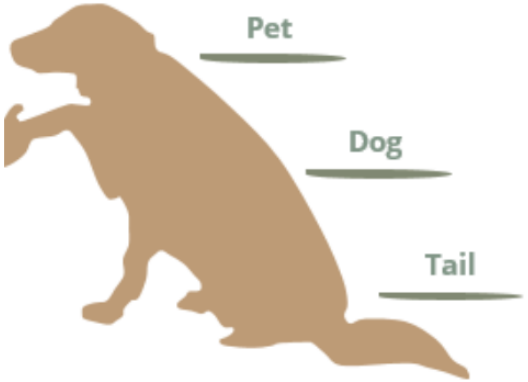
Inheritance Hierarchy



Mechanisms for Reuse

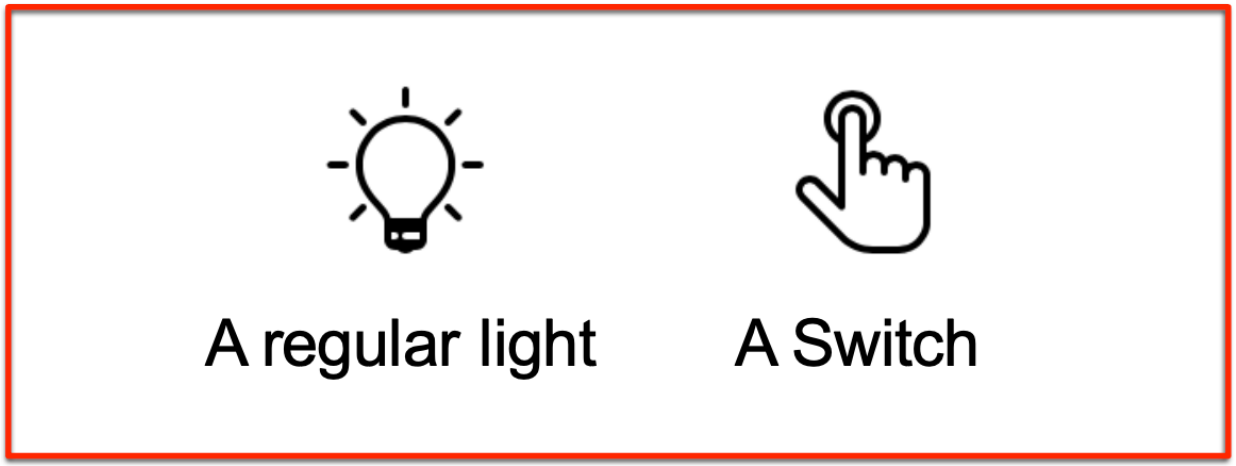
- **Composition over inheritance** (or **composite reuse principle**) in object-oriented programming (OOP) is the principle that classes should achieve polymorphic behaviour and code reuse by their composition (by containing instances of other classes that implement the desired functionality) rather than inheritance from a base or parent class.

•a tail is a part of both dogs and cats (**composition**)
•a cat is a kind of pet (**inheritance**)



Composition

Room



We can model complex objects

Say we have a room, with a light, and a light switch

Composition

Room

A regular light

A Switch

Rather than write a single class, we write a class, Room, that is composed of the methods and actions required for a room, but reuses the classes we have already written.

Composition

Room

A regular light

A Switch

We can have variable names that reference light and switch objects in the same way we use strings and integers

Composition

Room

A regular light

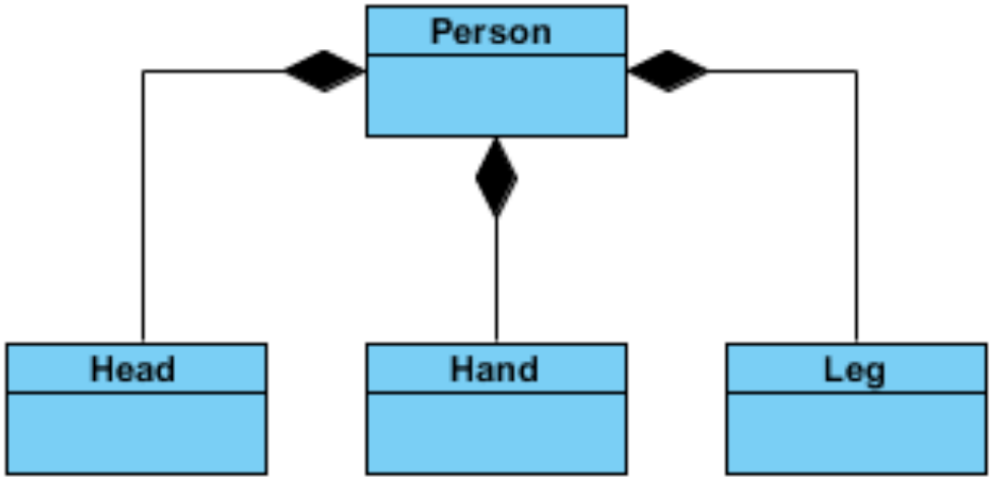
A Switch

This is useful in the same was inheritance was - less code written, fewer bugs, and any bugs fixed just once.

Composition

- When we create a new class, we are in effect creating a new data type.
- If we can create classes with sets of ints, floats, strings etc, why not build classes using instances of objects we've created.
  - This allows us to create simpler, more maintainable code.
  - This also allows us break out code that we use frequently into classes of functionality so that it can be reused.

Aside: UML



Unit Testing

- A software development process in which the smallest testable parts of an application, called units, are individually scrutinized for proper operation.
- Unit tests are can be automated tests or manual tests, written and run by software developers to ensure that a section of an application (known as the "unit") meets its design and behaves as intended.
- They are often performed by the developer who originally wrote the code, as a first line of defense before conducting further testing.
- Benefits:
  - **Early detection of problems in the development cycle**
  - **Reduced cost**
  - **Detects changes which may break a design**
  - **Test-driven development -** The same unit tests are run against that function frequently as the larger code base is developed either as the code is changed or via an automated process with the build. If the unit tests fail, it is considered to be a bug either in the changed code or the tests themselves.

Unit Testing

```python
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()
```

https://docs.python.org/3/library/unittest.html


Next Time:
Regular Expressions