Practical 10
Modelling the World with Objects

Learning Objectives

1. Understand and apply class relationships: composition, aggregation and inheritance
2. Understand and use exception handling

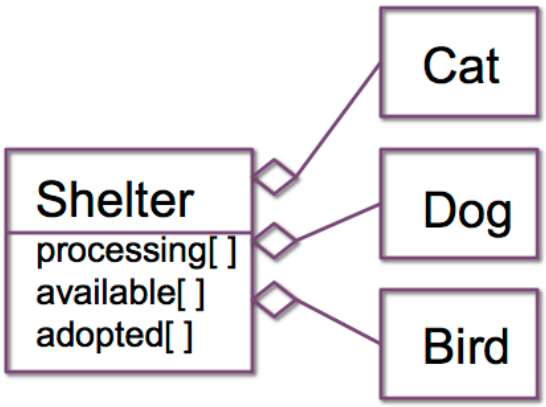
Overview

In this practical we will see how to create relationships between objects. Exceptions are an important concept in OO programming. We will add exceptions to our code.

Tasks

1. Setting up an animal shelter

The lecture notes outline an implementation of an animal shelter class. This class includes lists of animals – which is an aggregation relationship. The shelter "has" animals.



Type in the code below as **shelters.py** and copy and modify **animals.py** from Practical 9 as shown in the next code snippet. These match the code given in the notes in Lecture 10.

shelters.py

```
from animals import Dog, Cat, Bird, Shelter

print('\n#### Pet shelter program ####\n')

rspca = Shelter('RSPCA', 'Serpentine Meander', '123456')
rspca.newAnimal('Dog', 'Dude', '1/1/2011', 'Brown', 'Jack Russell')
rspca.newAnimal('Dog', 'Brutus', '1/1/1982', 'Brown',
                'Rhodesian Ridgeback')
rspca.newAnimal('Cat', 'Oogie', '1/1/2006', 'Grey', 'Fluffy')
rspca.newAnimal('Bird', 'Big Bird', '10/11/1969', 'Yellow', 'Canary')
rspca.newAnimal('Bird', 'Dead Parrot', '1/1/2011', 'Dead', 'Parrot')

print('\nAnimals added\n')

print('Listing animals for processing...\n')
```

```
rspca.displayProcessing()

# This code is commented out until you have implemented
# the methods in animal.py

#print('Processing animals...\n')

#rspca.makeAvailable('Dude')
#rspca.makeAvailable('Oogie')
#rspca.makeAvailable('Big Bird')
#rspca.makeAdopted('Oogie')

#print('\nPrinting updated list...\n')

#rspca.displayAll()
```

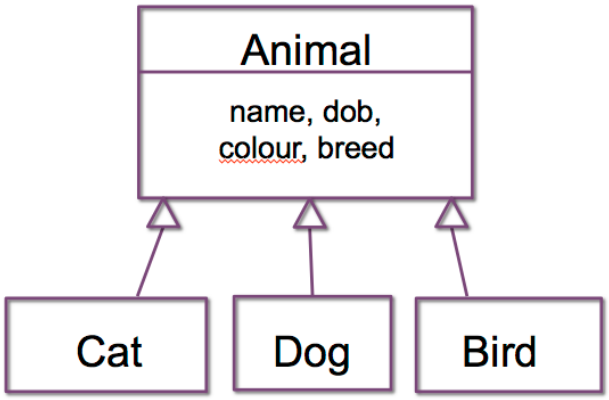
animals.py

```
# Cat, Dog, Bird definitions from Prac 10 should be here

class Shelter():

    def __init__(self, name, address, phone):
        self.name = name
        self.address = address
        self.phone = phone
        self.processing = []
        self.available = []
        self.adopted = []

    def displayProcessing(self):
        print('Current processing list:')
        for a in self.processing:
            a.printit()
        print()

    def displayAvailable(self):
        ... # add your code here

    def displayAdopted(self):
        ... # add your code here

    def displayAll(self):
        self.displayProcessing()

    def newAnimal(self, type, name, dob, colour, breed):
        temp = None
        if type == 'Dog':
            temp = Dog(name, dob, colour, breed)
        elif type == 'Cat':
            temp = Cat(name, dob, colour, breed)
        elif type == 'Bird':
            temp = Bird(name, dob, colour, breed)
        else:
            print('Error, unknown animal type: ', type)
```

```
        if temp:
            self.processing.append(temp)
            print('Added ', name, ' to processing list')
            self.displayAll()

    def makeAvailable(self, name):
        ... # add your code here

    def makeAdopted(self, name):
        ... # add your code here
```

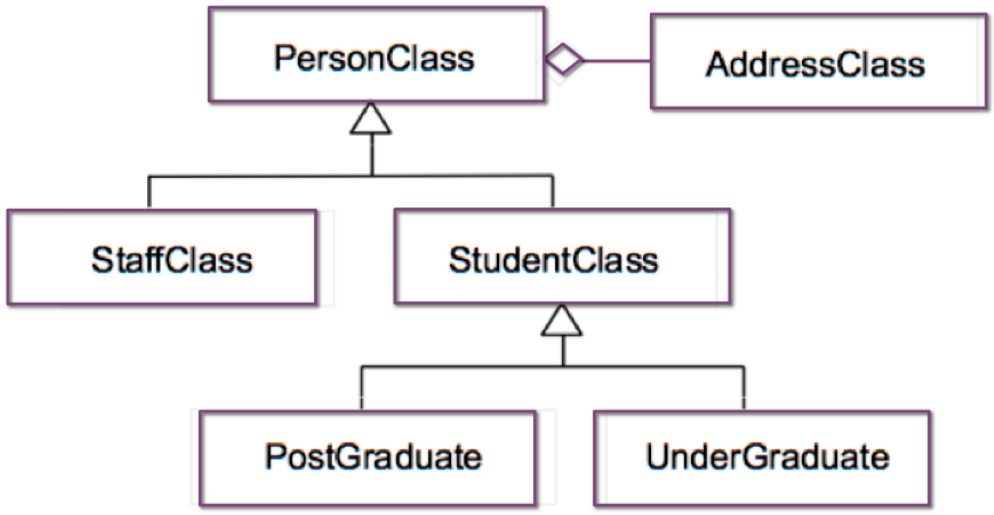
2. It's a family affair

In the lecture we created a parent (super) class Animal to factor out the repetition in Cat, Dog and Bird. This is an inheritance relationship – Cat "is an" Animal.



Edit animals.py and add in the modified class definitions for **Dog** and **Bird**. Re-run shelters.py after making the changes to see that everything still works.

```
class Animal():

    myclass = "Animal"

    def __init__(self, name, dob, colour, breed):
        self.name = name
        self.dob = dob
        self.colour = colour
        self.breed = breed

    def __str__(self):
        return(self.name + '|' + self.dob + '|' + self.colour + '|' +
               self.breed)

    def printit(self):
        spacing = 5 - len(self.myclass)
        print(self.myclass.upper(), spacing*' ' + ': ',
              self.name,'\tDOB: ', self.dob,'\tColour: ',
              self.colour,'\tBreed: ', self.breed)

class Dog(Animal):
    myclass = "Dog"

class Cat(Animal):
    myclass = "Cat"

class Bird(Animal):
    myclass = "Bird"
```



3. People are People

In the lecture we saw the above class diagram for people, students and staff. We will now implement that structure and then read information from files using regular expressions to split out the data.

people.py

```
class Address():

    def __init__(self, number, street, suburb, postcode):
        self.number = number
        self.street = street
        self.suburb = suburb
        self.postcode = postcode

    def __str__(self):
        return(self.number + ' ' + self.street + ', ' + self.suburb +
', ' + self.postcode)

class Person():

    def __init__(self, name, dob, address):
        self.name = name
        self.dob = dob
        self.address = address

    def displayPerson(self):
        print('Name: ', self.name, '\tDOB: ', self.dob)
        print('       Address: ', str(self.address))
```

testPeople.py

```
from people import Address, Person

print('#### People Test Program ###')

testAdd = Address('10', 'Downing St', 'Carlisle', '6101')
testPerson = Person('Winston Churchill', '30/11/1874', testAdd)
testPerson.displayPerson()
```

Run testPeople.py to see its output. Add in another test person and re-run the program.

Now we can add in a sub-class for Staff. We don't want to duplicate code, so, where possible, we will use super() to call the methods from the parent class. Modify people.py and testPeople.py to include the code below.

testPeople.py

```
from people import Address, Person, Staff

print('#### People Test Program ###')

testAdd = Address('10', 'Downing St', 'Carlisle', '6101')
testPerson = Person('Winston Churchill', '30/11/1874', testAdd)
testPerson.displayPerson()
print()

testAdd2 = Address('1', 'Infinite Loop', 'Hillarys', '6025')
testPerson2 = Staff('Professor Awesome', '1/6/61', testAdd2,
'12345J')
testPerson2.displayPerson()
print()
```

people.py

```
# existing code here

class Staff(Person):

    myclass = 'Staff'

    def __init__(self, name, dob, address, id):
        super().__init__(name, dob, address)
        self.id = id

    def displayPerson(self):
        super().displayPerson()
        print('       StaffID: ', self.id)
```

4. Shower the People

So, we have People and Staff classes. We can follow the same pattern to create students, postgrad students and undergrad students: (update people.py)

- Student class
  - extend Person
  - add a Student ID instance variable
  - update the myclass class variable to be 'Student'
- Postgrad class
  - extend Student
  - update the myclass class variable to be 'Postgrad'

- Undergrad class
  - extend Student
  - update the myclass class variable to be 'Postgrad'

Update **testPeople.py** to include values to test the new classes.

5. Regular Reading

Now we can connect up some concepts from across the semester. We will read in people data from a file, split it by a delimiter (':'), extract fields with regular expressions, then load up a list of people.

What we will need in our code is:

1. Import classes
2. Create variables
3. Open file
4. For each line in the file
   a. Split line into class, name, dob, address
   b. Use regular expressions to separate the parts of the address
   c. Create the address object
   d. Create person object to match first field in the entry
5. Print person list

Here is the sample file – you can copy and paste it into vim:

people.csv

```
Staff:Professor Michael Palin:1/6/61:1 Infinity Loop, Balga, 6061
Postgrad:John Cleese:1/9/91:16 Serpentine Meander, Gosnells, 6110
Undergrad:Graham Chapman:7/9/97:80 Anaconda Drive, Gosnells, 6110
Undergrad:Connie Booth:8/9/98:10a Cobra St, Dubbo, 2830
```

We can split the input on ':' to separate the class, name, dob and address.

To use the regular expressions, we need to import re, then split out the address in the same way as we did it in Prac Test 4. Use the code below as a starting point:

```
import re

with open('people.csv', 'r') as f:
    lines = f.readlines()

# Lets create a pattern and extract some information with it
reg = re.compile(r'''(
                  (\d+[a-zA-Z]?)  # (2) house number
                  (\s+)
                  (\w+\s\w+)      # (4) street name and type
                  (,\s+)
                  (\w+)           # (6) suburb
                  (,\s+)
                  (\d{4})         # (8) postcode
```

```
                  )''', re.VERBOSE)

for line in lines:
    splitline = line.strip().split(':')
    myclass = splitline[0]
    name = splitline[1]
    dob = splitline[2]
    inaddress = splitline[3]
    result = reg.match(inaddress)  # returns the matching groups
    if result:
        num = result.group(2)
        street = result.group(4)
        suburb = result.group(6)
        postcode = result.group(8)
        print(myclass,'|',name,'|',dob,'|',num,'|',street,
              '|',suburb,'|',postcode)
    else:
        print('Address not matched: ', inaddress)
```

6. Exceptional

As a final part to this final practical, we can add exception handling to our program. One of the riskiest areas of code is working with files. We can update the code from Task 5 to add exception handling around the file open/read/close. We should always have exception handling around file input and output.

```
try:
    with open('people.csv', 'r') as f:
        lines = f.readlines()
except OSError as err:
    print('Error with file open: ', err)
except:
    print('Unexpected error: ', err)
```

Submission

Create a README file for Prac10. Include the names and descriptions of all of your files from today.

All of your work for this week's practical should be submitted via Blackboard using the link in the Week 10 unit materials. This should be done as a single "zipped" file.

Reflection

1. **Knowledge**: define class variables and instance variables
2. **Comprehension**: If a class variable is changed, e.g. BankAccount.interest_rate = 0.02 in accounts.py, which objects are affected?
3. **Application**: How would you set up multiple shelters in shelters.py?

4. **Analysis**: In Task 2 the method __str__ was added to the Animal class. What does it do and how does it improve the classes?
5. **Synthesis**: What could you do to make your code more robust before publishing it as a package? (Hint: testing and exceptions)
6. **Evaluation**: Two approaches to sharing code are notebooks and publishing packages. Describe each approach and the situations each is suited to.

Challenge

These challenges extend the work we've done in the practicals and are a good test of your understanding.

1. Extending Task 5, add phone numbers to the input file people.csv, then add in code using regular expressions to read in the phone numbers.
2. Add exception handling to the accounts programs from last week. Each numeric input should be protected and checked with try/except clauses.
3. As in Prac 9, add in a class to represent rabbits and then write code to test it.
4. As in Prac 9, add an instance variable for holding the microchip information for cats and dogs in the animal shelter example. Birds do not have microchips, so it shouldn't be in their class definition
5. Code was given in Lecture 12 for parts of this code. Compare your code and update where it improves your code – or leave your code as-is if you're happy with your solution.