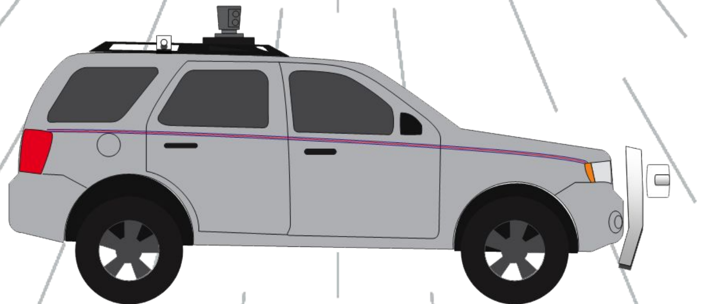
THE UNIVERSITY OF ARIZONA

CAT VEHICLE REU 2019

A crash course on Object-oriented programming

Rahul Bhadani <rahulbhadani@email.arizona.edu>



Before we proceed …

- This session assumes that you all have some sort of programming background.
- You are familiar with basic programming constructs such variables, keywords, if, switch, for loop, etc.
- I had posted on piazza last week for you to go through a Udacity course on C++ https://www.udacity.com/course/c-for-programmers--ud210

Procedural programming

- In traditional programming, a program is divided into functions (also called as procedures, hence procedural programming) to give it modularity
- Generally, no link between data and functions
- Flat structure of the program
- Data scope or visibility is only limited to functions
- Difficult to manage a large program

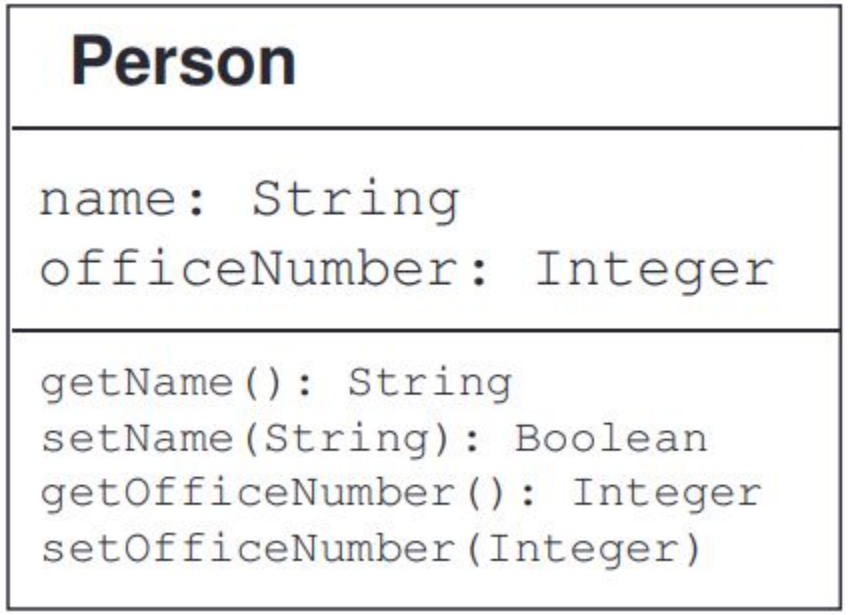
Object-oriented programming (OOP)

- As programs grow it is difficult to manage them
- Procedural programming doesn’t hide not required to be exposed
- OOP overcomes above shortcomings
- OOP hide data, only to be exposed by relevant functions
- Creates program in nested modularity
- Provides different ways of accessing mechanism: public, private and protected.

Central concepts in OOP

- Encapsulation
- Polymorphism
- Inheritance

Encapsulation



- Everything in C++ revolved around class.
- A class is an abstract data type (ADT)
- A class contains data definitions and implementation of procedures
- A class can be used to create different instances

Encapsulation

```
#include<iostream>
#include<cstring>
#include<cstdlib>
using namespace std;
```

```
class person
{
    private:
    string name;
    unsigned int ssn;
    public:
        void setName(string n) { this->name = n; }
        void setSsn( unsigned int SSN) { this->ssn = SSN; }
        string getName() { return this->name; }
        unsigned int getSsn() { return this->ssn; }
};
```

scope

Encapsulation

```
int main()
{
    person tony;
    tony.setName("tony");
    tony.setSsn(144142342);
    cout << tony.getName()<<endl;

    return 0;
}
```

Save it as person.cpp

Compiling the program:

In the terminal, navigate to the directory where you saved the file **person.cpp**

```
$ g++ person.cpp
```

This will create an executable a.out

```
$ ./a.out
```

Output will be:

```
tony
```

A few things about scope

- In previous slide you must have note **`public`**, **`private`** keywords
- These are called as scope.
- There are three of them: **`public, private, protected`**
- **`public:`** accessible through dot operator from anywhere outside the class, but within the program
- **`private:`** can't be access or viewed by dot operator outside the class. Only the class and its friend function can access it
- **`protected`**: similar to private but can be accessed in child class; it will be clear when we talk about inheritance.

Encapsulation

- Similar we can create other 'instances' of `person`:
- `person peter;`
- `person steven;`
- `person natasha;`

Polymorphism

- In a class, a function can have many definition, there are making it more flexible depending on the type of arguments and/or number of arguments passed to it. This is called as polymorphism.

```
void setName(string n)
{
    this->name = n;
}
```

```
void setName(string firstname, string lastname)
{
    this->name = firstname + " " + lastname;
}
```

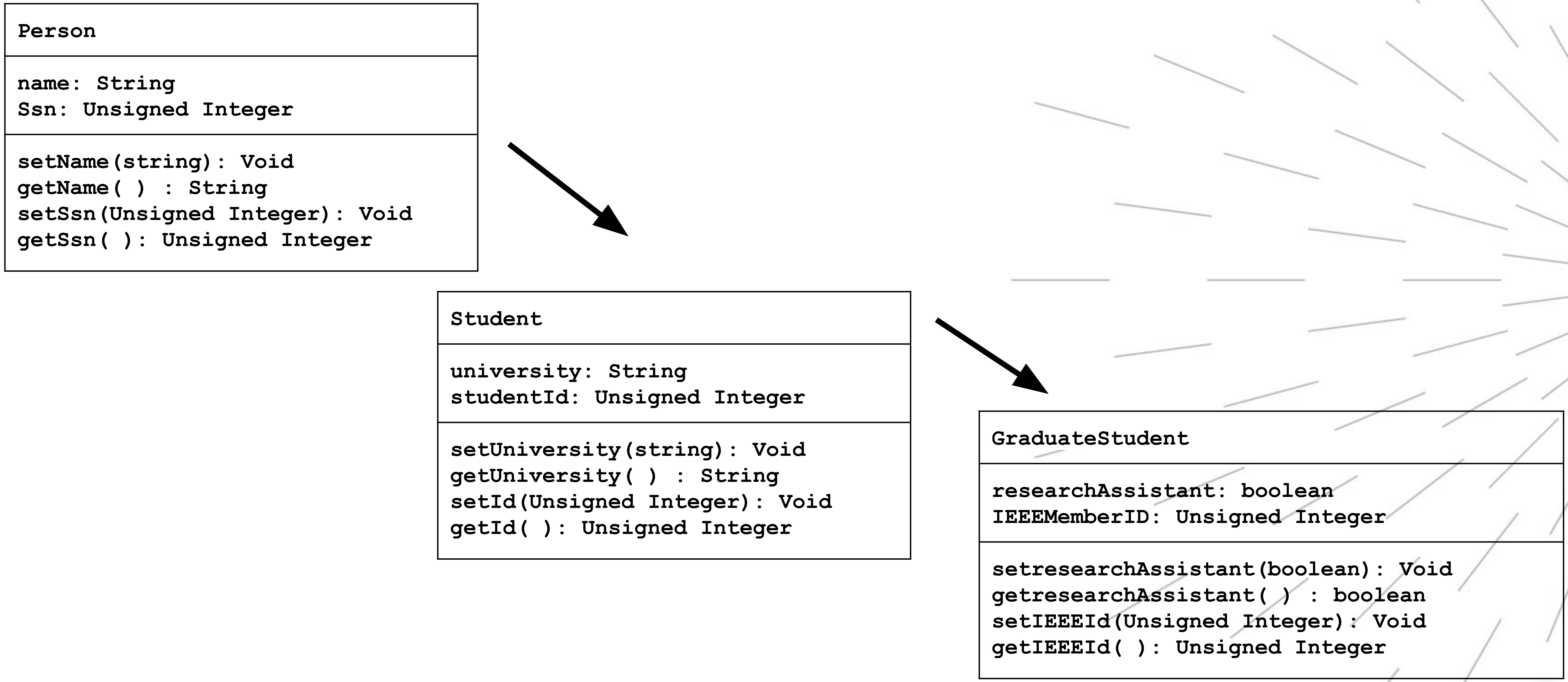
- The above polymorphism is function polymorphism. There is another polymorphism called as operator polymorphism. I am not going to cover it today, **however, you can look up it.**

THE UNIVERSITY OF ARIZONA

Inheritance

- Think about what is the dictionary meaning of inheritance.
- In OOP, inheritance is more or less the same.
- A child class, also known as subclass inheritance some properties from its parent class called as superclass.
- However, there is some different between dictionary meaning of inheritance and OOP inheritance.
- Inheritance can be public, private and protected

Inheritance: An abstract example



```
Person
name: String
Ssn: Unsigned Integer
setName(string): Void
getName( ) : String
setSsn(Unsigned Integer): Void
getSsn( ): Unsigned Integer
```

```
Student
university: String
studentId: Unsigned Integer
setUniversity(string): Void
getUniversity( ) : String
setId(Unsigned Integer): Void
getId( ): Unsigned Integer
```

```
GraduateStudent
researchAssistant: boolean
IEEEMemberID: Unsigned Integer
setresearchAssistant(boolean): Void
getresearchAssistant( ) : boolean
setIEEEId(Unsigned Integer): Void
getIEEEId( ): Unsigned Integer
```

Inheritenace

Inheritance mode

```
class student: public person
{
     private:
          unsigned int studentId;
          string university;
     public:
          void setStudentId(unsigned int sid)
          {
               this->studentId  = sid;
          }
          Void setUniversity(string uni)
          {
               this->university = uni;
          }
          string getUniversity()
          {
               return this->university;
          }
          unsigned int getStudentId()
          {
               Return this->studentId;
          }

}
```

There are three kind of inheritance mode:

1. **`public`**: public members of superclass become the public members of the subclass, protected members of the superclass become protected member of the subclass. Private members of the superclass cannot be directly accessed by subclass.

2. **`private`**: public and protected members of superclass become the private members of the subclass. Private members of the superclass cannot be directly accessed by subclass.

3. **`protected`**: public and protected members of superclass become the protected members of the subclass. Private members of the superclass cannot be directly accessed by subclass.

OOP in other languages

- OOP concept also exists in MATLAB and Python as well, however syntax are slightly different.

Example in MATLAB

Save this as **Person.m**

```
classdef Person

properties(GetAccess='private',SetAccess='private')
         name;
         ssn;
     end

     methods
         function obj = setName(obj, n)
             obj.name = n;
         end
         function obj = setSsn(obj, s)
            obj.ssn = s;
         end
         function SSN = getSsn(obj)
             SSN=  obj.ssn;
         end
         function nm = getName(obj)
             nm =  obj.name;
         end

         function printName(obj)
             disp(obj.name);
         end
     end
end %End of classdef
```

Implementation: save this file with any name ending with **.m**. Then run it.

```
p = Person;

p = p.setName('Thor');

p.getName()

p.printName();
```

Example in MATLAB: Inheritance

Save this as **Student.m**

```
classdef Student < Person


properties(GetAccess='private',SetAccess='private')
         studentId;
         university;
     end

     methods
         function obj = setUniversity(obj, uni)
             obj.university = uni;
         end
         function obj = setSid(obj, sId)
            obj.studentId = sId;
         end
         function SID = getSid(obj)
             SID=  obj.studentId;
         end
         function uni = getUniversity(obj)
             uni =  obj.university;
         end
     end

end %End of classdef
```

Implementation: save this file with any name ending with **.m.** Then run it.

```
s = Student;
s = s.setName('Loki');
s.getName()
s = s.setSid('2342022');
s.getSid()
```

Example in Python

To run code in python, install jupyter notebook.

- `sudo apt-get install jupyter-notebook`

Creating simplest class

```
class Person:
    pass #empty block - pass keyword does nothing
p = Person
print(p)
```

Example in Python

Functions in class definition with self

```
class Person:
    def say_hi(self):
        print('How are you?')
p = Person()
p.say_hi()
 # Previous 2 lines can also be written as Person().say_hi()
```

Using __init__

The __init__ method is run as soon as an object of a class is instantiated. This is like constructor in C++ and does the job of any initialization required for class members, etc.

```
class Person:
    def __init__(self, name):
        self.name = name
    def say_hi(self):
        print('Hello, my name is ', self.name)
p = Person('Wolverine')
p.say_hi()
```

Here, we define __init__ method with parameters name and self. Here, also by writing self.name, we created a member variable of class Person with the name name, although it is different from argument name being passed. Hence, a way of defining object variable is to write self.<variableName> in the __init__ function.

THE UNIVERSITY OF ARIZONA

Example in Python

Inheritance

In this example, the superclass is SchoolMember and subclass is Student.

```
class SchoolMember:
 '''Represents any school member'''
  def __init__(self, name, age):
      self.name = name
      self.age = age
      print('(Initialized SchoolMember:{})'.format(self.name))
  def tell(self):
      '''Tell my details'''
      print('Name:"{}", Age:"{}"'.format(self.name, self.age), end=" ")
```

```
# Class student inherits from SchoolMember
class Student(SchoolMember):
  '''Represents a student'''
  def __init__(self, name, age, marks):
      SchoolMember.__init__(self, name, age)
      self.marks = marks
      print('(Initialized Student:{})'.format(self.name))
  def tell(self):
      '''Tell my details'''
      print('Marks:"{}"'.format(self.marks))


s = Student('Cyclops', 24, 99)
```

Example in Python

More examples

```
class Vehicle:

   def __init__(self, wheels_num, tanktype, seat_num,
max_vel):

        self.wheels_num = wheels_num

        self.tanktype = tanktype

        self.seat_num = seat_num

         self.max_vel = max_vel
```

In python, there is a special way of writing getter and setter of a member variable

```
    @property

    def wheels_num(self):

        return self.__wheels_num

    @wheels_num.setter

    def wheels_num(self, number):

        self.__wheels_num = number

def make_noise(self):
        print('VRUUM VRUUM')
```

Example in Python

```python
toyota_prius = Vehicle(4, 'hybrid', 5, 100)
print(toyota_prius.wheels_num)
toyota_prius.wheels_num = 2
print(toyota_prius.wheels_num)
toyota_prius.make_noise()
```

But this is not enough

- This was a crash course in OOP with some examples in C++, MATLAB and Python
- I have not covered static variables, constructors, destructors, friend functions, etc.
- I encourage you to look up an learn about it.

Some useful references

- https://www.udacity.com/course/c-for-programmers--ud210
- https://www.mathworks.com/help/matlab/matlab_oop/hierarchies-of-classes-concepts.html
- http://www.archer.ac.uk/training/course-material/2018/02/oofortran-daresbury/Lectures/L02-IntroductionToOO.pdf
- https://www.coursera.org/specializations/data-science-python