THE UNIVERSITY OF WESTERN AUSTRALIA

Lecture 24
Objects

Objectives of this Lecture

- To get familiar with Objects
- To understand the concept of objects and how they can be used to simplify programs
- Understand that in Python, everything is actually an object

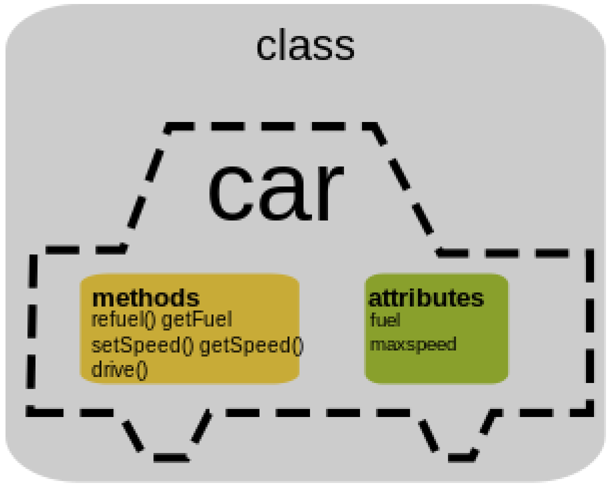
Overview

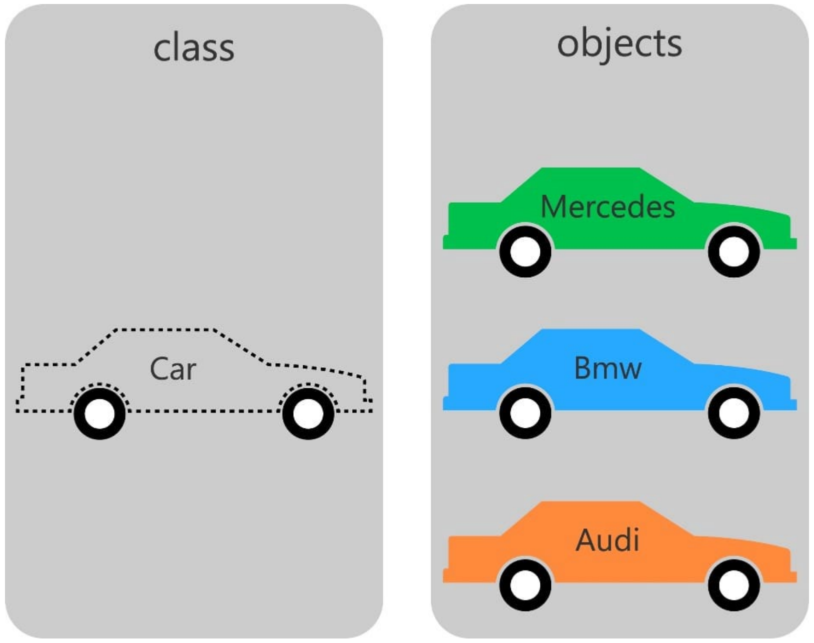
- So far, we saw that each data type can represent a certain set of values, and each had a set of associated operations.
- The traditional programming view is that data is passive – it is manipulated and combined using active operations.
- Modern computer programs are built using an object-oriented approach.

Objects and Object Oriented Programming

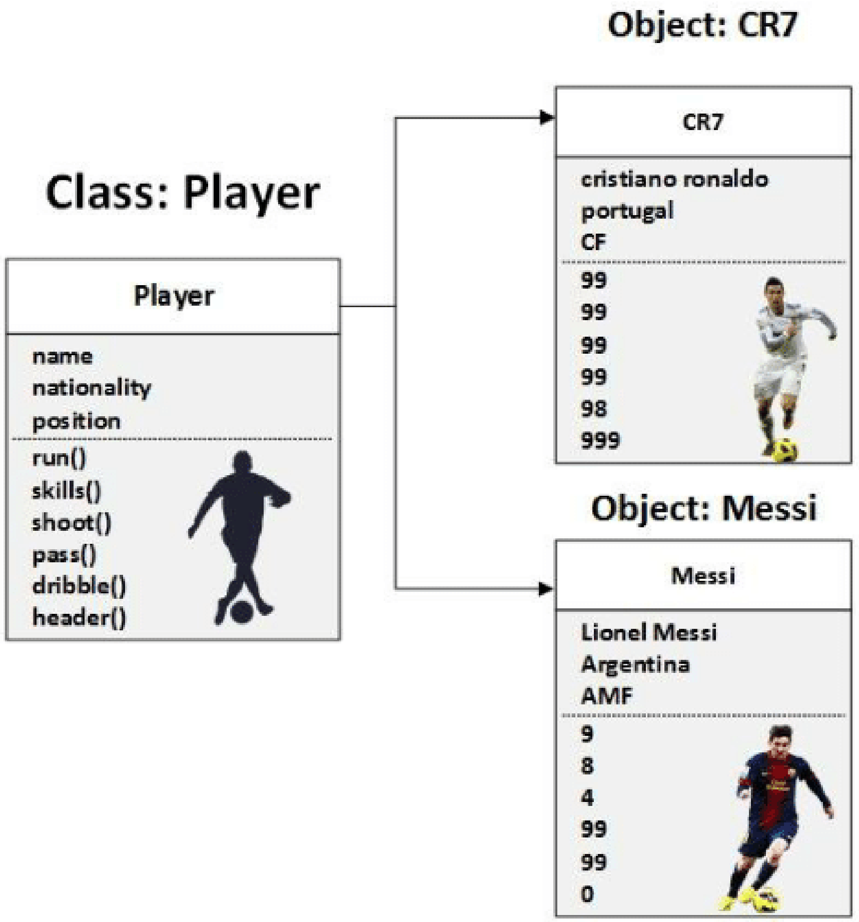
- Basic idea – view a complex system as the interaction of simpler objects.
- An object is a kind of active data type that combines data and operations.
  - *Objects know stuff (contain data) and they can do stuff (have operations).*
- Objects interact by sending each other messages *(requests to do stuff)*.

OOP concept


class
car
methods
refuel() getFuel
setSpeed() getSpeed()
drive()
attributes
fuel
maxspeed


class
Car
objects
Mercedes
Bmw
Audi

Example



How to learn OOP using football

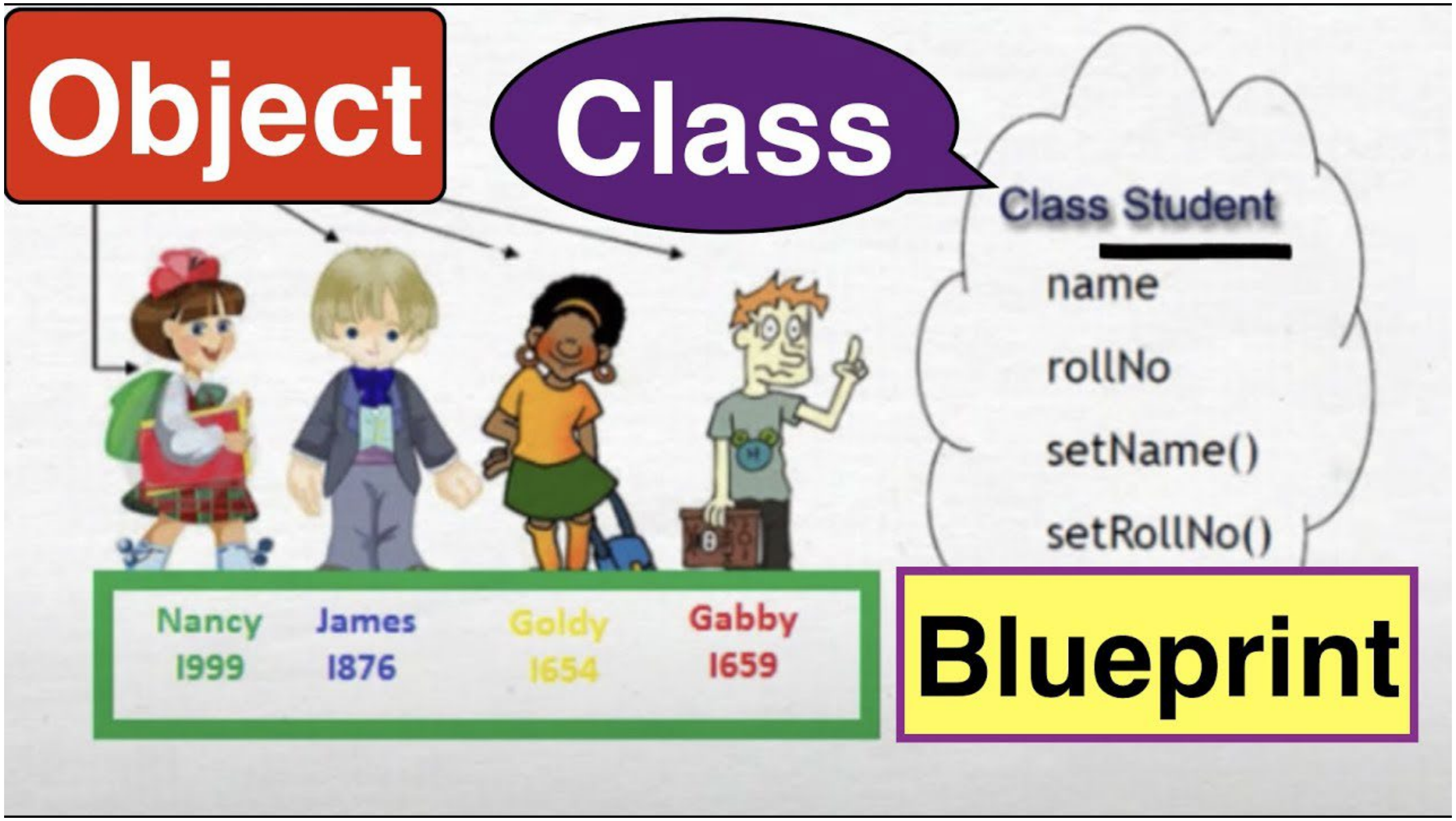
Creating a New object

- To create a new object of a class, we use a special operation called a *constructor*. □□□
  <class-name>(<param1>, <param2>, …)
- A <class-name> is the name of the class we want to create a new object of, e.g., `Car` or `Player`.
- *The parameters are required to initialize the object. For example,* `Player` *requires three values: name, nationality and position.*
  - `Player("Messi", "Argentina", "CF")`

Static and non-static

- Static (also known as class properties)
  - *Variables and methods which are shared by all the objects*
- Non-static
  - *Variables and methods belonging to individual objects*
- Think about car class. Which one is car ? Why or why not ?

Example (2)


Object
Class
Class Student
name
rollNo
setName()
setRollNo()
Nancy
1999
James
1876
Goldy
1654
Gabby
1659
Blueprint

Writing a class

```
class student:
    university = "University of Western Australia" # static attribute
    def __init__(self,name,rn):  # constructor
        self.name = name       # attribute
        self.rollNo = rn    # attribute
    def setName(self,nam):          # method
        self.name = nam
    def setRollNo(self,rn):             # method
        self.rollNo = rn
    def printName(self):
        print("The name of student is:",self.name)
    def printRollNo(self):
        print("The roll number of student",self.name,"is:",self.rollNo)
    def setUni(self,uni):
        student.university = uni  #static attributes are accessed via class name
```

Create object and access attributes and methods

```
>>> s1 = student("Ali",32131)
>>> s2 = student("Sara",11111)
>>> s1.printName()
The name of student is: Ali
>>> s2.printRollNo()
The roll number of student Sara is: 11111
>>> s1.setName("Chris")
>>> s1.printRollNo()
The roll number of student Chris is: 32131
>>> print(s1.university)
University of Western Australia
>>> print(s2.university)
University of Western Australia
>>> s1.setUni("UWA")
>>> print(s1.university)
UWA
>>> print(s2.university)
UWA
```

Objects: Explained with an Example

- Suppose we want to develop a data processing system for a university.
- We must keep records of students who attend the university.
- We also need to keep records for units offered in the university.
- Each student is enrolled in units while each unit has students.

Attributes of student and unit classes

- What information would be in a student object?
  - *Name*
  - *Roll number*
  - *Units enrolled by the student*
- What information would be in a unit object?
  - *Name*
  - *Code*
  - *Students enrolled in the unit*

Methods for student and unit classes

- The student object should do:
  - *Set attributes*
  - *Print attributes*
  - *Enrol student in units*
  - *Print list of units enrolled by the student*
- The unit object should do:
  - *Set attributes*
  - *Print attributes*
  - *Add students in the unit*
  - *Print list of students enrolled in the unit*

Objects within Objects

- An object can have one or more objects inside it
- For example, the unit-object will have student-objects inside it
- Similarly, the student-object will have unit-object in it.

Student class

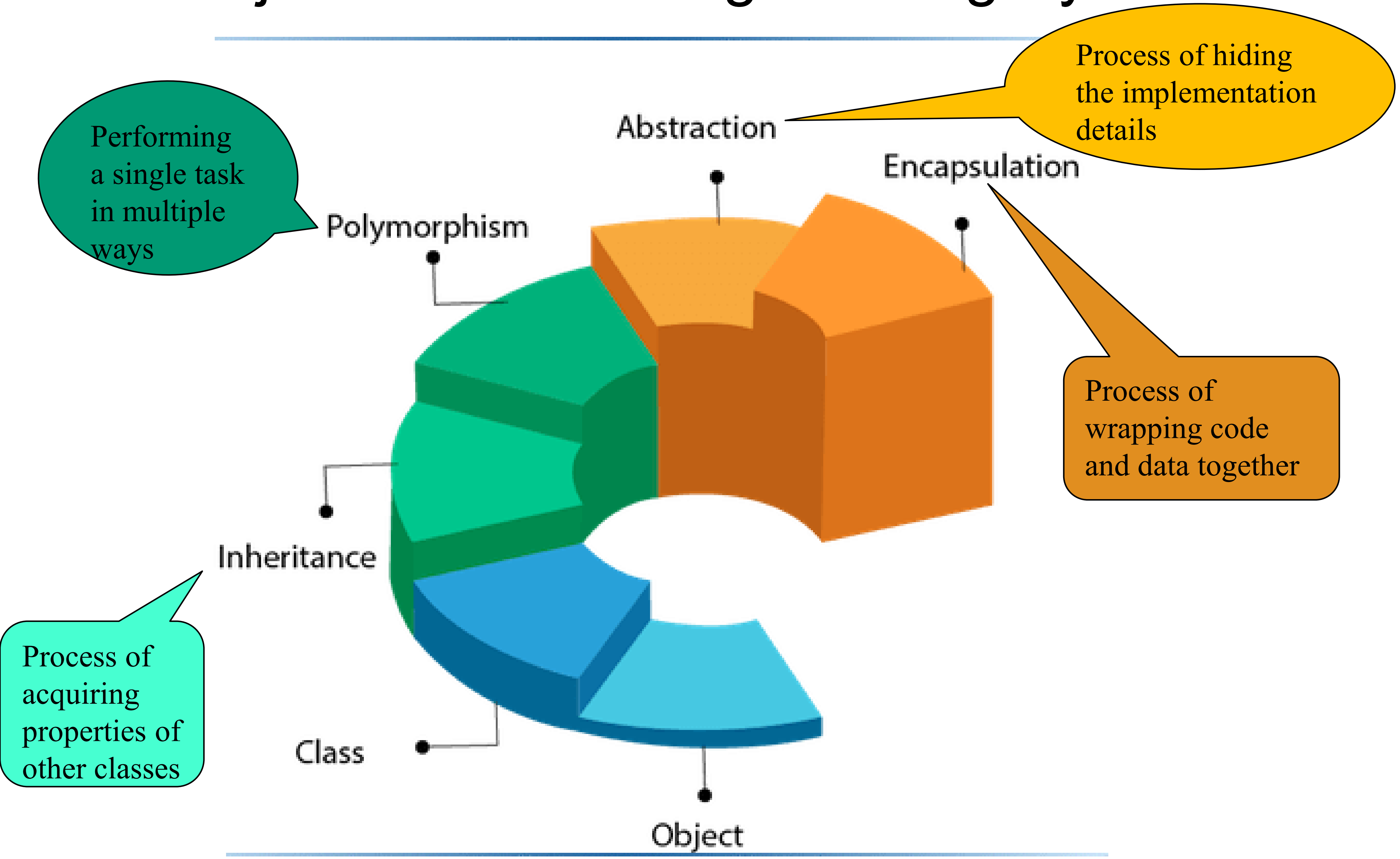
```
class student:
    university = "University of Western Australia"
    def __init__(self,name,rn):
        self.name = name
        self.rollNo = rn
        self.units = []
    def setName(self,nam):
        self.name = nam
    def setRollNo(self,rn):
        self.rollNo = rn
    def printName(self):
        print("The name of student is:",self.name)
    def printRollNo(hmm):
        print("The roll number of student",hmm.name,"is:",hmm.rollNo)
    def setUni(self,uni):
        student.university = uni
    def enrolUnit(self,unit):
        self.units.append(unit)
    def printUnits(self):
        print("The units enrolled by",self.rollNo,self.name,"are:")
        for unit in self.units:
            print(unit.code,unit.name)
```

Unit class

```
class unit:
    def __init__(self,code,name):
        self.code = code
        self.name = name
        self.students = []
    def setname(self,name):
        self.name = name
    def setcode(self,code):
        self.code = code
    def printUnit(self):
        print("The unit code is",self.code,"and name is",self.name)
    def addStudent(self,stu):
        self.students.append(stu)
    def printstudents(self):
        print("Students enrolled in",self.code,self.name,"are:")
        for st in self.students:
            print(st.name)
```

```
>>> s1 = student("Ali",32131)
>>> s2 = student("Tim",11111)
>>> u1 = unit("CITS1401","Computational Thinking with Python")
>>> u2 = unit("CITS4403","Computational Modelling")
>>> s1.enrolUnit(u1)
>>> s2.enrolUnit(u1)
>>> s2.enrolUnit(u2)
>>> u1.addStudent(s1)
>>> u1.addStudent(s2)
>>> u2.addStudent(s2)
>>> s2.printUnits()
The units enrolled by 11111 Tim are:
CITS1401 Computational Thinking with Python
CITS4403 Computational Modelling
>>> u1.printstudents()
Students enrolled in CITS1401 Computational Thinking with Python are:
Ali
Tim
```

Object Oriented Programming System


Process of hiding the implementation details
Abstraction
Encapsulation
Performing a single task in multiple ways
Polymorphism
Process of wrapping code and data together
Inheritance
Process of acquiring properties of other classes
Class
Object

Summary

- We learned some basics of Object Oriented programming.
- We learned what are objects and how to use them in our programs.
- We learned the difference between classes and objects.
- We haven't learned about many aspects of OOP. This will be covered in the unit CITS2004 "Object Oriented Programming".