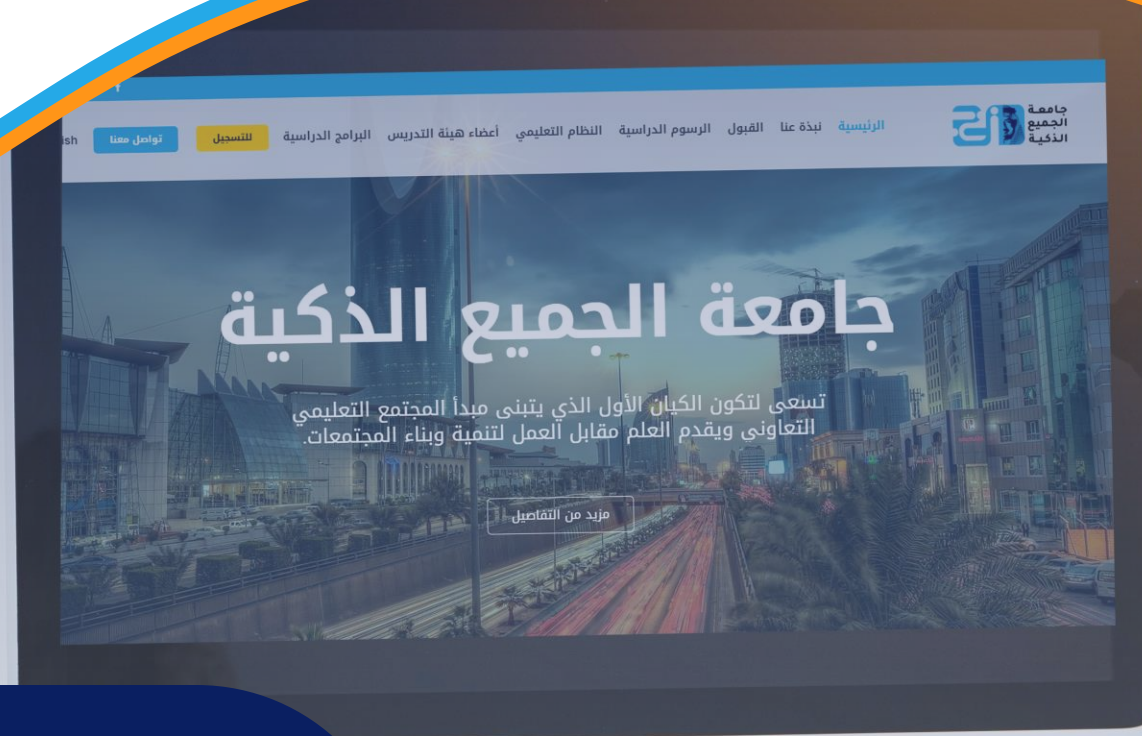




جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY



Object-Oriented Programming II
Prepared by Dr. Hamada I. AbdulWakel
Lecture 9

CYB550
Programming Foundations In
Python

- Learn more classes behaviors.
- Explain how to use operator overloading.
- Explain the meaning and uses of inheritance and polymorphism.
- Explain how to handle exceptions.

- The `getattr` is a built-in function, not a method on the object: it takes the object as its first parameter. The second parameter is the name of the variable as a string.
- The `setattr` allows us to set the value of an attribute.
- The `hasattr` detects whether an attribute exists.

- Example

```
class test:
```

```
    def __init__(self): self.x = 2; self.y = 1
```

```
t1 = test();
```

```
print(getattr(t1, 'x')); print(setattr(t1, 'y', 5))
```

```
print(getattr(t1, 'y')); print(hasattr(t1, 'z'))
```



- You can create list of objects by appending class instances to list.
- Example

```
class test:
```

```
    def __init__(self): self.x = 2; self.y = 1
```

```
Testarray = []
```

```
t1 = test(); t2 = test();
```

```
Testarray.append(t1); Testarray.append(t2); #or Testarray[test(),test()]
```

```
for obj in Testarray:
```

```
    print(obj.x)
```



- You can change the meaning of an operator in Python depending upon the operands used.
- For example, the + operator will perform arithmetic addition on two numbers, merge two lists, or concatenate two strings.
- Operators to overload:
 - + by `__add__()`
 - - by `__sub__()`
 - * by `__mul__()`
 - / by `__div__()`



- Example

```
class Point:
```

```
    def __init__(self, x=0, y=0):  
        self.x = x;    self.y = y
```

```
p1 = Point(1, 2)
```

```
p2 = Point(2, 3)
```

```
print(p1+p2) #TypeError: unsupported operand type(s) for +: 'Point' and 'Point'
```

- Example: Overloading the + Operator

class Point:

```
def __init__(self, x=0, y=0):
```

```
    self.x = x;    self.y = y
```

```
def __add__(self, other):
```

```
    x = self.x + other.x;    y = self.y + other.y;    return Point(x, y)
```

```
p1 = Point(1, 2);    p2 = Point(2, 3);    print(p1+p2)
```

- Python does not limit operator overloading to arithmetic operators only. We can overload comparison operators as well.
- Logical operators to overload:
 - < by `__lt__()`
 - <= by `__le__()`
 - == by `__eq__()`
 - != by `__ne__()`
 - > by `__gt__()`
 - >= by `__ge__()`

- Example: Overloading the + Operator

class Point:

```
def __init__(self, x=0, y=0):
```

```
    self.x = x;    self.y = y
```

```
def __lt__(self, other):
```

```
    self_cords = (self.x ** 2) + (self.y ** 2)
```

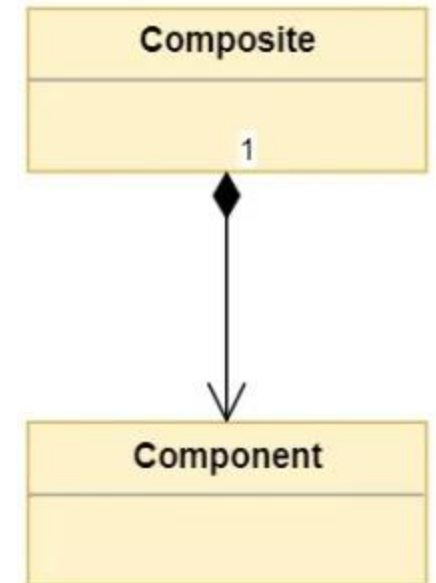
```
    other_cords = (other.x ** 2) + (other.y ** 2);    return self_cords < other_cords
```

```
p1 = Point(1, 2);    p2 = Point(2, 3);    print(p1<p2)
```



- **Inheritance** and **composition** are two major concepts in OOP that model the relationship between two classes.
- Both enable code reuse, but they do it in different ways.
- Use class object inside another class

- Composition is a concept that models a **has a** relationship.
- It enables creating complex types by combining objects of other types.
- This means that a class Composite can contain an object of another class Component.



- Example:

```
class Course:
```

```
    def __init__(self, name):  
        self.cname = name;    self.hours = 3
```

```
class Student:
```

```
    def __init__(self, name):  
        self.course = Course("OOP"); self.name = name
```

```
s = Student("ahmed")
```

```
print(s.course.cname)
```



- Use class object inside another class

```
from test import Course
```

```
class Student:
```

```
    def __init__(self, name):
```

```
        self.course = Course("OOP")
```

```
        self.name = name
```

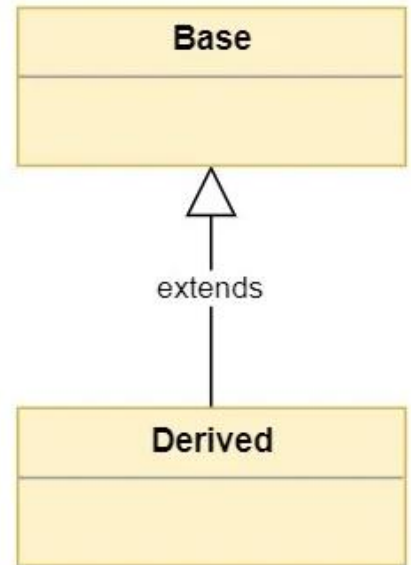
```
s = Student("ahmed")
```

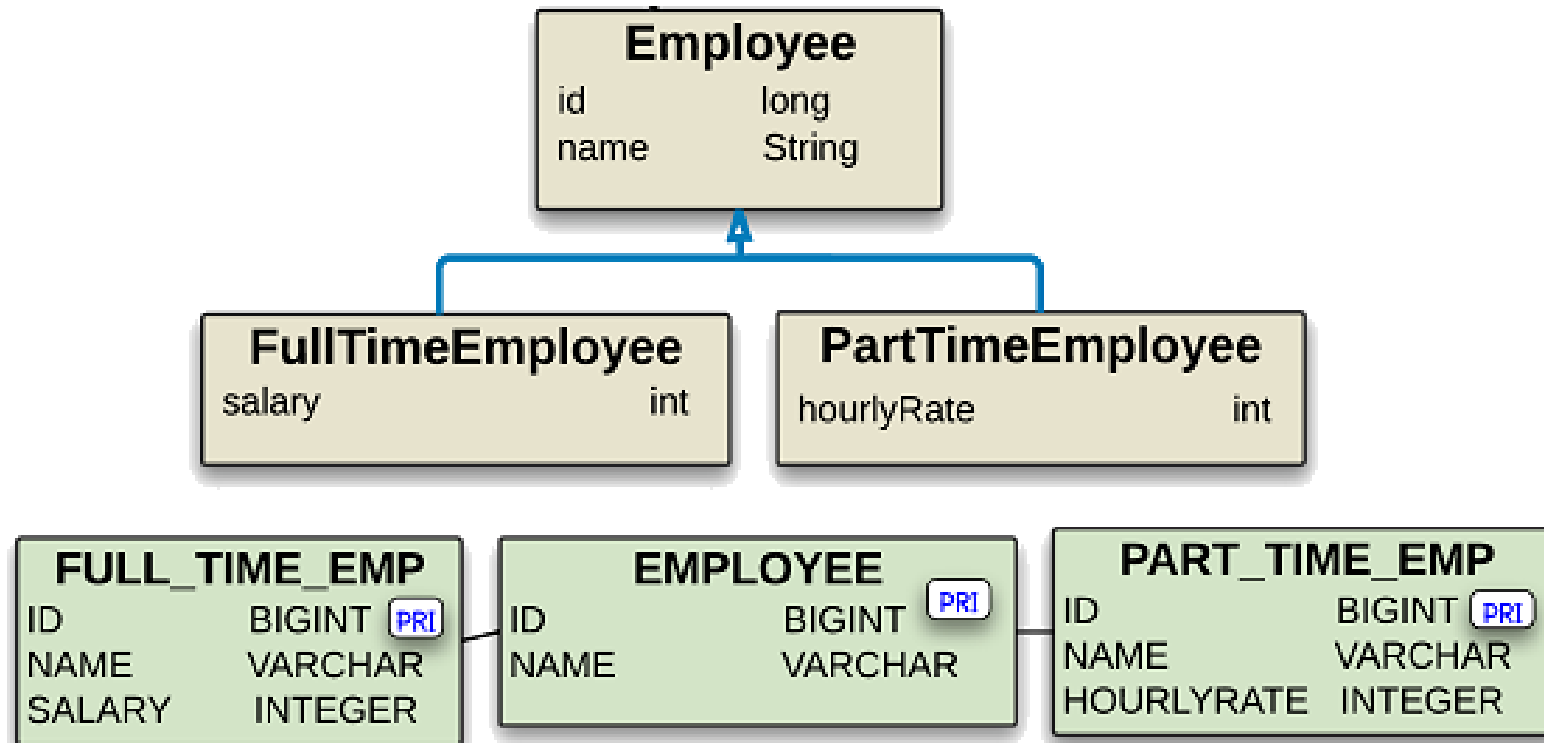
```
print(s.course.cname)
```



- Inheritance models what is called an **is a** relationship.
- This means that when you have a Derived class that inherits from a Base class, you created a relationship where Derived is a specialized version of Base.

```
class DrivedClassName(BaseClassName):  
    # body
```





- Example

class Point:

```
def __init__(self, x, y):  
    self.x = x;    self.y = y
```

```
def __str__(self):  
    return f"The point is located in ({self.x}, {self.y})"
```

```
def displayX(self):  
    print("The point x coordinate = ", self.x)
```

- Example

```
class Circle(Point):  
    def __init__(self, r, x, y):  
        Point.__init__(self, x, y);          self.radius = r  
    def __str__(self):  
        print(Point.__str__(self)); return f"The circle radius = {self.radius}"
```

```
c1 = Circle(2.5, 2, 3); print(c1); c1.displayX()  
print(isinstance(c1,Circle)); issubclass(Circle, Point)
```



- Polymorphism is taken from the Greek words **Poly** (many) and **morphism** (forms). It means that the same function name can be used for different types.
- There are different methods to use polymorphism:
 - Using functions.
 - Using class methods.

- You can create a function that can take any object, allowing for polymorphism.
- Create a function called func() which will take **an object**. Now, let's give the function something to do that uses the 'obj' object we passed to it.

- Example

class Tomato:

```
def type(self): print("Vegetable")
```

```
def color(self): print("Red")
```

class Apple:

```
def type(self): print("Fruit")
```

```
def color(self): print("Red")
```



```
def func(obj):  
    obj.type()  
    obj.color()
```

```
obj_tomato = Tomato()  
obj_apple = Apple()  
func(obj_tomato)  
func(obj_apple)
```



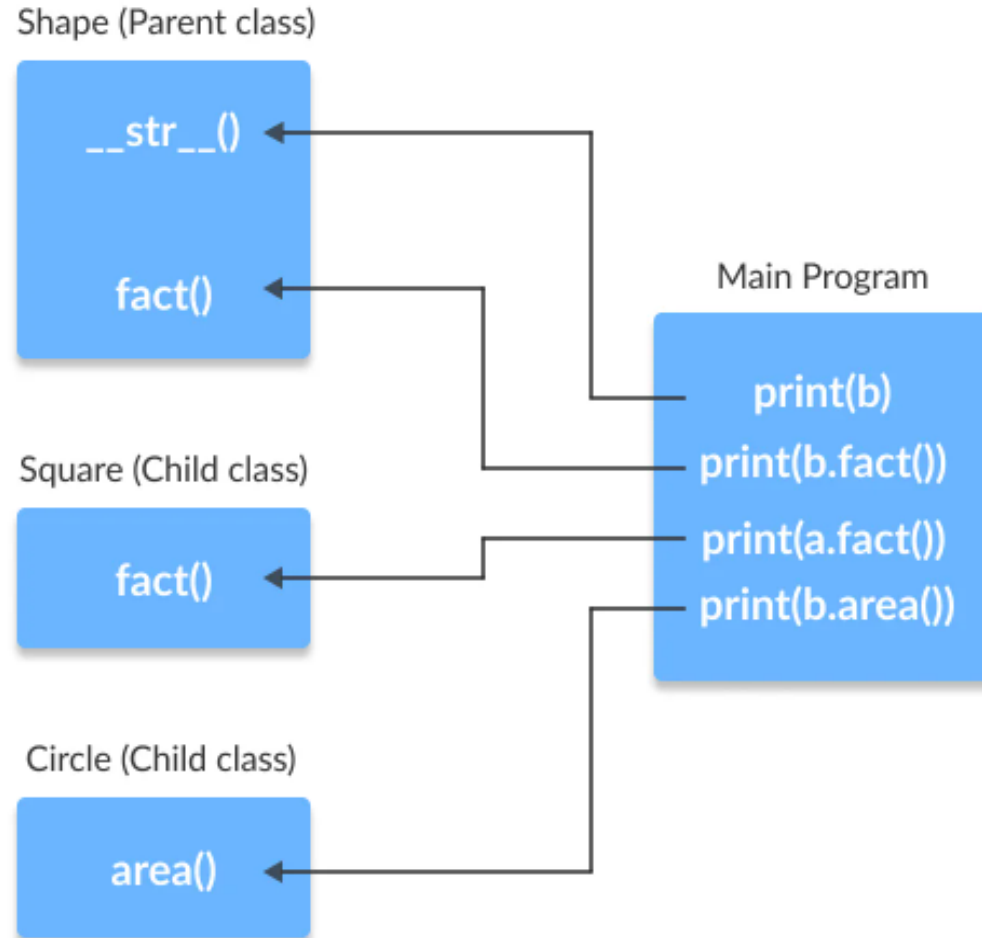
- Polymorphism defines methods in the child class that have the same name as the methods in the parent class.
- In inheritance, the child class inherits the methods from the parent class. Also, it is possible to modify a method in a child class that it has inherited from the parent class.
- This process of re-implementing a method in the child class is known as **Method Overriding**.



Lecture 9

Polymorphism with Inheritance

Cont'd



- Example

```
class Bird:
```

```
    def intro(self):    print("There are different types of birds")
```

```
    def flight(self):   print("Most of the birds can fly but some cannot")
```

```
class parrot(Bird):
```

```
    def flight(self):   print("Parrots can fly")
```

```
class penguin(Bird):
```

```
    def flight(self):   print("Penguins do not fly")
```



- Example

```
obj_bird = Bird();    obj_parr = parrot();    obj_peng = penguin()
```

```
obj_bird.intro();    obj_bird.flight()
```

```
obj_parr.intro();    obj_parr.flight()
```

```
obj_peng.intro();    obj_peng.flight()
```



- Errors (e.g., Syntax error) are the problems in a program due to which the program will stop the execution.
- Exceptions are raised when some internal events occur which changes the normal flow of the program.
- **Exception** is the base class for all the exceptions in Python.

- Example

```
marks = 10000
```

```
# perform division with 0
```

```
a = marks / 0
```

```
print(a) #ZeroDivisionError
```

- You can choose to throw an exception if a condition occurs.
- To throw (or raise) an exception, use the **raise** keyword.
- You can define what kind of error to raise, and the text to print to the user.

- Example

`x = -1`

`if x < 0:`

`raise Exception("Sorry, no numbers below zero")`



- Example

```
x = "hello"
```

```
if not type(x) is int:
```

```
    raise TypeError("Only integers are allowed")
```



- Example

```
a = int(input("Enter a positive integer: "))
```

```
if a <= 0:
```

```
    raise ValueError("That is not a positive number!")
```

- **Try** and **except** statements are used to catch and handle exceptions.
- Statements that can raise exceptions are kept inside the **try clause** and the statements that handle the exception are written inside **except clause**.
- A try statement can have more than one except clause, to specify handlers for different exceptions.

- Example

```
a = [1, 2, 3]
```

```
try:
```

```
    print("Second element = ", a[1])
```

```
    # Throws error since there are only 3 elements in array
```

```
    print("Fourth element = ", a[3])
```

```
except:
```

```
    print ("An error occurred")
```



- Example

```
def fun(a):
```

```
    if a < 4:  b = a/(a-3)
```

```
    print("Value of b = ", b)
```

```
try:
```

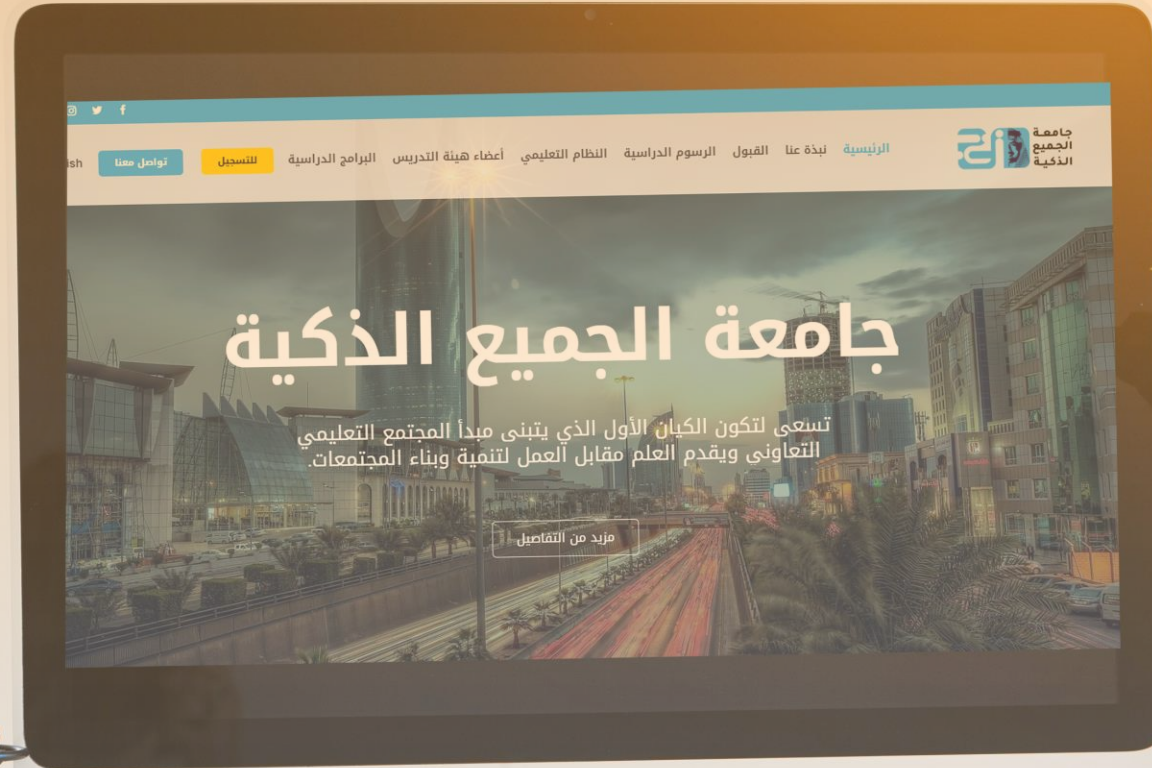
```
    fun(3)
```

```
    fun(5)
```

```
except ZeroDivisionError:  print("ZeroDivisionError Occurred and Handled")
```

```
except NameError:  print("NameError Occurred and Handled")
```





جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

THANK YOU