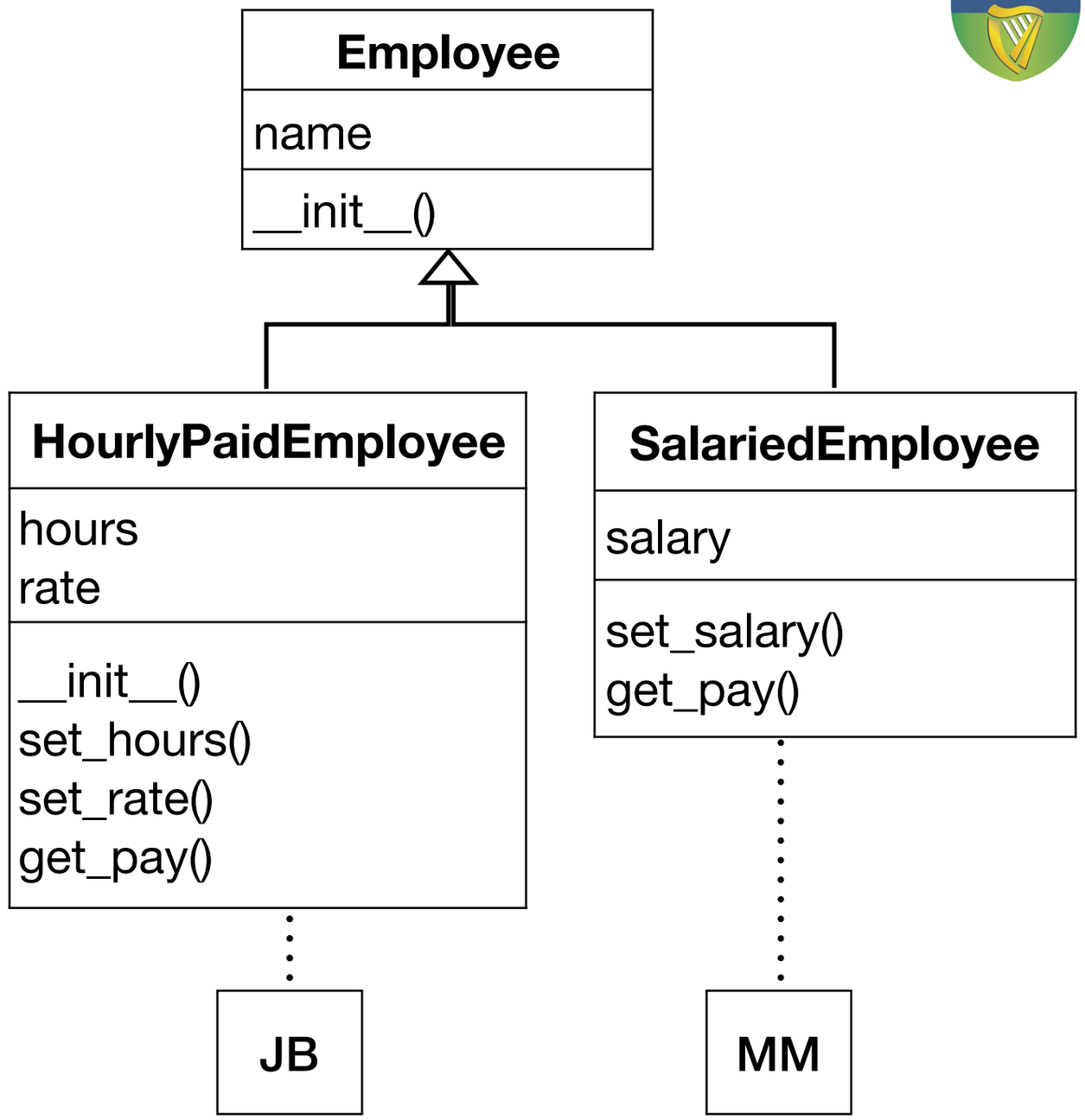
Principles of OOP

- **Encapsulation**
  - Encapsulation is the mechanism of hiding of data implementation by restricting access to public methods
- **Inheritance**
  - Inheritances expresses "is a" relationship between two objects. Using proper inheritance, In derived classes we can reuse the code of existing super classes
- **Polymorphism**
  - It means one name many forms. Details of what a method does will depend on the object to which it is applied.
- Also
  - **Instantiation**
  - **Abstraction**
  - **Modularity**

First Example

```
class Employee():
    def __init__(self, name):
        self.name = name

class HourlyPaidEmployee(Employee):

    def __init__(self, name):
        Employee.__init__(self, name)
        self.hours = 0
        self.rate =

    def set_hours(self, hours):
        self.hours = hours

    def set_rate(self, r):
        self.rate = r

    def get_pay(self):
        return self.rate * self.hours

class SalariedEmployee(Employee):

    def set_salary(self, sal):
        self.salary = sal

    def get_pay(self):
        return self.salary / 12
```




```
JB = HourlyPaidEmployee("Joe Bloggs")
MM = SalariedEmployee("Marvelous Mary")
JB.set_hours(121)
JB.set_rate(10.50)
MM.set_salary(45000)
```

Salary Example

```
JB = HourlyPaidEmployee("Joe Bloggs")
MM = SalariedEmployee("Marvelous Mary")
SB = SalariedEmployee("Sally Bloggs")
JB.set_hours(121)
JB.set_rate(10.50)
MM.set_salary(45000)
SB.set_salary(54000)
```

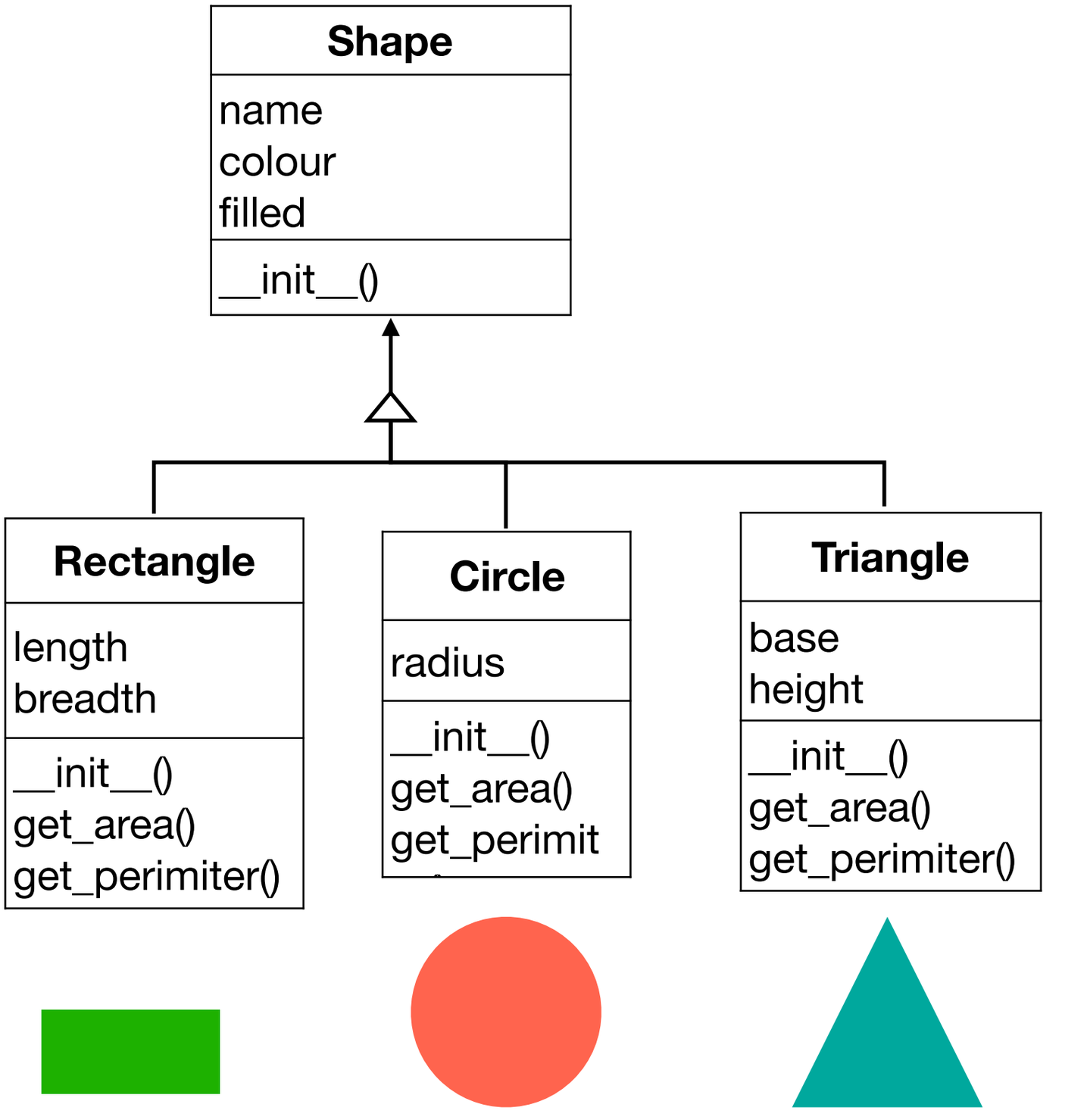
Function that actually gets called depends on class of member

```
In [22]:
gang = (SB,MM,JB)
for member in gang:
    print(member.name, "gets", member.get_pay(), "per month.")
```

```
Sally Bloggs gets 4500.0 per month.
Marvelous Mary gets 3750.0 per month.
Joe Bloggs gets 1270.5 per month.
```

Shape Example

- Polymorphic methods
  - get_area()
    get_perimiter()



Shape Example

```
r1 = Rectangle('R1', 10, 2)
r2 = Rectangle('R2', 10, 5)
r1.filled = True
r1.color = "orange"

c1 = Circle('C1', 12)
c2 = Circle('C2', 10)
c1.filled = True
c1.color = "blue"

t1 = Triangle('T1', 10, 5)
t2 = Triangle('T2', 10, 8)

shapes = [r1,c1,t1,r2,c2,t2]

for s in shapes:
    print(s.__class__.__name__, s.name, 'has area %4.1f' %(s.get_area()))
```

```
Rectangle R1 has area 20.0
Circle C1 has area 452.4
Triangle T1 has area 25.0
Rectangle R2 has area 50.0
Circle C2 has area 314.2
Triangle T2 has area 40.0
```

Different method called for each Shape class

Road Tax Example

- Ireland has two rules for calculating road tax
  - Cars registered after July 2008
    - based on CO2 emissions
  - Cars registered before July 2008
    - based on engine size in CC
    - http://www.mywheels.ie/motor-tax-rates-ireland/

CO2 Model	
Price	CO2
€120.00	0 - 1
€170.00	2 - 80
€180.00	81 - 100
€190.00	101 - 110
€200.00	111 - 120
…	…

CC Model	
Price	CC
€199	0 - 1000
€299	1001 - 1100
€330	1101 - 1200
€358	1201 - 1300
€385	1301 - 1400
…	…

Road Tax Example

- Create classes `Car`, and subclasses `PreJul2008` and `PostJul2008`.
  - The subclasses have a method `get_tax_rate`.
  - `PreJul2008` has an attribute `CC`
  - `PostJul2008` has an attribute `CO2`
- Write a function `total_tax` that will take a list of car objects as argument and return the total tax bill for these cars.
- Test it on a fleet with two pre July 2008 cars (1250CC and 1400CC) and a post July 2008 car with an emission level of 110.

Road Tax Example

```
class Car():
    pass

class PreJul2008(Car):
    def __init__(self, CC):
        self.CC = CC
    def get_tax_rate(self):
        if self.CC < 1001:
            return 199
        elif self.CC < 1101:
            return 299
        elif self.CC < 1201:
            return 330
        elif self.CC < 1301:
            return 358
        elif self.CC < 1401:
            return 385
```

```
class PostJul2008(Car):
    def __init__(self, co2):
        self.co2 = co2
    def get_tax_rate(self):
        if self.co2 < 2:
            return 120
        elif self.co2 < 81:
            return 170
        elif self.co2 < 101:
            return 180
        elif self.co2 < 111:
            return 190
        elif self.co2 < 121:
            return 200
```

What is bad about this code?

```
my_wreck = PreJul2008(1250)
my_wreck.get_tax_rate()
Out[4]:
358
```

Road Tax Example

```
my_wreck = PreJul2008(1250)
my_wheels = PostJul2008(110)
my_first_car = PreJul2008(1400)

my_fleet = [my_wreck, my_wheels, my_first_car]

def total_tax(fleet):
    tax = 0
    for car in my_fleet:
        tax += car.get_tax_rate()
    return tax
In [7]:
total_tax(my_fleet)
Out[7]:
933
```

Car Tax Example V2

- Modify the code in the Car Tax Example so that the limits and prices are not hardwired into the code. Instead these should be loaded from parameter files.
  - Sample parameter files are available on the Moodle page (`CC_limits.dat` and `co2_limits.dat`).
  - Change both `get_tax_rate` methods to use these parameter files.
  - **Hint:** Load the limits data into two dictionaries and store these dictionaries as class attributes.

Given

1. Write code to load both dictionaries from the data files.
2. Add these dictionaries as class variables in the classes.
3. Rewrite the get_tax_rate methods to use the dictionaries.
4. Extend the sample data in the limit files and test again, i.e. add one or two more limits.

Reading Parameter Files

```
def load_dict_file(d,file):
    with open(file, 'r') as co2_file:
        for line in co2_file.readlines():
            line = line.strip()
            # This next line is a bit complicated,
            # it might be clearer to use intermediate variables
            # to store the result of the split.
            a, d[a] = list(map (int, line.split(',')))
            print(a,d[a])
In [4]:
co2d = dict()
ccd = dict()
load_dict_file(co2d,'co2_limits.dat')
print('=========')
load_dict_file(ccd,'CC_limits.dat')
```

```
1 120
80 170
100 180
110 190
120 200
=========
1001 199
1101 299
1201 330
1301 358
1401 385
```

Dictionaries - Assignment

- What happens when a dictionary appears on the right hand side of an assignment? e.g. d2 = d1?
  - Is it a copy or a new handle?
  - Hint: change d1 after assignment and check d2 or use id() function.
- What happens with tuples?
- What happens with sets?

Polymorphism - Summary

- Redefining methods for sub-classes
- Method that is invoked depends on the sub-class
- Effectively ***Method Overriding***

- Other OOP languages such as Java have ***Method Overloading***
  - An object can have more than one version of a method
  - e.g. constructors with different arguments
- No Method Overloading in Python
  - Next lecture…