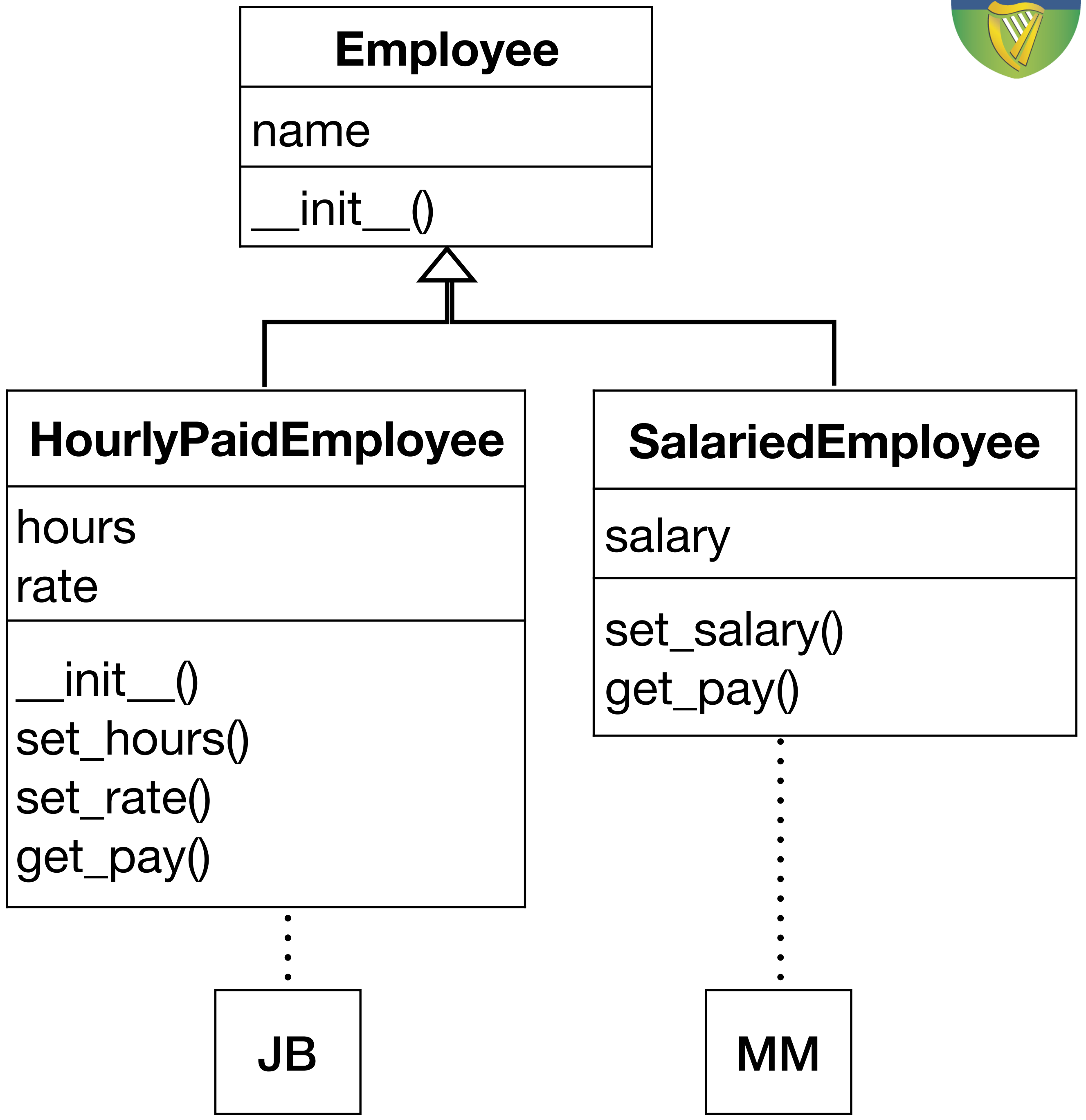
Principles of OOP

- **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

```python
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 = 0

    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
```



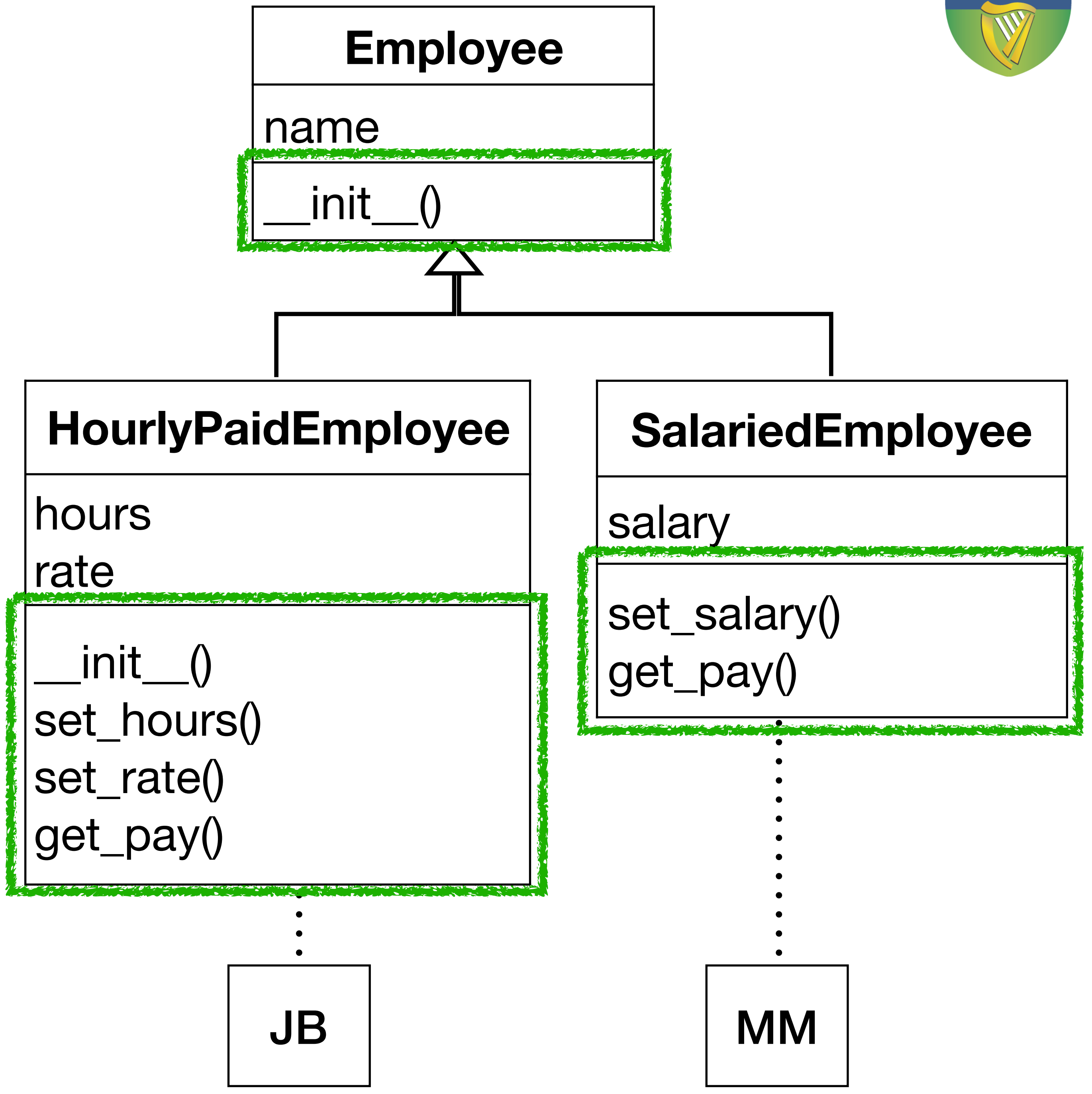
```python
JB = HourlyPaidEmployee("Joe Bloggs")
MM = SalariedEmployee("Marvelous Mary")
JB.set_hours(121)
JB.set_rate(10.50)
MM.set_salary(45000)
```

WHISTLE
STOP

Encapsulation

```
class Employee(
    def __init_
        self.na

class HourlyPa

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

    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
```

Implementation details are 'hidden' within the object. Object is accessed through methods

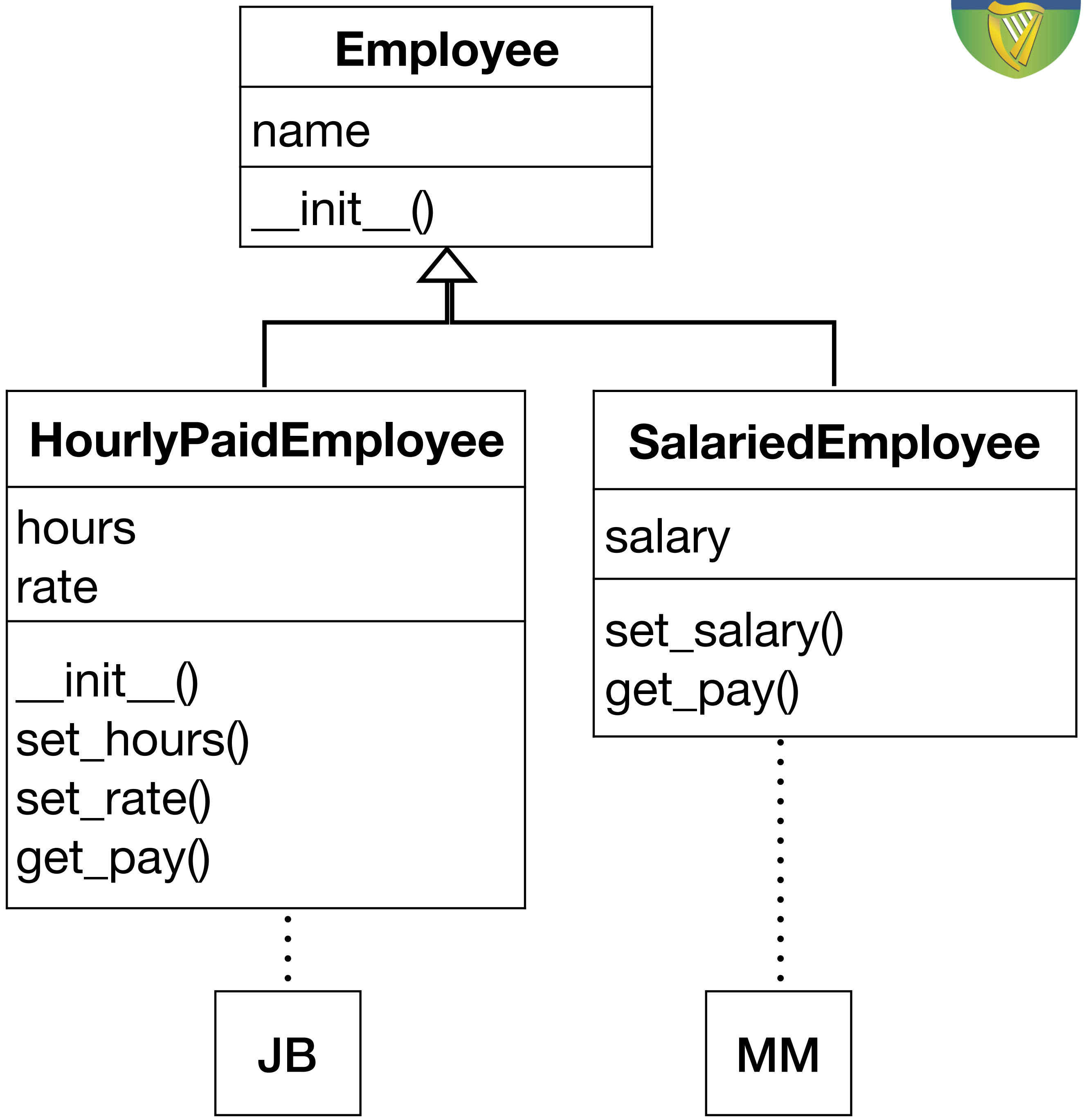


```
JB = HourlyPaidEmployee("Joe Bloggs")
MM = SalariedEmployee("Marvelous Mary")
JB.set_hours(121)
JB.set_rate(10.50)
MM.set_salary(45000)
```

Inheritance

```
class Employee(
    def __init_
        self.na

class HourlyPai

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

    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
```

HourlyPaidEmployee & SalariedEmployee are **subclasses** of Employee
They inherit data & methods from their **superclass**



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

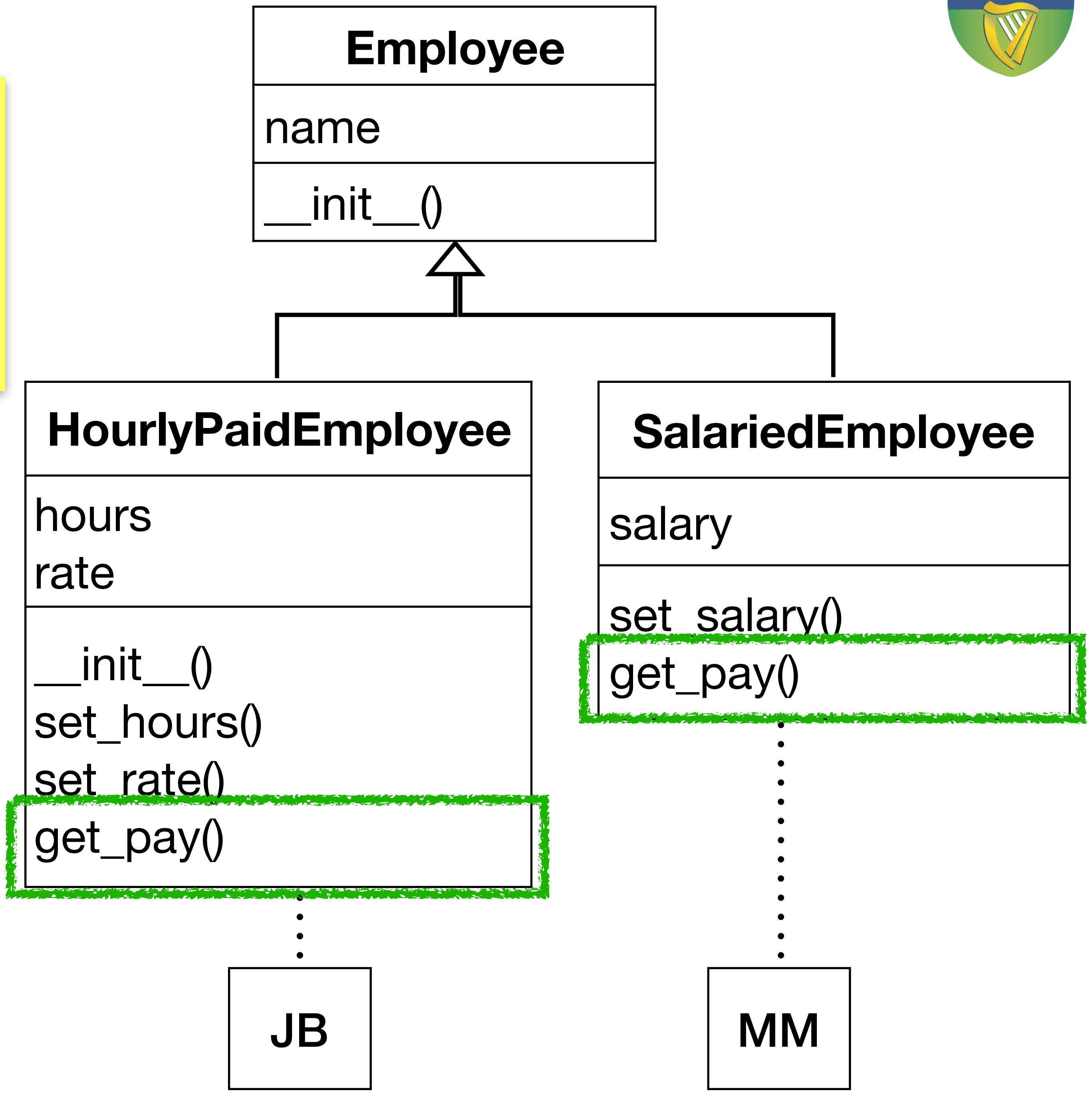
Polymorphism

```
class Employee(
    def __init_
        self.na

class HourlyPai

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

    def set_hours(sel
        self.hours =

    def set_rate(self
        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
```

One name, many meanings.

Different get_pay() methods for each class.

```
p1 = JB.get_pay()
p2 = MM.get_pay()
p1,p2
Out[7]:
(1270.5, 3750.0)
```



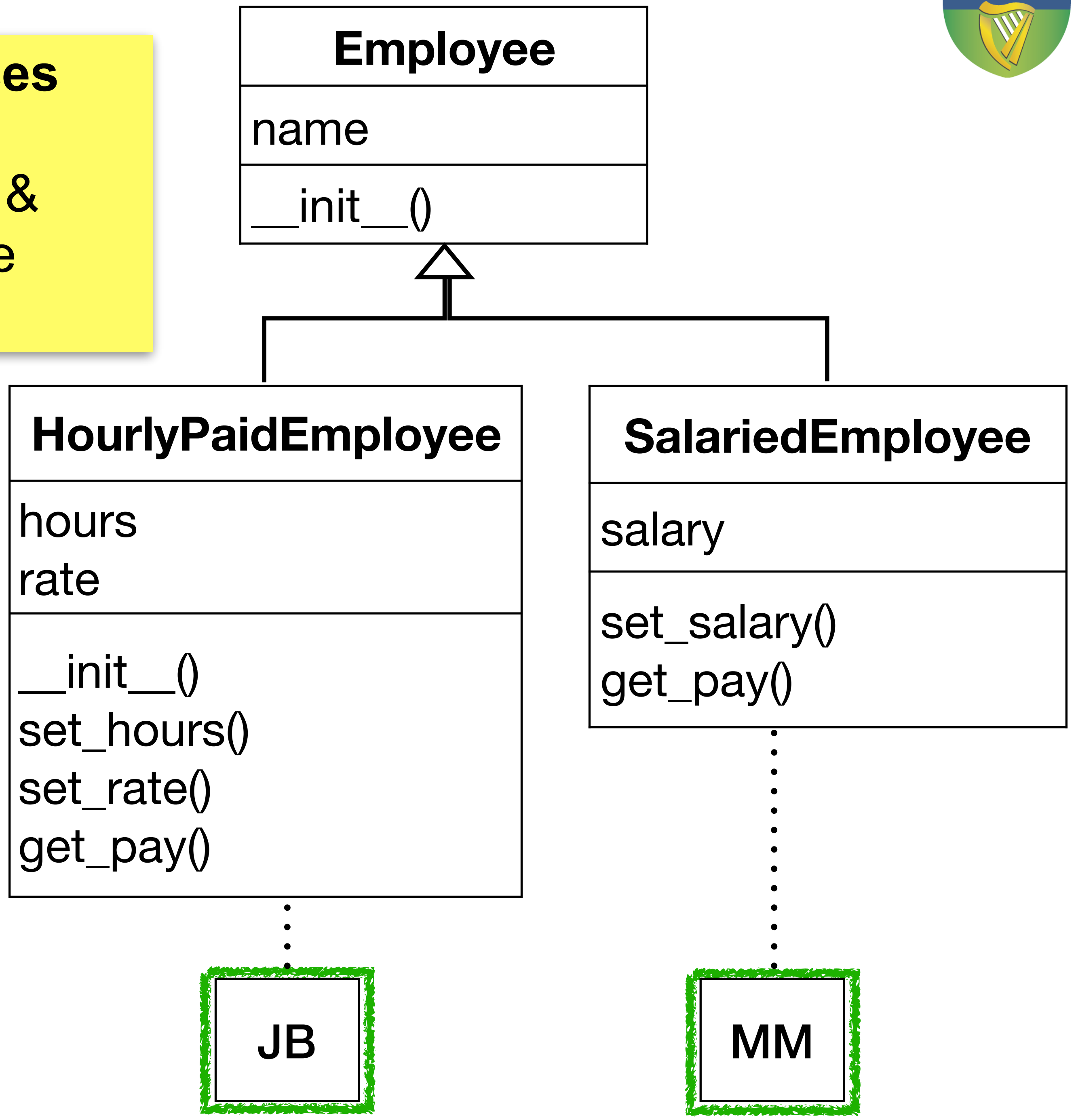
```
JB = HourlyPaidEmployee("Joe Bloggs")
MM = SalariedEmployee("Marvelous Mary")
JB.set_hours(121)
JB.set_rate(10.50)
MM.set_salary(45000)
```

Instantiation

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

class HourlyPaidEmploy

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

    def set_hours(sel
        self.hours =

    def set_rate(self
        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 & MM are **instances**
Employee,
HourlyPaidEmployee &
SalariedEmployee are
classes

```
p1 = JB.get_pay()
p2 = MM.get_pay()
p1,p2
Out[7]:
(1270.5, 3750.0)
```



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

Exercises

1. Assuming the minumum wage is €9.25, set that as the default rate in the **`HourlyPaidEmployee`** class. Create a new hourly paid employee to show that this works.
2. Update the **`SalariedEmployee`** class to cover the concept of a part-time worker, i.e. a salaried employee working three days a week would be paid 60% of the full salary.
   - **Hint:** SalariedEmployee should probably have an **`__init__`** function now. Use the one from **`HourlyPaidEmployee`** as a template.