technoteach

technocamps

Llywodraeth Cymru
Welsh Government

Prifysgol Abertawe
Swansea University

Cardiff University
Prifysgol Caerdydd

Prifysgol Bangor University

Cardiff Metropolitan University
Prifysgol Metropolitan Caerdydd

University of South Wales
Prifysgol De Cymru

Cyngor Cyllido Addysg Uwch Cymru
Higher Education Funding Council for Wales
hefcw

Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Prifysgol Aberystwyth University

Prifysgol Glyndŵr Wrecsam
Wrexham Glyndŵr University

institute of CODING in wales
technocamps

Python Review

What are these?

Python Review

```
print("Hello World!")
print(27+3)
print(2*2)
print(2**2)

myName="technocamps"
print(myName)

myLastName = input("Please provide your last name: ")
print(myLastName)

grade=70
if grade >= 70 :
    print("You got an A")
elif grade >= 60 :
    print("You got a B")
else :
    print("You got a C")

counter=1
while(counter<5):
    print("The counter is ", counter)
    counter+=1
print("End")

count=0
for count in range(0,5):
    print("The count is ", count)
print("End")
```

Python Review:
Local and Global variables

	Local	Global
Scope	It is declared inside a function.	It is declared outside the function.
Value	If it is not initialized, a garbage value is stored	If it is not initialized zero is stored as default.
Lifetime	It is created when the function starts execution and lost when the functions terminate.	It is created before the program's global execution starts and lost when the program terminates.
Data sharing	Data sharing is not possible as data of the local variable can be accessed by only one function.	Data sharing is possible as multiple functions can access the same global variable.
Parameters	Parameters passing is required for local variables to access the value in other function	Parameters passing is not necessary for a global variable as it is visible throughout the program
Modification of variable value	When the value of the local variable is modified in one function, the changes are not visible in another function.	When the value of the global variable is modified in one function changes are visible in the rest of the program.
Accessed by	Local variables can be accessed with the help of statements, inside a function in which they are declared.	You can access global variables by any statement in the program.
Memory storage	It is stored on the stack unless specified.	It is stored on a fixed location decided by the compiler.

Python Review:
Local and Global variables

Local

- The use of local variables offer a guarantee that the values of variables will remain intact while the task is running
- If several tasks change a single variable that is running simultaneously, then the result may be unpredictable. But declaring it as local variable solves this issue as each task will create its own instance of the local variable.
- You can give local variables the same name in different functions because they are only recognized by the function they are declared in.
- Local variables are deleted as soon as any function is over and release the memory space which it occupies.
- The debugging process of a local variable is quite tricky.
- Common data required to pass repeatedly as data sharing is not possible between modules.
- They have a very limited scope.

Global

- You can access the global variable from all the functions or modules in a program
- You only require to declare global variable single time outside the modules.
- It is ideally used for storing "constants" as it helps you keep the consistency.
- A Global variable is useful when multiple functions are accessing the same data.
- Too many variables declared as global, then they remain in the memory till program execution is completed. This can cause of Out of Memory issue.
- Data can be modified by any function. Any statement written in the program can change the value of the global variable. This may give unpredictable results in multi-tasking environments.
- If global variables are discontinued due to code refactoring, you will need to change all the modules where they are called.

technoteach
technocamps

Object Oriented Programming

Object Oriented Programming

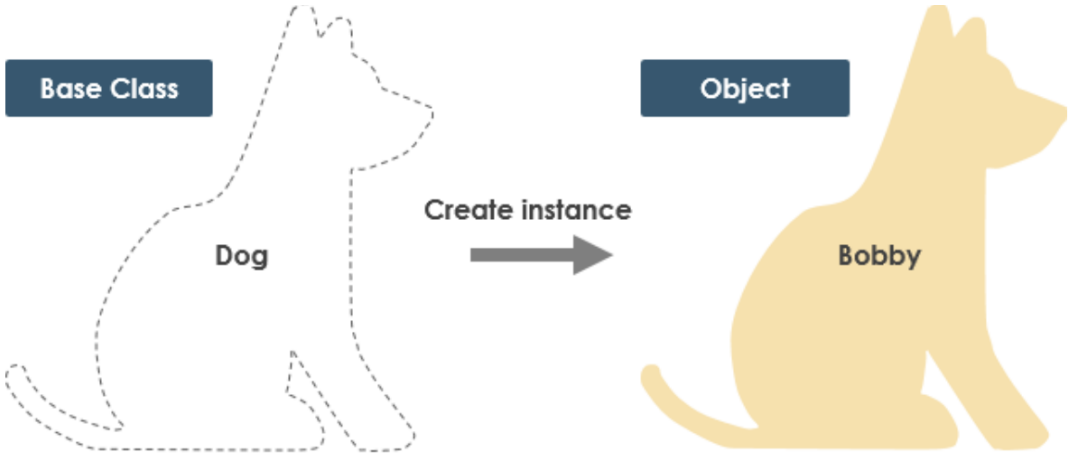
Object Oriented Programming (OOP) contains:

- Classes with properties and methods
- Objects (instances of the class)

Object Oriented Programming

Some basic terminologies

- Objects
- Classes
- Methods
- Instances



Properties	Methods
Color	Sit
Eye Color	Lay Down
Height	Shake
Length	Come
Weight	

Property Values	Methods
Color: Yellow	Sit
Eye Color: Brown	Lay Down
Height: 17 in	Shake
Length: 35 in	Come
Weight: 24 pounds	

Object Oriented Programming

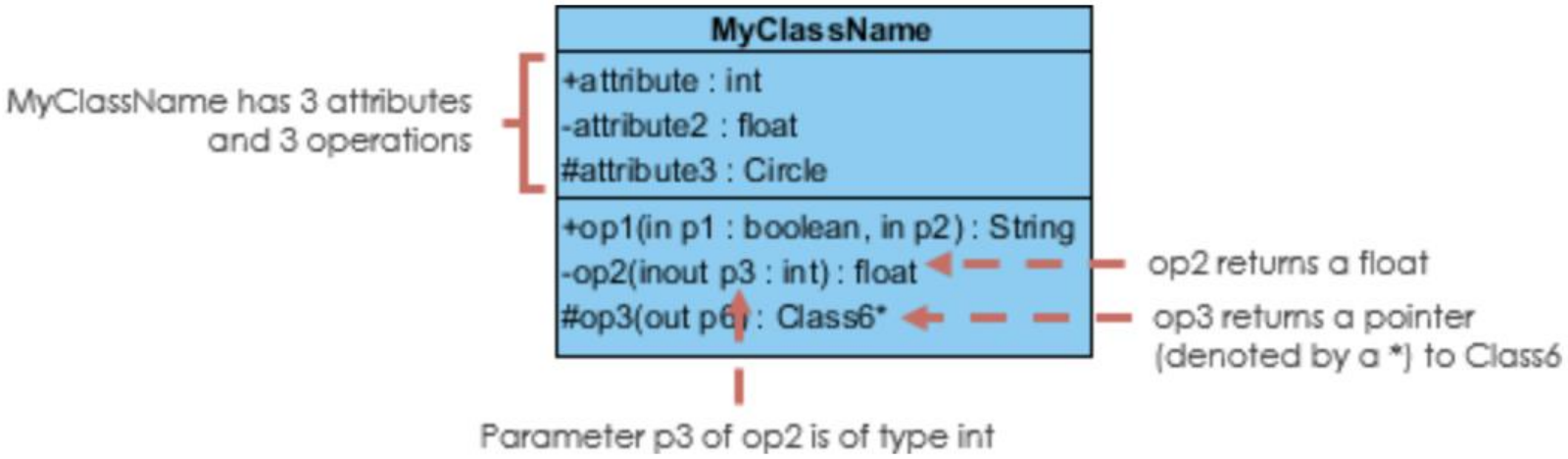
There are 4 main concepts of OOP:

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

UML Diagrams

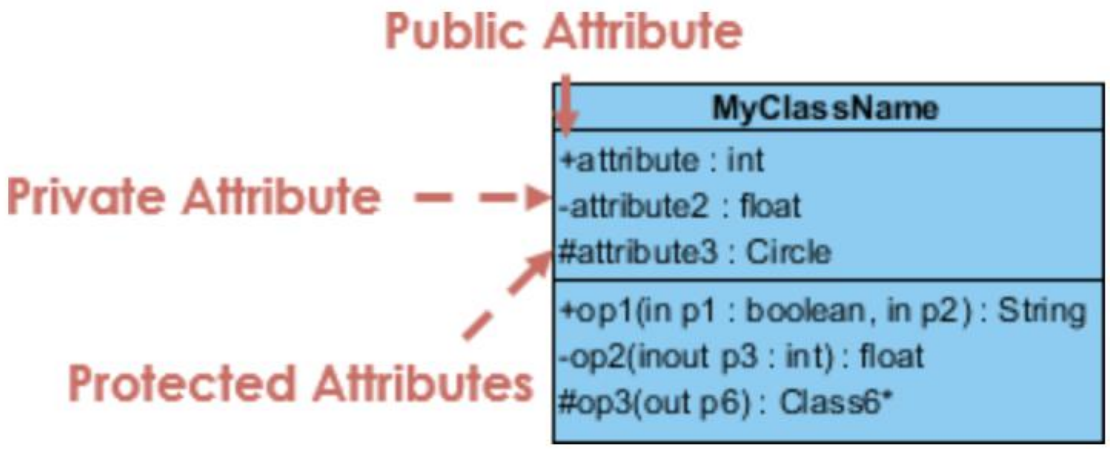
- Class name is the only mandatory information

UML Diagrams



UML Diagrams: Class Visibility

The +, -, and # symbols before the attribute and operator names in a class denote the visibility of the attribute and operation



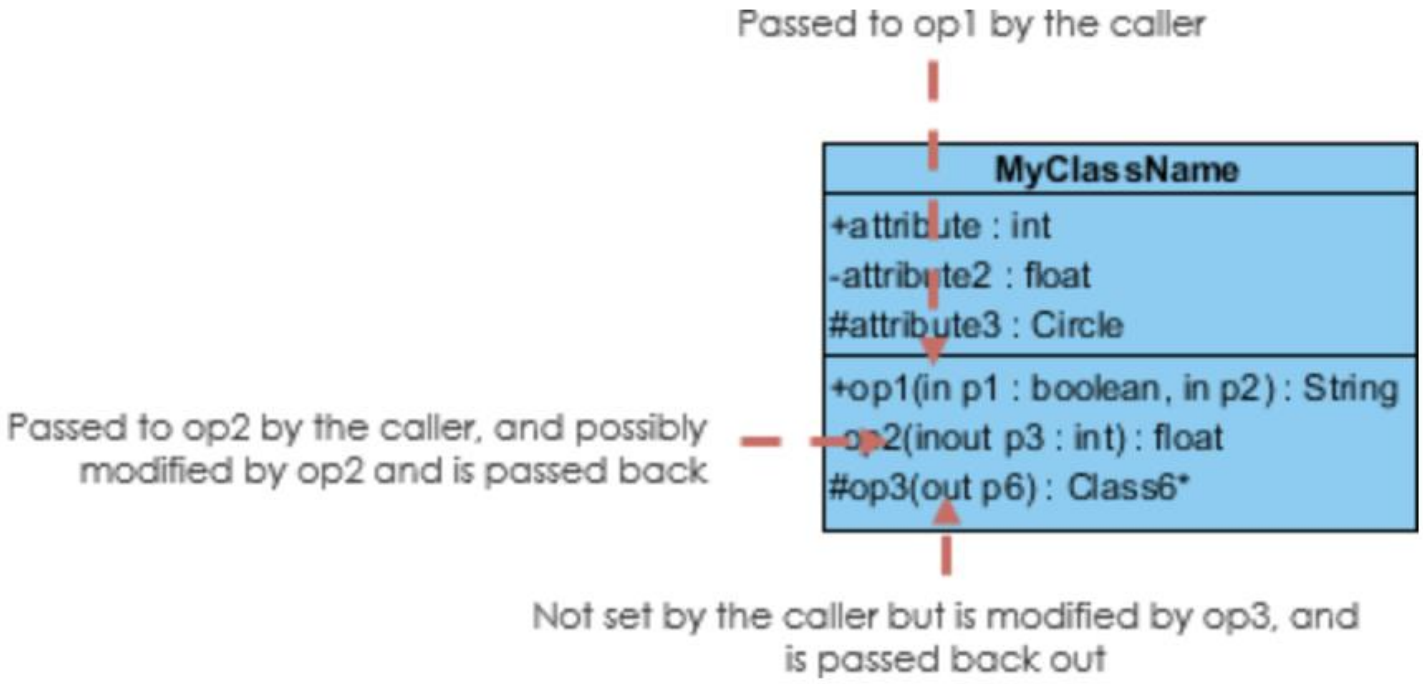
+ denotes **public** attributes or operations

- denotes **private** attributes or operations

# denotes **protected** attributes or operations

UML Diagrams: Class Directionality

Each parameter in an operation(method) may be denoted as an in, out, or inout which specifies its direction with respect to the caller.



Activity: Collect Information

Activity: Collect Information

Let's look back at the information you collected!

Name: Casey

Age: 10

Number of Siblings: 6

Number of Pets: 1

Teacher/Tutor's Name: Stewart

All of this information is about Students. We can create a class from this.

technoteach
technocamps

Classes

Classes

A **Class** is like recipe. It is used to build each object.

To create a class, use the keyword class.

Here we have a **class** Car with some defined **properties:** doors and colour.

```
1  class Car:
2      doors = 1
3      colour = "N/A"
```

Classes Activity

Think back to the information we collected.

What would we call the class to store this information?

What properties would the class have?

What should we set the default property values to? What values do you all share?

Write a class to hold this information.

Write your definition of a class in your workbook.

technoteach
technocamps

Objects

Objects

An **Object** is an **instance** of the **class.**

An object can either use the default values set by the class or it can change the values to suit itself.

```
1  class Car:
2      doors = 1
3      colour = "N/A"
4
5  car1 = Car()
6  print("This ", car1.colour, " has ", car1.doors, " doors")
```

uses default values

```
5  car1 = Car()
6  car1.doors = 4
7  car1.colour = "black"
8  print("This ", car1.colour, " has ", car1.doors, " doors")
```

sets it's own value for doors and colours

Objects Activity

Create an object that holds the correct information about your classmate.

Which properties do we need to set ourselves?

Which properties can use the default values?

Write your definition of an object in your workbook.

technoteach
technocamps

Methods

Methods

All classes have a function called __init__(), which is always executed when the class is being initialised. We call this the constructor.

The __init__() function allows you to assign values to object properties, or complete other operations that are necessary to do when the object is created.

```
1  class Car():
2      doors = 1
3      colour = "N/A"
4
5      def __init__(self,doors,colour):
6          self.doors = doors
7          self.colour = colour
8
9  car1 = Car(4,"black")
10 print("This", car1.colour, "car has", car1.doors, "doors")
```

Methods

Classes can also have other methods which the object can call.

Think of them as instructions.

If you had an object “Dog” what can you tell it to do?

Methods

"self" refers to the object that is calling the method.

```
1  class Car():
2      doors = 1
3      colour = "N/A"
4
5      def __init__(self,doors,colour):
6          self.doors = doors
7          self.colour = colour
8
9      def start(self):
10         print("Vroom, vroom")
11
12 car1 = Car(4,"black")
13 car1.start()
```

Methods

```
# default constructor

def __init__(self):

# initializing variable instance

self.number_variable=1001
```

```
# parameterized constructor

def __init__(self ,  id , name , age , gender, doj , dob ):

self.id_value = id

self.name_value = name

Text

self.age_value =  age

self.gender_value = gender

self.doj_value = doj

self.dob_value = dob
```

Methods Activity

Think of 3 methods that you can apply to your class/object.

Write these into your project and test they work.

Write your definition of a method in your workbook.

Object Oriented Programming

There are 4 main concepts of OOP:

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

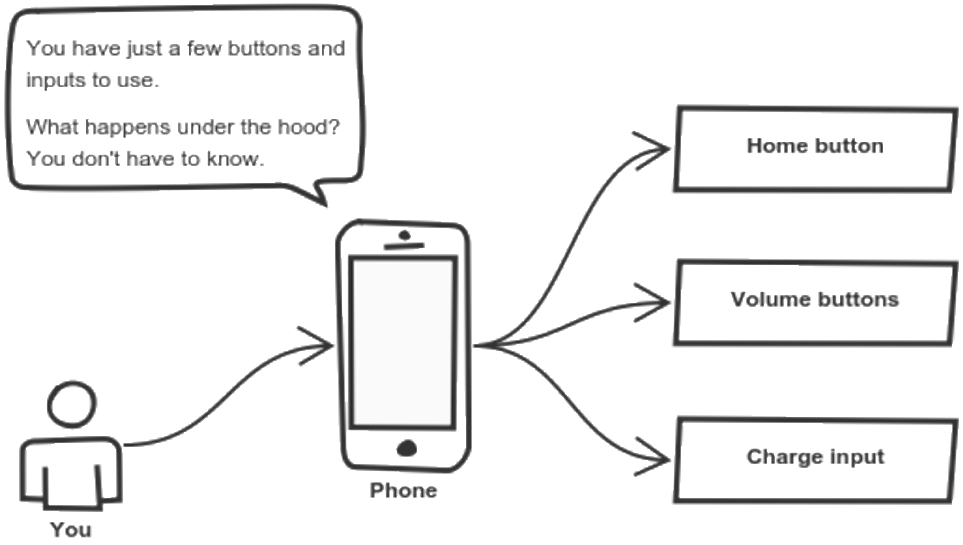
technoteach
technocamps

Abstraction

Abstraction

Applying abstraction means that each object should **only** expose a high-level mechanism for using it.

Think—a coffee machine. It does a lot of stuff and makes quirky noises under the hood. But all you have to do is put in coffee and press a button.



Abstraction Activity

Think of other every day objects you see/use but don't know what goes on behind the scenes.

Write down how you interact with the object and the outcome.

technoteach
technocamps

Encapsulation

Encapsulation

Encapsulation is achieved when each object keeps its state **private**, inside a class. Other objects don't have direct access to this state. Instead, they can only call a list of public functions—called methods.

In python __ sets a variable to private.

technoteach technocamps

Encapsulation

```
1  class Car():
2      doors = 1
3      colour = "N/A"
4      __seats = 1
5
6      def __init__(self,doors,colour):
7          self.doors = doors
8          self.colour = colour
9
10     def start(self):
11         print("Vroom, vroom")
12
13     def printNumberSeats(self):
14         print("This car has", self.__seats, "seats")
15
16 car1 = Car(4,"black")
17 car1.printNumberSeats()
```

technoteach technocamps

Encapsulation

```
1  class Car():
2      doors = 1
3      colour = "N/A"
4      __seats = 1
5
6      def __init__(self,doors,colour):
7          self.doors = doors
8          self.colour = colour
9
10     def start(self):
11         print("Vroom, vroom")
12
13     def printNumberSeats(self):
14         print("This car has", self.__seats, "seats")
15
16 car1 = Car(4,"black")
17 car1.__seats = 4
18 car1.printNumberSeats()
```

technoteach technocamps

Encapsulation

```
1  class Car():
2      doors = 1
3      colour = "N/A"
4      __seats = 1
5
6      def __init__(self,doors,colour):
7          self.doors = doors
8          self.colour = colour
9
10     def start(self):
11         print("Vroom, vroom")
12
13     def setNumberSeats(self,seats):
14         self.__seats = seats
15
16     def printNumberSeats(self):
17         print("This car has", self.__seats, "seats")
18
19 car1 = Car(4,"black")
20 car1.setNumberSeats(4)
21 car1.printNumberSeats()
```

technoteach
technocamps

Inheritance

Inheritance

Objects are often very similar. They share common logic. But they're not **entirely** the same.

So how do we reuse the common logic and extract the unique logic into a separate class? One way to achieve this is inheritance.

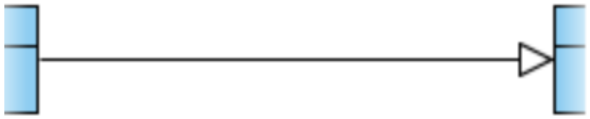
It means that you create a (child) class by deriving from another (parent) class. This way, we form a hierarchy.

The child class reuses all fields and methods of the parent class (common part) and can implement its own (unique part).

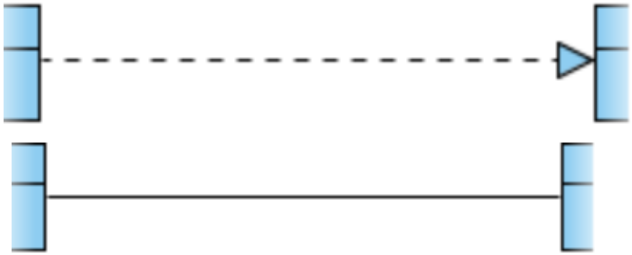
Think – *subclass* **is a** type of *parent*.

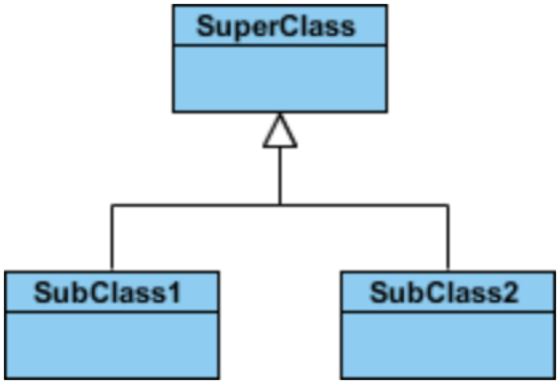
i.e. Dog is a type of Animal

UML Diagrams: Inheritance



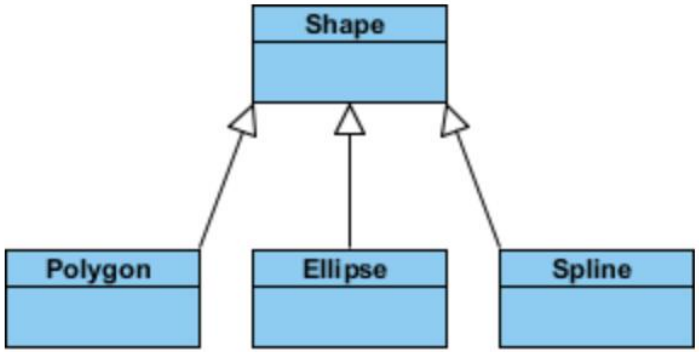
Inheritance



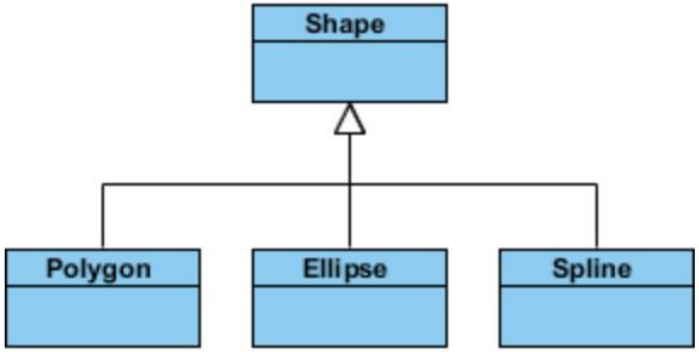


- Represents an “is-a” relationship

UML Diagrams: Inheritance Example

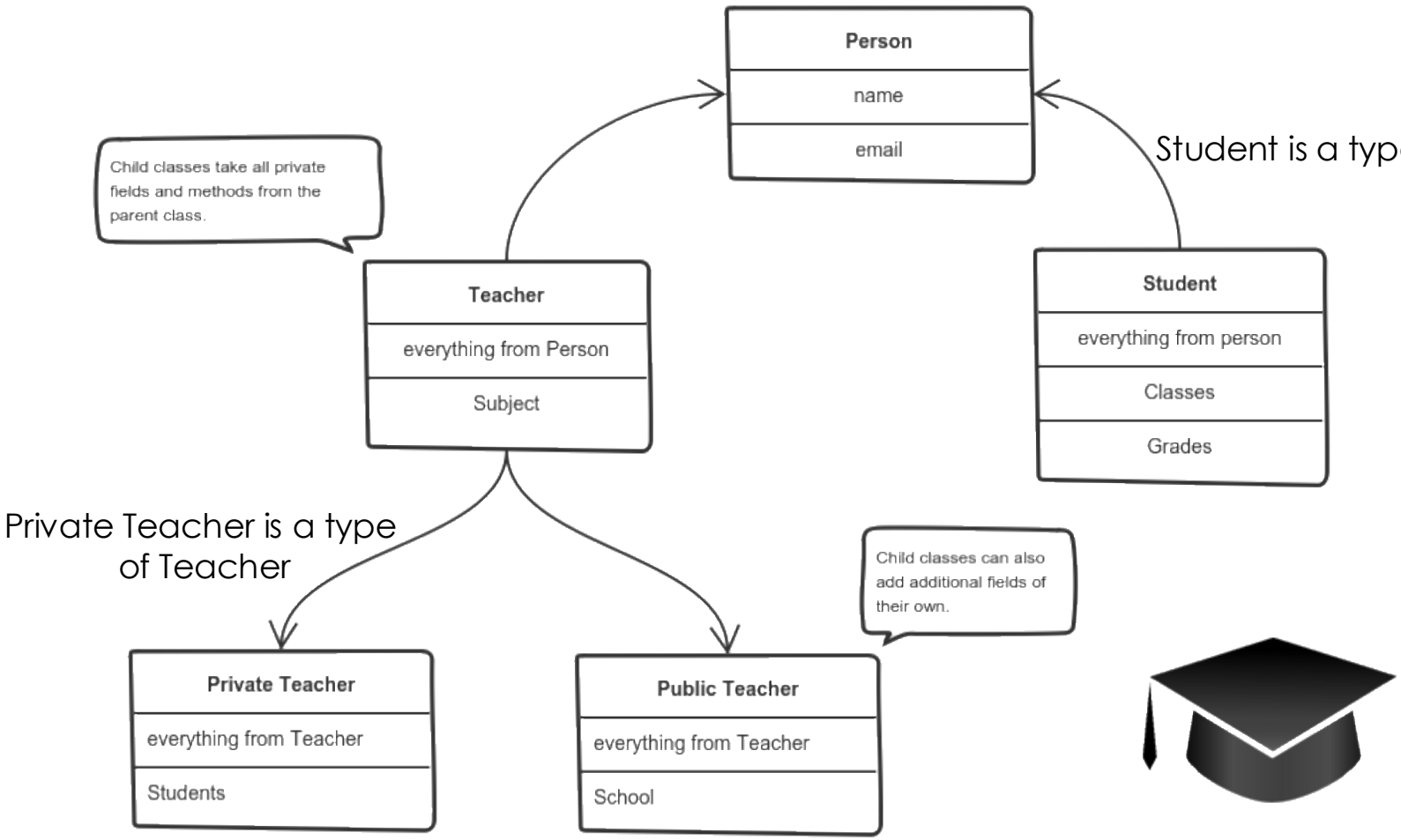


Style 1: Separate target



Style 2: Shared target

Inheritance



technoteach technocamps

Inheritance

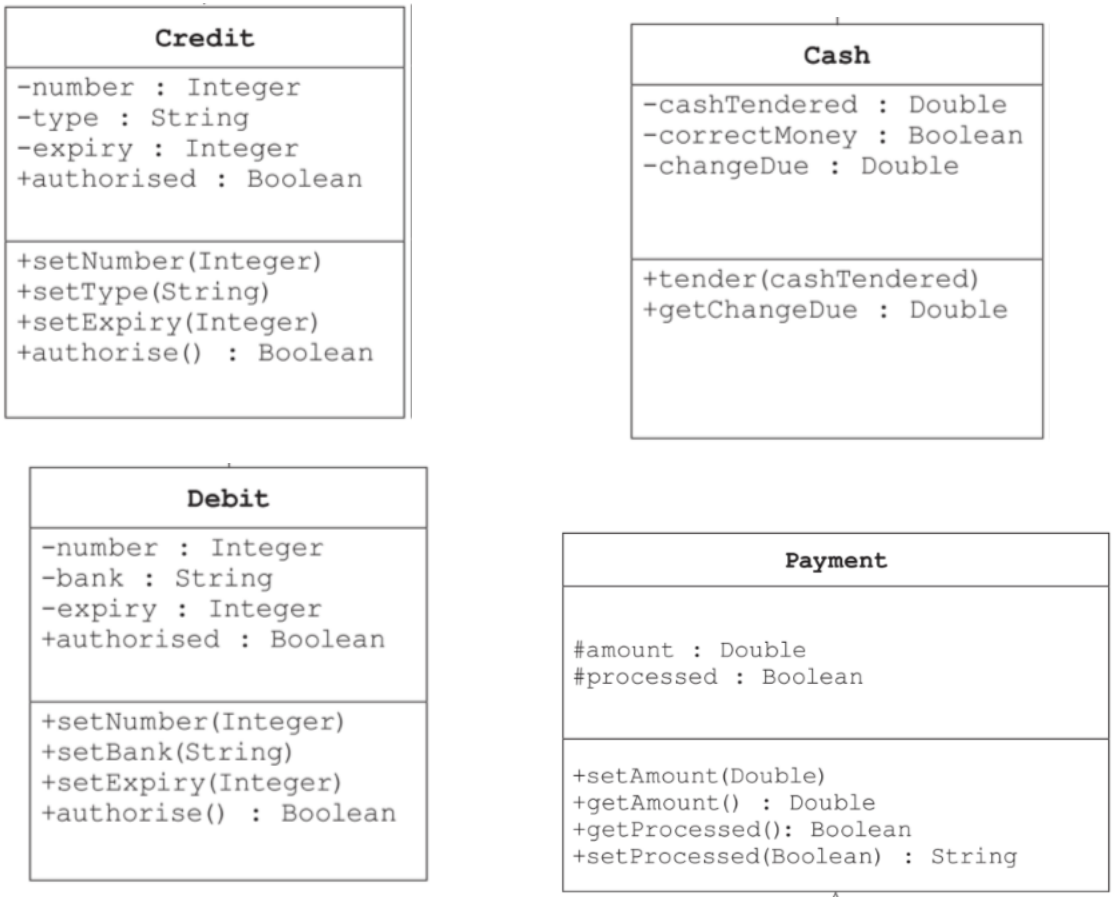
```
1  class Vehicle():
2      doors = 1
3      colour = "N/A"
4      __seats = 1
5      wheels = 1
6
7      def __init__(self,doors,colour):
8          self.doors = doors
9          self.colour = colour
10
11     def start(self):
12         print("Vroom, vroom")
13
14     def setNumberSeats(self,seats):
15         self.__seats = seats
16
17     def printNumberSeats(self):
18         print("This car has", self.__seats, "seats")
19
20 class Car(Vehicle):
21     wheels = 4
22
23 car1 = Car(4,"black")
24 car1.setNumberSeats(4)
25 car1.printNumberSeats()
```

Inheritance Activity

Match the subclasses to the parent class. Think carefully about what the subclasses will inherit.

Hint: Some parent classes may have their own parents!

Inheritance Activity



technoteach
technocamps

Polymorphism

Polymorphism

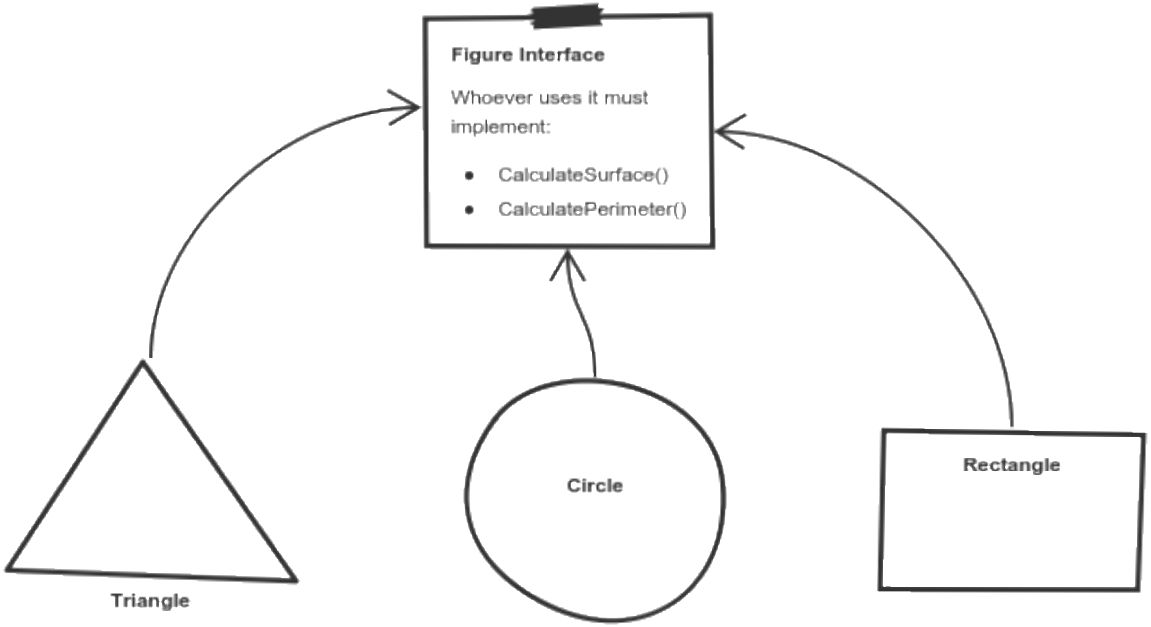
Polymorphism gives a way to use a class exactly like its parent so there's no confusion with mixing types. But each child class keeps its own methods as they are.

This typically happens by defining a (parent) interface to be reused. It outlines a bunch of common methods. Then, each child class implements its own version of these methods.

Polymorphism is perhaps the most complex concept.

- Polymorphism means that different types respond to the same function.
- Polymorphism is very useful as it makes programming more intuitive and therefore easier.

Polymorphism



Triangle, **Circle** and **Rectangle** inherit the Figure interface or abstract class.

They implement their own versions of **CalculateSurface()** and **CalculatePerimeter()**.

They can be used in a mixed collection of Figures.

Polymorphism - Types

```
1  class Vehicle():
2      doors = 1
3      colour = "N/A"
4      __seats = 1
5
6      def __init__(self,doors,colour):
7          self.doors = doors
8          self.colour = colour
9
10     def start(self):
11         print("The vehicle is starting: Vroom, vroom")
12
13     def setNumberSeats(self,seats):
14         self.__seats = seats
15
16     def printNumberSeats(self):
17         print("This car has", self.__seats, "seats")
18
19 class Car(Vehicle):
20     wheels = 4
21
22 class Bike(Vehicle):
23     wheels = 2
24
25 car1 = Car(4,"black")
26 bike1 = Bike(1,"pink")
27
28 vehicles = [car1,bike1]
29 for v in vehicles:
30     v.start()
```

Polymorphism – Method Overriding

```
1  class Vehicle():
2      doors = 1
3      colour = "N/A"
4      __seats = 1
5
6      def __init__(self,doors,colour):
7          self.doors = doors
8          self.colour = colour
9
10     def start(self):
11         print("The vehicle is starting: Vroom, vroom")
12
13     def setNumberSeats(self,seats):
14         self.__seats = seats
15
16     def printNumberSeats(self):
17         print("This car has", self.__seats, "seats")
18
19 class Car(Vehicle):
20     wheels = 4
21
22     def start(self):
23             print("The car is starting: Vroom, vroom")
24
25 class Bike(Vehicle):
26     wheels = 2
27
28     def start(self):
29             print("The bike is starting: Vroom, vroom")
30
31 car1 = Car(4,"black")
32 bike1 = Bike(1,"pink")
33
34 car1.start()
35 bike1.start()
```

Polymorphism

Select two parent classes from the inheritance activity and think of two or more methods that could be defined in the parent class and then overridden in the subclass.

Zoo Activity

Zoo Activity

Create a program that holds the following information about animals in a zoo.

- 3 Ducks – Daffy, Donald, Daisy. Daffy is 5 years old, Donald is 3 years old and Daisy is 1. How many legs do ducks have? What sound do they make?
- 2 Giraffes – George and Gerald. George is 7 and Gerald is 10 years old. How many legs do giraffes have? What sound do they make?
- 1 Elephant – Nelly. Nelly is 12 years old. How many legs do elephants have? What sound do they make?

- Make a method that prints the following:

"Hello! My name is ____. I am a (type of animal). I have ___ legs. (Noise/Sound)"

Think! What defaults can be set? Do some animals share the same values?

Conclusion

There are 4 main concepts of OOP:

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

To ensure we use these concepts we need to utilise classes, objects and methods.

technoteach
technocamps

Text Adventure in Python

Let's Start our Text Adventure!

Think of the style you'd like your text adventure game to be; should it be an adventure, a horror, educational?

This game is entirely your own, we're just going to guide you on making it!

Where is the game set?

Who are the characters?

What is the goal?

Making our world!

When making a game it's a good idea to keep notes of your game before it becomes too big to keep track of.

As we have no graphical element (at least at this stage) for our text adventure, it is a good idea to make ourselves a map.

Using notepad, create a map that looks something like this:

```
y | x   0           1           2
0                Reception

1    ITClass     Hallway     DTClass

2                Cafeteria
```

Rooms

I want to be able to have multiple **rooms** that I can visit within my game.

I will therefore need to create a **Room object** that defines each room in my game - i.e. the information and methods belonging to each room.

If I want to be able to create **Room objects**. What will I need to create first?

Rooms

I want to be able to have multiple **rooms** that I can visit within my game.

I will therefore need to create a **Room object** that defines each room in my game - i.e. the information and methods belonging to each room.

If I want to be able to create **Room objects**. What will I need to create first?

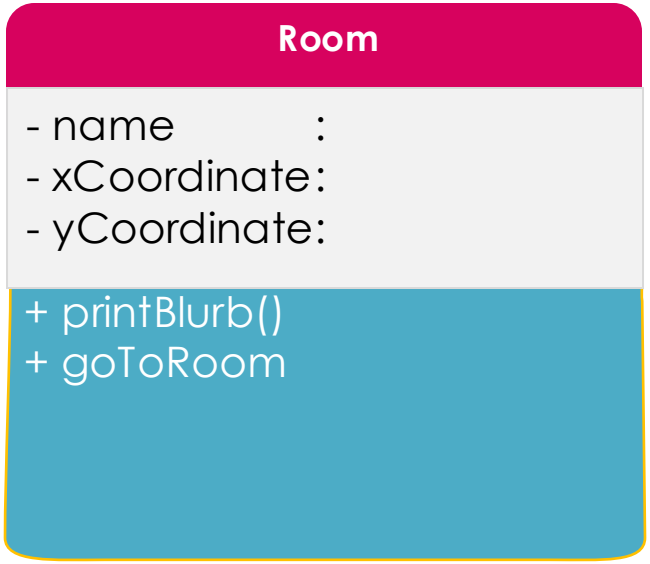
- A **Room Class**!

Your task:

- Design a Class to model a Room!

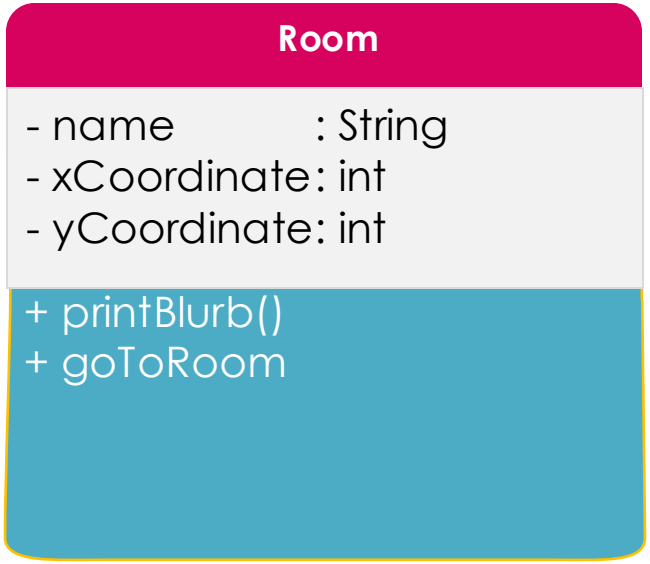
Room Class

Maybe something like this:



Room Class

Maybe something like this:



Got a different idea?

- There are a few variations here!

Room Objects

Okay so we now have a Room class that can be used to create Room objects for us!

But now we're going to need something that can hold onto all our rooms for us.

What could we make to hold our Room objects (**Hint:** remember we're using object-oriented programming)?

Room Objects

Okay so we now have a Room class that can be used to create Room objects for us!

But now we're going to need something that can hold onto all our rooms for us.

What could we make to hold our Room objects (**Hint:** remember we're using object-oriented programming)?

- A **Map** class!

Adding Rooms

Now we want to add our Room objects, as defined by those we made in our mini maps.

Ideally, when adding stuff into our game it should be stored separately to our code; this means it can be easily modified without having to modify the game code.

We will use a csv file to accomplish this (a csv file is just a text file with **comma separated values**). It should look something like this:

Adding Rooms

We can make our **csv** files using Notepad or a similar **basic** text editor.

Remember, your commas are separating the values, so we cannot use commas in our blurbs.

It should look something like this:

```
RoomName, xCoordinate, yCoordinate, Blurb

Reception,1,0,This seems to be the schools' reception. Broken glass and paper covers the floor.

Hallway,1,1,You enter the main hallway. It is pitch black. There are no windows at all. Classro
```

Importing Rooms

Now that we have our rooms described, we need to import these definitions into our program and start creating Room objects out of them!

We will create a new method in the **Map** class to handle this.

Type Casting

We want to be able to use the data from the room file for specific purposes, but currently everything we read from the file is a String.

What data do we need to change?

Type Casting

We want to be able to use the data from the room file for specific purposes, but currently everything we read from the file is a String.

What data do we need to change?

Our coordinates need to be of type int!

We will need a new method in our Room class to type cast our imported data.

Get A New Room

We already have a method in our Room class that works out the coordinates for each new Room object.

Now we need a method for our Map class that searches through the list of rooms to find a matching object!

Making our Game

Okay, so we have some essential classes set up!

We will need more of these as our game grows in complexity – remember this is OOP and everything in the game should be expressed as some sort of object.

Making our Game

Okay, so we have some essential classes set up!

We will need more of these as our game grows in complexity – remember this is OOP and everything in the game should be expressed as some sort of object.

However, for now, we can begin tying this all together by creating a class that will run the program.

Making our Game

Okay, so we have some essential classes set up!

We will need more of these as our game grows in complexity – remember this is OOP and everything in the game should be expressed as some sort of object.

However, for now, we can begin tying this all together by creating a class that will run the program.

This is often called Main, but we will call our main class **Game**, which will create a **Game object** from which the game is run!

Starting the Game

As we are performing object orientation, we will need to initialise an object of the Game class before running our program.

Again, with OOP this should be defined as a method.

The only thing outside our Game Class should be the creation of the Game Class object.

Moving Rooms

We now have a Game class and a partial method that runs the main game loop!

However, we're going to need to interface with the get Room methods we already created.

The Game object will need to use these methods to check if it is possible for the user to move rooms in the specified direction, and either:

1. Move rooms (i.e. change Room object)
2. Tell the user they cannot move

technoteach
technocamps

Text Adventure with OOP

OOP Recap

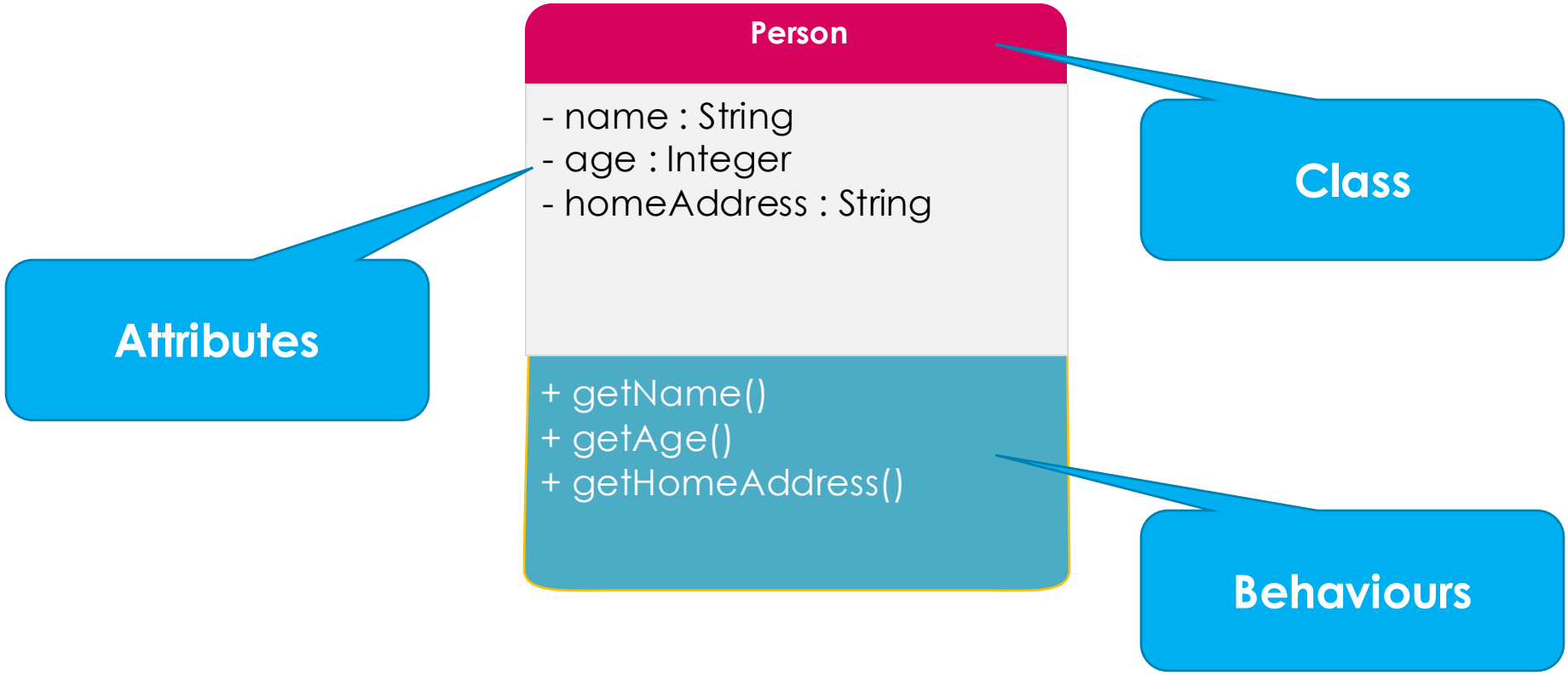
Object oriented programming is, as the name suggests, all about **objects**.

Classes are **blueprints** that can be used to create objects. A class defines:

- The **data (attributes)** an object will store.
- The **behaviour** an object will be capable of.
- How other objects can interact with it.

To create an Object, we must specify the blueprint (Class) for it!

OOP Recap


Person
- name : String
- age : Integer
- homeAddress : String
+ getName()
+ getAge()
+ getHomeAddress()
Attributes
Class
Behaviours

Inheritence

We regularly encounter Objects which are **specific types** of some abstract concept.

In day-to-day life, we often use the abstract concept to refer to things:

- "*Where are the **fruits**?*"
- "*Heavy **vehicles** are harder to drive*"
- "*Turn on the **heater***"
- "*Our shelter houses wounded **animals***"
- "*I will visit the **doctor***"

Inheritence

Each of these concepts on its own is too abstract to think about...

- But it is useful to group things with common **attributes** and **behaviours** together!

We can think of specific types of each of them

For example, **Animal**:

- Cat
- Dog
- Bird

Can you think of other examples?

The '*is - a*' Relationship

When we have a specific type of a more general concept, we use the ***is-a*** relationship between the two:

- "A dog ***is-a*** Animal"
- "A car ***is-a*** Vehicle"
- …

It's good to get in the habit of using the ***is-a*** relation!

What About Classes?

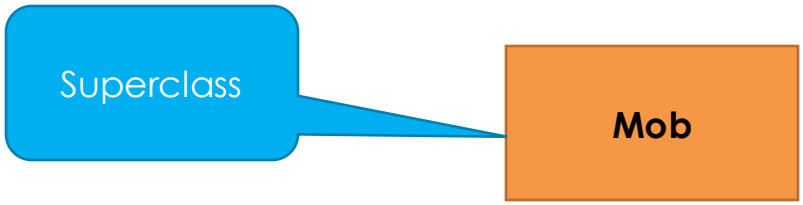
Inheritance is a core component between classes!

- The more general class is the **superclass**
- Specific types of this general class are **subclasses**

What About Classes?

Inheritance is a core component between classes!

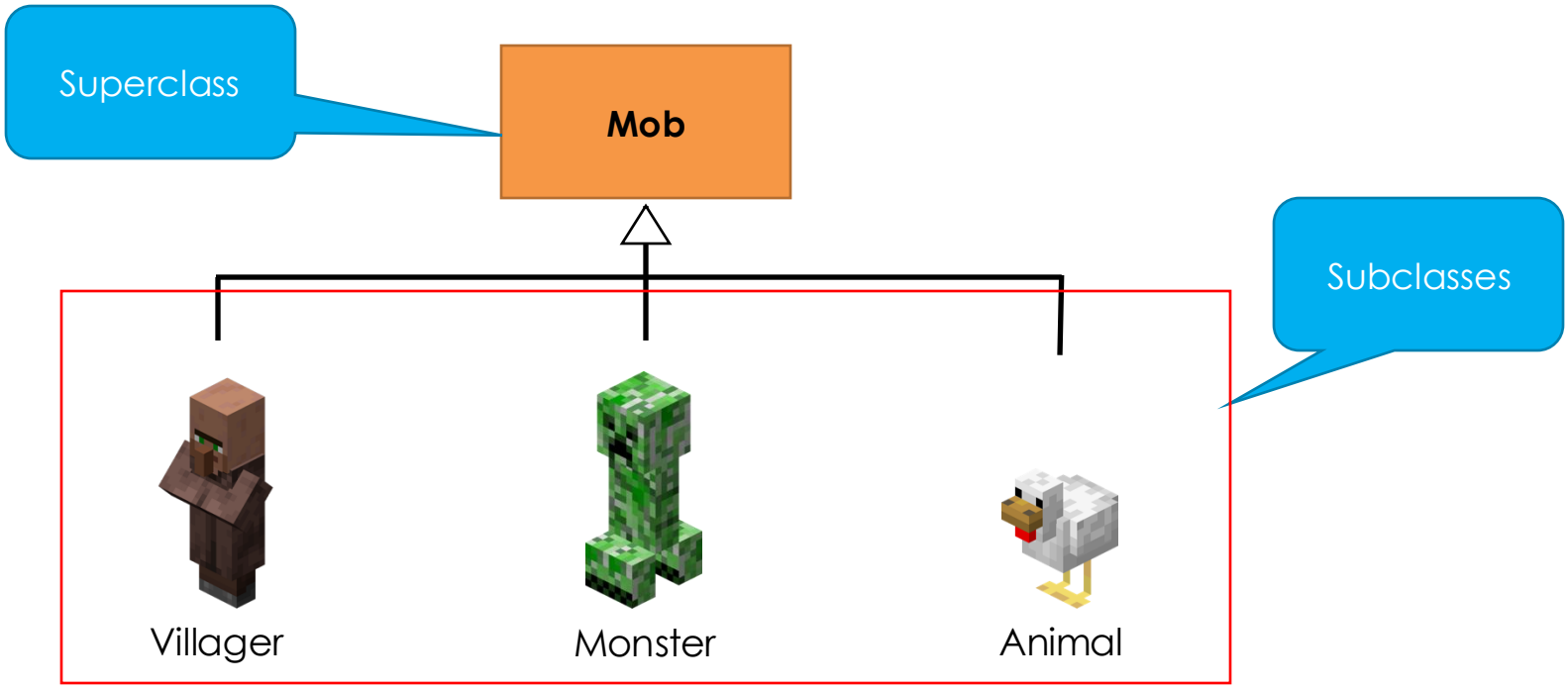
- The more general class is the **superclass**
- Specific types of this general class are **subclasses**



What About Classes?

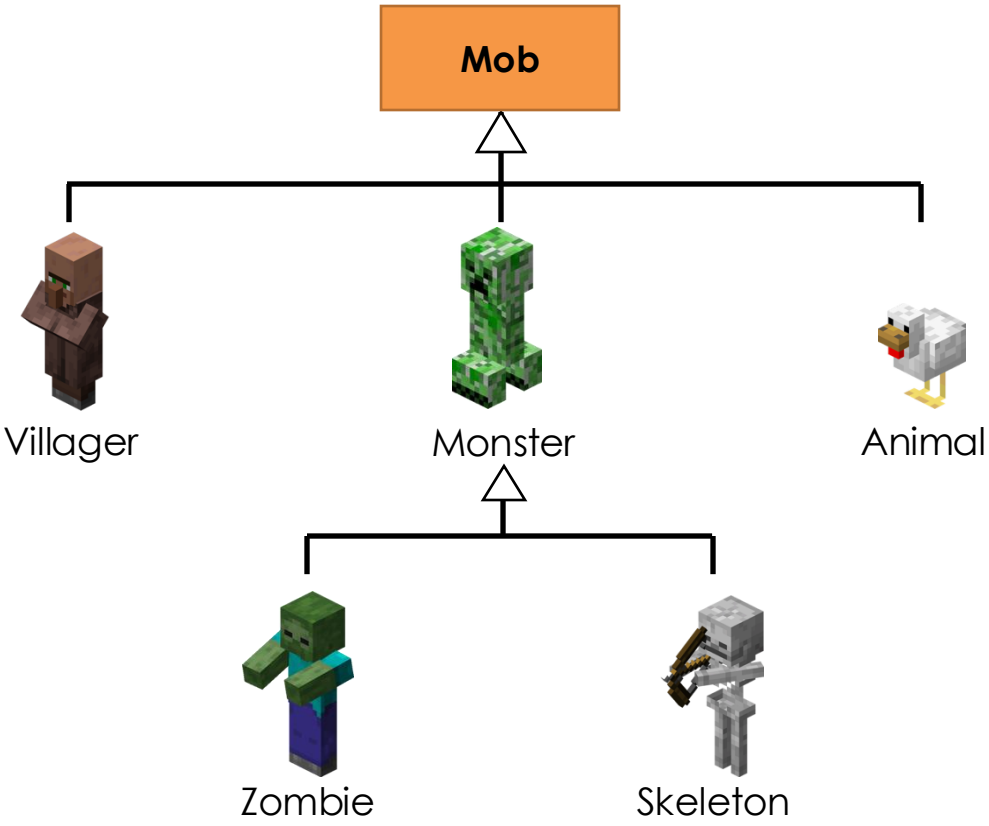
Inheritance is a core component between classes!

- The more general class is the **superclass**
- Specific types of this general class are **subclasses**



Fun Fact!

Inheritance is usually multi-level! Let's revisit the previous hierarchy:



Unified Modelling Language

We need a standardised way to talk about Classes

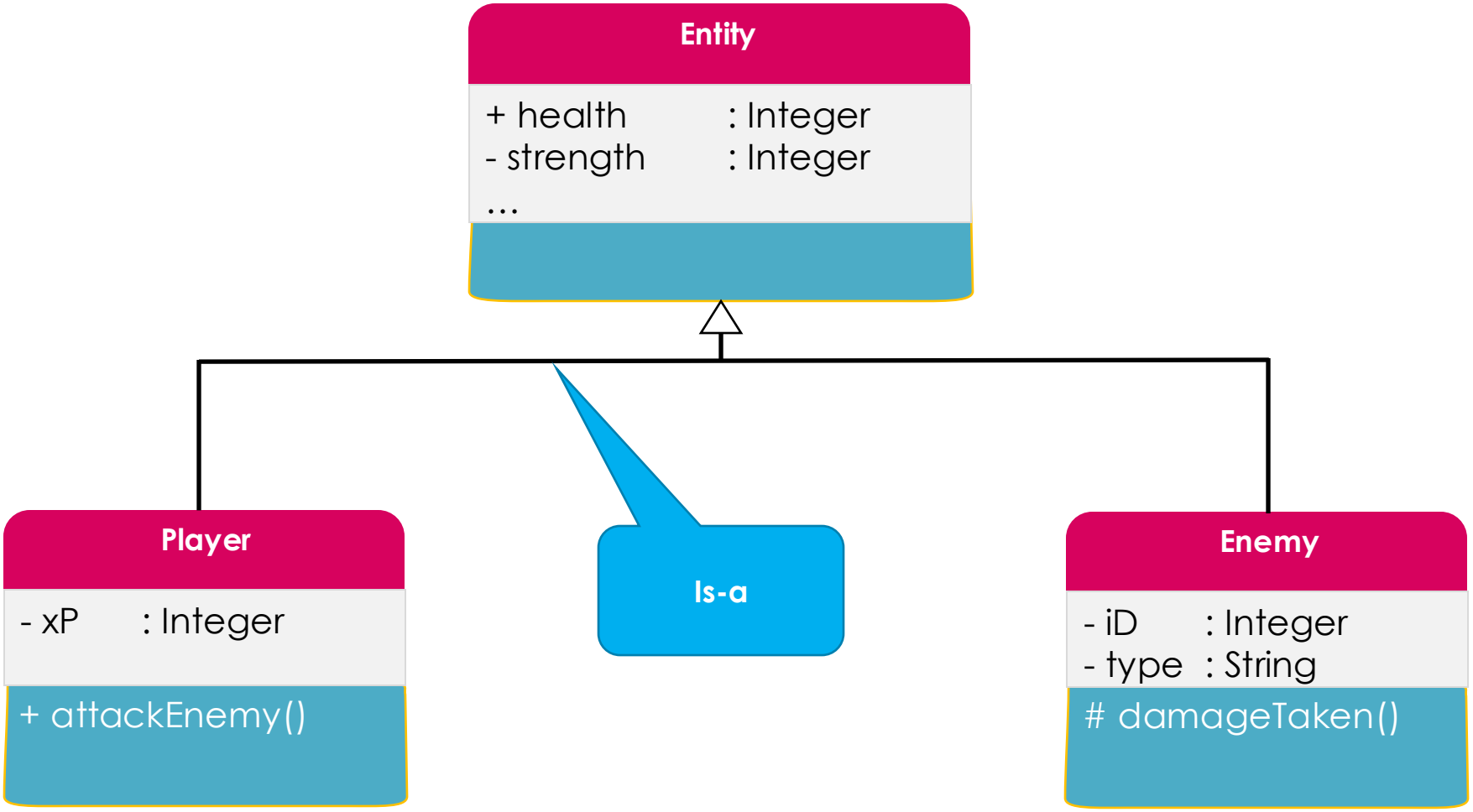
The Unified Modelling Language (**UML**) can do that!

- A very big, complicated language…
- But don't worry we'll only need to use a fraction of it!

Class Diagrams:

- Show Classes pictorially
- … We've used them already!
- There's just a bit more to them…

Visualising Relationships *ft. is-a*



The super() Method

When programming inheritance we generally want our subclasses to inherit all attributes (and possibly methods) from the superclass.

i.e. All **Entities** should have the attribute **Health**

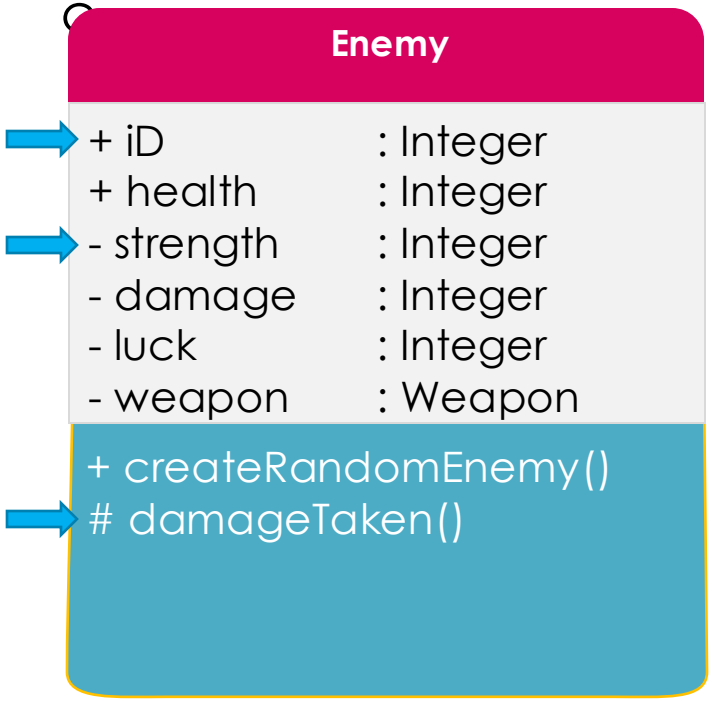
- this may well have different default values for different subclasses, but all entities have the attribute
- after all – that's why we made it an attribute of entities!

To inherit both attributes and methods in Python we can use the super() method.

What's up with the +/- symbols?

In a Class we must select which information is visible to the world

- And we may choose to limit, or restrict it completely!



Symbol	Meaning	Who can see it?
+	public	Everyone!
#	protected	The class & its subclasses
-	private	The class itself only

How do we do this in Python?

Python is good at emulating both scripting and object-oriented languages!

Unfortunately, in Python these are only conventions and they have no impact on the code itself (i.e. all variables are public and there is no such thing as private or protected)

Symbol	Meaning	Who can see it?	Naming Python Variables & Methods
+	public	Everyone!	variableName
#	protected	The class & its subclasses	_variableName
-	private	The class itself only	__variableName

Create Enemies

There's a few ways we could create enemies in our game:

- We could load them in from a file like we did with our rooms (this would be a good method for bosses with specific stats)
- We could generate them randomly

For now, we will generate them randomly, as we've already looked at reading from a file.

Printing Enemies

Now we have enemies in each room, but the game never tells us that – so the user would have no idea.

We should print out the enemies in each room as we enter!

Attack Enemies

Finally, we'll begin adding combat to our game!

We'll add a method to the **Player** to attack an **Enemy**, and a method to the **Enemy** to calculate the damage done by the **Player**.

We will then need to add this option to our user decisions.

Continue your game!

We have now looked at many methods for creating a game:

Continue to develop your game with these concepts!

- Finish the combat system to allow enemies to attack.
- Add an Items class with a Weapons subclass (this could change the damage dealt by entities)
- Read your items from files - a different file for each Item subclass would allow you to import unique attributes (i.e. value, weight, damage…)
- And whatever else you can imagine!