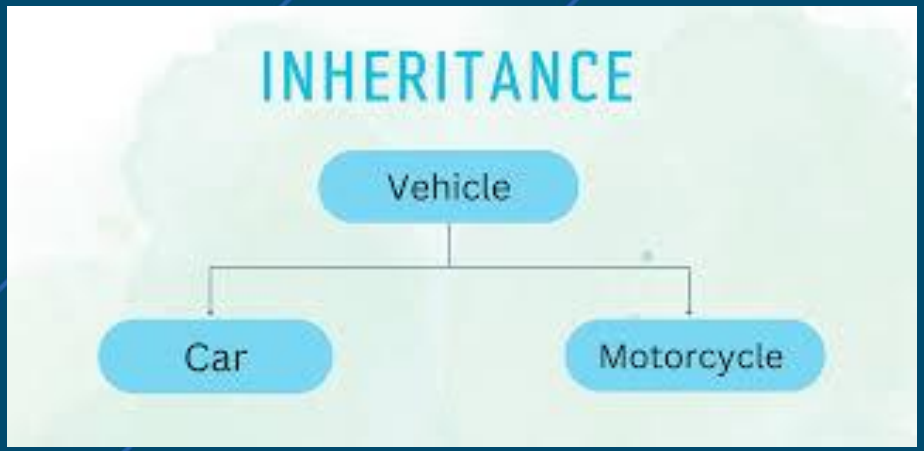
Féidearthachtaí as Cuimse
Infinite Possibilities



Inheritance

Object Oriented programming

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TU DUBLIN
TECHNOLOGICAL UNIVERSITY DUBLIN

Inheritance

- Dictionary

```
"To receive from predecessors"..
```

To date

- Classes
- Objects (Person p = new Person(“..etc”);
- Constructors
- Methods
- Method signatures
- Method overloading
- UML class diagrams

Scenario

College system – stores details on staff and students

Scenario

College system – stores details on staff and students

Student - spec

// attributes, inc programme, year of study etc

// behaviour (methods) includes: getters, setters, etc

etc

Staff

// attributes inc line manager, department etc

// behaviour (methods) includes: getters, setters, etc

Let’s look at the code….

Scenario

What’s the code overlap?

What’s the problem?

What’s the solution?

Inheritance in OO

“Is type of “

Person

- Student
- Staff

Purpose : to re-use code (avoid rewriting new code)

Inheritance in OO – multi layered

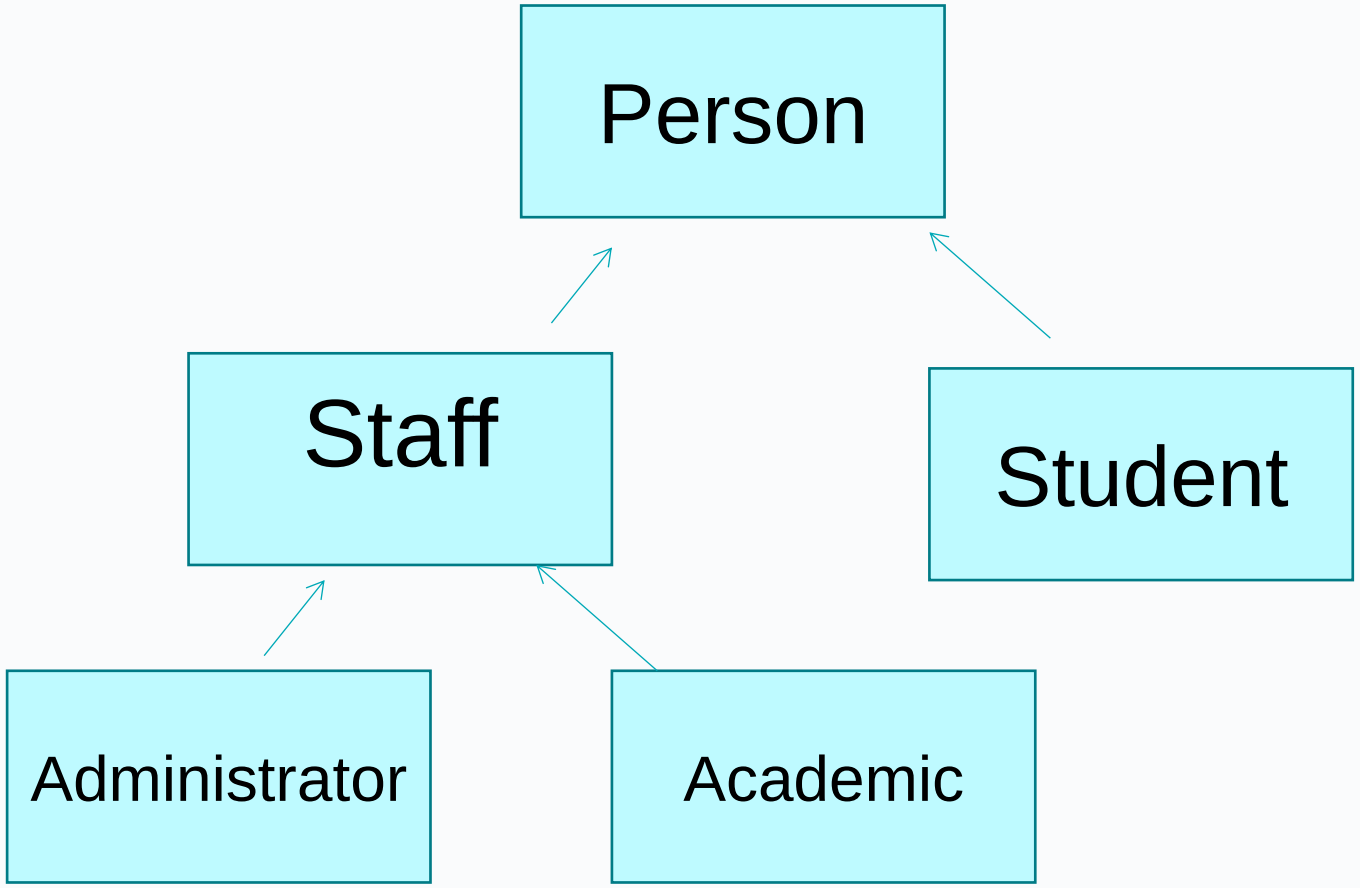
“Is type of “

Person

- Student
- Staff
  - Admin
  - Academic

Super classes and sub classes

How many
super classes and
sub classes
are shown?



can be identified from the class hierarchy

Subclass

- Inherits the members (attributes and methods)
- *Adds its own specific members (attributes and members)*
- Overrides methods (behaviour) as needed
- Example: Person/ Student/ Staff etc

To implement inheritance in java

```
public class Student extends Person
```

(In python `class Student(Person))`

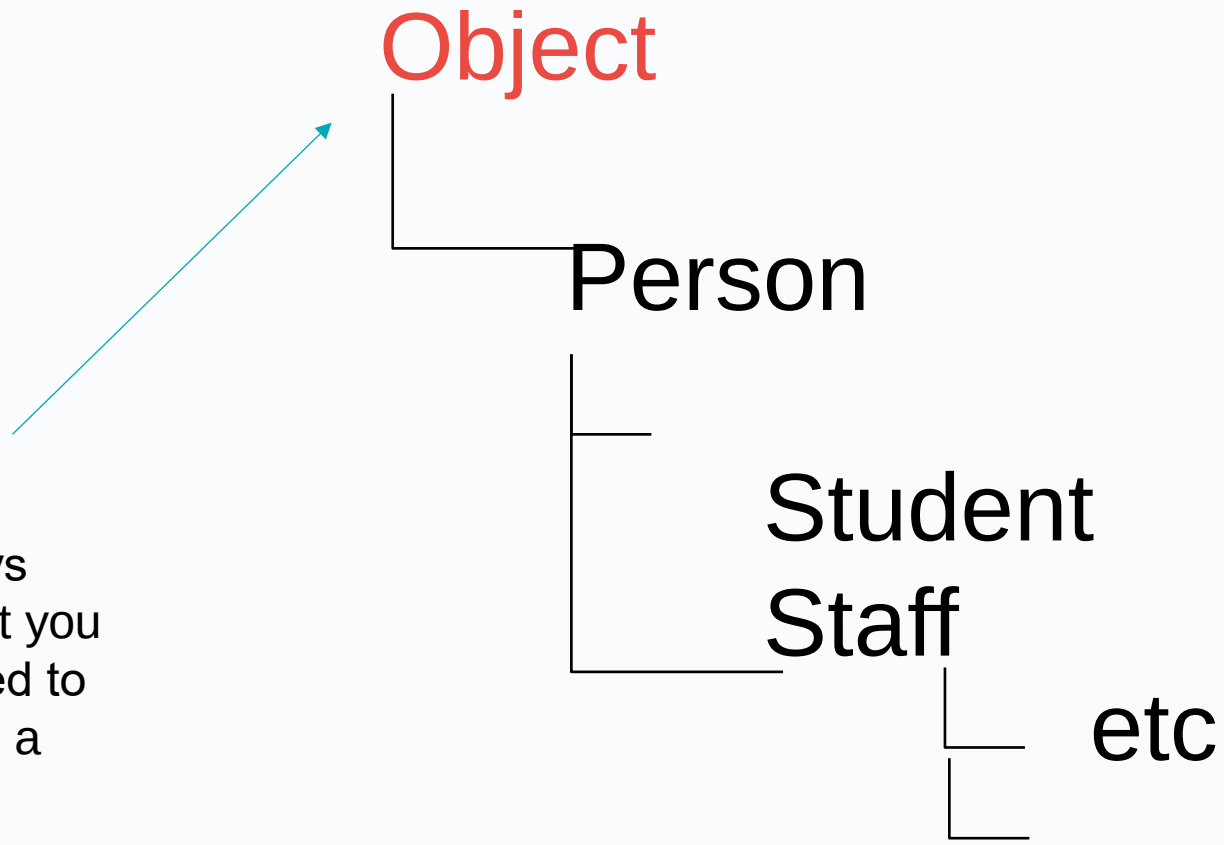
Note the constructor:
Use “super” to call constructor of superclass from subclass

Examine the code – and write the Staff class

Note: “ Object” The root class at the top

- The Object class
- “Adam and Eve” object
- A class with no superclass, extends this class
- toString() behaviour.. How is inheritance linked to this?

Object class

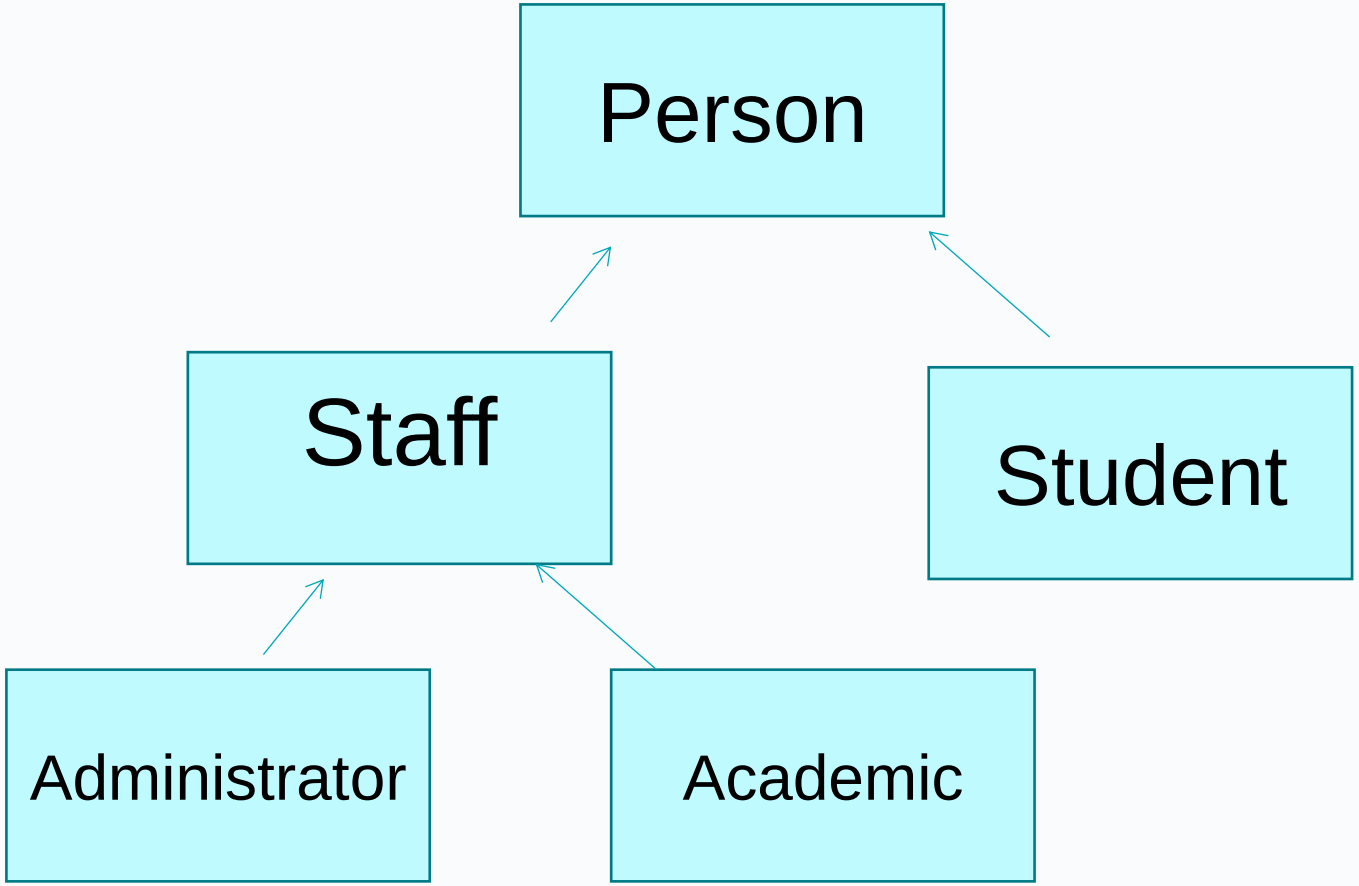


It’s always there, but you don’t need to draw it in a design !

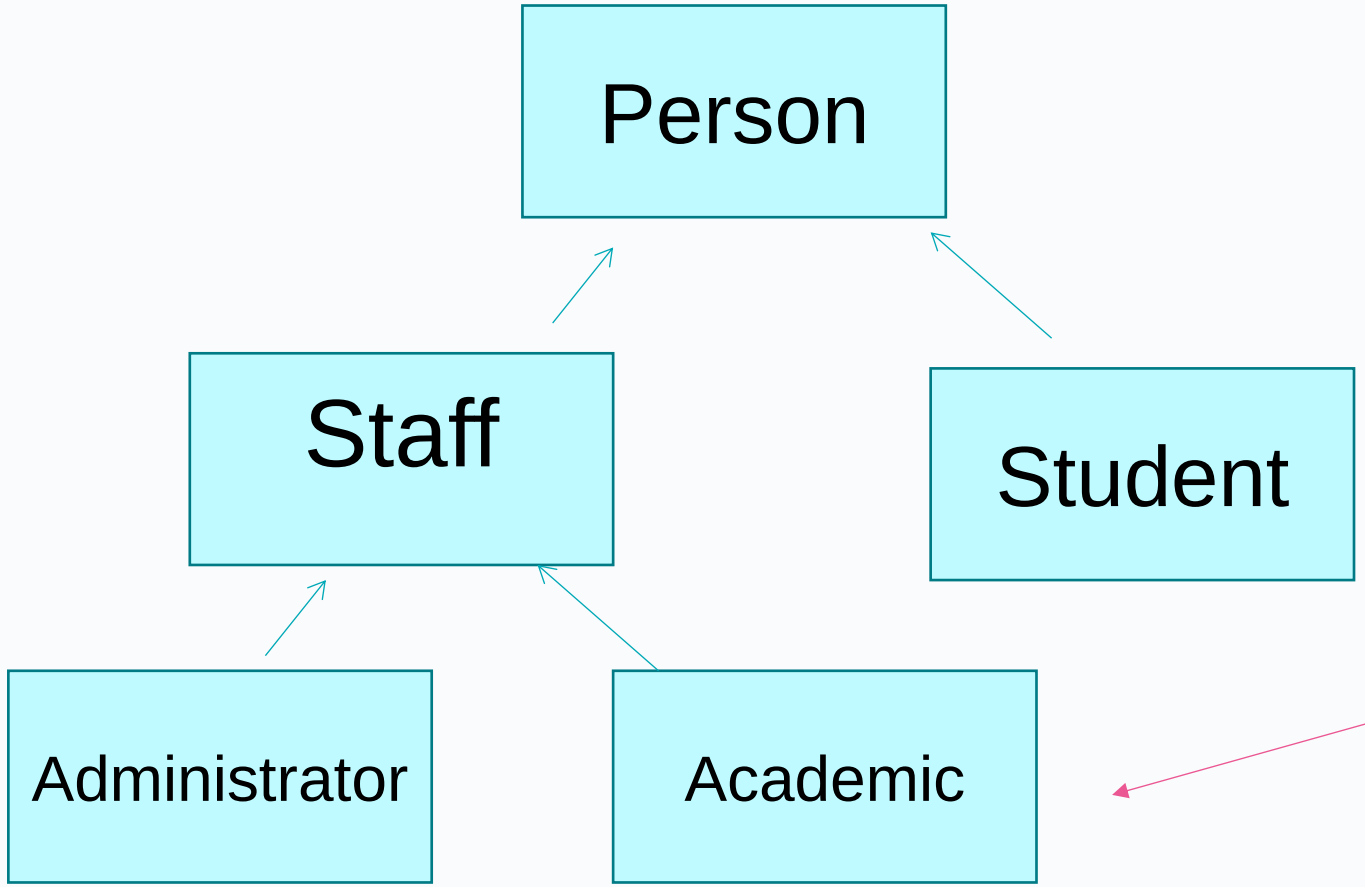
Object “Types”

An important concept in java

What



Object Types



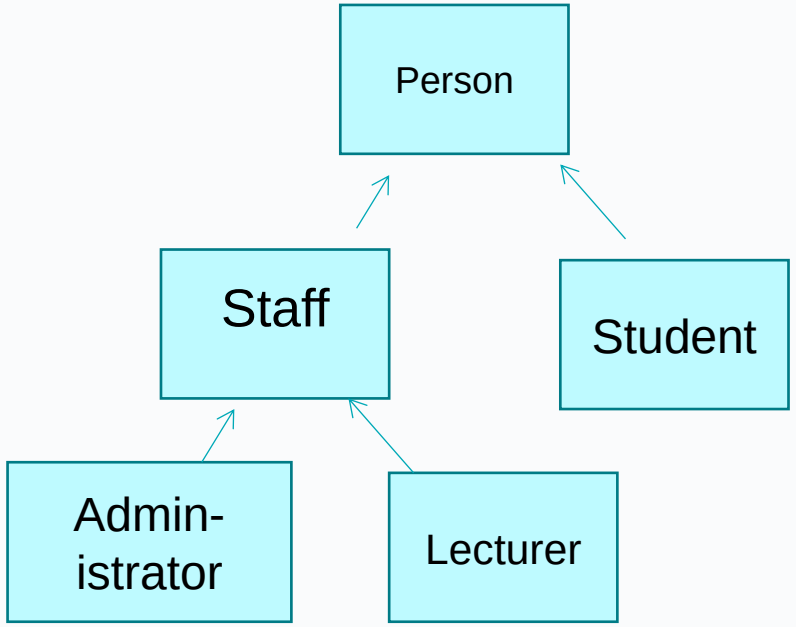
What is the object type(s) for an instance of this class

Or this class

Casting objects

"Casting" is taking an object of one type and converting into another type

In class hierarchies.. works a specific way:



Example

```
Person p1 = new Person();  // create a person object
Student s1 = (Student) p1;   // changes a person object called
p1 into a Student object
```

```
Person p1 = new Person();
Staff  a1 = (Staff)p1;
```

Method Overriding

- Different classes in the hierarchy do things in “their own way” – i.e. have their own version of a method

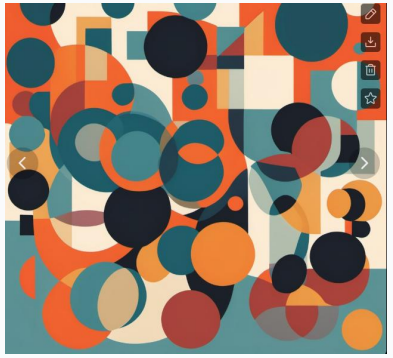
- Note: Use super.superclassmethod() from the subclass method if the superclass does part of the work.
  - avoiding code repetition

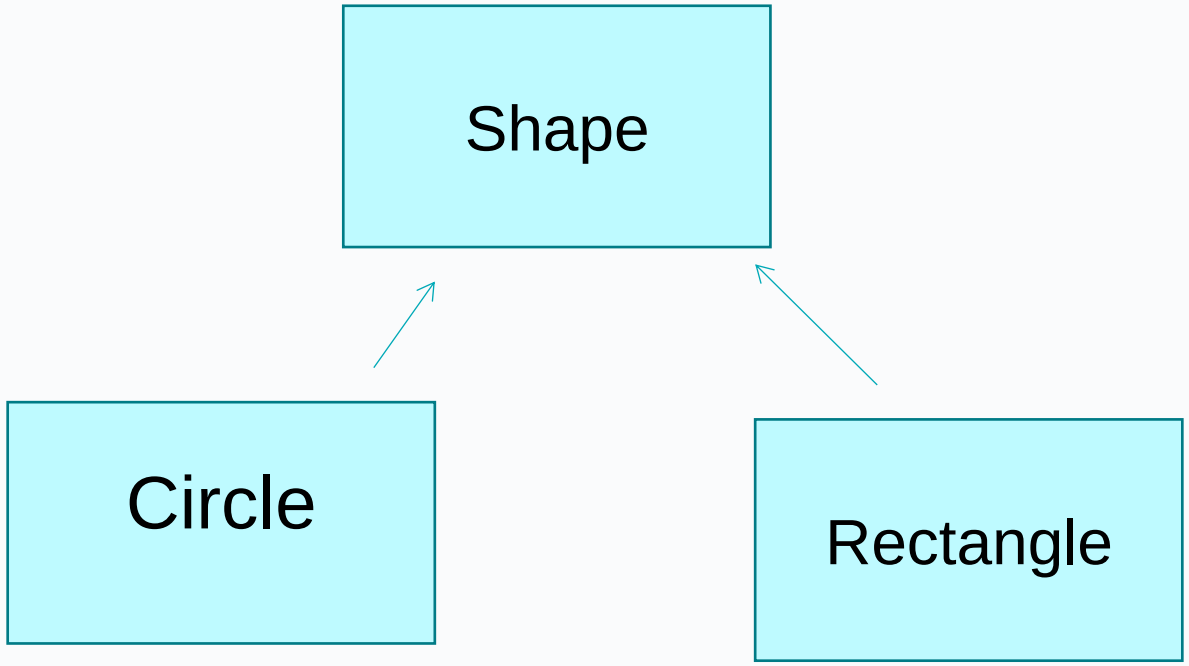
- An example in the toString() method

Another Scenario for inheritance

- Creating a game that will require different **shapes**:
- Circles, rectangles , etc

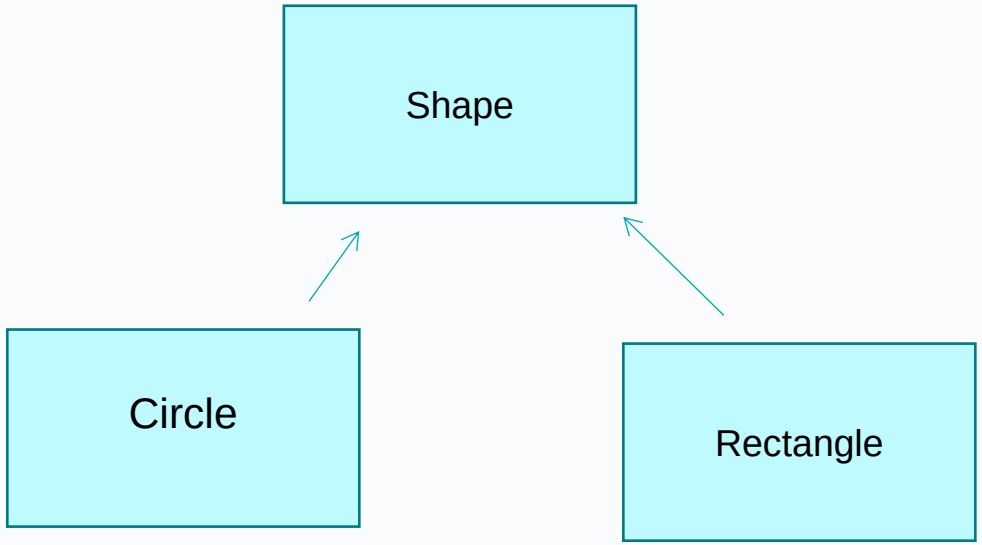
Another Scenario





- Shape at the top of the hierarchy
- Circle/ Rectangle inherit from it.
- Each one has a method to calculate its area

Another Scenario



- A class each..
- Constructor..
- Attributes..
- Its own calculateArea() method

An aside: Arrays

- In Java: Declare the type of objects it will hold – and either the length OR contents

```
String []  setOfWords= new String[4];
```

or

```
String []  setOfWords= {enne, meene,
                        miny, mo}
```

Any type of object e.g

```
Person []  people = new Person[20];
```

Back to Shapes example

- In Java: Declare the type of objects it will hold – and either the length OR contents

```
Shape[]  setOfShapes= new Shape[4];
```

Set each entry to either a Circle or a Rectangle

And loop around calling calculateArea()..

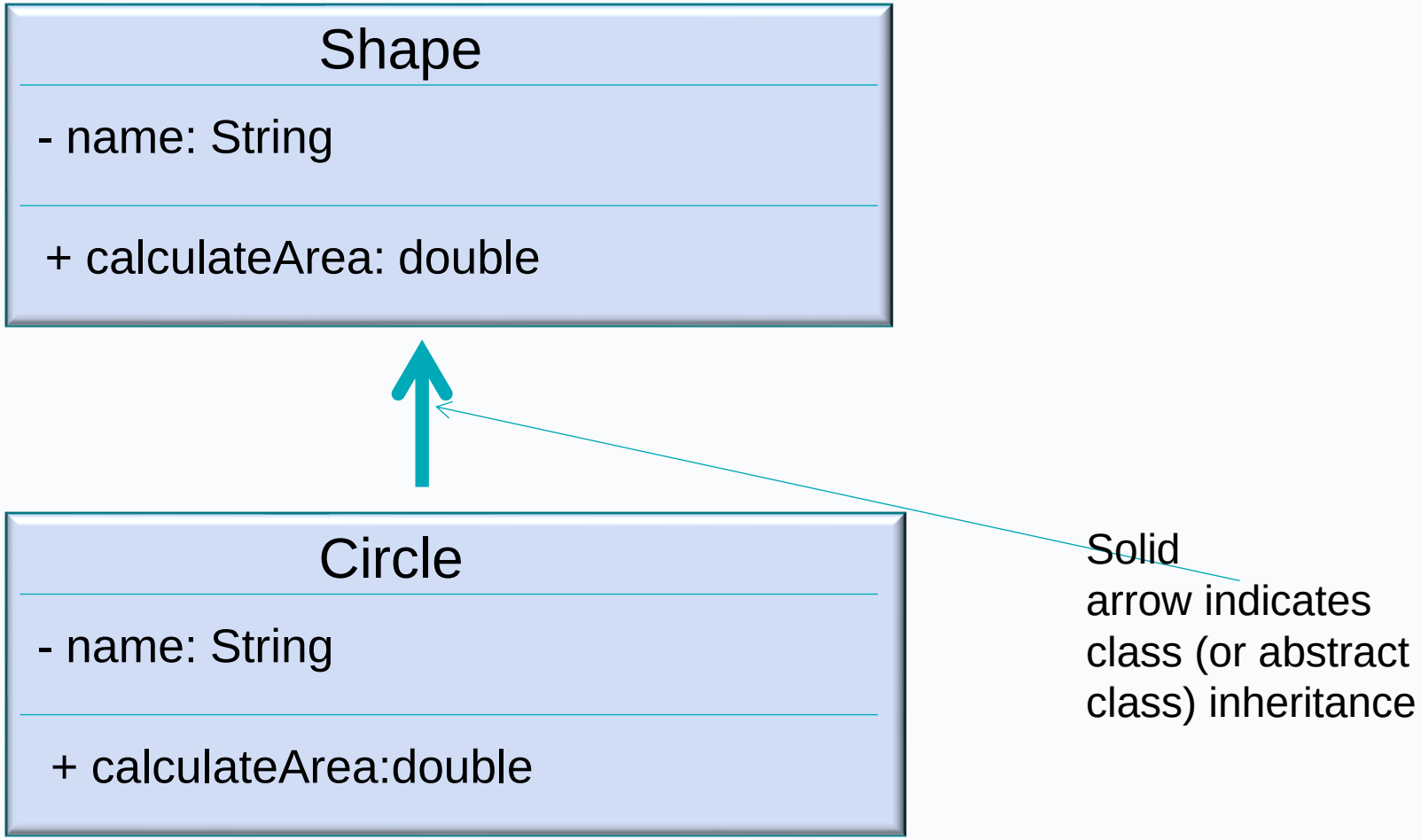
Polymorphism

- In a class hierarchy – “same” method in different classes;
- The behaviour differs depending on the object type
  - E.g. calculateArea

- Demo.. Using Shapes array

- Dynamic binding: the correct method (in this case, calculateArea()) is called depending on the object type..

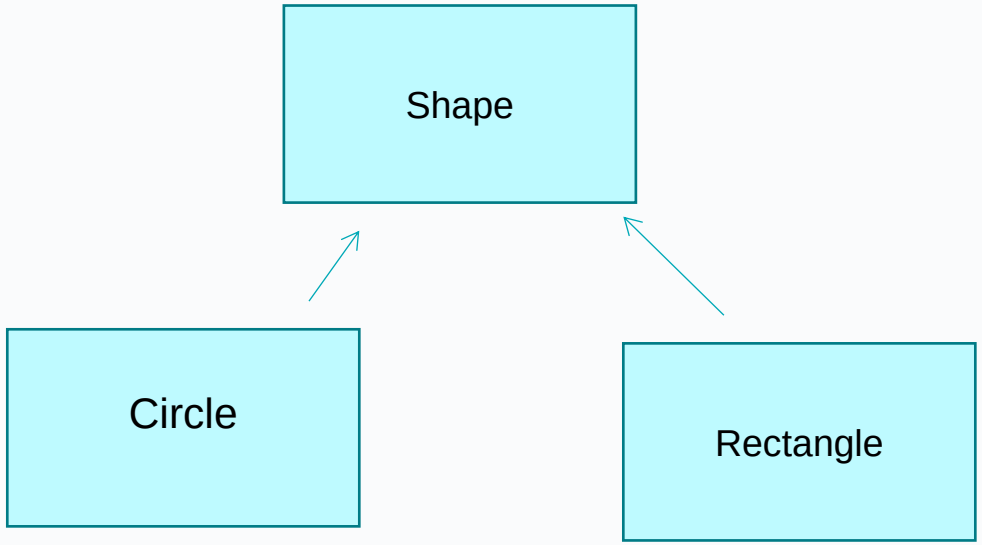
UML : class inheritance



Abstract Classes

- An abstract class or method is defined by the keyword abstract
  -
```
abstract class Shape {
        …
    abstract double calculateArea();
    abstract double
    calculateCircumference();
        …
  };
```
- Any class with an abstract method is automatically abstract and must be declared as such
- Opposite of concrete classes (which can be instantiated)

Scenario



- Shape wasn't ever a "real" thing – no attributes.. Only its subclasses make good objects
- Can make it abstract – and enforce the calculateArea() method to be implemented

“final” keyword

- Final attributes - can’t be changed
  - Used for constant values e.g. ?

```
public final double xPos;
```

- A Final class - can’t be subclassed

```
Public final class Person..
```

- A final method – can’t be overridden

```
Public final void someMethod()
```

What we covered

- Inheritance
  - Why it’s used - No 1 reason: code re-use
  - How it’s used - “extends”

- “Object” class 

- Object types / Casting
- Method overriding
- Polymorphism
- Abstract classes
- “final” keyword