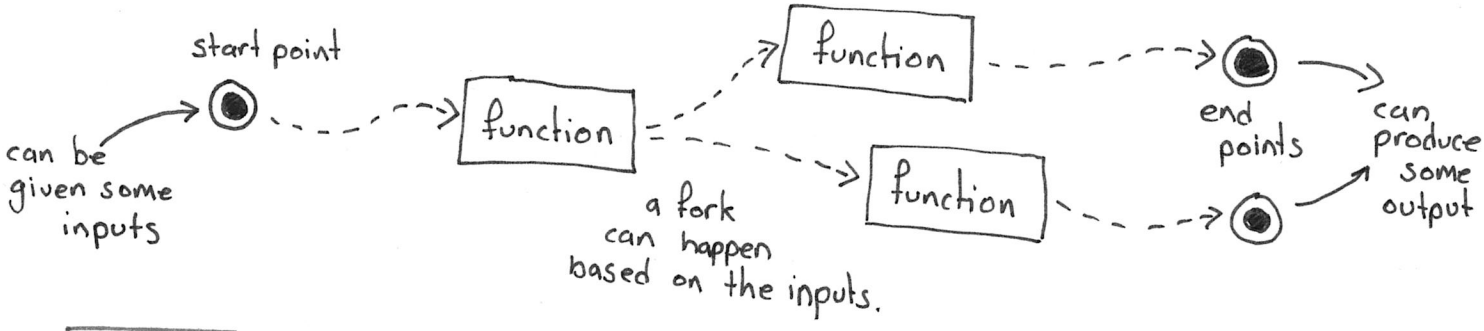
PROGRAMMING PARADIGM

The word paradigm means a typical pattern or model of something. Within programming we use paradigms to bring a structure to the problems and solutions we're working on. We use the programming paradigm to model the problem *domain* (the context and environment of the problem) and our intended solution.

There are 3 main paradigms used in programming: procedural, object oriented, and functional.

PROCEDURAL PARADIGM

Procedural programming, also known as scripting or imperative programming, is a paradigm which follows a linear, or top-down, pattern. We think of the program having a start point or entry point and one or more end points or exit points. The program will run between those points by calling functions, or procedures, which perform some computation.



Loops

In programming a loop is an instruction which allows us to perform a block of code multiple times.

There are two types of loop:

For loop:

```
for (int i : list) {
    //do something
}
```

The for loop iterates over a list or a range, performing the code block once per item.

While loop:

```
while (condition === true) {
    //do something
}
```

Here, the loop continues to run while it's condition is true, as soon as it becomes false it will stop running.

If Statement

An if statement is a conditional instruction, the block of code will be executed if the condition is true:

```
if (condition === true) {
    //do something
} else if (anotherCondition === true) {
    //do something else
} else {
    //do another thing
}
```

Here if the first condition is true then it's block will be executed and the others will not. If anotherCondition is true then it's will be executed, but if neither is true then the else block will be executed.

Map

A map is a data structure which stores key value pairs. In other programming languages they are sometimes called associative arrays (e.g. PHP) or dictionaries (e.g. Python).

```
HashMap<String, String> capitalCities =
        new HashMap<String, String>();
capitalCities.put("England", "London");
capitalCities.put("Scotland", "Edinburgh");
```

Here, we've created a HashMap where the key and value are both strings and we're using it to store the country as the key, and the city as the value.

OBJECT-ORIENTED PARADIGM

Object orientation is a very popular programming paradigm in which everything is described as an object which contains data and code. The data is stored in attributes and the code is stored in procedures / functions.

The easiest way to think of this is that the attributes are the things which describe the object and the functions are the behaviours of the object. E.g. imagine a car, it has attributes like colour, maker, model, horsepower, and brakepower. It also has behaviours like unlock, ignite engine, accelerate and brake.

Objects are created when the program is actually run, not by the programmers themselves, the programmer creates a class which describes what attributes and behaviours it's objects will have. That means we can create as many objects from a class as we need.

Classes

A class is a blue-print or description of an object and what it needs to have in it when it's created.

general practice is to give class names a capital letter.

private and public describe who has access to the class / variables / functions. We'll cover more on that later

```
public class Car {
    private String colour;
    private String maker;
    private String model;
    private int horsepower;
    private int brakepower;

    public void unlock () {
        // code to unlock the car
    }

    public void igniteEngine () {
        // code to ignite engine
    }
}
```

general practice is to give function names lower case letters and be camel case (caps for new words) or snake case (underscores between words)

Here we have defined a class called Car which contains a number of attributes and two behaviours.

If, in another part of our code, we want to create and use a car then we write:

```
Car myNewCar = new Car();
```

Then we can do stuff like:

```
myNewCar.unlock();
myNewCar.igniteEngine();
```

Which will run the functions on that specific instance of the car class.
The object I've created called myNewCar will exist until I get rid of / forget about that variable that refers to it.

The benefit of structuring our code into classes is that we encapsulate the logic. This means that we have our program logic arranged in re-usable, well defined, cohesive (meaning that everything which belongs together is all grouped together, in the same place), blocks of code.

Besides encapsulation, another aspect of the object oriented paradigm is abstraction. Abstraction is when we find some commonalities between objects and group them together in abstract classes which all of those objects extend.

Let's go back to the car example, and think of some other objects which are similar like Vans, Lorries, Motorcycles. All of these have similar attributes and behaviours so rather than duplicating them in all of the classes, we can create an abstract class called Vehicle which contains all of the shared attributes and behaviours.

Abstract class

An abstract class is the same as a class except that it can not be instantiated (meaning you can't make objects of it). It needs to be extended / sub classed and those sub classes will be able to use it's attributes and behaviours.

```
public abstract class Vehicle {

    private String colour;
    private int numberOfWheels;

    abstract public void igniteEngine();

    public void brake(){
        //code to brake
    }

}
```

this is an abstract method which means that every subclass of Vehicle must contain it's own implementation of this function.

this is not an abstract method so all subclasses of Vehicle will share this implementation of the function.

We use abstract methods to say that all instances of this abstract class must contain this behaviour but the way it will achieve the behaviour will be different for every one. E.g. here igniteEngine is abstract because it involves different things depending on whether the Vehicle is a car, a van or a motorcycle.

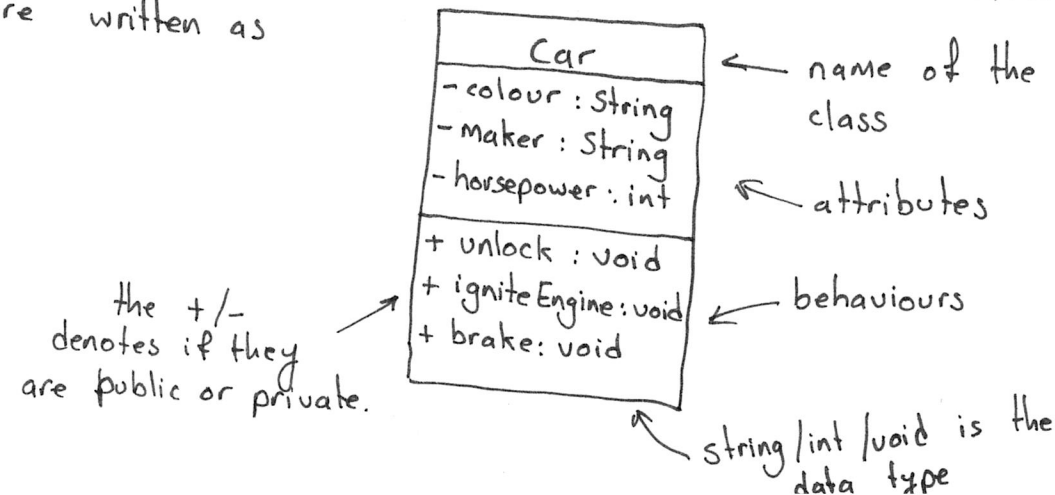
```
public class Car extends Vehicle {
    public void igniteEngine() {
        //the code for car to igniteEngine
    }
}
```

This shows how we use our new abstract class.

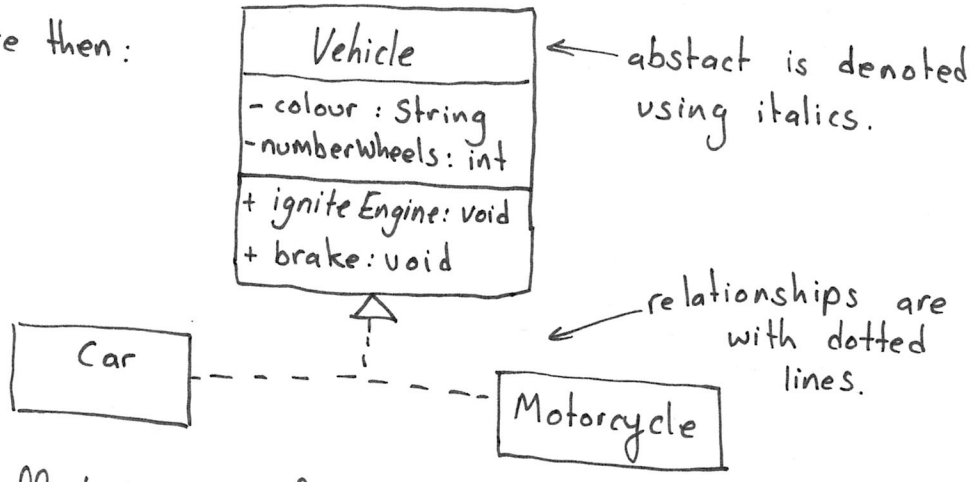
UNIFIED MODELLING LANGUAGE

The unified modelling language or UML diagram is a way in which we show the classes which we make and their relationship.

classes are written as



Abstract classes are then:



UML is an effective tool for keeping track of what classes you have and how they're related.

FUNCTIONAL PARADIGM

Functional programming treats all of the computation as the evaluation of mathematical functions and avoids creating different states or having mutable data (data which is changed after it is created).

It is very hard to write functional-style code in Java, other languages such as Haskal are much more suited to it.

It is very effective for performing complex and repetative tasks which can be represented as mathematical functions which makes it most useful in applications which analyse or mutate datasets.

SOURCE CODE MANAGEMENT

Source code management is a tool which is used by software engineers to store and version (e.g. store each incremental change to) their code. Each version is committed to a remote repository and stored as the new code base.

It also allows us to work collaboratively on the same code base by having one master version which everyone commits their changes to, and pulls everyone elses changes from.

There are a number of different SCM systems, subversion, mercurial, gitLab. We will be using git which is the most popular.