# Object-Oriented Programming (OOP)

The objective of these exercises is to consolidate your understanding of object-oriented programming (OOP) principles.

## Exercise 1 – OOP Principles

The following abbreviations apply:

[**C**]lass or classes, [**O**]bject or objects

[**Y**]es [**N**]o

Only one box should be ticked for each question.

Please tick the appropriate box (class(es) or object(s) ).

| | | C | O |
|---|---|---|---|
| 1 | Behaviour is defined in a(n) | ☐ | ☐ |
| 2 | Fields are defined in a(n) | ☐ | ☐ |
| 3 | Field values exist in a(n) | ☐ | ☐ |
| 4 | A message is (normally) sent to a(n) | ☐ | ☐ |
| 5 | Inheritance is a relationship between | ☐ | ☐ |

Please respond to the statements by ticking a box (yes or no).

| | | Y | N |
|---|---|---|---|
| 6 | An object is an instance of a class. | | |
| 7 | An object must belong to a class. | | |
| 8 | A class must have an object. | | |
| 9 | Behaviour is implemented by methods. | | |
| 10 | A message is a request from one object to another object. | | |
| 11 | An object always expects a response from a message. | | |
| 12 | Derived classes inherit behaviour from their base class(es). | | |
| 13 | A base class inherits the fields and methods of their child classes. | | |
| 14 | A class can have a field of another class. | | |
| 15 | A derived class is another name for a base class. | | |
| 16 | A derived class can have more than one base class. | | |
| 17 | In class 'Mammal', 'Weight' could be a field. | | |
| 18 | In the class 'StopWatch', 'Start' could be a behaviour. | | |
| 19 | A base class has the same number as or fewer fields than a derived class. | | |

**Solutions overleaf**

# Solutions: Exercise 1 – OOP Principles

| | | C | O |
|---|---|---|---|
| 1 | Behaviour is defined in a(n) | X | |
| 2 | Fields are defined in a(n) | X | |
| 3 | Field values exist in a(n) | | X |
| 4 | A message is (normally) sent to a(n) | | X |
| 5 | Inheritance is a relationship between | X | |

Please respond to the statements by ticking a box (yes or no).

| | | Y | N |
|---|---|---|---|
| 6 | An object is an instance of a class. | X | |
| 7 | An object must belong to a class. | X | |
| 8 | A class must have an object. | | X |
| 9 | Behaviour is implemented by methods. | X | |
| 10 | A message is a request from one object to another object. | X | |
| 11 | An object always expects a response from a message. | | X |
| 12 | Derived classes inherit behaviour from their base class(es). | X | |
| 13 | A base class inherits the fields and methods of their child classes. | | X |
| 14 | A class can have a field of another class. | X | |
| 15 | A derived class is another name for a base class. | | X |
| 16 | A derived class can have more than one base class. | | X |
| 17 | In class 'Mammal', 'Weight' could be a field. | X | |
| 18 | In the class 'StopWatch', 'Start' could be a behaviour. | X | |
| 19 | A base class has the same number as or fewer fields than a derived class. | X | |

# Exercise 2 – OOP Relationships

For each of the following questions, select the correct relationship (association / aggregation / inheritance / interface). Think carefully about the context.

Then, if you think it's an:

**Association**: Suggest a role name for each end of the association

**Aggregation**: Suggest a delegated operation

**Inheritance**: Suggest an inherited operation

**Interface**: Suggest an implemented operation

Remember:

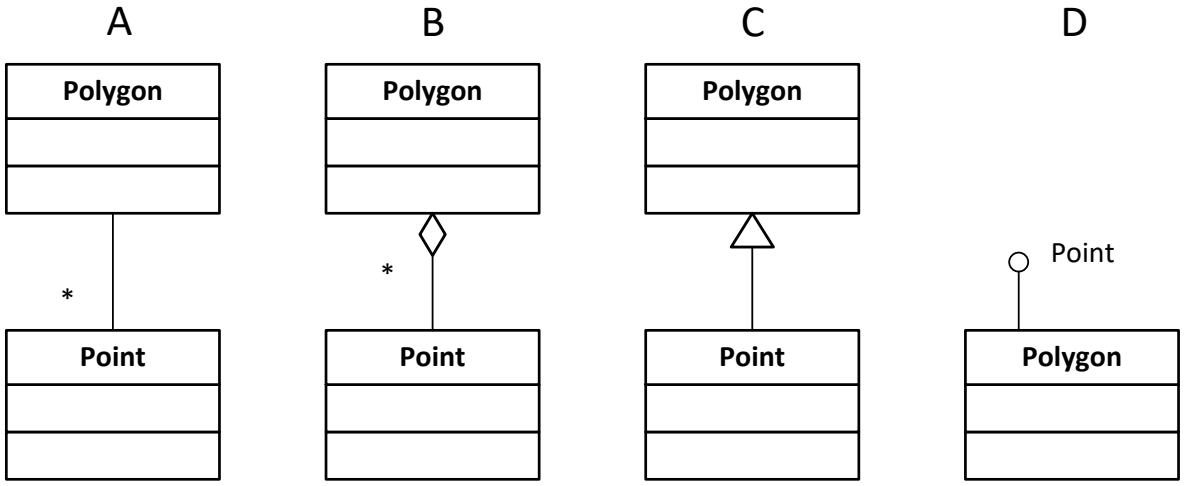
Association = *"Knows about"*

Aggregation = *"Made up of"*

Inheritance = *"Is a type of"*

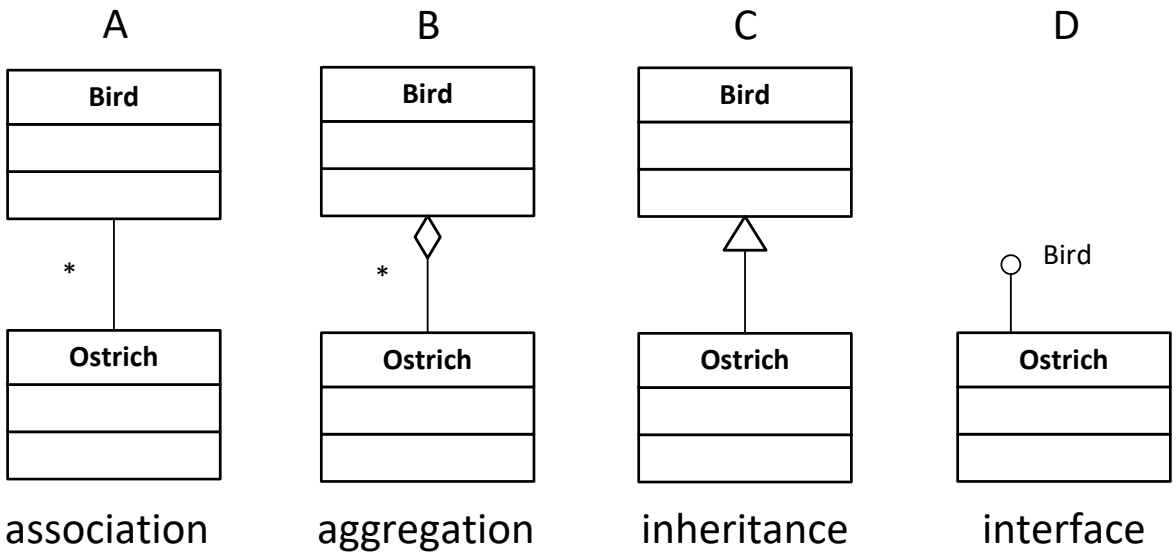
Interface = *"Can do"*

# Polygon / Point



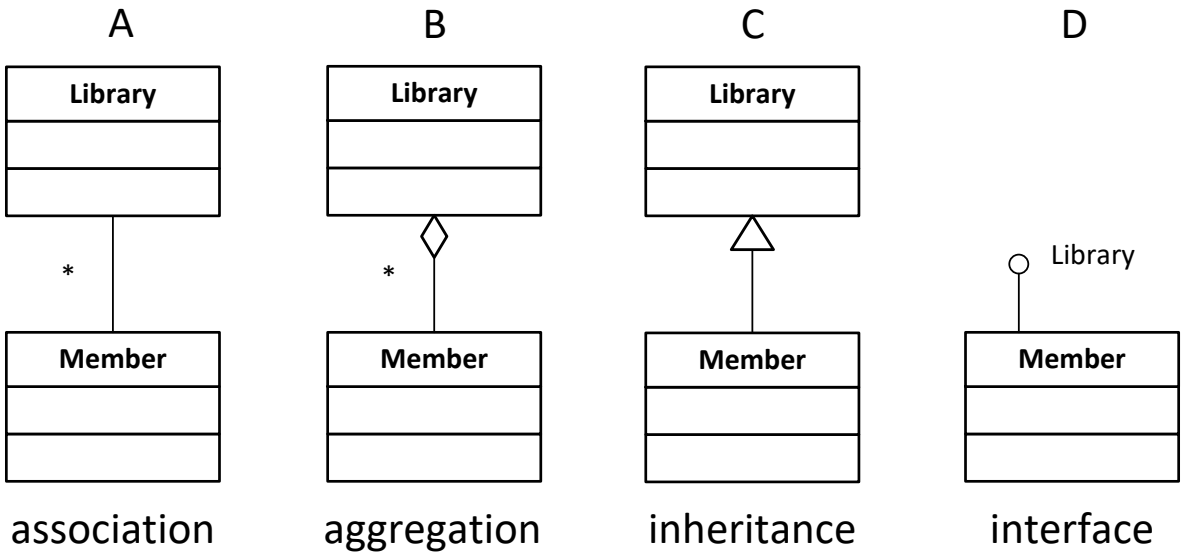

# Bird / Ostrich

Context: A Film Animation Company



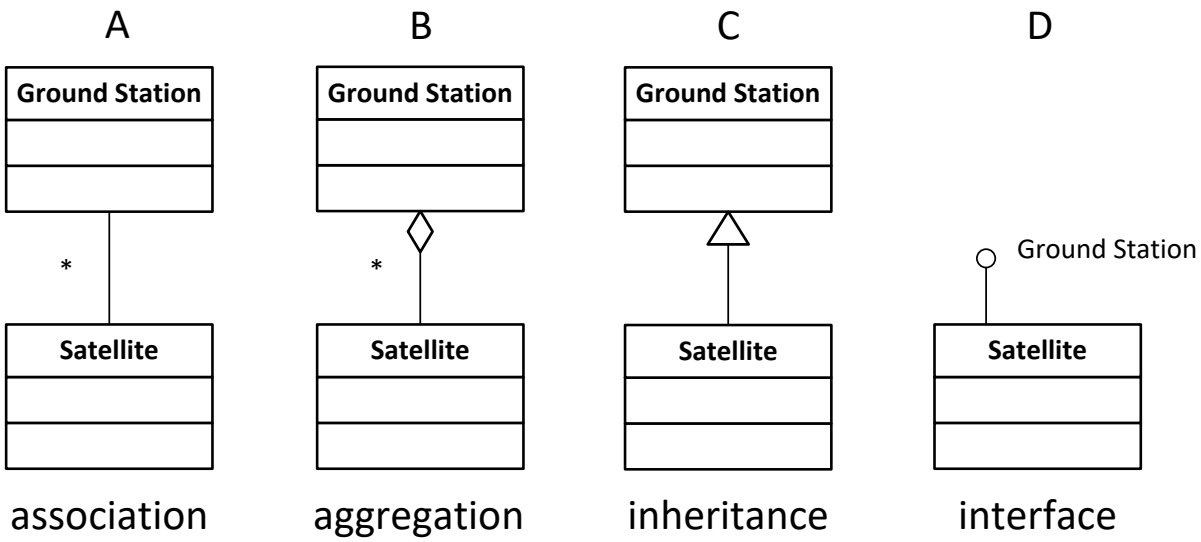
# Library / Member

Context: A Public Library Checkout system

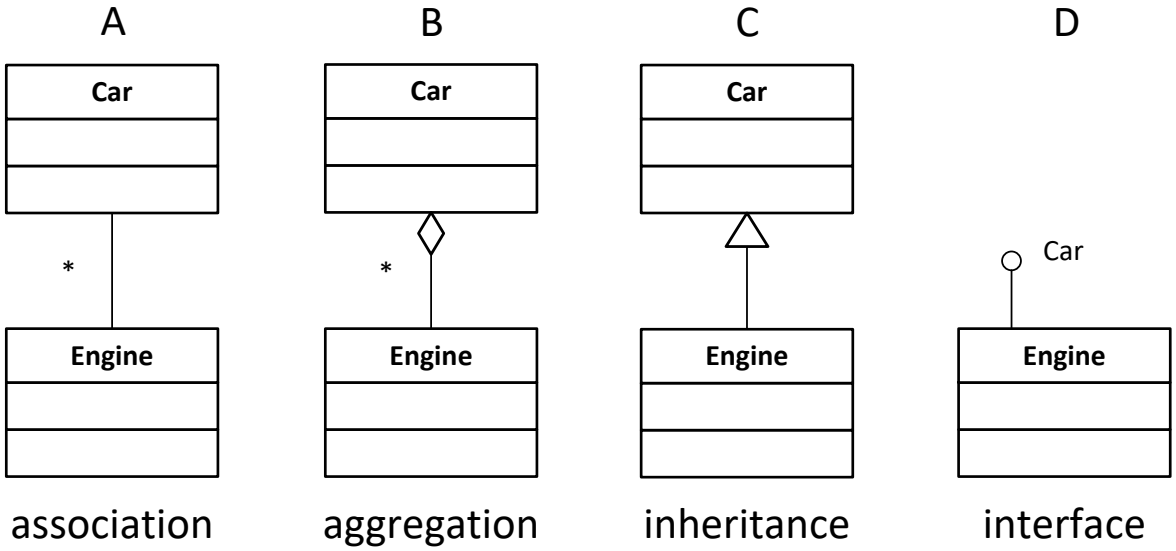



# Ground Station / Satellite

Context: Goonhilly, which monitors any satellite it can see

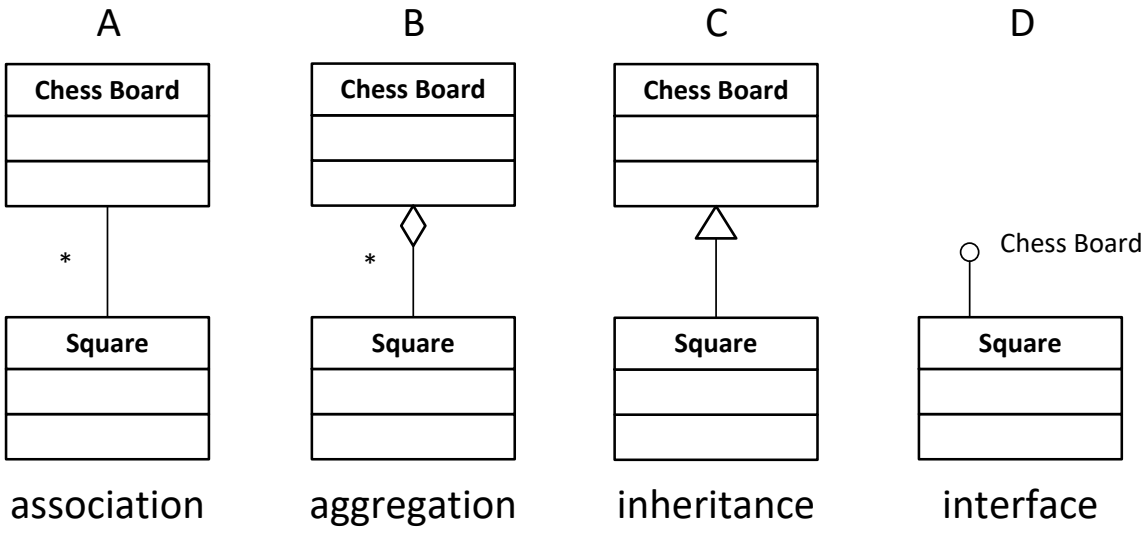


# Car / Engine




# Chess Board

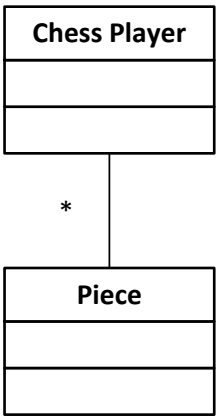
Context: A conference room repeater display of a chess match



# Chess Player

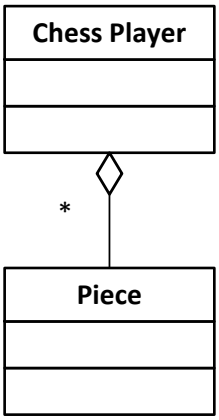
Context: The repeater board software. The "Player" is software, not the real player.

A



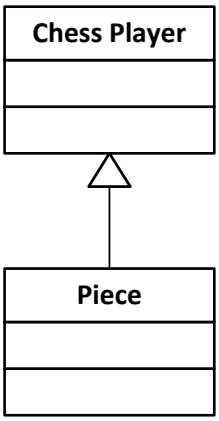

association

B



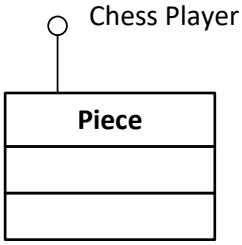

aggregation

C




inheritance

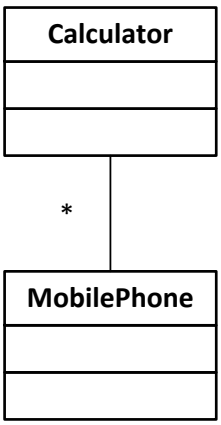
D




interface

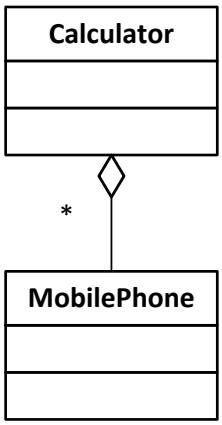
# A Mobile Phone's Calculator app

A



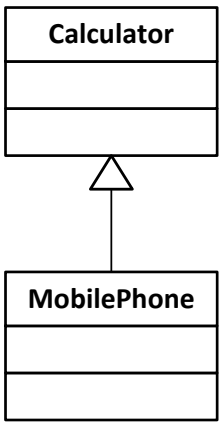

association

B



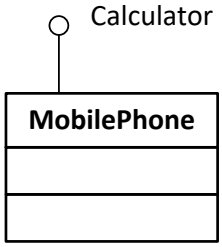

aggregation

C




inheritance

D




interface

**Solutions overleaf**

# Solution: Exercise 2 - OOP Relationships

***Polygon / Point***

Aggregation.

A delegated operation: Move()

***Bird / Ostrich***

It's inheritance, *provided* you can define meaningful behaviour for "Fly()" – assuming your Birds fly.

If your Birds really have to fly in the air then an Ostrich is *not* a type of Bird. It might be a type of Flightless Bird (i.e. a specialisation of Bird) provided there is also a sub-type Flighty Birds (which can fly). Of course, if the animation company is making a movie about an ostrich that flies, then there's no problem!

You could have Flyable as an interface.

An inherited operation: Fly() [or at least, FlapWings()!]

***Library / Member***

Aggregation.

A delegated operation: CheckoutBook()

***Ground Station / Satellite***

Association. If the Ground Station controlled the satellites, then there is an argument for aggregation.

Role Names of association: tracker – target

***Car / Engine***

Aggregation.

The car is the sum of its parts – of which Engine is one.

A delegated operation: Start()

***Chess Board***

Aggregation. The board is a collection of squares and achieves its purpose through the squares.

Delegated operation: Draw() (implemented by asking each square to draw itself)

***Chess Player***

Even though the English flows as a “Chess Player *has* Pieces” and the real human player achieves their purpose by moving Pieces, in the software world the Player is just the owner of the pieces and the software player does not move the pieces (this might be different for a chess-playing game, rather than our repeater software).

So, association.

Role Names: owner – owned by

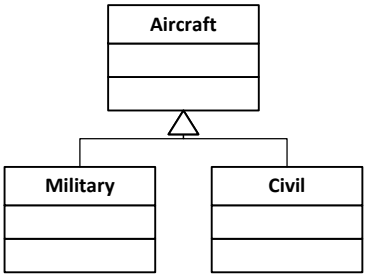
***A mobile phone that has a calculator***

This can be represented either as aggregation (but with the parent class being Mobile Phone) or by interface. It is not inheritance – a mobile Phone is *not* a type of calculator.

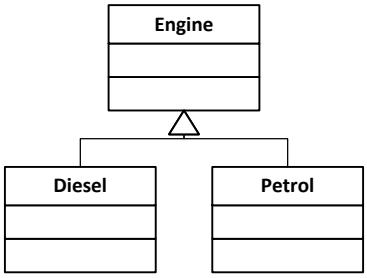
# Exercise 3 – OOP Subclassing

For these questions state whether, *given the supplied context*, sub-classing is correct or not. Give a yes/no answer and suggest a reason. The acid test is 'do they have different rules, or is it just a passive piece of data?'.

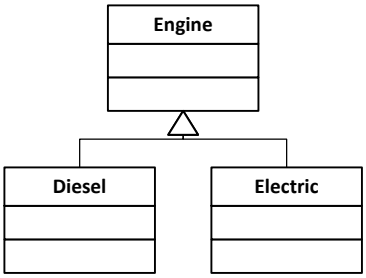
Context: Air Traffic Control



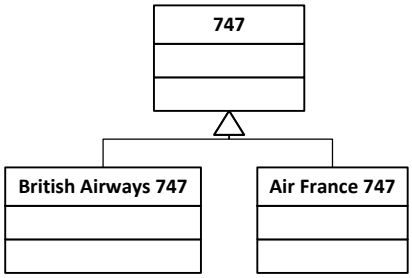

Context: Fuel Duty Payable



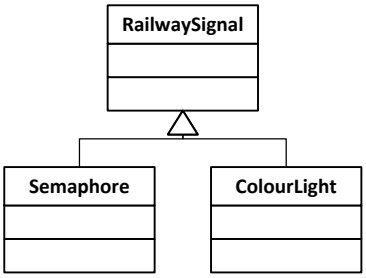

Context: Passenger making a train journey



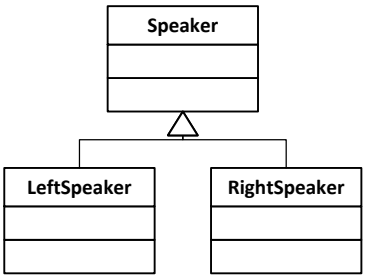

Context: Paris CDG Air Traffic Control



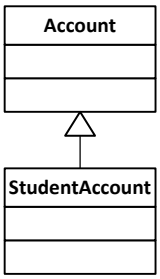

Context: Railway Signal



## Context: Hi-fi system

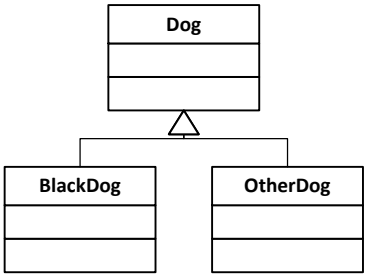



## Context: Bank Account

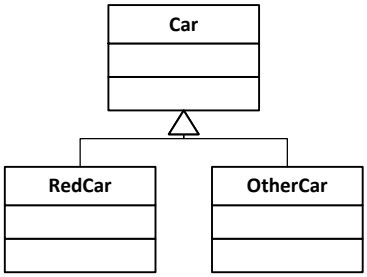



The only difference between a student account and a regular account is that on opening, a student gets a £50 “sweetener”.

## Context: Dog Training Club




## Context: Insurance Company




**Solutions Overleaf**

# Solution: Exercise 3 - OOP Subclassing

| Context | ? | Notes |
|---|---|---|
| Air Traffic Control | Yes | Military & Civil aircraft operate under different rules |
| Fuel Duty Payable | Yes | Again, different rules |
| Passenger making a journey | No | Do they care? |
| Paris Airport ATC | Yes | Air France planes get priority in France. |
| Signal | Yes | |
| Stereo Hi-Fi | No | We just have 2 speakers. We call one left & one right. |
| Student Account | No | It's just a bit of (starting) data that has no effect on subsequent behaviour. This should be handled by a parameter passed in at Object construction time. However, we do need to store this rule somewhere. |
| Dog Training Club | No | The colour of the dog is just a bit of data and does not affect behaviour. However, I have heard that Golden Retrievers just cannot be trained the same way as other dogs, so there may be sub-classing "Golden Retriever" and "Other". |
| Insurance Company | No | There are many more factors than just colour taken into account when assessing risk. |

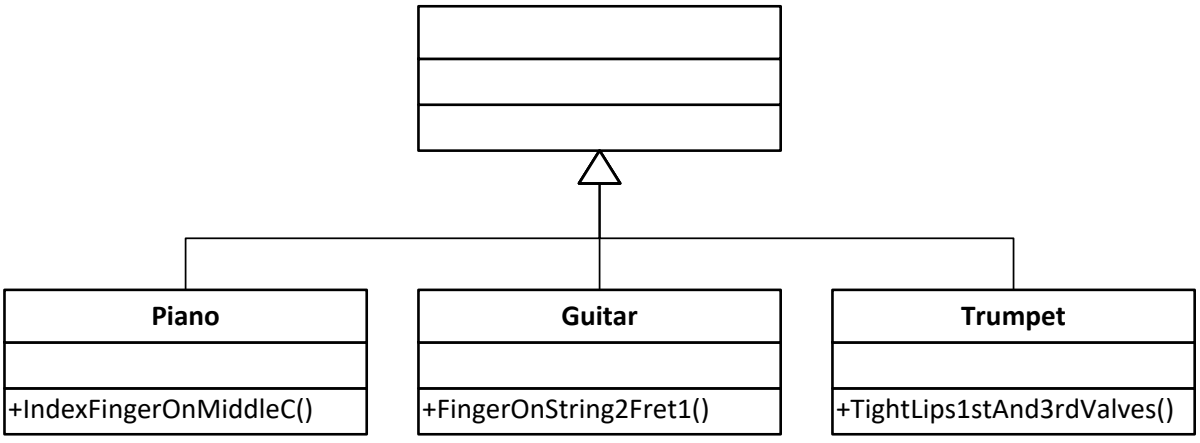
# Exercise 4 - OOP Superclass Operations

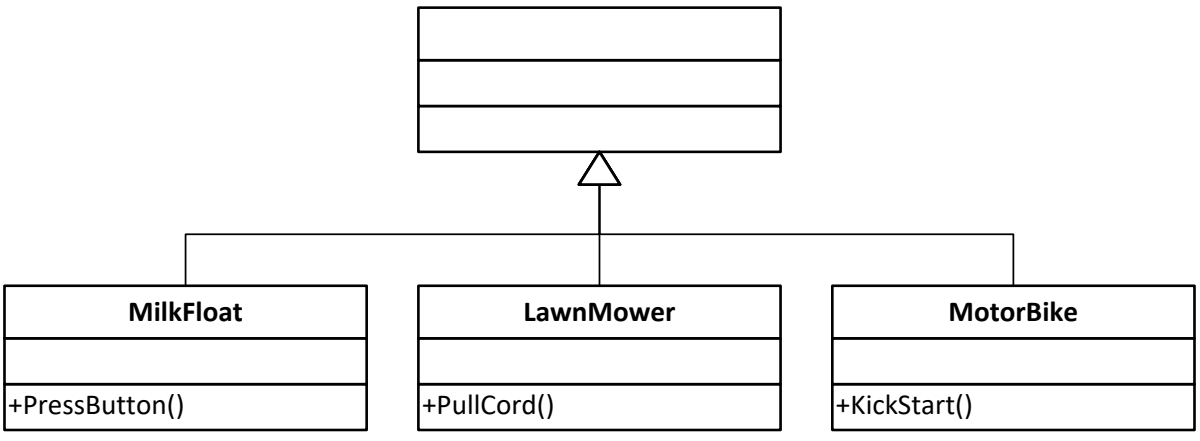
In the following two scenarios, some sub-classes are shown with methods.

Give a name to the superclass and state what the polymorphic operation might be at the superclass that may call the given methods in the subclasses. Assume that the polymorphic method at the subclass will merely call the stated method.

Remember that the superclass should have no knowledge of the subclass's behaviour; this extends to choosing a suitable name for the superclass operation so that the name doesn't inadvertently expose the behaviour.

Note that although the exercise refers to super classes, the same could equally be applied to interfaces.




**Solutions Overleaf**

# Solution: Exercise 4 - OOP Superclass Operations