OOP: concepts 1

Practice 2

OOP is a programming paradigm that differs from procedural programming. The statements that follow attempt to describe some core concepts of OOP but only **two** of the statements are accurate.

Select the **two accurate statements** from the list provided.

- [ ] A parent class inherits all of the attributes and methods of the child classes that it has.
- [ ] Access to the data of an object can be restricted using the `personal` and `public` access modifiers.
- [ ] To solve a problem, a program is divided into smaller parts called subroutines.
- [ ] A class is a series of step-by-step instructions that solve a problem.
- [ ] Multiple instances of a class can be created, each with different values for its attributes.
- [ ] The behaviour of a method that a child class has inherited from a parent class can be altered so that it behaves differently.

OOP: sequence code

Challenge 2

The following class has been defined using pseudocode.

Pseudocode

```
CLASS Radio
    PRIVATE volume: integer
    PRIVATE station: string
    PRIVATE on: Boolean

    PUBLIC PROCEDURE Radio(given_station)
        station = given_station
        volume = 3
        on = False
    ENDPROCEDURE

    PUBLIC FUNCTION get_volume()
        RETURN volume
    ENDFUNCTION

    PUBLIC FUNCTION get_station()
        RETURN station
    ENDFUNCTION

    PUBLIC FUNCTION is_on()
        RETURN on
    ENDFUNCTION

    PUBLIC PROCEDURE set_volume(new_volume)
        volume = new_volume
    ENDPROCEDURE

    PUBLIC PROCEDURE set_station(new_station)
        station = new_station
    ENDPROCEDURE

    PUBLIC PROCEDURE switch()
        IF on == True THEN
            on = False
        ELSE
            on = True
        ENDIF
    ENDPROCEDURE

ENDCLASS
```

In testing, it was found that the volume of the radio could be set to an unsafe level. The `set_volume` method must be updated so that it does not allow the volume to exceed a setting of 30. Drag and drop the given statements to create an updated version of the method.

You must use **all of the statements** with **correct indentation** in your solution.

Available items

```
ENDIF
```

```
volume = 30
```

```
IF new_volume > 30 THEN
```

```
ELSE
```

```
ENDPROCEDURE
```

```
volume = new_volume
```

```
PUBLIC PROCEDURE set_volume(new_volume)
```

Quiz:
STEM SMART Computer Science Week 22

Inheritance

Practice 1

Which of the following statements best describes **inheritance** in object-oriented programming (OOP)?

- The process of encapsulating data and methods into a single unit, known as a class.
- The process of creating a new class from an existing class, that allows the new class to acquire its attributes and methods.
- A principle that emphasises the ability of objects to behave differently based on their data types.

Quiz:
STEM SMART Computer Science Week 22

OOP: concepts 3

Challenge 1

Alex wants to develop a simple video game where players can choose between different characters, each with unique abilities.

He decides to implement a superclass to represent common attributes shared by all characters, such as health points and movement speed, and their common behaviours. He then creates subclasses for each character type, such as "Warrior", "Mage", and "Archer" with their unique behaviours.

Which core OOP concept is Alex applying when he creates his subclasses?

- Encapsulation
- Polymorphism
- Abstraction
- Inheritance

Quiz:
STEM SMART Computer Science Week 22

OOP: concepts 2

Challenge 2

Marina is programming an application that aims to help students revise for their Biology lessons. The definitions she has written (so far) for the `Animal` and `Frog` classes of the program is presented below.

Pseudocode

```
CLASS Animal
    PRIVATE habitat: String

    PUBLIC FUNCTION get_habitat()
        RETURN habitat
    ENDFUNCTION

    PUBLIC PROCEDURE display_habitat()
        PRINT("My natural habitat is " + habitat)
    ENDPROCEDURE
ENDCLASS

CLASS Frog EXTENDS Animal
    PRIVATE secondary_habitat: String

    PUBLIC FUNCTION get_secondary_habitat()
        RETURN secondary_habitat
    ENDFUNCTION

    PUBLIC PROCEDURE display_habitat()
        PRINT("I am an amphibian, I live in the " + habitat + " and also in the '
    ENDPROCEDURE
ENDCLASS
```

Select the OOP concepts that have been applied in this example.

- [ ] Encapsulation
- [ ] Polymorphism
- [ ] Decomposition
- [ ] Inheritance

Quiz:
STEM SMART Computer Science Week 22

Polymorphism

Practice 2

Olivia maintains the computer systems for a car manufacturer who has traditionally made cars with internal combustion engines (ICE) but is branching out into the production of electric vehicles. She has used the technique of **polymorphism** in the design of her classes.

Which of the following examples uses polymorphism?

Example 1 | Example 2 | Example 3 | Example 4

```
CLASS IceCar

    PRIVATE tank_capacity
    PRIVATE mpg
    PRIVATE registration_number

    PUBLIC PROCEDURE IceCar(given_reg_no, capacity, output)
        registration_number = given_reg_no
        battery_capacity = capacity
        power_output = output
    ENDPROCEDURE

    PUBLIC FUNCTION get_registration()
        RETURN registration_number
    ENDFUNCTION

    PUBLIC FUNCTION calculate_mileage()
        RETURN tank_capacity / mpg
    ENDFUNCTION
ENDCLASS

CLASS ElectricCar

    PRIVATE battery_capacity
    PRIVATE power_output
    PRIVATE registration_number

    PUBLIC PROCEDURE ElectricCar(given_reg_no, capacity, output)
        registration_number = given_reg_no
        battery_capacity = capacity
        power_output = output
    ENDPROCEDURE

    PUBLIC FUNCTION get_registration()
        RETURN registration_number
    ENDFUNCTION

    PUBLIC FUNCTION calculate_range()
        RETURN battery_capacity * power_output
    ENDFUNCTION
ENDCLASS
```

Example 1 Example 2 Example 3 Example 4

```
CLASS Car

    PRIVATE registration_number

    PUBLIC PROCEDURE Car(given_reg_no)
        registration_number = given_reg_no
    ENDPROCEDURE

    PUBLIC FUNCTION get_registration()
        RETURN registration_number
    ENDFUNCTION

ENDCLASS

CLASS IceCar EXTENDS Car

    PRIVATE tank_capacity
    PRIVATE mpg

    PUBLIC PROCEDURE IceCar(given_reg_no, capacity, output)
        SUPER(given_reg_no)
        battery_capacity = capacity
        power_output = output
    ENDPROCEDURE

    PUBLIC FUNCTION get_range()
        RETURN tank_capacity / mpg
    ENDFUNCTION
ENDCLASS

CLASS ElectricCar EXTENDS Car

    PRIVATE battery_capacity
    PRIVATE power_output

    PUBLIC PROCEDURE ElectricCar(given_reg_no, capacity, output)
        SUPER(given_reg_no)
        battery_capacity = capacity
        power_output = output
    ENDPROCEDURE

    PUBLIC FUNCTION get_range()
        RETURN battery_capacity * power_output
    ENDFUNCTION
ENDCLASS
```

Example 1 Example 2 Example 3 Example 4

```
CLASS Car

    PRIVATE registration_number

    PUBLIC PROCEDURE Car(given_reg_no)
        registration_number = given_reg_no
```

```
    ENDPROCEDURE

    PUBLIC FUNCTION get_registration()
        RETURN registration_number
    ENDFUNCTION

ENDCLASS

CLASS IceCar EXTENDS Car

    PRIVATE tank_capacity
    PRIVATE mpg

    PUBLIC PROCEDURE IceCar(given_reg_no, capacity, output)
        SUPER(given_reg_no)
        battery_capacity = capacity
        power_output = output
    ENDPROCEDURE

    PUBLIC FUNCTION get_tank_capacity()
        RETURN tank_capacity
    ENDFUNCTION

    PUBLIC FUNCTION get_mpg()
        RETURN mpg
    ENDFUNCTION

ENDCLASS

CLASS ElectricCar EXTENDS Car

    PRIVATE battery_capacity
    PRIVATE power_output

    PUBLIC PROCEDURE ElectricCar(given_reg_no, capacity, output)
        SUPER(given_reg_no)
        battery_capacity = capacity
        power_output = output
    ENDPROCEDURE

    PUBLIC FUNCTION get_battery_capacity()
        RETURN battery_capacity
    ENDFUNCTION

    PUBLIC FUNCTION get_power_output()
        RETURN power_output
    ENDFUNCTION

ENDCLASS
```

Example 1 Example 2 Example 3 **Example 4**

```
CLASS IceCar

    PUBLIC tank_capacity
    PUBLIC mpg
    PUBLIC registration_number
```

```
    PUBLIC PROCEDURE IceCar(given_reg_no, capacity, output)
        registration_number = given_reg_no
        battery_capacity = capacity
        power_output = output
    ENDPROCEDURE

ENDCLASS

CLASS ElectricCar

    PUBLIC battery_capacity
    PUBLIC power_output
    PUBLIC registration_number

    PUBLIC PROCEDURE ElectricCar(given_reg_no, capacity, output)
        registration_number = given_reg_no
        battery_capacity = capacity
        power_output = output
    ENDPROCEDURE

ENDCLASS
```

- Example 1
- Example 2
- Example 3
- Example 4

Quiz:
STEM SMART Computer Science Week 22

Relationship between classes

Challenge 2

Sam is creating a game where each player can choose the character (or sprite) that they can play with. A part of the definitions of the `Sprite` and `Game` classes is presented below. In the main program, an instance of the `Game` class called `my_game` is created.

Select the statement that correctly describes the type of relationship between the `Sprite` and `Game` classes.

Pseudocode

```
CLASS Sprite
    PRIVATE score: Integer
    PRIVATE name: String

    PUBLIC PROCEDURE Sprite(given_name)
        score = 0
        name =  given_name
    ENDPROCEDURE

    PUBLIC FUNCTION get_name()
        RETURN name
    ENDFUNCTION

    PUBLIC FUNCTION get_score()
        RETURN score
    ENDFUNCTION
ENDCLASS

CLASS Game
    PRIVATE my_sprite: Sprite

    PUBLIC PROCEDURE Game()
        my_sprite = NEW Sprite("Nikita")
        PRINT(my_sprite.get_name())
    ENDPROCEDURE
ENDCLASS

// Main program
PROCEDURE new_game()
    my_game = NEW Game()
ENDPROCEDURE
```

- **Inheritance**, because through the `my_sprite` object, the `Game` class inherits all of the attributes and methods of the `Sprite` class.
- **Encapsulation**, because the `my_sprite` object is created within the `Game` class.
- **Composition**, because if the `my_game` object is destroyed, then the `my_sprite` object will also be destroyed.

- **Aggregation**, because the `Game` class 'has a' `Sprite` object called `my_sprite`.

Quiz:
STEM SMART Computer Science Week 22

Raspberry Pi Foundation

OOP: class diagram

Challenge 1

Ben is writing an OOP program for an online chess game. He has sketched a **class diagram** to show the relationships between some of his classes. This diagram is shown in in **Figure 1**.

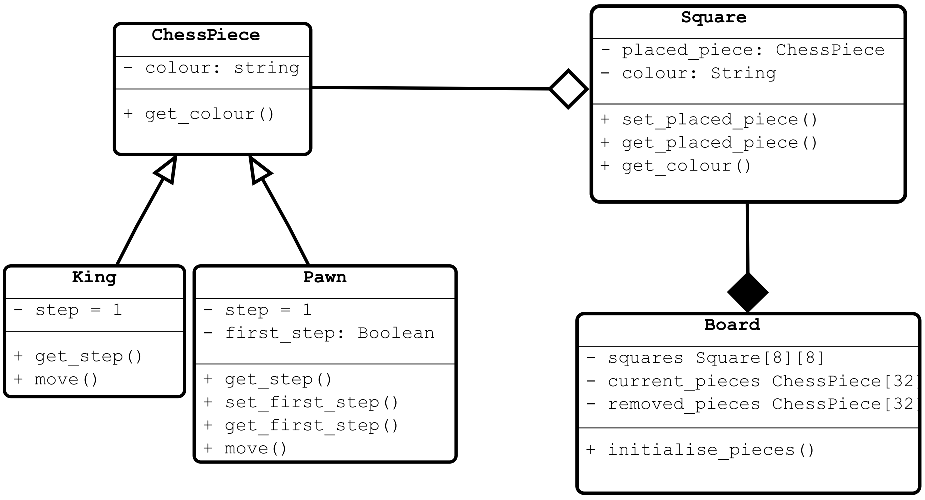

Figure 1: Ben's class diagram

The class diagram uses standard notation (UML) to show the relationship between the classes. These relationships are core OOP concepts.

Match each OOP concept to its description in the table.

OOP concept	Description
	`Square` objects are instantiated within the `Board` class and cannot exist separately from it.
	The implementation of the `move` method can be different for the `King` and `Pawn` classes, even though they have the same parent class.
	A `King` is a `ChessPiece`.
	A `Square` 'has a' `ChessPiece`, but the `ChessPiece` will already exist before it is placed on a `Square`, and it will cease to be linked to a specific `Square` as soon as it moves to a new one.

Items:

Composition Aggregation Inheritance Polymorphism

OOP: benefits and drawbacks 1

Challenge 1

All programming paradigms and languages have strengths, and also some weaknesses, and you need to understand these to pick the best option to work with.

Read each of the following statements and decide whether it is a correct statement relating to using an object-oriented programming language. Label each statement as **True** or **False** by dragging the correct label into the adjacent cell.

Statement	Label
1. OOP design techniques often make it easier to fully model a complete system.	
2. A system that relies on high volumes of message passing can degrade performance.	
3. Classes are modular, making maintenance easier.	
4. Encapsulation prevents direct access to private attributes.	
5. Classes cannot be extended to add extra functionality.	

Items:

True False