OOP: concepts 4

Practice 1

In the object-oriented programming paradigm, code is organised into classes. Within a class you will find attributes and methods with access modifiers that specify which of these properties can be accessed from outside the class.

Study the following class definition:

Pseudocode

```
CLASS Elf
    PRIVATE strength: Integer
    PRIVATE speed: Integer
    PUBLIC power: String

    PUBLIC PROCEDURE Elf(given_strength, given_speed)
        strength = given_strength
        speed = given_speed
        power = "Archery"
    ENDPROCEDURE

    PUBLIC FUNCTION get_strength()
        RETURN strength
    ENDFUNCTION
ENDCLASS

aegnor = NEW Elf(20, 50)
```

The table below gives a list of terms that are relevant to OOP classes. Use the class definition above to pick an appropriate example for each of the terms. Drag the example into the cell next to the term.

Term	Label
An attribute	
A method	
A class	
A reference variable	
An access modifier	
A data type	

Term	Label
A parameter	

Items:

strength Integer PRIVATE Elf aegnor get_strength given_strength

Encapsulation

Practice 1

Select one statement that describes why the principle of **encapsulation** is important for the design of OOP programs.

- ○ It ensures that child classes can take the attributes and methods of the parent class.
- ○ It allows the methods of child classes to behave in different ways to those inherited from a parent class.
- ○ It enables the creation of objects with specific states and behaviour.
- ○ It ensures that any interaction with an object, specifically the manipulation of its data, is only allowed via its public interface.

Quiz:
STEM SMART Computer Science Week 45

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 45

Attribute and method accessibility

Practice 2

●●

Classes are the main building blocks of an object-oriented system.

Explain how public, protected, and private attributes or methods are different. [3 marks]

Quiz:
STEM SMART Computer Science Week 45

OOP: complete code 1

Challenge 2

Mike is creating a game inspired by *The Lord of the Rings*. Each character of the game can belong to one of the following tribes: Elves, Dwarves, Hobbits, Men, Wizards, Orcs, and Trolls. The characters that belong to the tribe of Elves have a commonly known name and a secret elven name.

A part of the definitions of the `Elf` and `Character` classes is presented below. `SUPER` is used to call the constructor of the `Elf` parent class from the `Character` child class.

In the main program, an instance of the `Character` class called `my_character` is created, and then **an output statement** is used to demonstrate the value of an attribute.

Pseudocode

```
CLASS Elf
    PRIVATE strength: Integer
    PRIVATE speed: Integer
    PUBLIC power: String

    PUBLIC PROCEDURE Elf(given_strength, given_speed)
        strength = given_strength
        speed = given_speed
        power = "Archery"
    ENDPROCEDURE

    PUBLIC FUNCTION get_strength()
        RETURN strength
    ENDFUNCTION
ENDCLASS

CLASS Character EXTENDS Elf
    PRIVATE elf_name: String
    PUBLIC name: String

    PUBLIC PROCEDURE Character(given_strength, given_speed, given_elf_na
        SUPER(given_strength, given_speed)
        elf_name = given_elf_name
        name = given_name
     ENDPROCEDURE

    PUBLIC FUNCTION get_elf_name()
        RETURN elf_name
    ENDFUNCTION
ENDCLASS

// Main program
PROCEDURE new_character()
    my_character = NEW Character(200, 1000, "Greenleaf", "Legolas")
    PRINT(..................) // Missing code for the output statement
ENDPROCEDURE
```

Look at the list of output statements and select the **two** statements that will cause an error if they are used in the main program to display the value of an attribute.

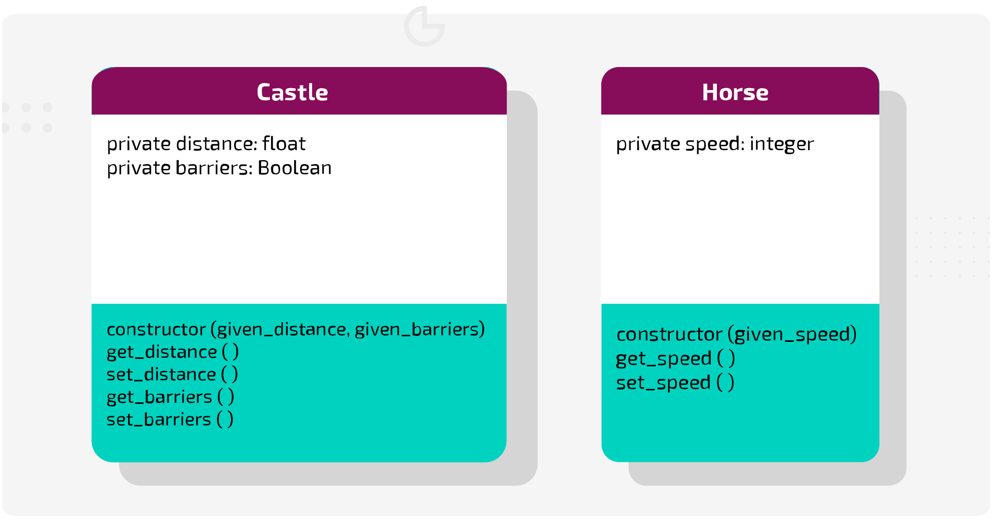
- [ ] `PRINT(my_character.speed)`
- [ ] `PRINT(my_character.get_elf_name())`
- [ ] `PRINT(my_character.name)`
- [ ] `PRINT(my_character.get_strength())`
- [ ] `PRINT(my_character.power)`
- [ ] `PRINT(my_character.elf_name)`

Quiz:
STEM SMART Computer Science Week 45

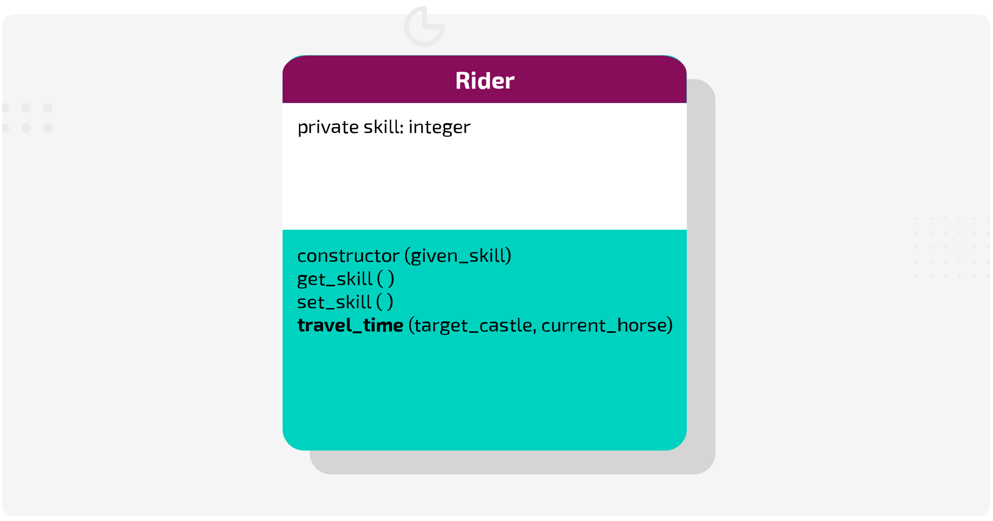
OOP: complete code 4

Challenge 1

Maxime is programming a horse riding simulation where a rider travels to various historic castles in the UK with different breeds of horses. Each breed of horse can travel at a different speed, for example, a `Thoroughbred` can travel at `88 km/h`, a `QuarterHorse` at `70 km/h`, and an `Appaloosa` at `65 km/h`. She has created three classes for her program.



`Castle` and `Horse` classes


Rider
private skill: integer
constructor (given_skill)
get_skill ( )
set_skill ( )
travel_time (target_castle, current_horse)

`Rider` class

The `travel_time` method of the `Rider` class has the following functionality:

- If the `skill` of the person travelling is greater than 8 and there are no `barriers` on the way to the castle, then the **time** required to reach the castle is calculated as the `distance` to the castle divided by the `speed` of the horse
- In any other scenario, the time taken to reach the castle is **doubled**.

Part A

Enter the statement that completes the code under the `IF` statement.

Pseudocode

```
PUBLIC FUNCTION travel_time(target_castle, current_horse)
    IF skill > 8 AND target_castle.get_barriers() == False
        time_required =  ........................................ // Missing code
    ELSE
        // It takes double the time
    ENDIF

    RETURN time_required
ENDFUNCTION
```

Part B

Based on the implementation of the `travel_time` method, what is the time required to reach the castle if the below set of objects are used in the simulation? Give your answer as a number.

Pseudocode

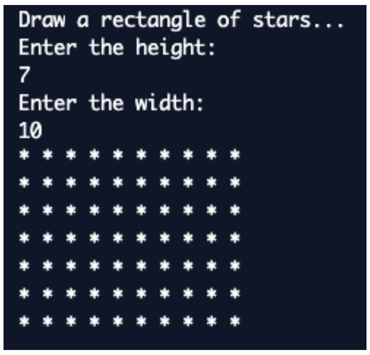
```
    my_castle = new Castle(130, True)

    my_appaloosa = new Horse(65)

    my_rider = new Rider(9)
```

OOP: complete code 5

Practice 1

Steve is learning object-oriented programming. He wants to write some code that will draw a simple rectangle, so he has created a class called `Rectangle`, with two attributes: `height` and `width` and a method `draw`.

The `draw` method will display a rectangle of stars of a given height and width. For example, a rectangle of height 7 and width 10, will look like this:



```
Draw a rectangle of stars...
Enter the height:
7
Enter the width:
10
* * * * * * * * * * *
* * * * * * * * * * *
* * * * * * * * * * *
* * * * * * * * * * *
* * * * * * * * * * *
* * * * * * * * * * *
* * * * * * * * * * *
```

Example output from Steve's program

Pseudocode

```
CLASS Rectangle
    PRIVATE width
    PRIVATE height

    PUBLIC PROCEDURE Rectangle(given_width, given_height)
        width = given_width
        height = given_height
    ENDPROCEDURE

    PUBLIC PROCEDURE draw()
        FOR row = 0 TO height
            FOR column = 0 TO width
               PRINT("* ")
            NEXT column
        PRINT() # Print new line.
        NEXT row
    ENDPROCEDURE
ENDCLASS

PRINT("Draw a rectangle of stars…")
input_height = INPUT("Enter the height: ")
input_width = INPUT("Enter the width: ")



```

```
    my_shape = NEW Rectangle(_______, _______) // Missing code
    my_shape.draw()
```

In the pseudocode shown above, a new Rectangle object is instantiated and its `draw` method is called. However, the line of code that instantiates the object has missing arguments. Can you identify what they should be?

Enter your answer as `argument1, argument2` using a comma to separate the arguments.

Quiz:
STEM SMART Computer Science Week 45

Features of polymorphism

Challenge 1

In object-oriented programming, polymorphism is a key concept that allows objects to be treated as instances of their parent class.

State two features of polymorphism.

[2 marks]

Quiz:
STEM SMART Computer Science Week 45

Inheritance

Challenge 1

🟦⬜

Inheritance is a key concept in object-oriented programming (OOP) that involves relationships between classes. It enables certain functionalities and benefits in software development.

Describe two features of inheritance in object-oriented programming (OOP).

[2 marks]

Quiz:
STEM SMART Computer Science Week 45

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**.

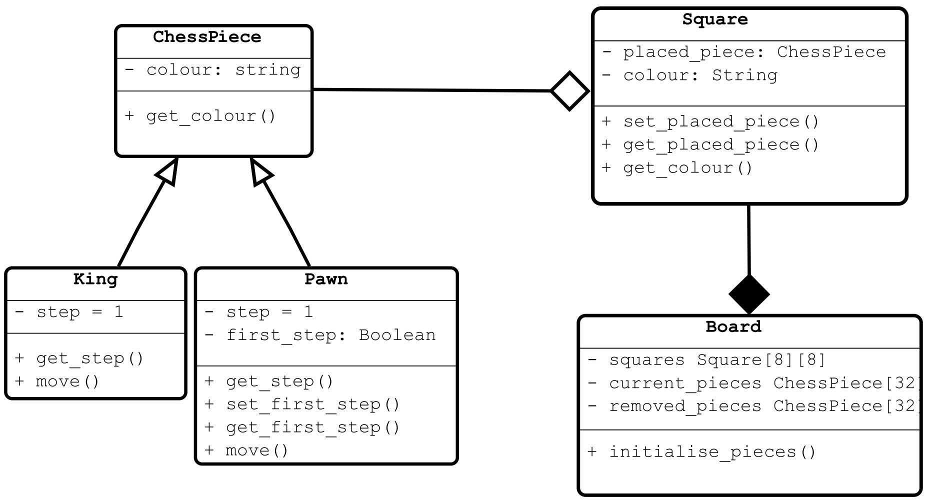


Figure 1: Ben's class diagram

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