ANSWERS

A1.8 Inheritance, polymorphism and overriding

Programming techniques and object-oriented programming · A level · OCR H446 1.2.4, AQA 7517 4.1.2.3, Eduqas A500QS 1.4 · about 25 min

What this lesson is about

Subclasses and super, overriding, polymorphism, and abstract, virtual and static methods, with robot behaviours as subclasses.

- Inheritance
- Overriding
- Polymorphism
- Abstract, virtual and static methods
- Drawing inheritance

Questions

5 marks in all

1. What does this program print? [1 mark]

```
class Robot:
    def __init__(self, name):
        self.name = name
    def greet(self):
        return "I am " + self.name

class Rover(Robot):
    def greet(self):
        return super().greet() + " and I rove"

class Drone(Robot):
    pass

for r in [Rover("R1"), Drone("D1")]:
    print(r.greet())
```

Answer:

```
I am R1 and I rove
I am D1
```

Rover overrides greet and extends the superclass version with super(); Drone does not override it, so it inherits Robot's.

2. What is polymorphism? [1 mark]

- ☐ **A** The same method call behaving differently depending on the class of the object it is called on
- ☐ **B** A class inheriting from two superclasses
- ☐ **C** Hiding an object's attributes
- ☐ **D** Creating many objects from one class

Answer: A. A loop can call act() on any behaviour, and each object's own version runs.

3. What is true of an abstract class? [1 mark]

- ☐ **A** It cannot be instantiated, and its abstract methods must be overridden by subclasses
- ☐ **B** It has no attributes
- ☐ **C** It cannot have subclasses
- ☐ **D** All its methods are static

> **Answer:** A. An abstract class exists to be inherited from; its abstract methods have no body to run.

4. A static method is called as Motor.clamp(120). What can it not do? [1 mark]

- ☐ **A** Use the attributes of a particular object
- ☐ **B** Take parameters
- ☐ **C** Return a value
- ☐ **D** Be called from another method

> **Answer:** A. A static method belongs to the class, so it has no self and no object's attributes.

5. Inheritance models which relationship between a subclass and its superclass? Answer with two words. [1 mark]

> **Answer:** is a. A Chirp is a Behaviour; has a relationships are aggregation and composition.

The task: behaviours by inheritance

The starter has the superclass `Behaviour`, with a `name` attribute and an `act()` method. Write three subclasses from the class diagram. Each constructor calls `super().__init__` with the behaviour's name, and each overrides `act()` so that it does its job and then prints `<name> done`: - `Chirp(note)`: name `chirp`; stores `note` (a frequency in Hz) and its `act` plays that note twice, 0.2 seconds each time. - `Square(size)`: name `square`; stores `size` (cm) and its `act` drives a square with sides of `size` cm, turning right 90 degrees at each corner. - `Spin()`: name `spin`; its `act` turns right a full 360 degrees. Then make the list `routine = [Chirp(523), Square(20), Spin(), Chirp(784)]` and call `act()` on each object in order with one loop. Do not use `isinstance` or `type`: polymorphism chooses the method.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

class Behaviour:
    def __init__(self, name):
        self.name = name

    def act(self):
        print(self.name, "does nothing")

routine = [Behaviour("chirp"), Behaviour("square")]
for behaviour in routine:
    behaviour.act()
```

The hint students can ask for: The base class already has a name and an act method. Each subclass passes its own name up to the base constructor and overrides act with what it does. The loop at the end calls the same method on every object and never asks which class it is.

A solution

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

class Behaviour:
    def __init__(self, name):
        self.name = name

    def act(self):
        print(self.name, "does nothing")

class Chirp(Behaviour):
    def __init__(self, note):
        super().__init__("chirp")
        self.note = note

    def act(self):
        tone(self.note, 0.2)
        tone(self.note, 0.2)
        print(self.name, "done")

class Square(Behaviour):
    def __init__(self, size):
        super().__init__("square")
```

```
        self.size = size

    def act(self):
        for i in range(4):
            forward(60, distance=self.size)
            turn_right(30, angle=90)
        print(self.name, "done")

class Spin(Behaviour):
    def __init__(self):
        super().__init__("spin")

    def act(self):
        turn_right(40, angle=360)
        print(self.name, "done")

routine = [Chirp(523), Square(20), Spin(), Chirp(784)]
for behaviour in routine:
    behaviour.act()
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.