1.2.4 Object-Oriented Languages

Object-Oriented Programming (OOP)

Object-oriented programming is a programming paradigm where programs are organised around **objects**.

An object represents something from the real world and contains:

- **Attributes** (data)
- **Methods** (functions that operate on the data)

Key Concepts of OOP

Classes

A **class** is a blueprint for creating objects.

It defines:

- It defines the attributes an object stores
- and the methods it can perform.

```
class Employee:
    def __init__(self, name, position):
        self.__name = name
        self.__position = position
        self.__hours_worked = 0
        self.__hourly_rate = 0

    def get_name(self):
        return self.__name

    def set_hours_worked(self, hours):
        self.__hours_worked = hours

    def set_hourly_rate(self, rate):
        self.__hourly_rate = rate

    def calculate_pay(self):
        return self.__hours_worked * self.__hourly_rate
```

Objects

An **object** is an instance of a class.

Instantiation

Instantiation is the process of creating an object from a class. The constructor is called when the object is instantiated.

```
employee1 = Employee("John Doe", "Software Engineer")
```

Private, Protected and Public Specifiers

- **Private** – Accessible only in the class in which it was defined
- **Protected** – Accessible in the class in which it defined and its subclasses
- **Public** – Accessible anywhere in the program

Encapsulation

Encapsulation means combining data and methods into one object and controlling access to the data.

Attributes are private and only accessible via public methods.

```
class Student:
    def __init__(self, name, age):
        self.__name = name
        self.__age = age

    def get_name(self):
        return self.__name
```

Inheritance

Inheritance allows one class to take characteristics from another class.

A subclass takes on the attributes and methods of a superclass and can declare additional

Attributes and methods of its own.

```
class Vehicle:
    def __init__(self, x, forward_speed):
        self._position = (x, 0)
        self._forward_speed = forward_speed

    def move(self):
        self._position[0] = self._position[0] + self._forward_speed


class Helicopter(Vehicle):
    def __init__(self, x, y, forward_speed, upward_speed):
        super().__init__(x, forward_speed)
        self._position = (x, y)
        self.__upward_speed = upward_speed

    def move(self):
        super().move()
        self._position[1] = self._position[1] + self.__upward_speed
```

`Helicopter` has inherited the attributes `position` and `forward_speed` from the `Vehicle` class. It has also inherited the method `move`.

It has also declared the additional attribute `upward_speed`.

Polymorphism

Polymorphism means different objects can respond differently to the same method call.

A method in a subclass overrides an inherited method to provide new behaviour.

In the code above, `Helicopter` has overridden the `move` method inherited from `Vehicle`. It extended it to also change its vertical position.