

20.1 Programming Paradigms

Name: _____ Class: _____ Date: _____ **Total: 18 marks**

KEY WORDS attributes 属性 • methods 方法 • objects 对象 • inheritance 继承
• polymorphism 多态

Objective

Build the skills to answer exam questions on **programming paradigms** 编程范式 — low-level, imperative, object-oriented and declarative.

You must be able to:

- explain what a **paradigm** is and the four kinds
- describe the **OOP** features (encapsulation, inheritance, polymorphism, abstraction)
- give examples of **declarative** (functional, logic, SQL) programming

1 Worked examples

Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

■ The four paradigms

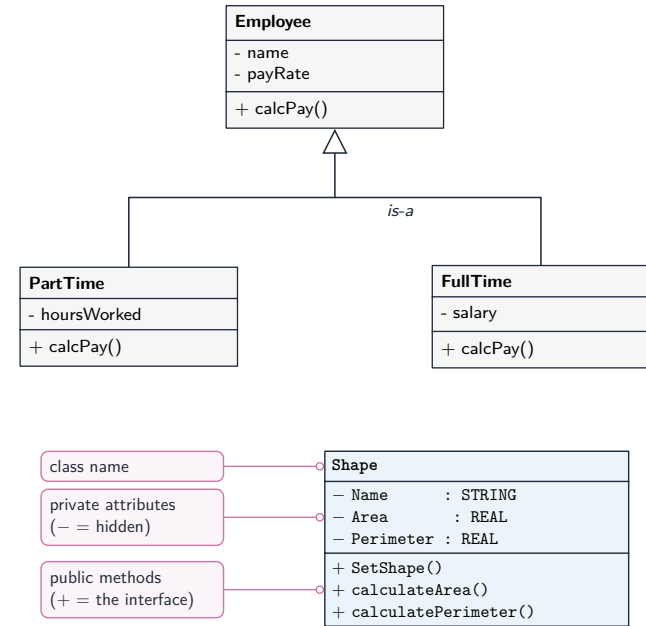
A **paradigm** is a style of structuring programs.

- **Low-level** — machine/assembly code, close to the hardware (fast, but architecture-specific).
- **Imperative (procedural)** — a **sequence of commands** that change state (C, Python).
- **Object-oriented (OOP)** — programs built from **objects** (data + methods).
- **Declarative** — say **what**, not **how** (SQL, Prolog, Haskell).

■ OOP features

An **object** is an **instance** of a **class** (data = **attributes**, operations = **methods**). The pillars:

- **encapsulation** — data is **hidden** behind methods (outside code uses public methods only).
- **inheritance** — a **subclass** specialises a **superclass** (an “is-a” relationship):



A class diagram models an object-oriented design

- **polymorphism** — the **same method call** behaves differently per object type (each **Shape** has its own **Area()**).
- **abstraction** — show a simple interface, hide the implementation.

■ Writing a class (pseudocode)

A class declares **private** attributes and **public** methods, including a **constructor** (**NEW**) that sets up a new object; a subclass uses **INHERITS**:

```
CLASS Employee
  PRIVATE Name : STRING
  PRIVATE PayRate : REAL

  PUBLIC PROCEDURE NEW(GivenName : STRING, GivenRate : REAL)
    Name ← GivenName
    PayRate ← GivenRate
  ENDPROCEDURE

  PUBLIC FUNCTION GetName() RETURNS STRING
    RETURN Name      // a "get" method reads a private attribute
  ENDFUNCTION
ENDCLASS

CLASS Manager INHERITS Employee
  PRIVATE Bonus : REAL
ENDCLASS
```

■ Declarative

Functional uses **pure functions** (no side effects); **logic** states facts/rules (Prolog); **SQL** queries data — `SELECT * FROM Customer WHERE Country = 'UK'` says **what**, not **how**.

2 Practice

Now apply the methods above.

2.1 State what a **programming paradigm** is. [1]

2.2 Name the **four** features (pillars) of **OOP**. [2]

2.3 State what **encapsulation** means. [1]

2.4 Give **one** example of a **declarative** language. [1]

3 Exam-style questions

3.1 Describe what is meant by the OOP feature **inheritance**, with an **example**. [3]

3.2 Identify the OOP feature that **restricts external access to an object's data**, and state **one** benefit of it. [2]

3.3 Explain what is meant by **polymorphism**. [2]

3.4 State **one** characteristic of **low-level** programming and **one** of **declarative** programming. [2]

3.5 A class **Book** has a private attribute **Title** of type **STRING**. Write pseudocode for a **constructor** that sets **Title**, and a **get method** **GetTitle()** that returns it. [4]

Solutions

Each answer shows the steps, then the **result**.

2.1

- a **style/approach to structuring programs**, with its own concepts and language features

2.2

- **encapsulation, inheritance, polymorphism, abstraction**

2.3

- an object's **data is hidden** and accessed **only through its methods** (not directly)

2.4

- any one: **SQL, Prolog, Haskell**, Lisp, F#

3.1

- a **subclass** inherits the **attributes and methods** of a **superclass**
- and can add or **override** its own
- example: a **Manager** inherits from **Employee** ("is-a")

3.2

- **encapsulation**
- benefit: it **protects the data** from invalid changes / lets the internals change without breaking other code

3.3

- different object types respond to the **same method call** in their **own way**
- so the caller need not know the exact type (e.g. **Circle** and **Rectangle** each implement **Area()**)

3.4

- low-level: any one — close to hardware, direct register/memory access, fast/compact, architecture-specific
- declarative: any one — you state **what** not **how**, no explicit step-by-step control flow

3.5

- constructor **NEW** with a parameter; getter returns **Title**

```
PUBLIC PROCEDURE NEW(GivenTitle : STRING)
    Title ← GivenTitle
ENDPROCEDURE

PUBLIC FUNCTION GetTitle() RETURNS STRING
    RETURN Title
ENDFUNCTION
```