

# 3.1 Abstraction and Program Design

---

Name: \_\_\_\_\_ Class: \_\_\_\_\_ Date: \_\_\_\_\_

**Total: 8 marks**

**KEY WORDS** abstraction 抽象 • method decomposition 方法分解 • procedural abstraction 过程抽象  
• overloading 重载

## Objective

---

Build the skills to answer exam questions on **abstraction and program design**.

**You must be able to:**

- describe **abstraction** 抽象 as hiding details to focus on the main idea
- explain how **procedural abstraction** 过程抽象 lets code use a method by what it does
- break a large problem into smaller behaviours using **method decomposition** 方法分解

## 1 Worked examples

---

Study these first. Each one shows the method for a question type used later.

### ■ Abstraction

**Abstraction** hides details so you can focus on the main idea of a problem.

### ■ Procedural abstraction

Lets you use a method by **what it does**, not **how** it does it — you can call `Math.sqrt` without knowing its internal algorithm.

### ■ Method decomposition

Break a large problem into smaller, well-named methods, each handling one behaviour.

## 2 Practice

---

**2.1** Define abstraction. [1]

---

**2.2** State what procedural abstraction lets you do. [1]

---

**2.3** State what method decomposition means. [1]

---

## 3 Exam-style questions

---

**3.1** Abstraction means [1]

- **A** adding more detail
  - **B** hiding detail to focus on the main idea
  - **C** running faster
  - **D** a syntax error
- 

**3.2** Using a method by what it does, not how, is [1]

- **A** overloading
  - **B** procedural abstraction
  - **C** casting
  - **D** recursion
- 

**3.3** A large program is split into small methods, each doing one task.

(a) Name this approach. [1]

(b) State one benefit. [1]

(c) Name the idea of hiding detail. [1]

## Solutions

---

**2.1** hiding details to focus on the main idea of a problem.

**2.2** use a method by what it does, without knowing how it works inside.

**2.3** breaking a large problem into smaller methods, each handling one behaviour.

**3.1** B.

**3.2** B.

**3.3** (a) method decomposition. (b) easier to read, reuse, or test (any one). (c) abstraction.