9.1 Computational Thinking Skills

Name: _____ Class: _____ Date: _____ **Total: 37 marks**

KEY WORDS computational thinking • abstraction • decomposition • sub-problems • program modules

Objective

Learn the two **computational thinking** skills of the syllabus: **abstraction** and **decomposition**. On this first sheet you learn what each one means, why it is used, and how to tell them apart. Sheet 9.1.1 then uses both skills on real systems.

You must be able to:

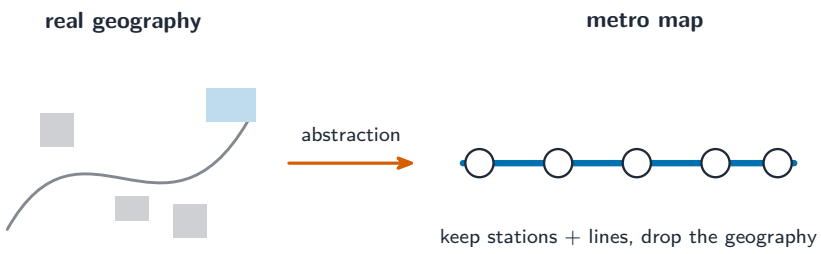
- state what abstraction is, and describe its purpose and its benefits
- state what decomposition is, and describe its benefits
- explain how decomposition leads to **program modules** (procedures and functions)
- recognise which of the two skills a designer is using

1 Worked examples

Study these first. Each one shows a **method** that a question later on this sheet needs.

■ Abstraction: decide what to leave out

Abstraction means **filtering out** the details that are **not needed** for the problem, and keeping only the **essential** details. The result is a simpler **model** of the real thing.



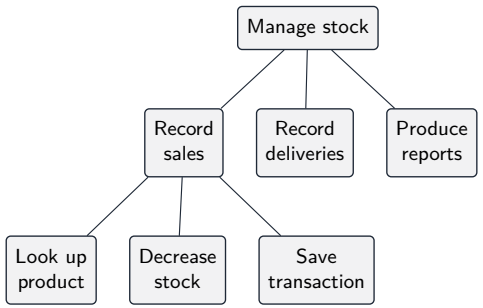
An underground railway map keeps the stations, the lines and their order, because a passenger needs them. It leaves out the real distances, the streets above and the bends in the tunnels.

Why abstraction is used (learn these: they are the marks):

- the problem becomes **simpler**, so it is easier to understand and to solve;
- the program is **smaller** and takes **less time** to design and write;
- the solution contains only what the **user needs**, so it is easier to use.

■ Decomposition: split the problem into parts

Decomposition means **breaking a problem down into smaller sub-problems**. Each sub-problem is broken down again, until every part is small enough to solve.



Each sub-problem is then written as a **program module**: a **procedure** or a **function**.

Why decomposition is used:

- each sub-problem is **easier to understand and to solve** than the whole problem;
- different modules can be given to **different programmers**, who work at the same time;
- each module can be **tested and corrected separately**, so errors are easier to find;
- a module can be **reused**, in this program or in another one;
- the program is easier to **maintain**: one module can be changed without changing the others.

■ Which skill is it?

- **Abstraction** is about **details**: which to keep, and which to leave out.
- **Decomposition** is about **parts**: how to split the task.

the designer ...	skill
decides which details to **keep** and which to **ignore**	abstraction
makes a **simple model** of something real	abstraction
splits a task into smaller tasks	decomposition
gives **parts of the work** to different people	decomposition
plans the **modules** of a program	decomposition

A weather app stores each city's temperature and chance of rain, but not the name of the person who measured them. Which skill? A detail is **left out** because it is not needed → **abstraction**.

■ Writing a definition that earns both marks

A two-mark definition needs **two separate points**.

- Abstraction [2]: *removing the details that are not needed* (1) *so that only the essential details are kept / to give a simpler model* (1).
- Decomposition [2]: *breaking a problem down into smaller sub-problems* (1) *until each one can be solved, or written as a module* (1).

"Making it shorter" or "a summary" is **not** abstraction: nothing is summarised. Details are **chosen** for a purpose.

Past-paper practice: complete sheets 9.1 and 9.1.1 before attempting past-paper set 9.1.1.

2 Practice

Now use the methods above. The first questions need one idea each.

2.1 Complete the sentences. Use words from the list. Each word is used once. [5]

abstraction decomposition

essential · *model* · *modules* · *sub-problems* · *unnecessary*

Abstraction means removing the ..... details of a problem and keeping only the ..... ones. The result is a simpler ..... of the real situation.

Decomposition means breaking a problem down into smaller ..... . These can then be written as program ..... .

2.2 Tick (✓) **one** box in each row to show which skill is being used. [5]

What the designer does	abstraction	decomposition
Splits "run a hotel" into bookings and payments.		
Leaves shoe size out of a hotel booking system.		
Draws a road junction without its trees and shops.		
Gives each of three programmers one part to write.		
Stores a book's title and author, not its weight.		

2.3 State what is meant by **abstraction**. [2]

abstraction

2.4 State what is meant by **decomposition**. [2]

decomposition

2.5 A bus company prints a map of its routes for passengers.

(a) State **one** detail that the map should keep, and give a reason. [2]

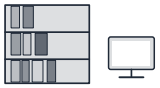
(b) State **one** detail that the map should leave out, and give a reason. [2]

3 Exam-style questions

3.1 A team is designing the software for a school's new library system. They begin by using abstraction.

(a) **State** the purpose of abstraction. [1]

(b) Give **two** benefits to the team of using abstraction. [2]



3.2 Describe three benefits of using decomposition when a large program is developed. [3]

3.3 A student writes: *"Abstraction means writing a shorter version of the program."*

Explain why this is not a correct description of abstraction. [2]

3.4 Explain how decomposition leads to the use of procedures and functions in a program. [3]

3.5 A company is designing a program to run a city's bicycle-hire scheme. The sentences describe what the designers did.

1. They listed everything that is known about a bicycle, and chose only the bicycle's number, its location and whether it is on hire.
2. They divided the program into: hire a bicycle, return a bicycle, take payment, report a fault.
3. They divided "take payment" into: work out the cost, charge the card, send a receipt.
4. They asked two programmers to write "hire a bicycle" and "return a bicycle" at the same time.

(a) **State** the computational thinking skill used in sentence 1. **Justify** your answer. [2]

(b) **State** the computational thinking skill used in sentences 2 and 3. **Justify** your answer. [2]

(c) **Explain** why sentence 4 is only possible because of the work done in sentence 2. [2]

(d) A fault is later found in the way the cost is worked out. **Explain** how the design in sentences 2 and 3 makes this fault easier to find and to correct. [2]

Solutions

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

2.1

- abstraction removes the **unnecessary** details and keeps the **essential** ones
- what is left is a simpler **model**
- decomposition gives smaller **sub-problems**, written as program **modules**

2.2

- splitting "run a hotel" into parts → **decomposition**
- leaving out the shoe size → **abstraction**
- a simple diagram without trees and shops → **abstraction**
- one part for each programmer → **decomposition**
- only title, author and code are kept → **abstraction**

2.3

- removing (filtering out) the details that are not needed for the problem
- so that only the essential details are kept, giving a simpler model

2.4

- breaking a problem down into smaller sub-problems
- until each sub-problem is small enough to be solved, or written as a module

2.5

(a) a detail a passenger needs, for example the names and the order of the stops, or which route goes where

- reason: the passenger needs it to choose a bus and to know where to get off

(b) a detail a passenger does not need, for example the exact shape of each road, the buildings, or the real distances

- reason: it does not help the passenger to plan a journey, and it makes the map harder to read

3.1

(a) to remove the details that are not needed, so that the model contains only what is essential to the problem

(b) any **two** of:

- the problem is simpler, so it is easier to understand and to solve
- the program is smaller, so it takes less time to design and to write
- the system contains only what its users need, so it is easier to use

3.2 any **three** of:

- each sub-problem is smaller, so it is easier to understand and to solve
- different modules can be given to different programmers, who work at the same time
- each module can be tested and corrected on its own, so errors are easier to find
- modules can be reused in this program or in other programs
- the program is easier to maintain, because one module can be changed without changing the others

3.3

- abstraction does not shorten or summarise anything: it **chooses** which details of the real problem are needed
- the details that are not needed for the purpose are left out, which gives a simpler model of the problem; the length of the program is not the aim

3.4

- decomposition breaks the problem down into smaller and smaller sub-problems
- it stops when each sub-problem is one clear task that is simple enough to program
- each of these sub-problems is then written as a program module, a procedure or a function, and the main program calls the modules

3.5

(a) **abstraction**

- from all the details of a bicycle, only the ones the system needs were kept, and the others were left out

(b) **decomposition**

- the problem was broken down into smaller sub-problems, and one sub-problem was broken down again

(c) the program was divided into separate modules that do not depend on each other's code

- so each programmer can be given one whole module and they can work at the same time

(d) "work out the cost" is a small module of its own, so the programmer knows where to look

- that module can be tested and corrected on its own, without changing the rest of the program