OCR A Level Computer Science

Your notes

6.1 Thinking Abstractly

Contents

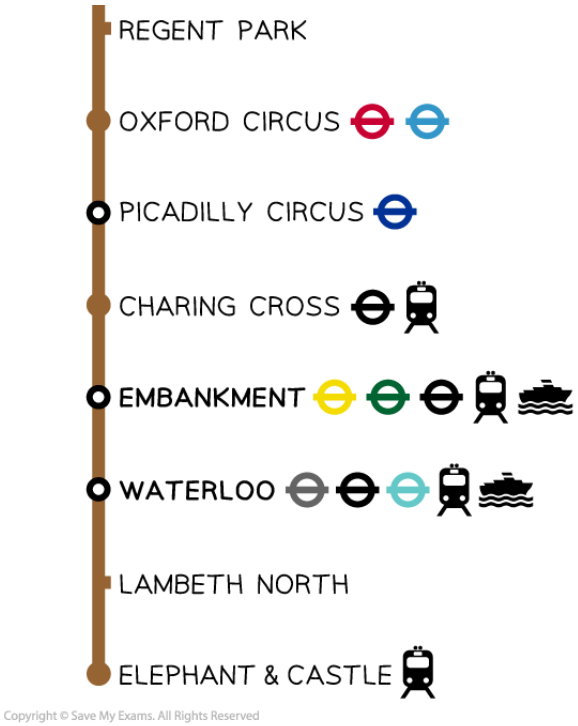
Your notes

The Nature & Need of Abstraction

What is the nature and need for abstraction?

- Abstraction is the process of removing unnecessary details of a problem to focus on the important features to implement in a solution
- Examples of abstraction include modelling a real life object, environment, action, sequence of actions or concept. Implementations of these include:
  - a simulator such as a car or flight simulator,
  - a representation of a building or house in a program or game or,
  - a map of a bus or train route in a city
- When creating a program, developers must identify important features that will contribute to solving the problem or have a role to play in the solution
- A specific example of abstraction would be the London underground train route map; travellers do not need to know the geographical layout of the routes, only that getting on at stop A will eventually transport you to stop B

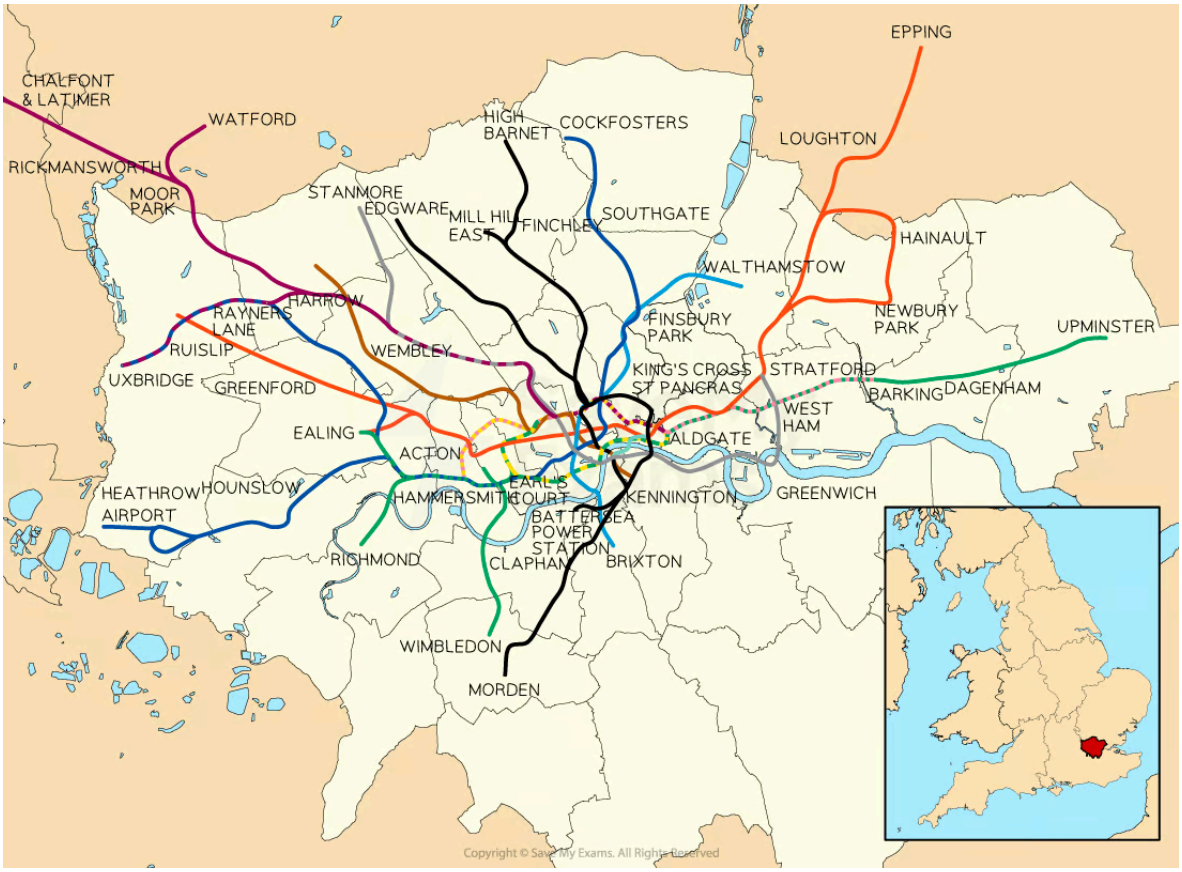
Your notes



London Underground train route map
Source: Wikipedia

Your notes

Your notes



The geographical London underground train map
Source: Wikimedia

- Another example of abstraction would be implementing the trajectory of a projectile, such as a ball or dart, or the physics of a snooker ball on a snooker table
  - Is gravity or air resistance taken into account, if applicable? Is friction?
- The closer the implementation is to reality, the less abstract the solution becomes
  - Pong is an example of a highly abstracted game of tennis or badminton
  - The momentum of the ball is constant and there are no extraneous factors that affect the game such as friction or gravity

Your notes

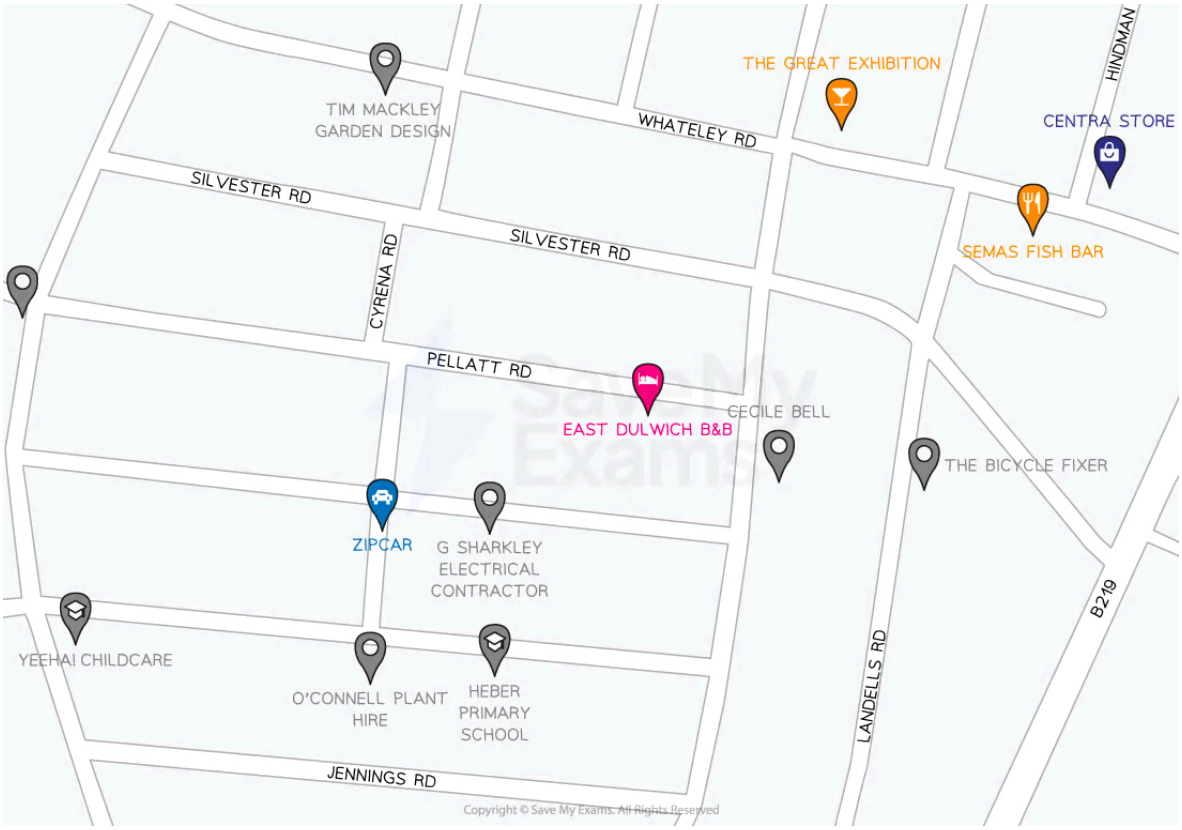
The difference between abstraction and reality

What is the difference between abstraction and reality?

- The real world is very complex and has many, many variables that factor into problems
- An aerial view of a city with all of its roads, junctions and streets forms the basis of reality when travelling from point A to point B. Roads weave and turn and may have different governing laws depending on location, such as maximum speed limits or no left turnings

A street view of a city showing roads, junctions and streets
Source: Google Maps

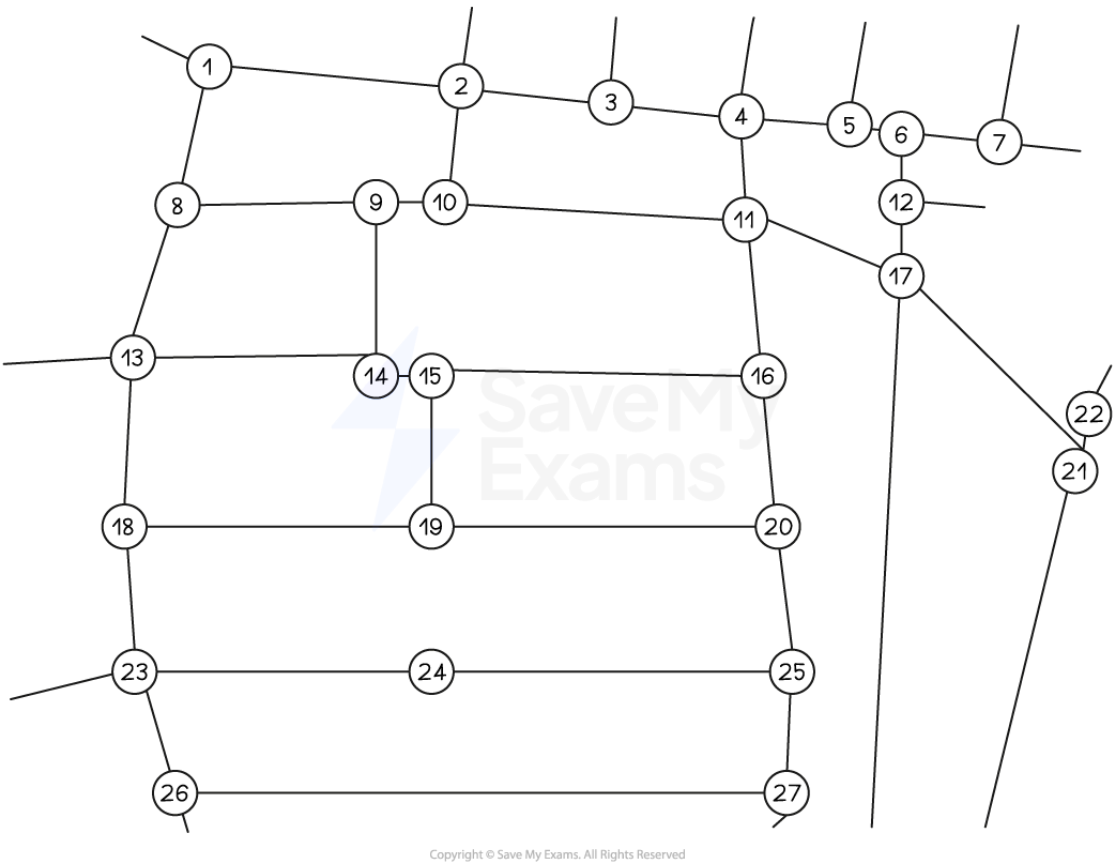
- When travelling by vehicle, the greenspace and buildings of a city are usually unnecessary detail. When creating a useful map of a city, these elements can be removed, leaving only roads and junctions



An abstracted street view, with all buildings and greenspace removed
Source: Google Maps

- Quick and efficient travel has always been a considerable problem, usually done by humans using maps. Computers however can process routes much quicker but require a suitable map to calculate these routes
- To provide a usable map to a computer, the map must be abstracted. After removing unnecessary detail, such as greenspace and buildings, a simplified representation of a map can be created
- All junctions can be represented by graph nodes and each road is represented as an edge or arc between each junction. Elements such as no left turnings can be represented by directed edges or arcs. The distance between each junction can be represented by an appropriate weight on each arc or edge
- The final product is a graph to which graph theory and algorithms such as Dijkstra's shortest path or A* search can be applied to

Your notes



A further abstracted street view, showing a collection of labelled and connected junction nodes

- Apps such as Google Maps or Uber use abstracted maps such as this to calculate the shortest routes from a starting point to a destination. Other factors such as traffic, roadworks and weather conditions can also be factored in, enabling more accurate calculations
- The final abstracted map above has removed all elements that do not impact or affect the finding of the shortest distance between two points

Your notes


Your notes

Devising an Abstract Model

How can you Devise an Abstract Model for a variety of situations?

- In order to create abstract models, the following questions must be answered carefully:
  - What is the specific problem to be solved?
  - Can the problem be broken down into milestones?
  - What elements will impact the solution for each milestone?
  - Alternatively, if an element were to be removed, would it impact the solution in any way?

Programming language abstraction

- Programming languages are themselves an abstraction of other languages
- Machine code, also known as binary, is a first generation language, using 0's and 1's to create programs.
  - This process is time consuming, tedious and error prone even for simple programs
- Assembly language, a second generation language, was developed to improve developers ability to create programs. Assembly language uses mnemonics to represent groups of binary digits, for example 1011 might represent the instruction ADD.
  - The advantage of assembly language was that it was easier, quicker and less error prone to create languages. The underlying implementation of 0's and 1's was abstracted away from the developer. They did not need to see how the program was actually run on the hardware
  - The disadvantage of assembly language is that each processor has its own version, a chip instruction set, that can only run on that particular family of processors. Programs had to be rewritten to run on other instruction set machines
- High level languages such as BASIC and FORTRAN were developed in the 1960's, which became the third generation set of languages. Other languages such as Python, Java, and C are considered third generation
  - The advantages of third generation languages were that they abstracted long, complicated sequences of instructions into shorter instructions. For example, multiplying a number in assembly language takes many lines of instructions, where as in Python, this takes only one line; A = B*C

- High level languages allow developers to ignore how data is stored in memory and the specifics of how instructions are carried out in the processor. Instead this allows them to focus on creating more complex programs much quicker and easier than using assembly language

- Abstraction in the context of languages allows developers to focus on solving the problem rather than worrying about the technical details. A comparison would be that a car driver doesn't necessarily need to know how the engine or mechanics of the car work in order to drive.
    - Similarly a programmer doesn't need to know all of the underlying technical complexity to create complex programs


Your notes

Data abstractions

- As with programming language abstraction, data can also be abstracted
- Programmers generally do not need to worry how primitive data types, such as integers, strings or booleans are stored and represented in a computer. These implementations are hidden to make creating programs easier
- Higher level languages allow programmers to create abstracted data types to represent logical structures such as modelling the queue of a fast food restaurant
- Queues are first-in, first-out structures where new people are added to the back and removed from the front once served and usually have a maximum capacity
- Data structures such as queues are actually modified arrays, which in turn are collections of variables, which are collections of bytes, which are collections of bits, which are collections of flip-flops
- A programmer does not need to worry about the underlying implementation of a queue and how its stored in memory, only that they can add to and remove data from a queue
- In this way, powerful analogous structures can be created that closely resemble, and operate similarly to, real life
    - This makes problems easier to solve and easier to understand and model
- Overall, abstraction is about separating what a program does from the implementation or how it does it