TABLE OF CONTENTS

Section 4

APPLICATION OF COMPUTER SCIENCE

4.1 Binary Representation

Binary Digit (Bit)

Basic unit of information, represented as 1 or 0.

Byte: 8 bits Kilobyte: 1000 bytes Megabyte: 1000 kilo-bytes

Minimal unit of storage that can be set to 1, or 0.

Decimal or 'Denary'

Positional system that uses 10 digits to represent a number. Also known as 'Base 10'.

Binary has only two states: 0 or 1, which is why it is known as 'Base 2' - however Decimal has ten possible states: 0 to 9, which is why it is known as 'Base 10'.

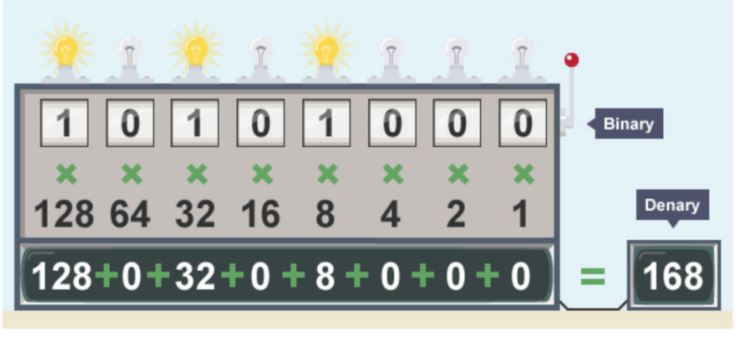
Thousands 1000s (10^3)	Hundreds 100s (10^2)	Tens 10s (10^1)	Ones 1s (10^0)
6	4	3	2

Or think of it as:

$(6 \times 1000) + (4 \times 100) + (3 \times 10) + (2 \times 1) = 6432$

Each base has its own placeholders. For example, in Decimal 6432 can be thought of as 6 1000s, 4 100s, 3 10s and 2 1s.

In Binary the same is true:



- Nibble - 4 bits (half a byte)
- Byte - 8 bits
- Kilobyte (KB) - 1000 bytes
- Megabyte (MB) - 1000 kilobytes
- Gigabyte (GB) - 1000 megabytes
- Terabyte (TB) - 1000 gigabytes

The values fall into the binary pattern (1, 2, … 128) like Connect Four… putting it into the highest value then putting the remainder into the next… and so on.

Here, 10101000 becomes 168 in Decimal.

Binary Representation (2.1.10)

Revised

In computers, an 'on' state, when electricity is flowing, represents true. The 'off' state, no electricity flowing, represents false. This can be represented with 0 and 1.

8 BITS

0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0
128	64	32	16	8	4	2	1	0

= 256 states / values

Integers 256 values can be represented by 8-bit numbers. When representing negative numbers, the most significant bit (MSB, leftmost bit) is either 0 (positive) or 1 (negative) so only 7 bits are left for the number.

Characters American Standard Code for Information Interchange (ASCII) character encoding scheme represents text. Each character (lower or uppercase latin letters, numbers, punctuation and control characters) is assigned a value from 0 to 127. Unicode can also have symbols and other languages.

Strings (e.g. UTF-8 Unicode is used - on average 40 bits per word is required).

Colours Each pixel has a colour value, often represented by a Hexadecimal RGB 6-digit number (3 values), every two digits show how much red, green or blue there is in the colour. Can also be represented in binary; more bits means more colours. Resolution is the width * height in pixels.

A colour will be split into three components (Accept RGB as an example); Each component will be assigned a certain number of bytes.

BASE PLACEHOLDERS

Denary *("Base 10")*	.. 1000 (10^3)	100 (10^2)	10 (10^1)	1 (10^0)
Binary *("Base 2")*	.. 8 (2^3)	4 (2^2)	2 (2^1)	1 (2^0)
Hex *("Base 16")*	.. 4096 (16^3)	256 (16^2)	16 (16^1)	1 (16^0)

HEXADECIMAL

This was created to simplify the way Binary is represented.
It is 'Base 16' due to having digits:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Each Hex digit represents 4 Binary digits:

DENARY	BINARY	HEXADECIMAL
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

For example; D4 is 11010100

Hexadecimal is only really used for Human comprehension.
It is always converted back into binary for mach interpretation.

Hexadecimal can be used to represe colours using the format #RRGGBB with 256 possible shades for each red, green, or blue - and over 16 million colours in total (256 ^3).

16^3	16^2	16^1	16^0
4096	256 X 1	16 X B	1 X 5

Hexadecimal representation of 1B5 - decimal value = (1*256)+(11*16)+(5*1) = 437

Algorithm: keep dividing by 16 and keep remainder, then put them together (write letters for d over 9). Put the first value into the rightmost box.

Example:

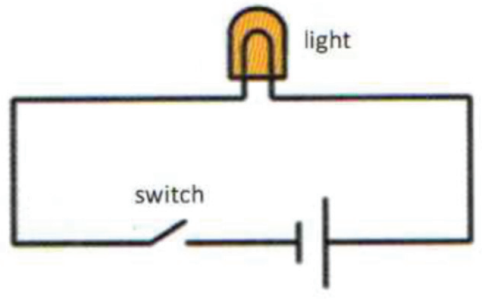
Using placeholders, we can convert any denary number into Binary or Hex

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		
	128	64	32	16	8	4	2	1		
Binary	0	1	1	0	1	1	0	1	=	109

	16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0		
	-	-	-	-	4096	256	16	1		
Hex	-	-	-	-	-	-	6	D	=	109

BOOLEAN LOGIC (2.1.11)

Revised



Transistors are electrically controlled switches, made up of two electrodes and one control wire:

When you apply electricity to the control wire, it lets current flow through one electrode through the transistor, to the other electrode.

The control wire can be considered to be the input, and the wire from the bottom electrode as the output.When the input is true, the output is true. When the input is false, the output is false: this is the most basic Boolean Logic.

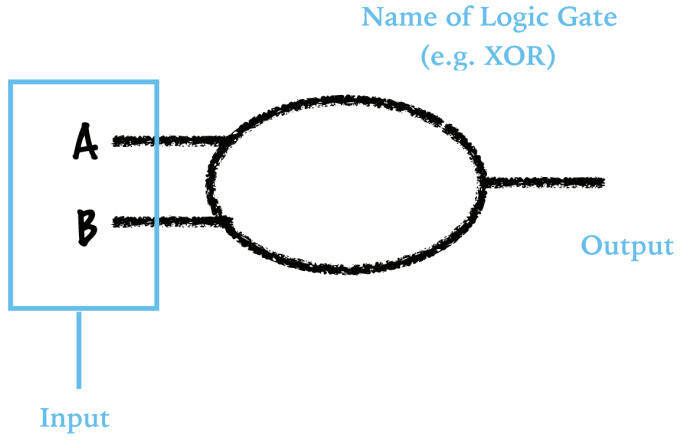
AND	$\wedge$	Both A and B must be true for the output to be true
OR	$\vee$	Either A or B or Both may be true for the output to be true
XOR (Exclusive Or)	$\veebar$	Either A or B may be true for the output to be true, but NOT BOTH
NAND (Negation And)	$\neg\wedge$	Negation of output for AND
NOR (Negation Or)	$\neg\vee$	Negation of output for OR

DRAWING LOGIC GATES

When asked to draw a Logic Gate diagram, shape specifics do not matter.

- Inputs are always leftmost.
- 'OR' is always the last (rightmost) gate, when drawing multiple Logic Gates at once.

Boolean Logic is given as 1 (true) or 0 (false).



Concurrent Processing (4.1.14-16)

Revised

This is when one or more processes are using the same resources at the same time (e.g. Building a house, Production lines, Division of labour). Concurrent Processing may be evaluated according to if it has saved time/resources.

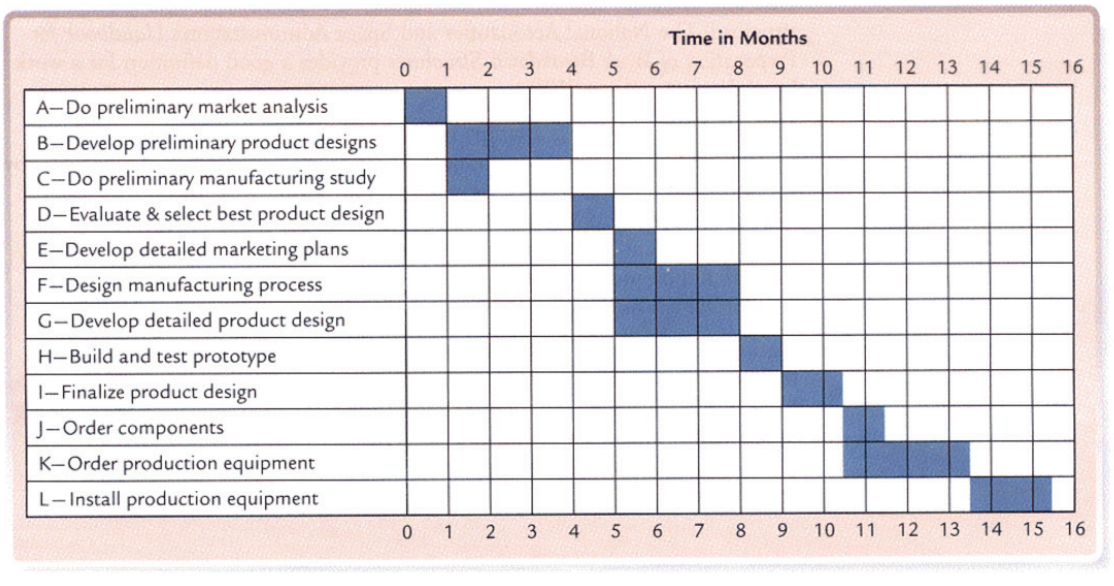
In a Concurrent Processing system, one job uses several processors to execute sets of instructions in parallel. The advantages of Concurrent Processing are:

- Increases computation speed.
- Increases complexity of programming language and hardware (machine communication).
- Reduces complexity of working with array operations within loops, conducting parallel searches in databases, sorting files etc.

On the GANTT Chart, any tasks carried out at the same time have been carried out Concurrently. In contrast, any tasks which are not carried out Concurrently have been carried out in Sequence, and are thus examples of **Sequential Programming**.

GANTT CHARTS

Here, the horizontal axis represents the span (time) of the project, and is broken into increments such as days, weeks or months. The vertical axis represents the tasks that make up the project.



Reason For:

- Efficient organisation; helps establish time-frame.
- Highly visual, easy to stick to.

Reason Against:

- Potentially over-complex.
- Needs to be constantly updated if project requirements change.
- Doesn't show whole picture; details of task.

Batch Processing:

Large amount of input happens over time, then whole set of input is processed in one go.

e.g. Monthly Mobile phone Bills (all inputs/access is added up and charged at the end of the month)

Online Processing:

Input processed almost immediately.

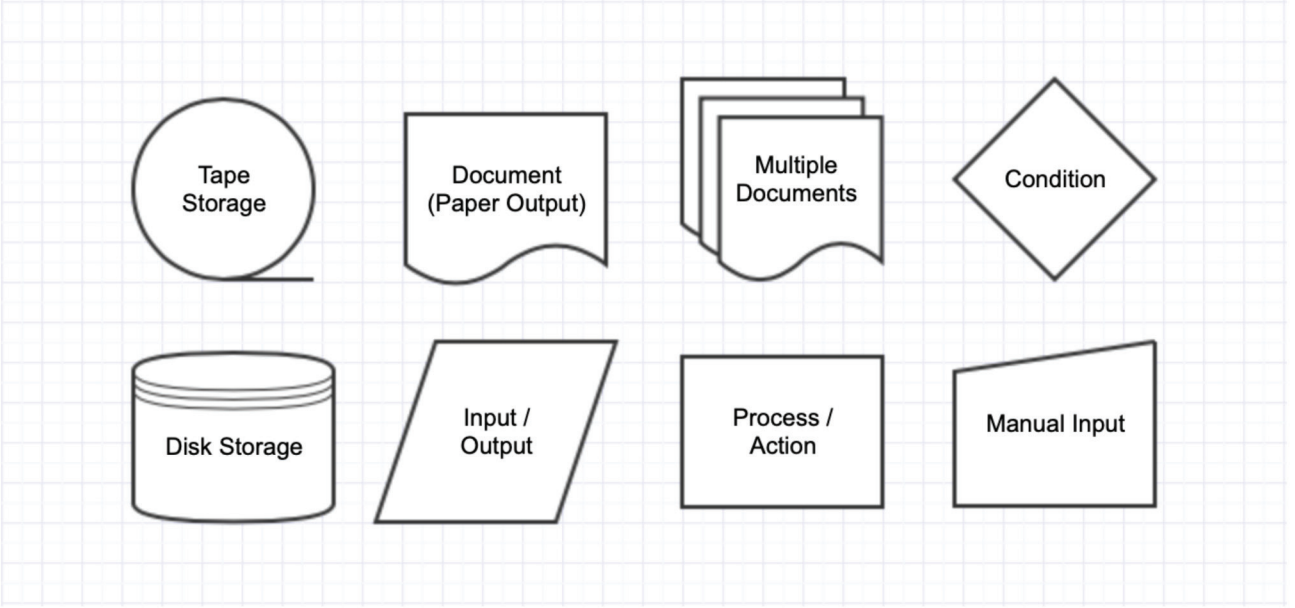
e.g. Flight seat booking system; as soon as you book the seat is yours.

Real-Time Processing:

Input processed immediately and continuously; generally without any user input, of which comes from system sensors.

e.g. Autopilot on Aircraft; large volume data harvested from multiple sensors continuously, system reacts in real time.

FLOWCHART SYMBOLS



TYPES OF FLOWCHARTS

Algorithm Flowchart

Each process represented by a flowchart symbol, stated in detail.

System Flowchart

Most basic, general overview of the system itself, with no detail.

Abstraction (4.1.17)

Revised

Abstraction is the process of taking away or removing characteristics from something in order to reduce it to a set of essential characteristics. This enables one to concentrate on the essential aspects of the problem, whilst ignoring any distracting details. It is a simple way of dealing with complexity.

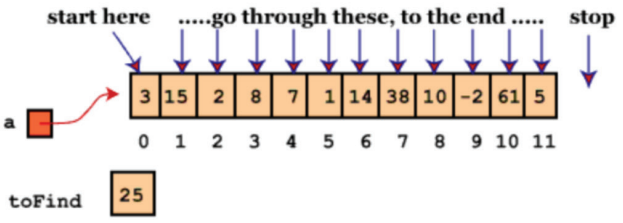
Standard Algorithms on Linear Arrays (4.2.1)

Revised

SEQUENTIAL SEARCH (A.K.A "LINEAR TO FIND")

Also known as a 'Linear Search'; this is an algorithm used to find an item in a list.

- It starts with the first element and compares each element to the one it's looking for, until it is found.

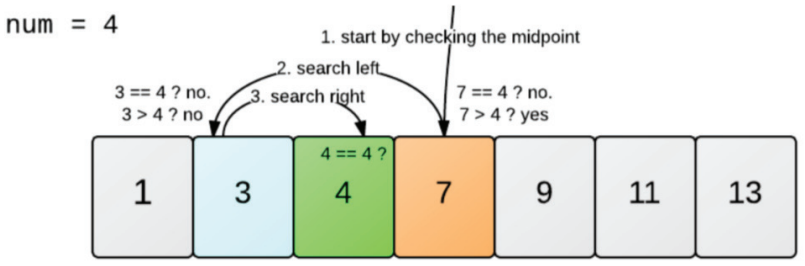



```
NAMES = "Bob","Betty","Kim","Lucy","Dave"

output "These names start with D"

loop while NAMES.hasNext()
    NAME = NAMES.getNext()
    if firstLetter(NAME) = "D" then
      output NAME
    end if
end loop
```

BINARY SEARCH (A.K.A "SORTED AND SPLIT")

Also known as a 'Half-Interval Search'; this is a search algorithm that finds the position of a target value within a sorted array. Crucially, it only applies to sorted arrays, where there are no duplicate values; or where duplicates do not matter.

- It works by comparing the target value to the middle element of the array;
- If they are unequal, the lower of upper half of the array is eliminated depending on the result and he search is repeated in the remaining sub-array until it is successful.



BUBBLE SORT (A.K.A "SWAP TO SORT")

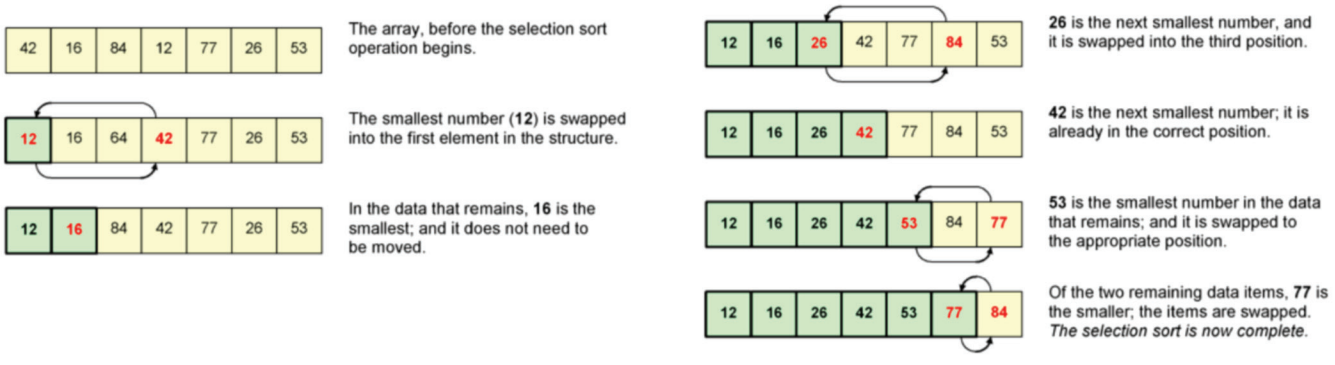
Known as 'Bubble Sort' because of way smaller elements bubble to the top of the list. Although the algorithm is simple in nature, it is often too slow and impractical for most problems.

- This a sorting algorithm that repeatedly steps through the list to be sorted, compares each pair of adjacent items and swaps them if they are in the wrong order.
- It will repeatedly pass through the list until there are no swaps needed, which indicates that the list is sorted.

SELECTION SORT (A.K.A "SUBLIST TO SORT")

- Algorithm divides the input list into two parts: the sublist of items already sorted, which is built up from left to right; initially the sorted sublist is empty and the unsorted sublist is the entire input list.

- The algorithm proceeds by finding the smallest (or largest, depending on sorting order) element in the unsorted sublist and swapping it with the leftmost unsorted element; then moving the sublist boundaries one element to the right.



Higher Level Language and Machine Code (4.3.5)

Revised

Compiler

When a high-level language is translated into lower-level language (done in a batch).

Translation into machine code for final execution.

Interpreter

When a high-level language is translated into an intermediate code which will be immediately executed by the CPU (done line by line).

Warns syntax error, shows outputs for tested processes.

For example, the Java Virtual Machine allows the same Java code to be installed on various different types of hardware; this means the programmer does not need to write many different versions of the program for each different device type.

Use of Programming Languages (4.3.6)

Revised

Variable — a Storage Location for Data in a program. They are a way of naming a memory location for later usage (to put a value into). Each variable has a name and a data type that is determined at its creation and cannot be changed.

Constant — an Identifier with an associated value which cannot be altered by the program during execution (the value is constant). This is the opposite of a *variable*.

Operator — a Set of characters which represents an Action. Types include:

- **Boolean**:
  `(AND/OR && ||)`
- **Arithmetic Operators:**
  `(+/- div mod)`
- **Assignment Operator:**
  `(=)`
- **Relational Operators**:
  `(> == != .equals() )`

Discuss the Need for Sub-Programmes and Collections (4.3.12)

Revised

Modular Programming is a beneficial approach to programming problems; as it allows the program to be easily organised (for the coder and team members); makes future maintenance easier and is efficient in its use of reusable code.

By breaking it down into modules, each compartment can be easily designed and tested. This also allows for 'distributed development' where multiple programmers can work in parallel.

Standard Operations of Collections (4.2.2)

Revised

Collections are unordered lists of unknown length or size. In Java, these collections are known as 'LinkedLists'; which are useful when you don't know how many items you'll be needing/using (compared to an array of fixed length). It is also an efficient use of RAM Memory and can be of any data type. It has operators (collection methods) such as:

- `addItem(data)` adds data item to the collection
- `resetNext()` starts at the beginning
- `hasNext()` tells whether there is another item in the list
- `getNext()` retrieves a data item from the collection
- `isEmpty()` check whether collection is empty

These are predefined Sub-programs, known as 'Collection Methods'

OBJECTS AS A PROGRAMMING CONCEPT

Revised

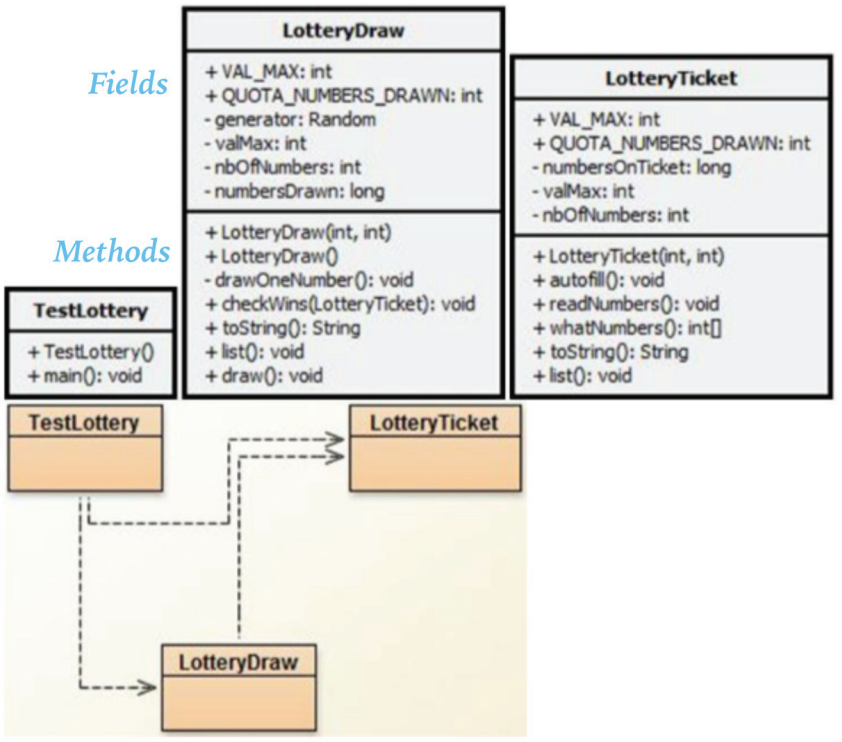
CLASSES AND OBJECTS (D1.1/2)

Revised

Object an Object is a representation of a real world entity (e.g. Book, Car, Student)

Class a template for creating these objects. A Class identifies what data/attributes need to be stored for these objects (this data are the Fields) and the procedures/functions/behaviours needed to access this data (known as the Methods).

UML DIAGRAMS



Here, the grey boxes are the UML Diagrams for the Classes.

The top half show the Fields and the bottom half show the Methods.

The notation (+) shows the variables are Public and (-) Private.

The Yellow Boxes show the Relationship between the Classes.

The constructor method of the Class has the same name as the Class (here it is TestLottery).

When a program is executed, the term Object has a different meaning. Here, an Object is created by a constructor method of the class; and this created Object is known as the **instance of the class**.

Here, the Class is a template for a lottery draw but would create a new lottery Object each draw.

Each draw would have its own set of winning numbers. This is how Objects are created, each has the same structure but its attributes (Fields) would be different (e.g different numbers).

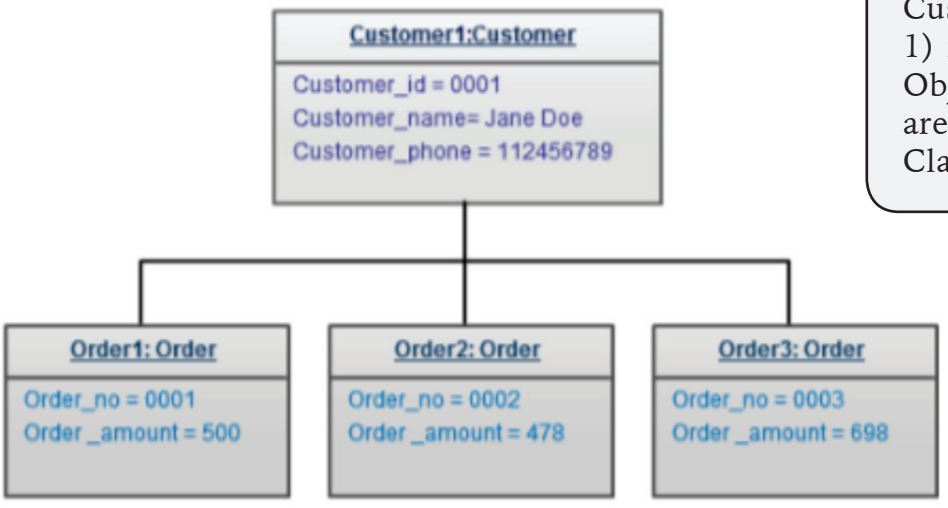
UML Diagrams (D1.3/4)

Revised

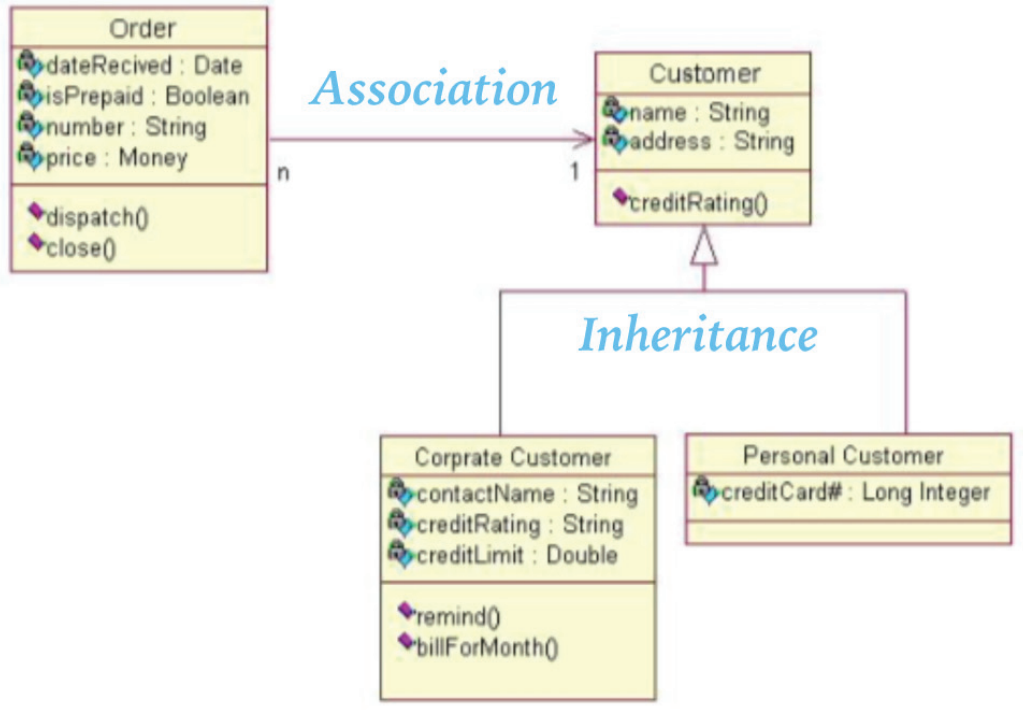
UML Diagrams are used to visualise computer thinking and program design. At IB, UML Diagrams are distinguished into two types:

OBJECT DIAGRAM

This is a diagram for Objects inside the Class. Here, the Customer Object (Customer 1) is associated with 3 other Objects (Order 1-3), which are instances of the Customer Class.



CLASS DIAGRAM



A Class diagram shows all the separate Classes inside a Program.

Each Class lists its Fields; for example Order has the Fields dateRecieved of type Date, isPrepaid of type Boolean, and so on.

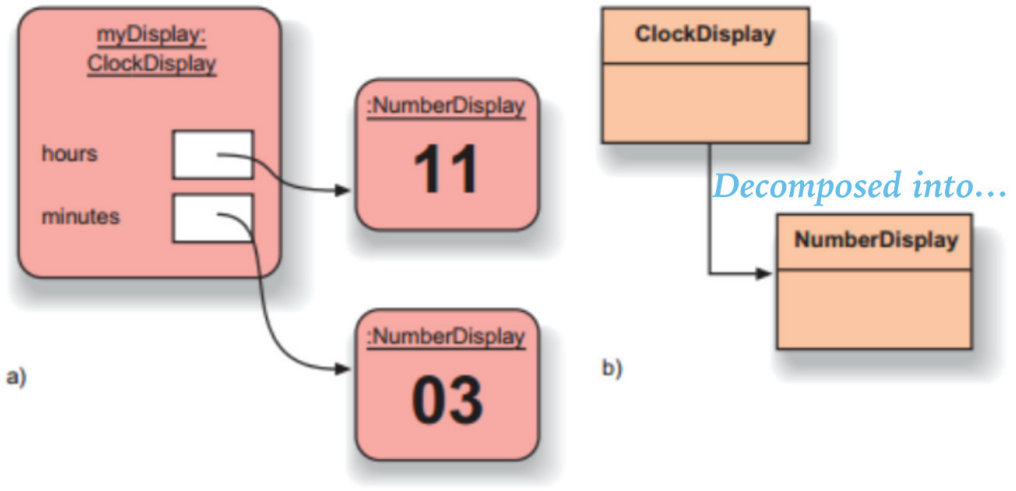
The Order Class has Methods dispatch() and close(), which show you have access to them. The name of the Method typically indicates what it does.

The Customer Class has Fields of Name and Address both of type String and a method called creditRating(). The Classes Corporate Customer and Personal Customer are types of the Customer Class (shown by the Relationship Arrows) and so **inherit the Fields and Methods** of the Customer Class.

These Classes Corporate Customer and Personal Customer may have their own specific Fields of which they do not share, for example creditLimit of type Double.

Decomposition into Related Objects (D1.5)

Revised



An example to demonstrate Decomposition into Related Objects can be seen with the 'Digital Clock'.

The clock is made of two 2-digit number displays. So Class numberDisplay was created to hold these two 2-digit numbers alongside Class numberDisplay.

For example, a Calendar is made up of days which can be grouped into weeks and grouped into months; this shows that you have decomposed the Calendar in days, weeks and months.

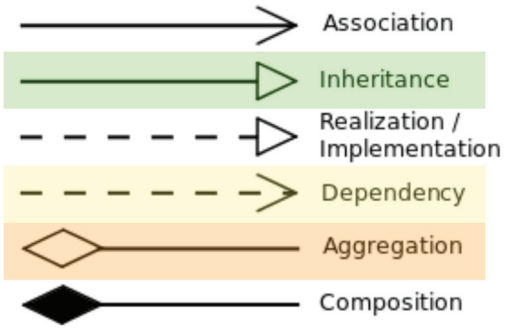
Describe the Relationship between Objects (D1.6)

Revised

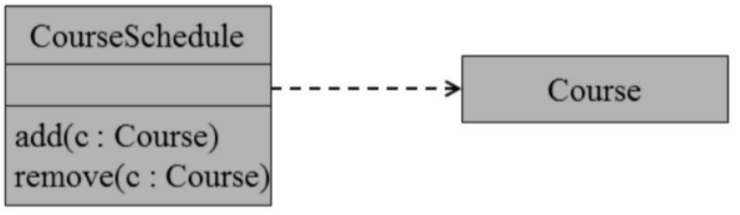
Objects can have three types of relationships:

- Dependency (x uses y)
- Aggregation (x has a y)
- Inheritance (x is a y)

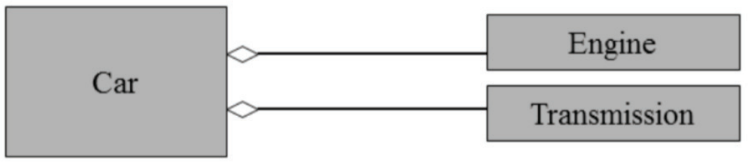
UML NOTATION



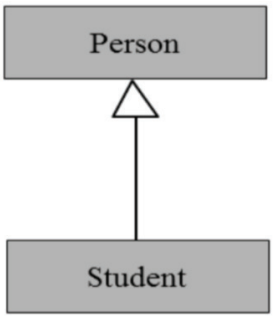
Dependency:



Aggregation:



Inheritance:



Here Class courseSchedule depends on Class course because the add() and remove() Methods both use the Class course.

Here Class Car has (includes) Fields of type Class Engine and Class Transmission (a Class can be a type).

A Class can inherit Fields and Methods from a super-Class.

Here the Class Student inherits the Fields and Methods from Class Person, but Class Student may have additional Fields and Methods itself.

Outline the Need to Reduce Dependencies between Objects (D1.7)

Revised ☐

Whenever Class A uses Class B, then A depends on B. Therefore, Class A is the 'Dependent' and Class B is the 'Dependency'.

Two Classes which use one another are 'Coupled'. The degrees of the relationship between Coupled Classes is continuous, not discrete.

Dependencies are bad because they decrease reuse. Reuse is found in code where parts are reiterated or have a replicated structure; this is helpful in code readability; code quality, etc.

Changes in one class will affect other (dependent) classes… which may increase errors in code.

Describe How Data can be Pass to/from Actions as Parameters (D1.10)

Revised ☐

Actions are Methods; and some need data it does not have direct access to. Because of this, specific data can be sent into the Method as an argument and accepted as a parameter.

Advantages of Libraries of Objects (D2.7)

Revised ☐

Libraries are made available for parts of code that are likely to be reused. These are imported into Java programs, and provide increased functionality. They work like pre-made methods.

import java.lang.Math.*;

Describe the Disadvantages of Object-Oriented Programming (D2.8)

Revised ☐

- **Size**

  OOP are much larger than other programs; and in the infancy of computer technology when storage capacity was poor, this made OOP unfavourable. Today, this is not an issue.

- **Effort**

  OOP require a great level of planning before any code is written.

- **Speed**

  OOP typically run slower, due to their size.

Discuss the Use of Programming Teams (D2.9/10)

Revised ☐

The modular nature of OOP means that objects can bee assigned to a different coder to program, as long as any shared Methods are known. This allows further efficiencies in being able to finish coding the program faster and allowing users to specialise to their strengths. Additionally, each module can be tested and debugged separately.

However, this could cause communication/coordination issues; diffusion of responsibility; conflict.

Program Development

Definition of Terms (D3.1-3)

Revised ☐

Class	Defines the Fields and Methods for a group of similar objects. An object is an instance of a Class.
Identifier	A broad term for the name of a Variable or Method.
Primitive (Data Type)	byte, int, long, double, char, boolean, etc
Instance Variable	The Fields in a Class. So when an object is created of that class type, it has its own version of the Fields.
Parameter Variable	Information sent into a method from the calling program. `pubic int add(int x, int y)`
Local Variable	A Variable which exists only as long as the code to which it belongs exists. So if declared in a Method which is no longer being run; the variable no longer exists/ it is overwritten.

Method A program routine or behaviour contained within an Object designed to perform a particular task on the fields within an object.

Accessor A Method that allows access to the fields within an object.

```
getName()
```

Mutator A Method that allows the values of the files in an object to be changed.

```
setName()
```

Constuctor The Method that correctly initialises and sets up an object when it is created. This method has the same name as the class it belongs to, and has no return data type.

```
public person()
```

Signature The Method name and its parameter types.

```
doSomething(String y)
```

Return Value The value returned by a method to the calling program.

Private Variables, Methods, Fields and Constructors that can only be accessed within the declared class itself. Classes cannot be private. An object is able to encapsulate itself and hide its data from outside access.

Protected Variables, Methods, Fields and Constructors declared protected in the super-Class can only be accessed by the sub-Classes.

Public Variables, Methods, Fields and Constructors declared public can be accessed from any other Class.

Extends This is used by a sub-Class to say that it inherits the fields and methods from a super-Class.

Static In Java, a static is a member of a class which isn't associated with the instance of a class. It belongs to the Class itself. As such, you can access the static member without first creating a class instance.

Modern Programming and Ethical Considerations (D3.9/10)

Revised

When organisations interact on an international basis, there may be issues of language differences.

- **Common character sets**

  Character sets used among many platforms and languages, like UNICODE. Unlike ASCII, UNICODE has a much larger number of characters so has letters and symbols from different alphabets/ languages.

- **Platform independent languages:**

  High level programming languages like Java enable code to run on many different platforms.

ETHICAL/MORAL OBLIGATIONS OF PROGRAMMERS

- **Adequate testing:**

  To prevent possibilities of commercial or other damage.

- **Plagiarism**

  Acknowledge the work of other programmers to avoid being accused of plagiarism.

- **Open source movement**

  Increasing support for open-source software, where you are given access to the code for free, allowing you to make changes/customisations to meet exact needs.

4.3 Java Object Oriented Programming

FOUNDATIONS OF OBJECT ORIENTED PROGRAMMING:

Object — A software bundle of related state (data) and behaviour (procedures). These procedures and functions (attributes) are known as Methods. It is also the 'Instance of a Class'.

In the real-world everything has state and behaviour, for example Dog has state (name, colour, breed) and behaviour (wagging, eating, running). What states can this object have, and what behaviours can it perform? In Java (object oriented programming) everything is an Object.

Advantages of objects:

- Objects can be easily passed around the program, and so are modular in nature.
- Allows for re-use for objects already created.
- Debugging ease; if a bicycle is broken you replace the bolt, not the whole bike. Similarly, you can replace and diagnose specific objects.

Members — Objects have their own variables, constants and methods associated to it. These are broadly known as 'Members' or 'Features' of an Object.

Class — A category of objects (template) for which objects are created. Models the state (data) and behaviour (methods) of a real-world object.

In the real world, there are thousands of Bicycles (Objects) for which all were built from the same set of blueprints and thus contain the same components. In OOP terms; your bicycle is the instance of the class of objects known as 'Bicycles'.

A Class is the blueprint for which individual bicycles (Objects) are created.

It can also hold Identifiers which are shared by, and accessible to all the Objects within the Class (static members) and can handle methods which deal with the entire Class rather than a specific object.

Inheritance The capability of defining a new class of objects that inherit state and behaviours from a parent/super-class.

New data and methods can be added to the class, whilst the data and methods of the parent class are available for objects in the new class.

Different kinds of objects often have certain amounts in common with one another. Mountain bikes, Road bikes and Tandem bikes all share characteristics of bicycles (speed, cadence, etc) — yet each has additional features that makes each different. Bicycle is the parent/super-Class of MountainBike and so on.

public class B extends A

This means that class B will have all the members (variables, constants and methods) in class A. 'extends' shows that class B inherits from class A; alternatively it can be said that class B is a sub Class of the parent Class, class A.

Advantages of objects:

- No need to re-write methods when sub-Classes can extend a parent Class.
- Any changes in the parent Class are inherited by the sub-Class automatically.

Encapsulation Mechanisms that allow each object to have its own data and methods (states and behaviours). Encapsulation is affected by having all method calls handled by objets that recognise the method.

Data and Methods are limited to the object in which they are defined. Private variables can be only accessed outside of its class by specific Accessor and Mutator methods.

```
void add(Data dat, Key k)
```

Called using: `t.add(dat, k)` *Receiver…* *…Message*

Where the method call is known as the 'message' and t is the 'receiver' of the message. The benefit of this approach is that there can be many methods named 'add' with different objects implementing them in different ways. This allows programmers to reuse names of methods.

Advantages of objects:

- Can restrict the way data/methods are altered or called — ensures variables are not accidentally changed by another part of the program, whilst still maintaining their functionality.
- Hide the way data is stored.
- Since relevant data/methods are linked makes it clearer to diagnose any run problems.

ACCESS MODIFIERS:

In Java, the keyword public or private indicates the level of accessibility. When private is used, the member will not be inherited by the sub-Classes.

Polymorphism The capability of having Methods with the same names exhibit different behaviours depending on the 'receiver'. Can send the same message to two different objects and they can respond (return) in different ways.

Commonly achieved through 'Overloading' which allows two objects to have the same name but different functionality. Values can be passed through the parameters (braces) of the method call and return a specific output — this changes the Method Signature which allows for Polymorphism.

Advantages of objects:

- External programs can use subclass methods without knowing any details.
- Reusability of same Method names with differing functionality.

JAVA CONCEPTS AND STATEMENTS

Variables Objects store their states in 'Fields' — these are also known as 'Identifiers' and include variables (value can change) and constants (value cannot change).

Instance Variables Also known as 'Non-static Fields' (declared without static keyword). Instance variables are unique to each instance of the class (Object).

`currentSpeed()` of one bicycle is independent of the `currentSpeed()` of another bicycle object.

Class Variables Also known as 'Static Fields' and tells the Compiler there is exactly one copy of this variable only.

`static int numGears = 6` could be declared since the same number of gears will apply to all instances.

Local Variables Similar to how an object stores its state in Fields, a Method will store its temporary state in Local Variables. Any variables stored within the braces '( )' of a Method are Local Variables.

Parameters Set the conditions for a program operation. Can be sent across Methods within the Class as an 'argument'.

```
System.out.
println("Hello
World!")
```

Here, "Hello World!" is the condition for which it will be printed.

Primitive Data Types: (D1.9)

Revised

In Java, all variables must first be declared before they can be used, by stating the variables type and name.

`int gear = 1` tells the program that a Field named 'gear' exists, holds numerical data and has a value of 1.

A Primitive Data Type is predefined by Java and is stated using a keyword. Those within the scope of SL IB Computer Science are:

`byte` an 8-bit group of binary digits operated on as a single unit.

`int` an Integer, which is any whole negative or positive number.

`float` numbers with fractional value (a rational number with finite decimal expansion).

`double` numbers with a decimal place

`boolean` only two possible values; true or false.

`char` a single character (can take range of Unicode characters)

`String` text values written inside quotations.

Operators Variables can be evaluated using Operators, which include:

Symbol	Definition	Examples	
=	is equal to	X = 4, X = K	If X = 4
>	is greater than	X > 4	if X > 4 then
>=	is greater than or equal to	X >= 6	loop while X >= 6
<	is less than	VALUE[Y] < 7	loop until VALUE[Y] < 7
<=	is less than or equal to	VALUE[] <=12	if VALUE[Y] <= 12 then
≠	not equal to	X ≠ 4, X ≠ K	
AND	logical AND	A AND B	if X < 7 AND Y > 2 then
OR	logical OR	A OR B	if X < 7 OR Y > 2 then
NOT	logical NOT	NOT A	if NOT X = 7 then
mod	modulo	15 mod 7 = 1	if VALUE[Y] mod 7 = 0 then
div	integer part of quotient	15 div 7 = 2	if VALUE[Y] div 7 = 2 then

Relational Operators (==, >, =>, <, <=, !=)

Arithmetic Operators (+, –, * , / , % , =)

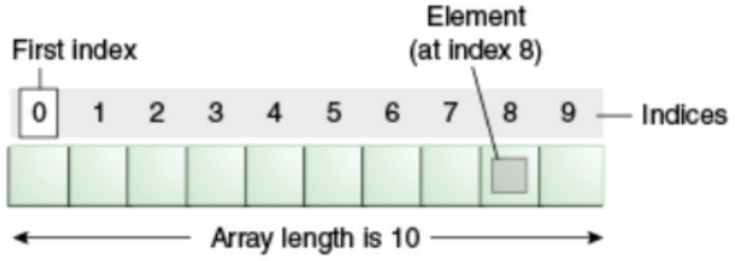
Unary Operators (++, - - , !)

Conditional Operators (&& , | |)

Arrays

An Array is a container object that holds a fixed number of values of a single type.

The length of the array is established when it is created, therefore its length is fixed.



An array of 10 elements.

Each item in the Array is known as the 'Element', and each Element is accessed by its numerical 'Index'. Numbering begins with 0, so the element: 9 is accessed at index: 8

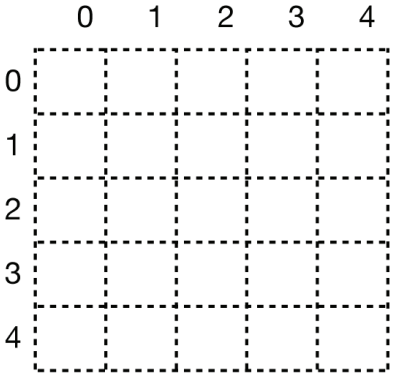
The Array type is expressed as: `type[]`

Where type is any data type and creates elements of this type.

`String[] stringArray` would create an Array of String values.

Multidimensional Arrays can be created using two or more sets of brackets:

`String[][] stringMultiArray` would look like:



ARRAY STATEMENT EXAMPLES

Create array w/ random values:

```
double[] arrayName = new double[10];
for (int j = 0; j<10; j++){
      arrayName[j] = Math.random();
}
```

Printing array values:

```
for(int = 0; j<10; j++){
      System.out.println(arrayName[j]);
```

Find average of array values:

```
double sum = 0.0;
      for (int j = 0; j<10; j++){
            sum = sum + arrayName[j];
      }
double average = sum/10;
```

Finding a min or max value:

```
for (int j = 0; j<10; j++){
      if(arrayName[j]>num)then{
            num = arrayName[j];
      }
```

EXPRESSIONS:

A Construct made up of variables, operators and method invocations which evaluate to a value.

Expressions are the core components of Statements; and these statements may be grouped into 'Statement Blocks'.

STATEMENTS:

Statements are like sentences, it is a complete unit of execution; each line in a program is a statement.

Methods are defined by the statements within them, since these create their behaviour.

Statements specify a sequence of actions which should be performed when the method is invoked.

STATEMENT BLOCKS:

Any sequence of statements can be grouped together to function as a single statement when expressed within braces — { }.

A statement block can be written inside another statement, this is known as 'nesting'. They are used to define methods.

```
double payRate = 15.75; {
      System.out.println(payRate);
      double newPayRate;
      newPayRate = 12.75;
      System.out.println(newPayRate)
}
```

CONTROL FLOW STATEMENTS:

Statements are executed sequentially, on their own. However, Java has control statements which break the sequential flow of code, and allow repetitive execution of statements and conditional execution of statements.

DECISION-MAKING STATEMENTS:

if-then	Only executes code if test (written in parameters) evaluates to true. If false, skips to end of if statement.
if-then-else	Same as if-then but goes to a second condition if test evaluates to false rather than skipping the statements.
case-switch-break/default	A Switch Statement can have any number of possible execution paths for int, char and byte data types.

LOOPING STATEMENTS:

while	Evaluates an expression, which must return a boolean value. The while statement continues testing the expression and executing the statement block beneath it until it evaluates to false.
for	A compact way to iterate over a range of values expressed in the parameters:

```
for(int i=1; i<11; i++){
     System.out.println("Count is: " + i);
          }
     }
```

BRANCHING STATEMENTS:

break	Terminates a loop when a value is found or statement block complete (expressed at the end of the statement block).
return	Exits from the current method and returns to where the method was originally invoked. By specifying a value with return, the exit can be achieved whilst sending the value back (e.g return num1;)

OBJECT CREATION:

A Class contains Constructor Methods that are invoked to create objects from the Class template.

Constructor Methods always share the same declaration name as the Class.

To create a new Bicycle object called myBike, the Constructor Method is called by the new operator. Creating an Object is the same as 'instantiating a Class'.

```
public Bicycle(int startGear, int startSpeed)
                        {

. . .

Bicycle myBike = new Bicycle(2, 0);
```

Therefore myBike is an instance of the class of Bicycles and has attributes startGear of 2 and startSpeed of 0 both of type int.

OBJECT REFERENCE:

In order to refer to the Fields within an object outside of the current class, an 'object reference' expression must be used with the dot operator:

Fields

```
motorBike.startGear;
```

You can use 'object reference' to invoke (call) an Object's Method. Additionally, any arguments within the parentheses can be provided:

Methods

```
myBike.setName(nameBike);
```

Parameters are the list of variables in a method declaration (also typically separated by commas).

Arguments are the actual values passed in when the method is invoked.

When an argument, stated inside the parameters, is passed through the Method; the internal processes of the Method use the argument temporarily in order to perform a calculation.