Notes Mandatory Study

Muhammad Haris Javed

Table of Contents

Digital

Number System

A system of writing , expressing or representation of a number of certain type in known as number system. But more logical meaning of a Number System is Collection of symbols usually numbers on which we can perform operation like Addition, Subtraction , Multiplication and Division and get a singular result.

For Example 2 + 3 = 5, But what is 'Hello' + 'World' Or 'Hello' - 'World' is nothing

There are some types of number system

1. Binary
2. Hexadecimal
3. Decimal
4. Octal

Base of Number System

Each system or collection has some finite amount of unique characters, Like when we are talking about Decimals they have 10 unique character [0 – 9] and Hexadecimals have 16 [0 – F] characters.

So when we are talking about base of a number It is equals to amount of unique characters they have.

For Example

Decimals [0 – 9] => 10 unique character so they has base 10,

Hexadecimal [0 – F] => 16 unique characters so they has base 16

Common properties of every number system is that All numbers start with 0 and ends with Base -1

Calculations

For calculation or performing some arithematical operations . It is required that all numbers are under same system or their bases are same

For Example we will not calculate 0b 0111 1111 and 0x EF

So for calculation of two different numbers system we must convert their bases under same number system

Conversion Of Number System

All number systems can be converted from 1 form to another this is further discussed when we are discussing each Number System specifically.

Counting in Number System

Counting uses the **B (**base of number system**)** amount symbols **first** through **last**. Counting begins with the incremental substitution of the least significant digit. When the available symbols for this position are exhausted, the least significant digit is reset to **first**, and the next digit of **higher significance** (one position to the left) is incremented (overflow), and incremental substitution of the low-order digit resumes. This method of reset and overflow is repeated for each digit of significance.

For Example:

Binary [0 - 1] => 0000 , 0001, 0010 , 0011, 0100, 1000 – 1111

Decimal [0 – 9] => 00, 01, 02, 03, … , 09 , 10 , … , 19 , 20 , … 99

Multiplication in Number System

A simple rule for multiplication of two digits in any base is to multiply them in decimal. If the product is less than the **base**, then we take it as the result. If the product is greater than the base we divide it by the base and take the remainder as the least significant digit. The quotient is taken as carry in the next significant digit.

For example, (3)4 × (1)4 = (3)4 but (3)4 × (2)4 = (12)4 since 3 × 2 = 6 is decimal and division of 6 by 4 has the remainder 2 and quotient 1.

Sign number VS Un-Sign numbers

Sign number are those numbers which have digits along with a plus or minus Sign. I.E Numbers from – Infinity to + Infinity are called sign numbers. While Un-Sign number doent have a sign so they have only positive numbers from 0 to + Infinity.

Binary

Binary numbers are collection of 0s and 1s and its base is 2, Each of its digit is called bit.

Because of its straight forward implementation In circuitry electronics by using logic gates. It is widely used in Digital Electronics and in Programming Languages which are closer to Electronics like Machine Language and Assembly Language.

Reason for its usage in electronics is that electronics signals only have two states [on ,off] and in binary we have [1,0].

Sign and Un-Sign Binary Numbers

As we know that binary numbers only contains 0s and 1s, (means no Sign -,+) that dosn't means binary numbers only contains Un- Sign numbers, they also contains – ve numbers and also +ve numbers.

In Sign numbers the most significance bit is called sign bit if is 0 the number is positive and if is 1 then number is negative

Lets look at example:

0100 , this is a positive number = 4

1100 , this is a negative number = -4

In the above example we saw that 1100 is a negative number but this number is also a positive 12.

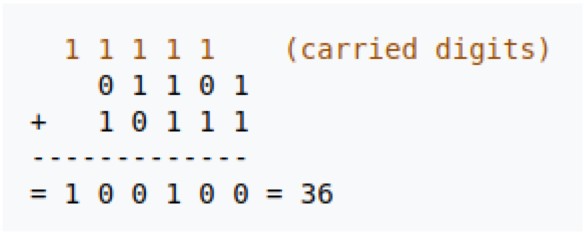
So It is upon us that how we treat binary number as sign number or un-sign number

If we treat number as sign than most significance bit represent a sign.

Arithmetic Operators

<u>Addition</u> It is a simplest operation in binary.

First we do addition with least significance bit of both numbers,
If either of them is 1 then we write 1 if both is 0 we write 0 , if both are 1 we write 1 and carry shifted towards left bit.



```
  1 1 1 1 1    (carried digits)
    0 1 1 0 1
+   1 0 1 1 1
-------------
= 1 0 0 1 0 0 = 36
```

<u>Substraction</u> The formula of subtraction is simple. If A and B are two sign binary digits then

A-B = A + NOT B + 1

A = 0011 = 3

B = 0010 = 2

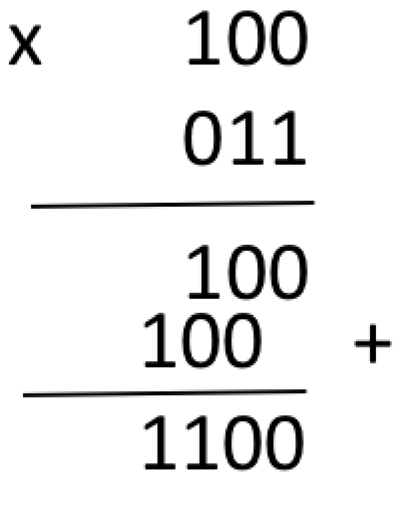
A - B = A NOT B + 1

```
A - B = 0011 + NOT 0010 + 1

A - B = 0011 + 1101 + 1


 0011

 1101

+

-------

10000 because we are using only 4 digit numbers so we neglect 5th


A - B = 0000 + 0001

A - B = 1;
```

Multiplication Of Binary

The Process of multiplying two numbers with each other is that

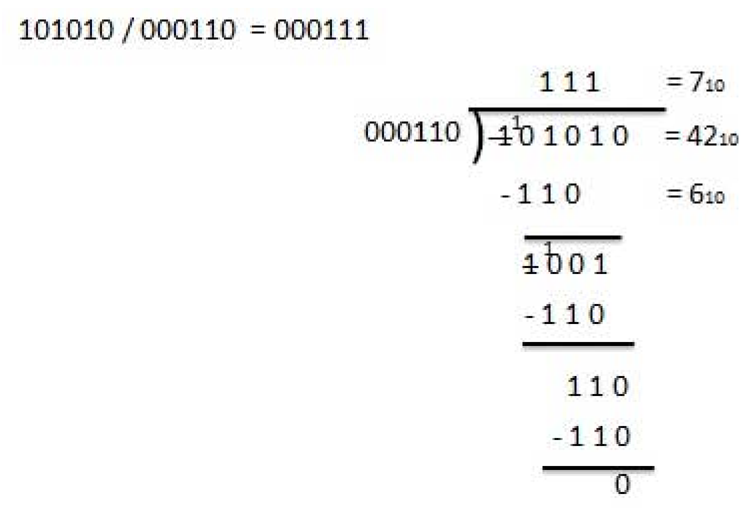
1. Multiplication of two bits is done by doing bitwise AND operation
2. Take LSB (Least Significant bit) from 2nd number multiply it bit by bit from 1st number.
3. Write down answer 1 line
4. Repeat the process until you iterate whole bits of 2nd number and writing answer line by line and also keep adding 0 to the tail of number (append right side of LSB)
5. finally add all answer you are writing line by line



Final result of 4 x 3

Divisions in Binary Numbers

Divisions in Binary is same as Decimals. It is also called long division process.



In Binary there is a lot of things we will discuss some of them in Boolean Algebra Topic

Octal

Octal numbers are 3 bit representation of binary numbers.

Octal numbers are from 0 – 7 having base 8, Octal numbers can be made from binary by groping three bits 1111111 = 001111111 = 001 , 111, 111 = 177_8

Conversion of numbers to octal

In Mathematics we know that 16 , 8 , 2 are of same base 2. means we will write 16 , 8 , 2 as 2^4 , 2^3 , 2^1 respectively. And we also know that in number system Hexadecimal, Octal , Binary have bases 16 , 8 , 2 respectively.
So it is correct to say that converting Hexadecimal, Octal , Binary to each other is easy then converting them to decimal.

Binary to Octal Conversion

As we already discuss converting binary to decimal is easy by grouping bits into 3 digits

1111111 = 001111111 = 001 , 111, 111 = 177_8

Why we are grouping them to 3 digits? Or From Where 3 is come from?

As we know binary has base 2 and octal has base 8 and 2^3 = 8, because of 2 exp 3 is equal to 8.

We also use wise versa process to convert octal to decimal

Hexadecimal to Octal Conversion

The easy process is that First convert Hexadecimal to binary then binary to Octal

Converting Hexadecimal to binary is again easy we can write each digit of hexadecimal to 4 bit binary and finally combine them.

For Example : FF as we know F = 1111 so we can write 1111 1111.

now group these binary number to 3 bits 011 111 111 = 377_8

Decimal to Octal

Converting Decimal to octal is simple just divide the decimal number and its quotient and append the reminder after MSB.

Here is an example of using repeated division to convert 1792 decimal to octal:

Decimal Number	Operation	Quotient	Remainder	Octal Result
1792	÷ 8 =	224	0	0
224	÷ 8 =	28	0	00
28	÷ 8 =	3	4	400
3	÷ 8 =	0	3	3400
0	done			

Arithmetic in Octal

Arithmetic in Octal is same as Arithmetic in decimal, or binary or in other number system.

Addition: Addition is same as decimals but after seven there must be 10 not 8

$456_8 + 123_8 = 601_8$

$$\begin{array}{rl} 1\ 1 & \text{carry} \\ 4\ 5\ 6 & = 302_{10} \\ +1\ 2\ 3 & = 83_{10} \\ \hline 6\ 0\ 1 & = 385_{10} \end{array}$$

Subtraction: In subtraction all rules are same in all number system. The only variation is in binary we borrow group of 2, 8, 10 and 16 for binary, octal , decimals and hexadecimal respectively.

Example:

$456_8 - 173_8 = 333_8$

$$\begin{array}{rl} 8 & \text{borrow} \\ {}^{3}4\ 5\ 6 & = 302_{10} \\ -1\ 7\ 3 & = 123_{10} \\ \hline 2\ 6\ 3 & = 179_{10} \end{array}$$

Multiplication:

As we discussed before the rules of mathematics in any number system in Multiplication in Number System.

Here is an example of multiplication in octal number system.

We have 6 × 3 = 18 in decimal, which when divided by 8 gives a remainder 2 and carry 2. Again 6 × 2 = 12 in decimal, and 12 + 2 = 14. This when divided by 8 gives a remainder 6 and a carry 1.

Hence $6_8 \times 23_8 = 162_8$	*$6 \times 3 = 18$* *18/8 = 2 with remainder 2 → l,s,d,* *$6 \times 2 = 12 + 2$ (carry) = 14* *14/8 = 1 with remainder 6.*

Hexadecimals

There for Hexadecimal are more human readable 4 bit representation of binary numbers in computer science.

Because computer are used to store data in memory registers, ram, … Counted on bytes, words

and Hexadecimal are perfect representation as for word we use single digit hexadecimal

and for byte we use double digit hexadecimal.

Conversion To Hexadecimal

Hexadecimal - Binary

Conversion Between Hexadecimal and binary is easy

Write binary in group of 4 bits and convert each 4 bit in its Hexadecimal equilent.

For Example: 1010 1111 = AF

Its wise versa convert Hexadecimal to binary.

Hexadecimal – Octal

Again conversion from Hexadecimal to Octal is an easy stuff.

Convert Octal to Binary , Group bits by 4 and convert to hexadecimal

$257_8 = 010\ 101\ 111_2 = 010101111_2 = 0000\ 1010\ 1111_2 = 0AF_{16} = AF_{16}$

Hexadecimal – Decimals

There is a formula to convert Hexadecimal to Decimal

$H_{n-1} \times 16^{n-1} + H_{n-2} \times 16^{n-2} + \ldots.. + H_2 \times 16^2 + H_1 \times 16^1 + H_0 \times 16^0$

Put Every digit of decimal equivalent Hexadecimal into H_n with respect to its position, put LSB to position 0 and you will get its decimal equivalent.

For Example: Here is a hexadecimal number $0d47a1_{16}$

In this case positions are $0_5\ d_4\ 4_3\ 7_2\ a_1\ 1_0$ this is a six digit number $h_{5-}\ h_0$

convert each digit into its decimal equivalent $0_5\ 13_4\ 4_3\ 7_2\ 10_1\ 1_0$

now put each digit into formula $0 * 16^5 + 13*16^4 + 4*16^3 + 7*16^2 + 10*16^1 + 1*16^0 = 870305_{10}$

The method to convert decimal to hexadecimal is Repeating Divide and Remainder method

Lets convert base 10 188 to base 16

DIVISION	**RESULT**	**REMAINDER (in HEX)**
188 / 16	11	C (12 decimal)
11 / 16	0	B (11 decimal)
ANSWER		BC

Arithmetic Operations in Hexadecimal

Hexadecimal borrows same rules for arithmetics in hexadecimal

Addition:

$4A6_{16} + 1B3_{16} = 659_{16}$

$$\begin{array}{rl} 1 & \text{carry} \\ 4\,A\,6 & = 1190_{10} \\ +1\,B\,3 & = \ 435_{10} \\ \hline 6\,5\,9 & = 1625_{10} \end{array}$$

Substraction:

$4A6_{16} - 1B3_{16} = 2F3_{16}$

$$\begin{array}{rl} 16 & \text{borrow} \\ {}^{3}4\,A\,6 & = 1190_{10} \\ -1\,B\,3 & = \ 435_{10} \\ \hline 2\,F\,3 & = 755_{10} \end{array}$$

Decimals

Decimal number system is that we are using it in our daily life. It has [0-9] 10 unique symbols to represent an amount , Therefore its base in 10.

Conversion To Decimals

For **n** amount of digit having $\mathbf{D_{n-1}}$ to $\mathbf{D_0}$ having base **b**

$D_{n-1}\,{}_x\,B^{n-1} + D_{n-2}\,{}_x\,B^{n-2} + \ldots\ldots + D_2\,{}_x\,B^2 + D_1\,{}_x\,B^1 + D_0\,{}_x\,B^0$

For Example :

$1011_2 = 1_x 2^3 + 0_x 2^2 + 1_x 2^1 + 1_x 2^0 = 8+0+2+1 = 11$

$FF_{16} = 15_x 16^1 + 15_x 16^0 = 240 + 16 = 255$

$755_8 = 7_x 8^2 + 5_x 8^1 + 5_x 8^0 = 448+40+5 = 493$

Conversion From Decimals

To convert any number system from decimal is same repeated dividing and writing remainder process

For any decimal number **d** converting to base **B.**

Take a **d** and divide it by **B** and write symbol for equivalent remainder in LSB

continue Dividing **d** and writing its remainder right along with LSB to get your converted number.

See Example in Conversion To Hexadecimal

Gates

In every number system we have some kind operators. In binary we have two types of operators

1. Arithmetic Operators
2. Bitwise Operators

Bitwise operators or logic gates are used when we have only 2 states of a symbol I.e [true, false] Or [0 , 1]

There are some brief defination of bitwise operators

OR If either operand is 1 it will give 1 otherwise 0

AND If either operand is 0 it will give 0 otherwise 1

NOT It is a singular operator it change the state of symbol Not 1 will give 0 and Not 0 will give 1

XOR If operands are different it will give 1 otherwise 0

XNOR If operands are same it will give 1 otherwise 0

Boolean Algebra

Boolean Algebra is used to simplify circuit logic in digital circuits, also used in software logic and design in conditions.

Rule in Boolean Algebra

Following are the important rules used in Boolean algebra.

- Variable used can have only two values. Binary 1 for HIGH and Binary 0 for LOW.
- Complement of a variable is represented by an overbar (-). Thus, complement of variable B is represented as. Thus if B = 0 then Not B= 1 and B = 1 then Not B = 0.
- ORing of the variables is represented by a plus (+) sign between them. For example ORing of A, B, C is represented as A + B + C.
- Logical ANDing of the two or more variable is represented by writing a dot between them such as A.B.C. Sometime the dot may be omitted like ABC.

Postulates and Basic Laws of Boolean Algebra

Or Laws:

$$x + 0 = x$$

$$x + 1 = 1$$

$$x + x = x$$

$$x + x' = 1$$

And Laws:

$$x.1 = x$$

$$x.0 = 0$$

$$x.x = x$$

$$x.x' = 0$$

Boolean Laws

Commutative law

This law states that the order in which the logic operations are performed is irrelevant as their effect is the same.

$$(i)\ A.B = B.A \qquad (ii)\ A + B = B + A$$

Distributive law

$$A.(B + C) = A.B + A.C$$

And Law

$$(i)\ A.0 = 0 \qquad (ii)\ A.1 = A$$

$$(iii)\ A.A = A \qquad (iv)\ A.\overline{A} = 0$$

Or Law

$$(i)\ A + 0 = A \qquad (ii)\ A + 1 = 1$$

$$(iii)\ A + A = A \qquad (iv)\ A + \overline{A} = 1$$

INVERSION law

This law uses the NOT operation. The inversion law states that double inversion of a variable results in the original variable itself.

$$\overline{\overline{A}} = A$$

Boolean Expression

Boolean algebra deals with binary logic, A boolean function is described by boolean expresion

$$\underset{\text{Boolean Function}}{F\,(A, B, C, D)} \quad = \quad \underset{\text{Boolean Expression}}{A + \overline{BC} + ADC}$$

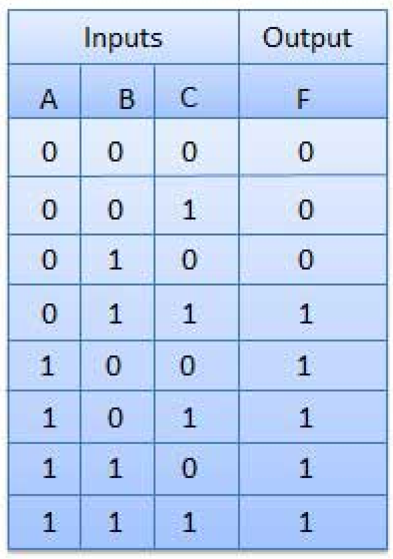
Truth Table Formation

A truth table represents a table having all combinations of inputs and their corresponding result.

Consider the following equation F(A,B,C) = A + BC

In this case the answer is high if A is high or BC or Both, BC is high if both is high

For above equation we have following truth table.



Inputs			Output
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

DeMorgan's Theorem

This theorem is usefull in finding the complement of boolean function.

It states that

The Complement of logical Oring of atleast two variable is equal to the complement of complement and to two variables

DeMorgan's theorem with 2 Boolean variables x and y can be represented as

(x + y)' = x'.y'

The dual of the above Boolean function is

(x.y)' = x' + y'

Duality Theorem

Group1	Group2
x + 0 = x	x.1 = x
x + 1 = 1	x.0 = 0
x + x = x	x.x = x
x + x' = 1	x.x' = 0
x + y = y + x	x.y = y.x
x + (y + z) = (x + y) + z	x.(y.z) = (x.y).z
x.(y + z) = x.y + x.z	x + (y.z) = (x + y).(x +

	z)

Methods to simplify the boolean function

The methods used for simplifying the Boolean function are as follows −

- Karnaugh-map or K-map, and
- NAND gate method.

Karnaugh-map or K-map

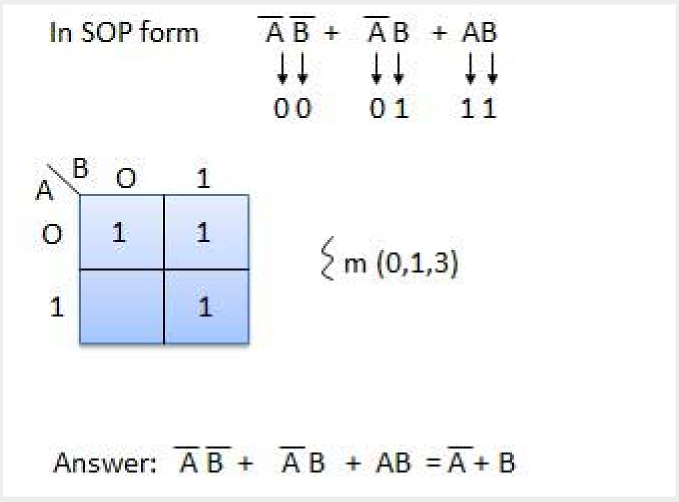
Boolean Logic , theorems and De-Morgan's theorems are useful in manipulating the logic. We can realize the no of gates used in this equation.
We can reduce the requirement of gates using K-Map method

Sum of Products (SOP) Form

Its name describes it very well when equation is written in such form where we should first done anding of variable than oring it.

For Example

AB + AC + BC


In SOP form $\bar{A}\bar{B} + \bar{A}B + AB$
00 01 11
A B 0 1
0 1 1
1 1
$\sum m(0,1,3)$
Answer: $\bar{A}\bar{B} + \bar{A}B + AB = \bar{A} + B$

Example 1

Let us ***simplify*** *the Boolean function, f = p'qr + pq'r + pqr' + pqr*

We can simplify this function in two methods.

Method 1

Given Boolean function, f = p'qr + pq'r + pqr' +pqr.

Step 1 − *In first and second terms r is common and in third and fourth terms pq is common. So, take the common terms by using* ***Distributive law****.*

⇒ f = $(p'q + pq')r + pq(r' + r)$

Step 2 *− The terms present in first parenthesis can be simplified to Ex-OR operation. The terms present in second parenthesis can be simplified to '1' using* ***Boolean postulate***

⇒ f = $(p \oplus q)r + pq(1)$

Step 3 *− The first term can't be simplified further. But, the second term can be simplified to pq using* ***Boolean postulate****.*

⇒ f = $(p \oplus q)r + pq$

Therefore, the simplified Boolean function is ***f = (p⊕q)r + pq***

Operating System

The Operating System is a program that is initially loaded into a computer by boot program. Manages all other programs in a computer.
Programs make use of operating system by requesting services by defined API's

Operating is an example of system software

Process

A process is an instance of a program currently in executing. Process consist of a set of memory addresses that holds data and instructions, a set of resources allocated by operating system to a process and Process Control Block (**PCB**) to store properties of a process.

A process is used to group resource together. A process can be of many several states like Running, Ready , Blocked or Terminated

PCB

PCB is a data structure in Operating System kernel containing the information needed to manage the scheduling of particular process. The PCB is a manifest of process in a operating system

Role

- Access and modified by most OS utilities including those involved with scheduling, memory and I/O resources access and performance monitoring
- Define current state of Operating system
- Pointers to other PCBs inside a PCB allow the creation of those queues of processes in various scheduling states ("ready", "blocked", etc.) that was previously mentioned.

Structure

Structure of PCB was classified into three main categories.

1. Process Identification Data
2. Process State Data

3. Process Control Data

The Approach commonly followed is to create and update status tables for each relevant entries

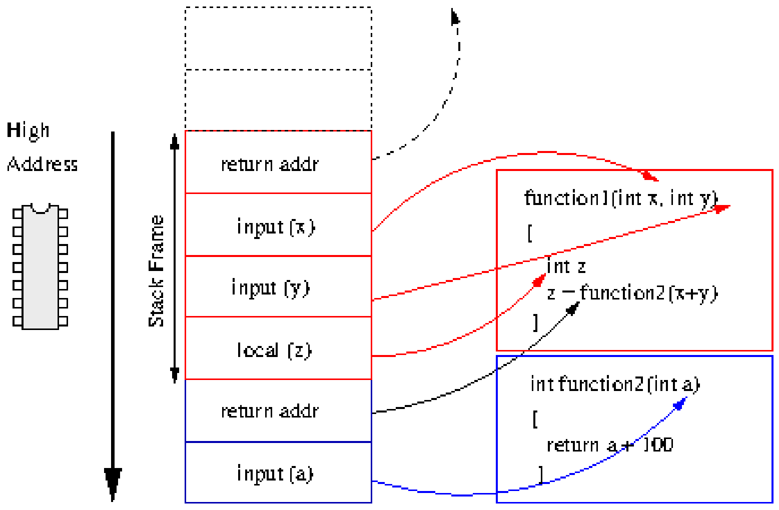
Elements of a Process

Process Id : Operating system assign id to a Process which is unique to other running processes.

Memory : OS assign memory to process which stores program code , its variables and other allocated storage. Parts of memory can be share between processes.

Code and Data : Program code and data should kept separately since they required different permission from OS. OS gives code read and execute permission on other hand give data to read and write permission not execute.

Stacks: Stacks are consider part of data section. Stacks are fundamentals to function calls. Each time when a function called it gets a new stack frame which usually contain at minimum, The address to return, input arguments to the function and space for local variables.



Stacks do make calling functions slower, because values must be moved out of registers and into memory

Thread

Thread are the entities schedule for executing on CPU.

Thread is a single sequence execution within the process. Thread have some properties of a Process therefore thread is also called Light weight process. The CPU switch rapidly back and forth among the thread creating the illusion that the threads are running in parallel. Thread are popular way to improve application through parallelism. Like process thread can be of any several states like Running, Blocked , Ready or Terminated. Each thread has its own stack space because threads may call procedures so it has its own executing history.

A Thread consist of Program Counter, Register Set, and a stack space.

Thread create is child thread.

Why Threads

Following are some reasons why we use threads in designing operating systems.

1.A process with multiple threads make a great server for example printer server.

2.Because threads can share common data, they do not need to use interprocess communication.

3.Because of the very nature, threads can take advantage of multiprocessors.

Threads are cheap in the sense that

1.They only need a stack and storage for registers therefore, threads are cheap to create.

2.Threads use very little resources of an operating system in which they are working. That is, threads do not need new address space, global data, program code or operating system resources.

3.Context switching are fast when working with threads. The reason is that we only have to save and/or restore PC, SP and registers.

But this cheapness does not come free - the biggest drawback is that there is no protection between threads.

Types of Threads

There are two type of threads

1. **User Level Thread**

   User Level Threads are implemented on user libraries or user program, Operating System knows nothing about user level threads. So it has a great advantage that we implement user level threads on those Operating Systems which does not support multiple threading.

2. **Kernel Level Thread**

   In this case Operating System knows about thread so no user level system or program in needed. And also Operating System manages more gratefully than user program

Process Scheduling

Work of physical processors is to process but the problem is when and which process is allowed to use processor. The solution of this problem is called **Process Scheduling.**

When more than one process is runnable than operating system must decide which one would run first. The part of operating system which takes this decision is called **scheduler**.

Dead Lock

A Set of process are in a deadlock state if each process in a set is waiting for an event that cause by another process in that set OR each deadlock process is waiting for a resource that can be released by only that process in a waiting list.

Virtual Memory

When a very large program store them selves in a memory in a form of pages while their execution and only required pages and processes are loaded in to main memory this technique is called **virtual memory**.

The technique is very useful as large virtual memory is available for user or user process.

In real scenarios, most processes never need all their pages at once, for following reasons :

- Error handling code is not needed unless that specific error occurs, some of which are quite rare.
- Arrays are often over-sized for worst-case scenarios, and only a small fraction of the arrays are actually used in practice.
- Certain features of certain programs are rarely used.

Key Points:

Local variables and input parameters are kept in Functions Stack Frame and Global variables are kept in separate area of data memory.

Because each frame has a reference one before it a debugger can walk backward it is called **stack trace**.

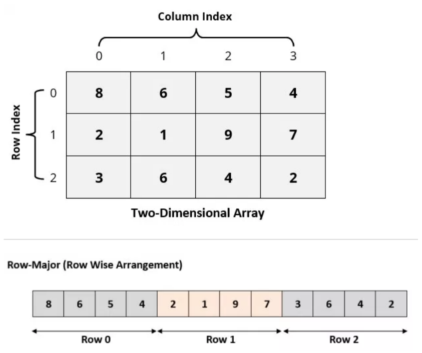
Data Structure

What are They?

Data Structure is to store data in an organized manner so that it can be retrieved more effectively.

Array

A data Structure in which data set is organized in such a way that its elements are placed one after other (In Linear way) and they are identified by at least one index or key. And their position is calculated by a mathematical formula.



Linked List

A data structure in which data is store in such a manner that each data element is linked with other element in a set.

How ever in Linked List data is not store as continuous memory location.

Each element has two things one is the data and other is the address of next element.

Trees

A data structure in which data represent as node connected with its parent node and child node.

Each node is connected with 1 parent node and multiple child nodes

A node with no parent is known as root node.

A node with no child is known as leaf.

The Benefits of tree on Array and Linked list is that Searching is faster and Addition and Deletion is faster.

Map

Map is an Abstract data structure represent as key value pair

Keys cannot be duplicated and use to identify element.

Maps are also called Associative Array.

Hashing Based Collection

Hashing means using some function or algorithm to map object data to some representative integer value. This so-called hash code (or simply hash) can then be used as a way to narrow down our search when looking for the item in the map.

Dictionary

A dictionary is also called a hash, a map, a hashmap in different programming languages (and an Object in JavaScript). They're all the same thing: a key-value store.

Stack

Stack is a collection in which data Inserted or removed only from Head or Top.

A stack is called last in first out data structure (LIFO) In which we use Push and POP functions to retrieve data.

Graph

A Graph consists of a finite set of vertices(or nodes) and set of Edges which connect a pair of nodes

Like Trees Graph is a Data Structure which is node based. But with differences.

In Graphs, nodes have multiple parents and multiple childs

The technically Graph node may have multiple references of its parents and childs

To Iterate All graph node we use **Breath First Approach,** and **Depth First Approach.**

Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like linkedIn, Facebook. For example, in Facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender, locale etc.

Queue

Queues are called first in first out Data structure. Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.

Heaps

A Heap is a special Tree-based data structure in which the tree is a complete binary tree. Generally, Heaps can be of two types:

1. **Max-Heap:** In a Max-Heap the key present at the root node must be greatest among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree.
2. **Min-Heap:** In a Min-Heap the key present at the root node must be minimum among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree.

Heaps have many uses like heap sort and in priority Queues.

Programming Algo

In Computer Science Algorithm is specification of how to solve a class of problem.

Algorithms have time amount of space and time.

Space and Time Complexity of Algorithms

Some times we have more than one way to solve the problem but question is how we could analysis that which algorithms in better. And The answer is by judge on the basis of space time trade off.

In every algorithms we may comprise over space (Memory) to gain speed or less time taken. Or some times we may compromise over Time to take less memory.

Compromising of time over space or space over time is called time space trade off.

Variable

In Programming algorithms have data values that may be fixed or variable

For Example

Fixed data:

for 1 to 5

Variable Data:

input a number from user

for 1 to that number:

Functions and Procedures

In a program there are sections of code that we want to reuse, those chunks of statements given a name those are called functions / procedures.

Functions may call them self or be called by some other functions.

The only difference between procedure and function is that Functions Return but procedures are not.

Method

Method is similar like functions but they are Integral part of a class. They share properties of whole object.

Methods used to manipulate more than one properties of an object.

For Example

here is an example of cd player

```
public void play(CD);

public void stop(CD);
```

this is a function.

```
Public class CDPlayer(CD){

        public void play();

        public void stop();

}
```

this is an example of methods

Decisions

In programming also we face some situations where we want a certain block of code to be executed when some condition is fulfilled.

if: if statement is the most simple decision making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

if-else: The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won’t. But what if we want to do something else if the condition is false. Here comes the else statement. We can use the else statement with if statement to execute a block of code when the condition is false.

nested-if: A nested if is an if statement that is the target of another if or else. Nested if statements means an if statement inside an if statement. Yes, it allows us to nest if statements within if statements. i.e, we can place an if statement inside another if statement.

s**witch-case** The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression

Loops

In computer programming, a loop is a sequence of instruction s that is continually repeated until a certain condition is reached.

For Loop: For loop is used when we know how many time we want to execute that piece of code.

While: While loop is used when we don’t know how many times does the loop executes but we know the condition when to stop the loop.

Do While: Do while is used when we want to execute our code atleast 1 time even if condition will not satisfy.

Iteration

In the context of computer programming, is a process wherein a set of instructions or structures are repeated in a sequence a specified number of times or until a condition is met. When the first set of instructions is executed again, it is called an iteration.

Iteration vs recursion

Historically, “iteration” in computer science is a special form of recursion for which no additional stack space is needed in other words, tail recursion. This form is computationally exactly equivalent to what we now colloquially know as “iteration”, namely a finite loop (such as a for loop with a fixed lower and upper bound).

Traverse

“Traversal” just means walking through (all or some) elements of a data structure.

Breath First

Traversal in such a way where we visit every node on a level before going to a lower level. This search is referred to as *breadth-first search*

Depth First

As the traversal is deepened as much as possible on each child before going to the next sibling.

DBMS

Relation

A relational database is composed of two-dimensional tables. (A table can also be called a relation, although relational "purists" would argue that there is a subtle distinction between the two.) Each "row" of a table is called a tuple. Each tuple is composed of fields, one for each attribute of the table. (The attributes are the names we associate with the fields/columns.)

One to One Relation

In a one-to-one relationship, one record in a table is associated with one and only one record in another table.

For example, in a school database, each student has only one student ID, and each student ID is assigned to only one person.

One to Many Relation

In a one-to-many relationship, one record in a table can be associated with one or more records in another table.

For example, each customer can have many sales orders.

Self Relation

In self relation one record in a table is associated with record with one or more records in same table.

For example, In Navigation menu each link is connected with one or more link, which are their childs.

Many to Many Relation

A many-to-many relationship occurs when multiple records in a table are associated with multiple records in another table. For example, a many-to-many relationship exists between customers and products: customers can purchase various products, and products can be purchased by many customers.

Functions in RDBMS

A function is compiled and executed every time whenever it is called. A function must return a value and cannot modify the data received as parameters.

Relational Algebra

a DBMS translates an SQL query submitted to it into a program that produces the answer to that query— as having as its purpose to translate the SQL query into an equivalent relational algebra query.

Queries in relational algebra are based upon the use of three elementary operations on tables: project, restrict, and join. (The "restrict" operation is usually called "select", but here we use the terminology of C.J. Date (prolific author on the subject of the relational model), in part because the SELECT verb in SQL has an entirely different meaning.)

Because the result of applying an operation is itself a table, we can compose operations in sequence. The obvious analogy is with functions mapping reals to reals, where the function (f o g), read "f of g", is defined by (f o g)(x) = f(g(x)). Indeed, the operators in relational algebra are functions, with tables as both domain and range.

For Examples

PROJECT <list of attributes> FROM <table>

In RA, this is written

$\Pi_{\text{<list of attributes>}}(\text{<table>})$

For example, the SRA expression

PROJECT Name, Class FROM Student

Tables

In relational databases, and flat file databases, a table is a set of data elements (values) using a model of vertical columns (identifiable by name) and horizontal rows, the cell being the unit where a row and column intersect. A table has a specified number of columns, but can have any number of rows.

Keys

Key plays an important role in relational database; it is used for identifying unique rows from table. It also establishes relationship among tables

Candidate Key – A super key with no redundant attribute is known as candidate key

Alternate Key – Out of all candidate keys, only one gets selected as primary key, remaining keys are known as alternate or secondary keys.

Composite Key – A key that consists of more than one attribute to uniquely identify rows (also known as records & tuples) in a table is called composite key.

Foreign Key – Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

Primary Key – A primary is a column or set of columns in a table that uniquely identifies tuples (rows) in that table.

Super Key – A super key is a set of one of more columns (attributes) to uniquely identify rows in a table

Normalization

Normalization is a database design technique which organizes tables in a manner that reduces redundancy and dependency of data. It divides larger tables to smaller tables and links them using relationships.

1 Normal Form

1NF (First Normal Form) Rules

- Each table cell should contain a single value.
- Each record needs to be unique.

2 Normal Form

- Rule 1- Be in 1NF
- Rule 2- Single Column Primary Key

3 Normal Form

- Rule 1- Be in 2NF
- Rule 2- Has no transitive functional dependencies

To move our 2NF table into 3NF, we need to again divide our table.

Transactions

A transaction, in the context of a database, is a logical unit that is independently executed for data retrieval or updates. In relational databases, database transactions must be atomic, consistent, isolated and durable--summarized as the ACID acronym.

Transactions are completed by COMMIT or ROLLBACK SQL statements, which indicate a transaction's beginning or end. The ACID acronym defines the properties of a database transaction, as follows:

- Atomicity: A transaction must be fully complete, saved (committed) or completely undone (rolled back). A sale in a retail store database illustrates a scenario which explains atomicity, e.g., the sale consists of an inventory reduction and a record of incoming cash. Both either happen together or do not happen - it's all or nothing.

- Consistency: The transaction must be fully compliant with the state of the database as it was prior to the transaction. In other words, the transaction cannot break the database's constraints. For example, if a database table's Phone Number column can only contain numerals, then consistency dictates that any transaction attempting to enter an alphabetical letter may not commit.
- Isolation: Transaction data must not be available to other transactions until the original transaction is committed or rolled back.
- Durability: Transaction data changes must be available, even in the event of database failure.

Relation cardinality & modularity in database

Cardinality refers to the relationship between a row of one table and a row of another table. The only two options for cardinality are one or many.

- One to many
- One to One

Modality : As cardinality is the maximum number of connections between table rows (either one or many), modality is the least number of row connections! Modality also only has two options, 0 being the least or 1 being the least

- Connection is nullable
- Connection is not nullable

Functional Dependency

The attributes of a table is said to be dependent on each other when an attribute of a table uniquely identifies another attribute of the same table.

For example: Suppose we have a student table with attributes: Stu_Id, Stu_Name, Stu_Age. Here Stu_Id attribute uniquely identifies the Stu_Name attribute of student table because if we know the student id we can tell the student name associated with it. This is known as functional dependency and can be written as Stu_Id->Stu_Name or in words we can say Stu_Name is functionally dependent on Stu_Id.

Formally:

If column A of a table uniquely identifies the column B of same table then it can represented as A->B (Attribute B is functionally dependent on attribute A)

OOP (Object Oriented Programming)

Defination

Object-oriented programming (OOP) is a programming language model organized around objects rather than "actions" and data rather than logic. Historically, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data.

Functional / Procedural Programming.

In computer science, functional programming is a programming paradigm—a style of building the structure and elements of computer programs—that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.

Classes

A Class is a collection of objects which has identical properties and common behaviour.

Access Modifier

Access modifiers (or access specifiers) are keywords in object-oriented languages that set the accessibility of classes, methods, and other members. Access modifiers are a specific part of programming language syntax used to facilitate the encapsulation of components. ... A class cannot be declared as private

Objects

A real world entity which has its own existance is known as object. It has its own behaviour and certain features.

In Programming languages Objects stored in memory and its reference is stored in variable.

Each object has unique reference.

Members

In Object Oriented Programming classes have their members which are called fields and methods.

Fields are the data and other objects associated with that class.

Methods are Fields manipulators and allows object to behave with respect to its fields

Access Modifier are used to set permisions of members.

Method and Signature

Method signature is the method name and the number, type and order of its parameters. Return types and thrown exceptions are not considered to be a part of the method signature.

Construction and Destruction

A class constructor is a special member function of a class that is executed whenever we create new objects of that class.

A constructor will have exact same name as the class and it does not have any return type at all, not even void. Constructors can be very useful for setting initial values for certain member variables.

Overloading / overriding

Overloading means creating methods with same name but different parameters. Overriding means re-defining body of a method of superclass in a subclass to change behavior of a method.

Polymorphism is a wide concept which includes overriding and overloading and much more in it's scope.

Object Modal

An object model is a logical interface, software or system that is modeled through the use of object-oriented techniques. It enables the creation of an architectural software or system model prior to development or programming.

An object model is part of the object-oriented programming (OOP) lifecycle.

Generalization/specialization/inheritance

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.

When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the base class, and the new class is referred to as the derived class.

The idea of inheritance implements the is a relationship. For example, mammal IS-A animal, dog IS-A mammal hence dog IS-A animal as well and so on

Polymorphism

The word polymorphism means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.

C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

Encapsulation

Data encapsulation is a mechanism of bundling the data, and the functions that use them and data abstraction is a mechanism of exposing only the interfaces and hiding the implementation details from the user.

Virtual methods / vtable

For every class that contains virtual functions, the compiler constructs a virtual table, a.k.a vtable. The vtable contains an entry for each virtual function accessible by the class and stores a pointer to its definition. Only the most specific function definition callable by the class is stored in the vtable

Interface

An interface describes the behavior or capabilities of a C++ class without committing to a particular implementation of that class.

The C++ interfaces are implemented using abstract classes and these abstract classes should not be confused with data abstraction which is a concept of keeping implementation details separate from associated data.

A class is made abstract by declaring at least one of its functions as pure virtual function. A pure virtual function is specified by placing "= 0" in its declaration as follows

Packages

A package is a namespace that organizes a set of related classes and interfaces. Conceptually you can think of packages as being similar to different folders on your computer. You might keep HTML pages in one folder, images in another, and scripts or applications in yet another.

Design Pattern

By definition, Design Patterns are reusable solutions to commonly occuring problems(in the context of software design). Design patterns were started as best practices that were applied again and again to similar problems encountered in different contexts. They become popular after they were collected, in a formalized form

Singleton

The singleton pattern is one of the simplest design patterns: it involves only one class which is responsible to instantiate itself, to make sure it creates not more than one instance; in the same time it provides a global point of access to that instance. In this case the same instance can be used from everywhere, being impossible to invoke directly the constructor each time.

Need

Sometimes it's important to have only one instance for a class. For example, in a system there should be only one window manager (or only a file system or only a print spooler). Usually singletons are used for centralized management of internal or external resources and they provide a global point of access to themselves.

Intent

- Ensure that only one instance of a class is created.
- Provide a global point of access to the object

Implementation

The implementation involves a static member in the "Singleton" class, a private constructor and a static public method that returns a reference to the static member.

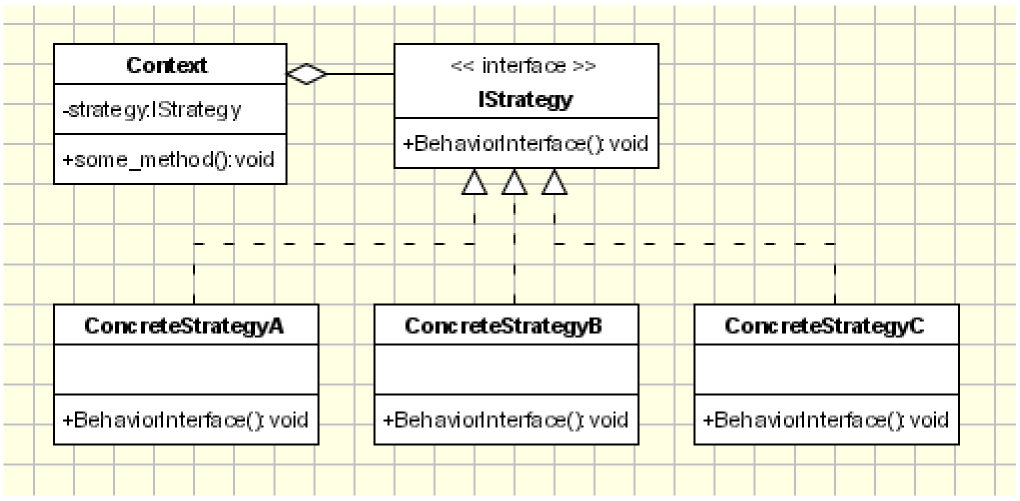
Strategy

There are common situations when classes differ only in their behavior. For this cases is a good idea to isolate the algorithms in separate classes in order to have the ability to select different algorithms at runtime.

```java
public interface IBehaviour {
        public int moveCommand();
}

public class AgressiveBehaviour implements IBehaviour{
        public int moveCommand()
        {
                System.out.println("\tAgressive Behaviour: if find another robot attack it");
                return 1;
        }
}

public class DefensiveBehaviour implements IBehaviour{
        public int moveCommand()
        {
                System.out.println("\tDefensive Behaviour: if find another robot run from it");
                return -1;
        }
}

public class NormalBehaviour implements IBehaviour{
        public int moveCommand()
        {
                System.out.println("\tNormal Behaviour: if find another robot ignore it");
                return 0;
        }
}

public class Robot {
        IBehaviour behaviour;
        String name;

        public Robot(String name)
        {
                this.name = name;
        }

        public void setBehaviour(IBehaviour behaviour)
        {
                this.behaviour = behaviour;
        }

        public IBehaviour getBehaviour()
        {
                return behaviour;
        }

        public void move()
        {
                System.out.println(this.name + ": Based on current position" +
                                         "the behaviour object decide the next move:");
                int command = behaviour.moveCommand();
                // ... send the command to mechanisms
                System.out.println("\tThe result returned by behaviour object " +
                                        "is sent to the movement mechanisms " +
                                        " for the robot '"  + this.name + "'");
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }
}


public class Main {

        public static void main(String[] args) {

                Robot r1 = new Robot("Big Robot");
                Robot r2 = new Robot("George v.2.1");
                Robot r3 = new Robot("R2");

                r1.setBehaviour(new AgressiveBehaviour());
                r2.setBehaviour(new DefensiveBehaviour());
                r3.setBehaviour(new NormalBehaviour());

                r1.move();
                r2.move();
                r3.move();

                System.out.println("\r\nNew behaviours: " +
                                "\r\n\t'Big Robot' gets really scared" +
                                "\r\n\t, 'George v.2.1' becomes really mad because" +
                                "it's always attacked by other robots" +
                                "\r\n\t and R2 keeps its calm\r\n");

                r1.setBehaviour(new DefensiveBehaviour());
                r2.setBehaviour(new AgressiveBehaviour());

                r1.move();
                r2.move();
                r3.move();
        }
}
```

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

Implementation



Strategy - defines an interface common to all supported algorithms. Context uses this interface to call the algorithm defined by a ConcreteStrategy.

Concrete Strategy - each concrete strategy implements an algorithm.

Context

contains a reference to a strategy object.

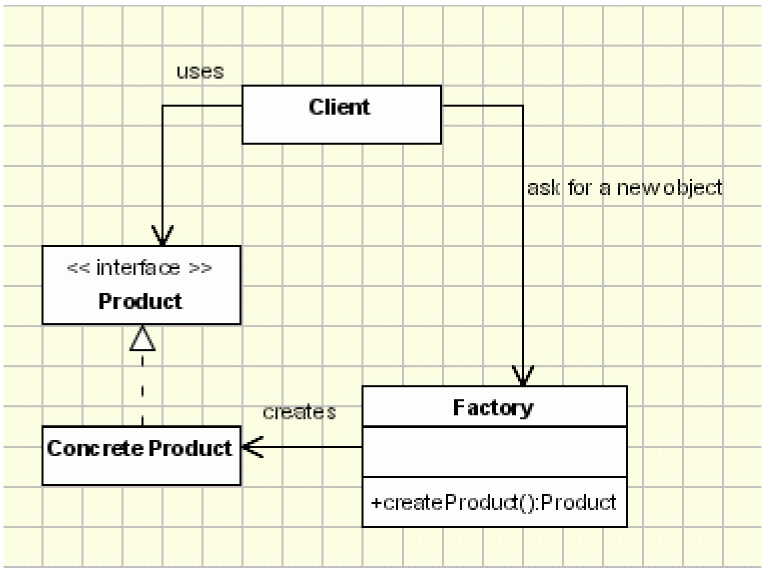
may define an interface that lets strategy accessing its data.

The Context objects contains a reference to the Concrete Strategy that should be used. When an operation is required then the algorithm is run from the strategy object. The Context is not aware of the strategy implementation. If necessary, addition objects can be defined to pass data from context object to strategy.

Factory

- creates objects without exposing the instantiation logic to the client.
- refers to the newly created object through a common interface

Implementation



The implementation is really simple

- The client needs a product, but instead of creating it directly using the new operator, it asks the factory object for a new product, providing the information about the type of object it needs.
- The factory instantiates a new concrete product and then returns to the client the newly created product(casted to abstract product class).
- The client uses the products as abstract products without being aware about their concrete implementation.

```
public class ProductFactory{
        public Product createProduct(String ProductID){
                if (id==ID1)
                        return new OneProduct();
                if (id==ID2) return
                        return new AnotherProduct();
                ... // so on for the other Ids

        return null; //if the id doesn't have any of the expected values
    }
    ...
}
```

MVC

MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate application's concerns.

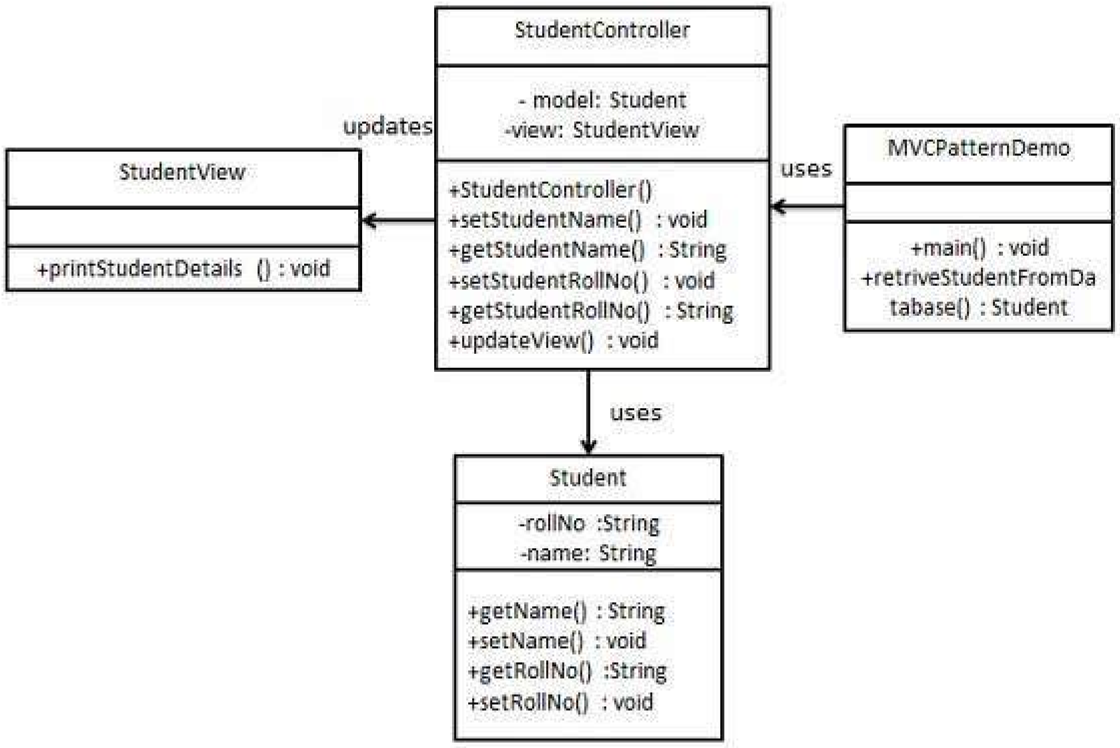
Model - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.

View - View represents the visualization of the data that model contains.

Controller - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

Implementation

We are going to create a Student object acting as a model.StudentView will be a view class which can print student details on console and StudentController is the controller class responsible to store data in Student object and update view StudentView accordingly.



MVCPatternDemo, our demo class, will use StudentController to demonstrate use of MVC pattern.