1. c-language

2. java

3. sql

4. html, css, javascript

5. xml

java

|- jse (core java)

|- jee (advance java)

|- jme

java

|- core java

|- advance java

|- frameworks (experience)

i. hibernate

ii. spring

|- tools

i. maven

ii. junit

iii. git

etc

workshop on frameworks

software company

|- 1. degree

|- 2. communication

|- 3. 2 programming languages (c + java)

------------------------------------------------------------------------------

c-language

----------

what is c-language?

--> c is a programming language, high level language and procedure oriented

programming language.

programming language

|- programs

|- software/application

|- task

--> as c-language uses english words and it is user-friendly thats why c is

called as high level language.

what is a program?

--> a program is a set of instructions and a program is written to perform

some task.

c-program

---------

#include<stdio.h>

#include<conio.h>

int main() each line --> instruction

{

int a = 10;

int b = 20;

int c;

clrscr();

c = a + b;

printf("result = %d",c);

return 1;

}

computer languages

------------------

--> computer languages are divided into 2 types.

1. high level languages

2. low level languages

i. assembly language

ii. binary language (machine language)

--> the language which uses english words and which is user-friendly such

language is called as high level language.

ex: c, c++, java, python etc

--> the language which uses special symbols is called as assembly language.

--> the language which uses only 0's and 1's is called as binary language.

--> only machine understands binary language.

translators

-----------

what is translator?

--> translator is a software or a program which converts one language into

another language.

--> there are 3 types of translators

1. compiler

2. interpreter

3. assembler

--> compiler is a translator which converts high level language into binary

language and also reports errors (syntax errors) at compile time.

--> interpreter is a translator which converts high level language into

binary language line by line.

--> assembler is a translator which converts assembly language into binary

language.

note:

--> all the high level languages use either compiler or interpreter.

--> only assembly language uses assembler.

------------------------------------------------------------------------------

c-language --> procedure oriented programming language

what is procedure oriented programming (pop)?

--> procedure oriented programming is a programming paradigm.

note: a programming paradigm is also called as programming methodology or

programming technique or programming style.

paradigm means principles

principles of pop

-----------------

--> in procedure oriented programming the program is divided into multiple

sub tasks and every sub task is written inside the function (procedure).

note: the advantage of using function is reusability of code.

--> as c-language follows the principles of procedure oriented programming

paradigm thats why c is called as procedure oriented programming language.

java --> object oriented programming language

what is object oriented programming (oop)?

--> object oriented programming is a programming paradigm.

principles of oop

-----------------

1. class

2. object

3. data hiding

4. abstraction

5. encapsulation

6. inheritance

7. composition

8. polymorphism

--> as java follows the principles of object oriented programming paradigm

thats why java is called as object oriented programming language.

aspectj --> aspect oriented programming language

what is aspect oriented programming (aop)?

--> aspect oriented programming is a programming paradigm.

principles of aop

-----------------

1. aspect

2. advice

3. join point

4. point cut

5. target

6. weaving

7. proxy

--> as aspectj follows the principles of aspect oriented programming paradigm

thats why aspectj is called as aspect oriented programming language.

programming paradigms

---------------------

1. procedure oriented programming (pop)

2. object oriented programming (oop)

3. aspect oriented programming (aop)

------------------------------------------------------------------------------

1972

|

c-language --> dennis ritchie

secure operating system --> unix

------------------------------------------------------------------------------

number systems

--------------

what is number system?

--> number system is a technique or a system to represent the numbers in

the computer system.

--> there are 4 numbers systems

1. decimal number system

2. octal number system

3. hexadecimal number system

4. binary number system

decimal = 10

octal = 8

hexadecimal = 16

binary = 2

1. decimal number system

---------------------

base - 10

range - 0 to 9

ex: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 99, 100, 123, 1000, 12345 etc

2. octal number system

-------------------

base - 8

range - 0 to 7

ex: 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 22, 23,

24, 25, 26, 27, 30, 31 .......... 37, 40, 41 ....... 47, 50, 51 ... 57,

60, 61 ..... 67, 70, 71, 72 ........ 77, 100, 101, 102 ...... 107, 110,

111, 112, 113 ....... 117, 120, 121, 122 ........ 127, 130, 131 ......

137, 140, 141, 142 ....... 147, 150, 151, 152 ....... 157, 160, 161,

162 ....... 167, 170, 171 ....... 177, 200, 201, 202 ..... 207, 210,

211, 212 ...... 217, 220 ............................................

......................................................................

......................................................................

777, 1000, 1001 ..... 1007, 1010, 1011 ..... 1017, 1020, 1021 .......

1027, 1030 ..... 1037, 1040, 1041 ..... 1047 ............. etc

programming languages --> prefix

decimal number --> no prefix

octal number --> 0

hexadecimal number --> 0x or 0X

binary number --> 0b or 0B

10 --> decimal number

010 --> octal number

0x10 --> hexadecimal number

0b10 --> binary number

hexadecimal number system

-------------------------

base - 16

range - 0 to 15 (0 to 9 a b c d e f)

ex: 2a, 7d3, 9f3c, ab, 10, 45, 5FC2, 2A7fc etc

binary number system

--------------------

base - 2

range - 0 and 1

ex: 101, 100, 1010, 1000, 10110, 1110101 etc

-----------------------------------------------------------------------------

keywords (reserved words)

--------

--> c-language has grammar, the grammar of c-language is represented by some

words which are reserved or fixed in c-language for some purpose and every

word has some meaning attach with it which is known to c-compiler such

words are called as keywords.

--> in c, there are 32 keywords.

ex: if, else, switch, case, default, for, while, do, auto, extern, static,

void, break, continue, return, goto, signed, unsigned, int, float, char,

long, short, double, union, struct, typedef, enum etc

spoken language

|- english

|- telugu

|- hindi

english

|- grammar

|- english words (keywords)

|- reserved or fixed in english language only

|- meaning

|- person

telugu

|- grammar

|- telugu words (keywords)

|- reserved or fixed in telugu language only

|- meaning

|- person

programming languages

ex: c, c++, java, python etc

scripting languages

ex: javascript, php etc

query language

ex: sql

markup languages

ex: html, xml

c-language (programming language)

|- grammar

|- some words (keywords)

|- reserved or fixed in c-language for some purpose

|- meaning

|- c-compiler

javascript (scripting language)

|- grammar

|- some words (keywords)

|- reserved or fixed in javascript for some purpose

|- meaning

|- interpreter

sql (query language)

|- grammar

|- some words (keywords)

|- reserved or fixed in sql for some purpose

|- meaning

|- database

html (markup language)

|- grammar

|- html tags (keywords)

|- reserved or fixed in html for some purpose

|- meaning

|- browser

--> xml is the only language which has no grammar or keywords.

------------------------------------------------------------------------------

identifiers

-----------

what is an identifier?

--> any name in c-language is called as an identifier.

ex: variable name, function name, array name, pointer name, structure name etc

rules for writing identifiers

-----------------------------

1. a to z

A to Z

0 to 9

\_

2. an identifier cannot start with a digit

3. an identifier can start with any alphabet or underscore only

4. >31 characters

5. keywords cannot be used as identifiers

6. identifiers are case sensitive

7. predefined function names cannot be used as identifiers

ex:

1. num123

2. 4xyz //invalid

3. \_123xyz

4. abc@123 //invalid

5. scanf //invalid

6. switch //invalid

------------------------------------------------------------------------------

constants (literals)

---------

what is a constant?

--> a fixed value which can be assigned to a variable is called as constant

or literal.

ex: int a = 10;

here is 10 is constant or literal

types of constants

------------------

--> there are 2 types of constants

1. numeric constants

i. integer constants

a. decimal constants

b. octal constants

c. hexadecimal constants

ii. floating/real constants

2. non-numeric constants

i. character constants

ii. string constants

decimal constants

-----------------

base - 10

range - 0 to 9

ex:

1. int num = 2;

2. int num = 7;

3. int num = 9;

4. int num = 10;

5. int num = 99;

6. int num = 1000;

7. int num = 12345;

octal constants

---------------

base - 8

range - 0 to 7

prefix - 0

ex:

1. int num = 03;

2. int num = 06;

3. int num = 07;

4. int num = 08; //error

5. int num = 010;

6. int num = 0452;

7. int num = 07236;

hexadecimal constants

---------------------

base - 16

range - 0 to 15 (0 to 9 a b c d e f)

prefix - 0x or 0X

ex:

1. int num = 0x10;

2. int num = 0x58;

3. int num = 0xab;

4. int num = 0xf37d;

5. int num = 0xA2B8;

6. int num = 0xc4F2d;

------------------------------------------------------------------------------

memory units (memory sizes)

------------

1 bit = either 0 or 1

1 byte = 8 bits

1 kilo byte = 1024 bytes

1 mega byte = 1024 kb

1 giga byte = 1024 mb

1 tera byte = 1024 gb

1 peta byte = 1024 tb

1 exa byte = 1024 pb

1 zeta byte = 1024 eb

1 yotta byte = 1024 zb

1 bronto byte = 1024 yb

1 geo byte = 1024 bb

datatypes

|- int, long, float, double, char etc

--> every datatype in c-language has some size.

int --> 2 bytes

long --> 4 bytes

char --> 1 byte

float --> 4 bytes

double --> 8 bytes

laptop

|- processor --> intel i5 latest generation, amd ryzen-5

|- ram --> 8gb (16 or 32 gb expandable)

|- memory --> 1. hdd 2. ssd (500gb)

processor

|- cores 1 core means 1 cpu

note:

1. by default every integer value is int type in c-language.

2. to represent an integer value as long type we use a letter l or L as

a suffix.

10 --> int type

10L or 10l --> long type

--> in c-language, we have sizeof operator using which we can find the

size of any datatype.

3. by default every floating value is double type in c-language.

4. to represent a floating value as float type we use a letter f or F as

a suffix.

2.5 --> double type

2.5f or 2.5F --> float type

5. c-language is called as loose or weak type checking.

float a = 10; //10.0

int b = 2.5; //2

floating constants

------------------

ex:

1. float a = 3.5;

2. float a = 3.5f;

3. double a = 3.5;

4. double a = 3.5f;

5. float a = 10; //10.0

6. float a = 011; //9.0

7. float a = 0x10; //16.0

8. float a = 12e3; //12000.0

9. float a = 5E2; //500.0

------------------------------------------------------------------------------

computer

|- memory

|- 1. primary memory

ex: ram (current working memory)

|- 2. secondary memory

ex: hard disk

--> the data is stored in memory in the form of binary that is 0's and 1's.

---------------------------------------------------------------------------

character constants

-------------------

--> a single alphabet or a single digit or a single special character

or a single white space character which is enclosed with single

quotes is called as character constant.

ex:

1. char ch = 'a';

2. char ch = '5';

3. char ch = '$';

4. char ch = ' ';

5. char ch = 'ab'; //invalid

6. char ch = '25'; //invalid

7. char ch = '#$'; //invalid

8. char ch = 97; //a

9. char ch = 0142; //b

10. char ch = 0x63; //c

string constants

----------------

--> a group of characters which are enclosed within double quotes is

called as string constant.

ex: "hyderabad", "abc123$#", "ravi kumar"

---------------------------------------------------------------------------

variables

---------

what is a variable?

--> a variable is a name or an identifier which is given to the memory

location internally.

--> variables are used in the program to store some data and using varia-

bles we access the data in the program to perform operations on the

data.

--> variable is a container to hold or store some data.

--> variable acts as place holder in the program.

--> any time any place in the program we can change or modify the value

of the variable in the program so variables are varying in nature.

note: before using variables in the program we must have to declare the

variable.

declaring the variable

----------------------

--> specifying the datatype of the variable is called as declaring the

variable.

syntax: datatype variable\_name = <initialization>;

here initialization is optional

ex:

1. int num;

2. float area;

3. char ch;

4. double amount;

5. long fact;

initialization of variable

--------------------------

--> assigning or storing the value to the variable is called as initiali-

zation of variable.

ex: int a; (or) int a = 10;

a = 10;

---------------------------------------------------------------------------

datatypes

---------

what are datatypes?

--> datatypes are keywords and datatypes are used to specify two things.

1. type of data

2. size of data

--> datatypes are used to specify the type of data a variable should

hold or store.

--> datatypes are used to specify the size of memory should be allocated

for the data.

types of datatypes

------------------

--> there are 2 types of datatypes

1. primary datatypes

i. primitive datatypes

ex: short, int, long, float, double, char etc

2. secondary datatypes

i. derived datatypes

ex: arrays, functions, pointers

ii. userdefined datatypes

ex: structure, union, enum

--> primitive datatypes allow a variable to store single value.

--> derived datataypes and userdefined datatypes allow a variable to store

multiple values.

--> primitive datatypes are divided into 3 types

1. integer datatypes

ex: signed short or short, unsigned short,

signed int or int, unsigned int, signed long

or long and unsigned long

2. floating datatypes

ex: float, double

3. character datatypes

ex: signed char or char, unsigned char

Integer Datatypes

------------------------------------------------------

datatype size (bytes) range

------------------------------------------------------

1. signed short 2 -32768 to 32767

or

short

2. unsigned short 2 0 to 65535

3. signed int 2 or 4 -32768 to 32767

or

int

4. unsigned int 2 or 4 0 to 65535

5. signed long 4 -2,147,483,648 to

or 2,147,483,647

long

6. unsigned long 4 0 to 4,294,967,295

-------------------------------------------------------

--> signed and unsigned are keywords in c-language.

--> signed keyword represents both +ve and -ve values where

as unsigned represents only +ve values.

--> int size --> 2 or 4 bytes

70's decade -> 1972 -> computers -> micro processor

|

16 bit

16 bit compiler --> int size is 2 bytes

32 bit compiler --> int size is 4 bytes

Floating Datatypes

---------------------------------------------------

datatype size range

---------------------------------------------------

1. float 4 -3.4e38 to 3.4e38

2. double 8 -1.7e308 to 1.7e308

---------------------------------------------------

--> float is 32 bit floating point datatype.

--> double is 64 bit floating point datatype.

--> float uses 6 digits of precision.

--> double uses 14 digits of precision.

123.45678

|

precision

Character Datatypes

---------------------------------------------------

datatype size range

---------------------------------------------------

1. signed char 1 -128 to 127

or

char

2. unsigned char 1 0 to 255

---------------------------------------------------

in c-language --> char --> 1 byte

in java --> char --> 2 bytes

c-language

|- ascii --> 0 to 255 (1 byte)

java

|- unicode --> 0 to 65535 (2 bytes)

----------------------------------------------------------------------------

void --> it is a keyword and it used as return type of the function which

indicates the function will not return any value.

---------------------------------------------------------------------------

functions

---------

what is a function?

--> a function is a reusable block of statements and a function is used

to perform some operation.

syntax: return-type function\_name(parameters/arguments)

{

statement(s);

return statement;

}

return type

-----------

--> before the function name if we write any datatype or void keyword then

it is called as return type of the function.

--> in c-language, a function may return some value or may not return some

value so the return type of the function indicates whether the function

returns a value or not.

--> the function whose return type is void such function never return any

value.

--> the function whose return type is other than void that is some datatype

then such function returns a value.

--> return type of the function is mandatory.

int f1()

{

return 1;

}

float f2()

{

return 2.5;

}

void f3()

{

return 1; //error

}

int, float, char, double, long, short --> datatypes

return statement

----------------

--> return statement is used to return the value to the function whose

return type is other than void.

--> we cannot use return statement to return the value to the function

whose return type is void otherwise compiler gives error.

--> return statement is optional, it depends on the return type of the

function.

--> return is a keyword in c-language and we use return keyword as return

statement to the value to the function.

parameters (arguments)

----------

what are parameters?

--> if we write any variables inside the parenthesis of the function then

such variables are called as parameters or arguments.

--> function uses parameters to pass the data and to receive the data.

--> parameters or arguments are optional.

types of parameters

-------------------

--> there are 2 types of parameters

1. actual parameters

2. formal parameters

--> function uses actual parameters to pass the data to formal parameters

and function uses formal parameters to receive the data from actual

parameters.

int x, y; //here x and y are variables

void add(int a, int b) //a and b are parameters or arguments

{

}

function body (implementation)

-------------

--> function body contains statements.

--> function body is also called as implementation.

--> function body is mandatory.

function name

-------------

--> function must have name and ends with parenthesis.

---------------------------------------------------------------------------

different sections of the function

----------------------------------

--> function has 2 sections

1. function header or function prototype

2. function body or implementation

--> function header or function prototype consists of function name, return

type and parameters or arguments.

ex: void show();

int add(int a, int b);

--> function body consists of statements.

ex: void show()

{

statement(s);

}

types of functions

------------------

--> there are 2 types of functions

1. predefined functions (built-in functions or library functions)

2. userdefined functions

--> the functions which are already developed and available in c-language

are called as predefined functions.

ex: printf(), scanf(), clrscr(), getch(), gets(), puts(), strcpy() etc

--> the functions which are developed by the programmer are called as

userdefined functions.

ex: show(), add(), substract(), multiply(), display(), calculate() etc

---------------------------------------------------------------------------

Note: if we want to work with userdefined functions in c-language then we

must have to write 3 things

1. function declaration

2. function definition

3. function caller

function declaration

--------------------

--> writing the name of the function, return type of the function and

parameters of the function is called as function declaration.

--------------

void show(); //function declaration

--------------

function definition

-------------------

--> writing the function body is called as function definition.

---------------

void show() //function definition

{

}

---------------

write the program -> compile the program -> run or execute the program

note: execution of every c-program always starts from main function so

operating system calls only main function but not userdefined

functions.

--> to call userdefined function we must have to call userdefined

function from main function by using function caller.

function caller

---------------

--> using function caller we can call userdefined function from other

function.

--> to write function caller we use function name.

---------

show(); //function caller

---------

what is calling function?

--> a function which calls other function is called as calling function.

what is called function?

--> a function which is called by calling function is called called function.

---------------------------------------------------------------------------

basic predefined functions in c :

-------------------------------

1. printf() :

--------

--> it is used to display the value to the standard output device (monitor).

syntax:

printf("control string with format specifiers",arg-1,arg-2,arg-3...arg-n);

2. scanf() :

-------

--> it is used to read inputs (values) from the standard input device

(keyboard) at runtime.

syntax:

scanf("format specifiers",&arg-1, &arg-2 .... &arg-n);

& ampersand symbol is called as address of operator in c-language

3. clrscr() :

--------

--> it is used to clear the output screen.

syntax: clrscr();

4. getch() :

-------

--> it is used to read a character from keyboard at runtime, till it

reads a character from keyboard it holds the output screen and if

we enter a character then it releases the output screen.

syntax: getch();

---------------------------------------------------------------------------

ways of writing printf()

------------------------

syntax:

printf("control string with format specifiers",arg-1,arg-2...arg-n);

--> control string, format specifiers and arguments are optional

--> double quotes are mandatory

1. printf(); //error

printf("");

printf("welcome"); //welcome

what is control string?

--> in printf() function inside the double quotes everything which is not

format specifier is called as control string and control string is

displayed as it is.

2. int a = 10;

printf("a"); //a

printf("%d",a); //10

format specifiers

-----------------

--> format specifiers start with % symbol.

--> format specifiers are used for input and output purpose.

input means reading values

output means displaying values

--> format specifiers are used to specify printf() function to display

specific type of values.

--> format specifiers are used to specify scanf() function to read specific

type of values from keyboard at runtime.

--> format specifiers act as place holders in printf() and scanf() functions.

1. int, short --> %d

2. long --> %ld

3. float --> %f

4. double --> %lf

5. char --> %c

6. string --> %s

7. unsigned --> %u

8. octal --> %o

9. hexadecimal --> %x

---------------------------------------------------------------------------

3. int a = 10;

output: a = 10

printf("a = %d",a); //a = 10

4. int a = 10, b = 20;

output: a = 10 and b = 20

printf("a = %d and b = %d",a,b); //a = 10 and b = 20

5. int id = 1;

char gender = 'm';

float salary = 1000.0;

output: id = 1, gender = m, salary = 1000.0

printf("id = %d, gender = %c, salary = %f",id,gender,salary);

6. int n = 5, i = 4;

output: 5 x 4 = 20

printf("%d x %d = %d",n,i,n\*i);

7. "

printf("""); //error: unterminated string

printf("\""); "

8. ""

printf("\"\""); ""

9. \

printf("\"); //error: unterminated string

printf("\\"); \

10. \\

printf("\\\\"); \\

11. \"\"

printf("\\\"\\\"");

12. int a = 10;

output: a = "10"

printf("a = \"%d\"",a);

--------------------------------------------------------------------------

escape sequences (escape sequence characters)

----------------

--> escape sequences or escape sequence characters start with \

--> in c-language, we have fixed set of escape sequences or escape

sequence characters.

\n - new line

\t - tabspace

\b - backspace

\r - carriage return

\a - alarm

\" - "

\\ - \

\' - '

Note: all the escape sequences or escape sequence characters can be used

as character constants in c-language.

ex:

char ch = 'a';

char ch = '3';

char ch = '$';

char ch = ' ';

char ch = '\n';

char ch = '\b';

char ch = '\r';

char ch = '\\';

char ch = '\"';

char ch = '\'';

char ch = '\x'; //invalid

---------------------------------------------------------------------------

structure of c-program

----------------------

--> c-program has a structure which consists of 5 sections.

section-1: comments

section-2: preprocessor directives

section-3: global declaration

section-4: void main()

{

}

section-5: sub-routine functions (userdefined functions)

comments

--------

what are comments?

--> comments are the non-executable statements which are present in the

program.

--> comments are used to write documentation in the program.

--> comments make the program more readable and easy to understand.

types of comments

-----------------

--> there are 2 types of comments

1. single line comments

//this is single line comment

2. multi line comments

/\*

this is multi line comment

\*/

preprocessor directives

-----------------------

--> preprocessor directives start with # symbol.

ex: #include, #define, #undef etc

--> preprocessor directives are used to perform preprocessing which

means before the source code (c-program) is given to c-compiler

for compilation preprocessor directives modify the source code

and gives modified source code to c-compiler for compilation.

header files :

------------

what is a header file?

--> a file which ends with .h extension is called as header file.

--> header files contain function declaration and function definition code

of predefined functions.

--> header files are called as c-library.

ex:

stdio.h --> printf(), scanf()

conio.h --> clrscr(), getch()

math.h --> sqrt(), pow(), sin(), cos() etc

string.h --> strlen(), strcpy(), strrev() etc

#include : (file inclusion directive)

--------

--> it is a preprocessor directive, it copies function declaration and

function definition code of predefined functions from header files

into the c-program before the c-program is given to c-compiler for

compilation.

ex: #include<stdio.h>

#include<conio.h>

#include<string.h>

#define :

-------

--> it is a preprocessor directive and it is used to

create or define userdefined macros.

syntax: #define MACRO\_NAME code

#undef :

------

--> it is a preprocessor directive and it used to

undefine a macro.

syntax: #undef MACRO\_NAME

what is a macro?

--> a macro is a name which is given to the piece of code.

--> macro's are constants which means once we define or create

a macro in next line we cannot modify.

types of macros :

---------------

there are 2 types of macros

1. predefined macros

ex: \_\_DATE\_\_ , \_\_TIME\_\_ , \_\_FILE\_\_ , \_\_LINE\_\_

2. userdefined macros

i. variable like macros

ii. function like macros

--------------------------------------------------------

local declaration :

-----------------

--> anything which is declared inside the function is called as

local declaration.

global declaration :

------------------

--> anything which is declared outside all the function is called

as global declaration.

local variable :

--------------

--> a variable which is declared inside the function is called as

local variable.

global variable :

---------------

--> a variable which is declared outside all the functions is called

as global variable.

int b = 20; //global declaration, global variable

f1()

{

int a = 10; //local declaration, local variable

printf("%d",a); //10

printf("%d",b); //20

}

f2()

{

printf("%d",a); //error

printf("%d",b); //20

}

scope --> accessibility and visibility

lifetime --> creation and destruction

--> both local variable and global variable are having

scope and lifetime.

--> the scope of local variable is within the function

but not outside the function.

--> the scope of global variable is through out the

program.

--> local variables are created when function execution

starts and destroyed when function execution is

completed this is called lifetime of local variables.

--> global variables are created when program execution

starts and destroyed when program execution is com-

pleted this is called lifetime of global variable.

--> execution of every c-program always starts from

main function so main function is the entry point

from where execution starts.

--> apart from main function we can write userdefined

functions to perform some operations such userdefined

functions are called as sub-routine functions.

note: section-1, section-3 and section-5 are optional

but section-2 and section-4 are mandatory.

writing first c-program :

-----------------------

IDE (Integrated Development Environment)

IDE --> software

ex: turboc++, DevC++, eclipse, code blocks, falcon c++ etc

program: write the c-program to perform addition of two numbers.

---------------------------------------------------------------------------

operators

---------

what is an operator?

--> an operator is a symbol which is used to perform some operations on

operands (variables).

ex: a + b

here + symbol is operator and a, b are operands (variables)

--> in c-language, 47 operators are there.

--> all the 47 operators are divided into 3 categories

1. unary category operators

--> this category operators use single operand to perform operations.

ex: ++ -- & ! ~ sizeof typecast etc

2. binary category operators

--> this category operators use two operands to perform operations.

ex: + - \* / % < > <= >= != == & | && || etc

3. ternary category operators

--> this category operators use three operands to perform operations.

ex: ?: (conditional operator)

--> all 47 operators are having precedence and associativity.

what is precedence of operator?

--> for every operator some rank is given which is called as

precedence and precedence is used to decide which operator

should be given first preference or priority.

ex: a = 2 + 5 \* 4; --> a = 2 + 20; --> a = 22;

what is associativity of operator?

--> an operator should be evaluated in some order and that order

is called as associativity.

--> associativity can be either left to right or right to left.

ex: i. 2 + 5 left to right

ii. a = 10; right to left

iii. a = 7 - 3 + 4; --> 4 + 4 --> 8

iv. a = 7 + 4 - 3; --> 11 - 3 --> 8

precedence means priority or preference

associativity means order

operators precedence table

|- 47 operators

|- precedence

|- associativity

---------------------------------------------------------------------------------

1. ++ and --

2. parenthesis

3. + - \* / %

4. unary -

5. unary +

6. assignment = += -= \*= /= %=

7. < > <= >= == !=

8. && || !

9. ?:

10. sizeof

11. type cast (type)

12. address of &

13. subscript []

14. value at address \*

[] subscript operator ---> arrays

. member operator ---> structure

-> pointer operator ---> structure with pointer

& | ^ ~ << >> ---> bitwise operators (c and java)

control flow statements (c and java)

functions (methods)

arrays (c and java)

string (c and java)

------------------------------------------------------------------------------

Java

|- 1. core java

|- 2. advance java

|- 3. frameworks

i. hibernate

ii. spring

|- 4. realtime tools

i. junit ii. log4j iii. git and github iv. maven etc

fresher --> core java, advance java

frameworks --> 20% to 30% knowledge --> out of 100 --> 3 or 4

experience --> core java, advance java, frameworks, realtime tools

frameworks --> workshop

sql, html, css, javascript, database, eclipse

------------------------------------------------------------------------------

core java

---------

what is java?

--> java is a programming language, high level language and object oriented

programming language.

programming language --> programs --> software or application --> task

--> a program is a set of instructions which is written to perform some

task or operation.

--> as java uses english words and it is user-friendly thats why java is

called as high level language.

--> java follows all the principles of object oriented programming paradigm

thats why java is called as object oriented programming language.

computer languages

|- 1. high level languages

|- 2. low lever languages

i. assembly language

ii. binary language (machine language)

--> the language which uses english words and which is user-friendly is

called as high level language.

ex: c, c++, java, python etc

what is object oriented programming?

--> object oriented programming is a programming paradigm.

paradigm means principles

--> programming paradigm is also called as programming methodology or prog-

ramming style or programming technique.

principles of object oriented programming (oop)

-----------------------------------------

1. class

2. object

3. data hiding

4. abstraction

5. encapsulation

6. inheritance (is-a relationship)

7. composition (has-a relationship)

8. polymorphism

--> if a language follows all the principles of object oriented programming

paradigm such language is called as object oriented programming language.

programming paradigms

|- 1. procedure oriented programming (pop)

|- 2. object oriented programming (oop)

|- 3. aspect oriented programming (aop)

Java --> Technology --> Sun Microsystems (Company)

Open Source Technology

--> every technology has api, as java is a technology so java has api.

what is api?

--> api stands for application programming interface

--> the predefined classes and interfaces which are developed and provided by

sun microsystems in java is called as api.

--> using api (predefined classes and interfaces) we write java programs to

develop or build software or application.

c-language

|- programs

|- functions

1. predefined functions --> api

2. userdefined functions

in c-language, api means predefined functions

java

|- programs

|- classes and interfaces

1. predefined classes and interfaces --> api

2. userdefined classes and interfaces

in java, api means predefined classes and interfaces

Java Editions/Java Platforms

----------------------------

1. JSE (Java Standard Edition)

2. JEE (Java Enterprise Edition)

3. JME (Java Micro/Mobile Edition)

--> sun microsystems has given different java editions or java platforms to

develop different types of applications.

types of applications

---------------------

1. stand-alone or desktop applications

i. gui applications

ii. cui applications

2. web applications

3. enterprise web applications

4. distributed applications

5. mobile applications

6. embedded systems

network concepts

|- network

|- advantages

|- types of networks

|- network topologies

|- types of network architectures

|- client and server applications

|- ip address

|- protocol

stand-alone or desktop applications :

-----------------------------------

--> the applications which runs locally on the client machine such

applications are called as stand-alone or desktop applications.

--> stand-alone or desktop applications are single user applications.

--> stand-alone or desktop applications are 2 types

i. gui applications

ii. cui applications

--> to use gui applications we no need to use any commands but to use cui

applications we must have to use commands.

--> gui applications are more user-friendly than cui applications.

--> gui applications provide graphical user interface where as cui

applications provide command line interface.

ex: text editors, calculator, adobe reader, media players, command prompt etc

web applications :

----------------

--> the applications which runs inside the web server such applications

are called as web applications.

--> web applications are multi-user applications.

ex: gmail, online tutorial websites etc

enterprise web applications :

---------------------------

--> the large web applications which are developed for business purpose

such applications are called as enterprise web applications.

ex: e-commerce websites, banking websites, ticket booking websites etc

distributed applications :

------------------------

--> two different applications which are running on two different servers

and which can share their business services over the network to each

other such applications are called as distributed applications.

ex: irctc.co.in and makemytrip.com

mobile applications :

-------------------

--> the applications which are developed to run on mobile devices are

called as mobile applications.

ex: mobile apps

embedded systems :

----------------

--> embedded systems is a combination of hardware and software, in embedded

systems the software is embedded inside the hardware.

ex: computer, television, set top box etc

JSE --> stand-alone or desktop applications (gui and cui applications)

JEE --> web applications, enterprise web applications, distributed applications

JME --> mobile applications, embedded systems

client side programming --> jse

server side programming --> jee

micro programming --> jme

------------------------------------------------------------------------------

features of java (java buzzwords)

----------------

1. simple

2. object oriented

3. robust

4. platform independent

5. portable

6. architectural neutral

7. secure

8. multi-threaded

9. distributed

10. high performance

11. interpreted

12. dynamic

simple

------

i. java follows the similar syntaxes of c and c++ languages

thats why learning java is simple.

ii. all the confusing concepts of c and c++ languages like

pointers, multiple inheritance and goto statement are

eliminated from java to make java as simple.

object oriented

---------------

--> java follows the principles of object oriented programming paradigm

so in java everything is represented as an object thats why java is

called as object oriented.

robust (strong)

------

i. java has better memory management mechanism

ii. java has better exception handling mechanism

--> in java, programmer is responsible for allocating the memory but

programmer is not responsible for deallocating the memory, memory

deallocation is automatically performed by garbage collector thats

why java has better memory management mechanism.

--> in java, sun microsystems has provided a rich api (predefined

classes and interfaces) to work with exception handling thats

why java has better exception handling mechanism.

what is memory management?

--> allocating and deallocating the memory is called as memory management.

types of memory management

--------------------------

--> there are 2 types of memory management

1. static memory allocation

--> allocating the memory at the time of writing the program before

compilation is called as static memory allocation.

ex: declaring variable, declaring array etc

2. dynamic memory allocation

--> allocating the memory at the time of running the program at

runtime is called as dynamic memory allocation.

ex: malloc(), calloc(), realloc(), free()

platform independent :

--------------------

-> after compiling the java program, java compiler generates class file

with byte code, class file is not platform dependent thats we can execute

the class file with byte code on any machine having any operating system

and any processor but on every machine jvm must be available this is

called as wora (write once and run anywhere) and thats why java is called

as platform independent.

portable :

--------

--> after compilation java compiler generates class file (.class file) which

we can carry on any operating system this is why java is portable.

Architectural Neutral :

---------------------

--> the size of int primitive datatype is fixed in java that is 4 bytes

irrespective of microprocessor architecture thats why java is called

as architectural neutral.

Secure :

------

--> before executing the java program, jvm loads the class which contains

byte code into jvm's method area then byte code verification will be

done to check whether the byte code which is loaded is valid or invalid,

if the byte code is valid then it will be executed by jvm inside jre and

if the byte code is invalid then jvm terminates the execution of the

program by giving an error this is why java is secure.

Multi-threaded :

--------------

--> In java, sun microsystems has provided a rich api (predefined clasess

and interfaces) to develop multi threaded applications thats why java

is called as multi-threaded.

--> Developing multi-threaded applications is easy in java because of api.

Distributed :

-----------

--> sun microsystems has provided several distributed technologies in java

like corba, rmi, ejb etc to develop distributed applications thats why

java is called as distributed.

High Performance :

----------------

--> In java along with interpreter sun microsystems has provided a special

compiler called as JIT compiler (Just In Time compiler) to improve the

performance thats why java is high performance.

Dynamic :

-------

--> in java, classloaders load the classes dynamically at runtime on demand

thats why java is called as dynamic.

Interpreted :

-----------

--> as java uses interpreter thats why java is called as interpreted.

------------------------------------------------------------------------------

what is class?

--> anything which does not exist physically which is imagination about an

object is called as class.

--> as a class is imagination about an object thats why a class will not

occupy any space or memory.

--> a class is a blueprint for creating the object.

--> a class represents the structure of the object.

--> a class is a logical form of the object.

--> a class contains variables and methods which are called as members of

the class.

--> variables are used to hold data thats why variables are called as data

members of the class and methods use data members to perform operations

on data thats why methods are called as member methods of the class.

--> a class will not have state and behaviours but a class represents the

state and behaviours of the object.

--> a class is a userdefined datatype in java so creating a class means

creating userdefined datatype and userdefined datatype allows a varia-

ble to store multiple heterogeneous or homogeneous.

what is object?

--> anything which exists physically in this real world is called as an object.

--> as an object is existing physically so an object will occupy memory or

space.

--> an object is the physical form of the class.

--> an object is called as an instance of the class.

instance means physical form

--> object has state and behaviours, what an object posses or owns is called

as state of the object and using the state an object performs operations

such operations are called as behaviours of the object.

--> in java, variables represent state and methods represent behaviours.

note:

--> we can create class without creating the object but we cannot create

object without creating the class so to create object class is mandatory.

---------------------------------------------------------------------------------

jdk 1.8 --> download --> http://www.oracle.com

javac --> javac.exe or java compiler or development tool

C:\Program Files\Java\Jdk1.8\bin\javac.exe

what is path?

--> path is an environment variable or system variable which is used by operating

system to search for the location of executable files.

note: by default operating system searches for executable files in the current

directory location, if executable files are not available in the current

directory location then operating system searches of executable files in

path which is an environment variable or system variables thats why we

have to set path.

ways to set the path

--------------------

--> there are 2 ways to set the path

1. command prompt level

--> if we set path at command prompt level then path will be available until we

close the command prompt, if we close and open command prompt then we have

to set the path again.

syntax: set path=location\_of\_executable\_files

2. operating system level

--> if we set path at operating system level then path becomes permanent.

------------------------------------------------------------------------------

--> set is a command which is used to set the path at command prompt level.

writing first java program

--------------------------

--> to write the java program 3 things are required.

1. Jdk software

2. IDE (Integrated Development Environment) --> Text Editor

3. Workspace

ide stands for integrated development environment --> software

java ide's --> eclipse, netbeans, myeclipse, jdeveloper, intellij etc

--> as a beginner in java and to learn internals use text editor instead of ide.

text editor's --> notepad, notepad++, editplus, wordpad, atom etc

--> the directory where we place the java programs is called as workspace.

--> a java program extension must be .java.

--> the file which ends with .java extension is called as java file.

comments

--------

--> comments are the non-executable statements present in the java program.

--> comments are used to write the documentation in the program.

--> comments make the program more readable and easy to understand.

types of comments

-----------------

--> there are 3 types of comments

1. single line comments

ex: //this is single line comment

2. multi line comments

ex: /\*

this is multi line comments

\*/

3. documentation comments

ex: /\*\*

this is documentation comments

\*/

why we use documentation comments?

--> if we want to create our own api document then we have to use

documentation comments.

api -> predefined classes & interfaces -> to write programs -> software/application

predefined classes & interfaces (api) -> rt.jar (class libraries)

api document

------------

--> sun microsystems has developed api document and api document

contains only the names of predefined classes and interfaces.

--> predefined classes and interfaces are physically available in

rt.jar file.

--> to know the names of predefined classes and interfaces we have

to use api document.

Java

|- Core Java --> JSE api document

|- Advance Java --> JEE api document

--> in c-language, to write the programs we use functions so writing the

function means writing the c-program

--> in java, to write programs we use classes so writing the class means

writing the java program

creating the class

------------------

syntax: modifier class ClassName

{

//variables and methods

}

--> here modifier, variables and methods are optional.

--> class keyword is used to create the class in java.

--> class name and the file name must be same.

--> execution of every java program always starts from main method.

--> in java, main method has standard signature.

public static void main(String args[])

--> in java, class name first letter should be capital.

--> String and System are predefined classes in java.

--> sun microsystems has placed String and System in java.lang package.

--> instead of using fully qualified name of the class name we can import

the package.

-------------------------------------------------------------------------

developer-1

-----------

package pkg1;

class Employee pkg1.Employee

{

}

developer-2

-----------

package pkg2; pkg2.Employee

class Employee

{ naming collision or naming conflict problem

}

developer-3

-----------

package pkg3; pkg3.Employee

class Employee

{

}

developer-4

-----------

package pack1.pack2;

class Employee pack1.pack2.Employee

{

}

-------------------------------------------------------------------------

com.projectname.modulename.submodulename.ClassName

--> if multiple developers create multiple classes with

the same name then we get naming collision or naming

conflict problem.

--> the solution for naming collision or naming conflict is

make the class names unique.

--> to make the class names unique sun microsystems has

provided packages.

--> package is used to make the class name unique.

--> package keyword is used to create the package in java.

syntax: package package-name;

--> package inside the package is called as sub package.

syntax: package package-name.sub-package-name;

--> when we create the class, first we should create the

package then create the class and place the class inside

the package.

--> if the class is placed inside the package then we

cannot use the class directly by its name, to use the

class which is placed inside the package we must have

to use fully qualified name of the class name.

--> fully qualified name of the class name means

package-name.ClassName

--> instead of using fully qualified name of the class name

we can import the package.

what is import?

--> import is a keyword and it is used to import the classes

and interfaces of a package in the java program.

--> if we import the package then we no need to use fully qualified

name of the class name, we can use the class name directly.

syntax: import package-name.sub-package-name.ClassName;

import package-name.sub-package-name.\*;

note: java.lang is the default package in every java program.

--> if we dont import any package in the java program then java

compiler automatically import java.lang package.

--> javac and java are commands or tools.

--> to run java compiler we have to use javac command or tool.

syntax: javac ClassName.java

--> after compilation java compiler generates class file (.class file)

with byte code.

--> byte code is special code only jvm understands the byte code.

--> to run jvm we have to use java command or tool.

syntax: java ClassName

--> the above commands we have to run in command prompt.

------------------------------------------------------------------------------

modifiers :

---------

what are modifiers?

--> modifiers are keywords and modifiers are used to give some instructions

to compiler and jvm.

ex: public, private, protected, default, synchronized, static,

abstract, final, transient, volatile, native, strictfp

types of modifiers :

------------------

--> there are 2 types of modifiers

1. access modifiers

ex: public, private, protected, default

2. non-access modifiers

ex: static, final, abstract, synchronized,

transient, volatile, strictfp, native

scope means accessibility and visibility

--> access modifiers are used to specify the scope of classes, interfaces,

methods and variables.

ex: class A

{

private void m1() {

}

}

class B

{

public void m2() {

}

}

--> non-access modifiers are used to give some special instructions to

compiler and jvm.

ex-1: final int a = 10; //constant variable

a = 20; //error

ex-2: class One

{

void methodOne() {

}

}

--> in java, to call the method of the class, object is required.

One obj = new One();

obj.methodOne();

--> if we want to call the method of the class without creating

the object then declare the method using static keyword.

class One

{

static void methodOne() { //static method

}

}

One.methodOne();

--> to call the static method of the class, object is not required, we

can call the static method using Classname directly.

--> jvm calls main method.

public static void main(String args[]) :

--------------------------------------

public :

------

--> public is an access modifier.

--> jvm calls main method so main method must be accessible and

visible to jvm thats why main method is public.

static :

------

--> static is a non-access modifier.

--> jvm calls main method without creating the object that is

by using classname internally thats why main method static.

void :

----

--> void is a keyword and void is the return type of main method

which indicates main method will not return any value.

main() :

------

--> it is the name of the main method from where execution of the

java program starts.

String args[] :

-------------

--> main method has a parameter whose name is args, whose type

is String[] and main method uses args parameter to receive

the command line arguments.

Note:

--> jvm always looks for main method as :

public static void main(String args[])

second java program

|- write the java program to perform addition of two numbers.

string concatenation operator +

-----------------------------

--> if atleast one operand or one value is string type then + acts as string

concatenation operator and the result of string concatenation is string.

--> anything + string is string or string + anything is string.

ex:

i. "welcome" + 10 = welcome10

ii. 20 + "hello" = 20hello

iii. "data" + "base" = database

iv. 10 + 20 = 30 //addition

-------------------------------------------------------------------------------------

java naming conventions

-----------------------

--> java naming conventions are the rules given by sun microsystems for

writing the class name, interface name, method name, variable name,

package name, constant variable name.

1. java naming conventions for class names and interface names

ex: String, Thread, Connection, Driver, StringBuffer, ArrayList, ResultSet,

PreparedStatement, InputStreamReader, ObjectOutputStream etc

2. java naming conventions for method names

ex: read(), equals(), readLine(), charAt(), compareToIgnoreCase() etc

3. java naming conventions for variable names

ex: id, age, city, firstName, lastName, addressLine1, dateOfBirth etc

4. java naming conventions for constant variable names

ex: SIZE, BUFFER, PI, MONTH, YEAR, MAX\_PRIORITY, DAY\_OF\_MONTH etc

5. java naming conventions for package names

ex: java.lang, java.util, java.sql, java.io, java.util.concurrent etc

6. java naming conventions for keywords

ex: if, else, class, import, interface, extends, implements, final,

break, continue, static, public, private, protected, for, do etc

keywords (reserved words)

--------

--> java has grammar, the grammar of java is represented by some words

which are reserved or fixed in java for some purpose and every word

has some meaning attach with it which is known to java compiler such

words are called as keywords.

--> in java, 49 keywords are there.

ex: class, interface, package, import, extends, implements, public, if

private, protected, break, continue, return, final, static, abstract,

void, try, catch, finally, throw, throws, native, this, super, else,

while, do, for, int, byte, short, long, float, double, boolean etc

identifiers

-----------

--> any name in java is called as identifier.

ex: class name, interface name, method name, variable name, package name,

array name etc

rules for writing identifiers

-----------------------------

1. a to z

A to Z

0 to 9

\_

$

2. an identifier cannot start with a digit

3. an identifier can start with alphabet or underscore or dollar

4. no length limit

5. keywords cannot be used as identifiers

6. identifiers are case sensitive

7. predefined class names and interface names can be used as identifiers,

it is valid but not recommended to use.

ex:

1. value123

2. abc#123 //error

3. System //valid but not recommended

4. $123abc

5. 7xyz //error

6. \_abc987

7. for //error

8. xyz$234

variable

--------

what is a variable?

--> a variable is a name or an identifier which is given to the memory

location internally.

--> variables are used in the program to hold or store some data and

using variables we access the data in program to perform opera-

tions on the data.

--> variable is a container to hold some data.

--> variables act as place holders in the program.

--> any time any place in the program we can change or modify the value

of the variable.

declaration of the variable

---------------------------

--> specifying the datatype of the variable is called as declaring the

variable.

syntax: datatype variableName = <initialization>;

here initialization is optional

ex:

1. int num;

2. float value;

3. char ch;

4. String str;

initialization of the variable

------------------------------

--> assigning or storing value to the variable is called as initialization.

ex: int a; --> int a = 10;

a = 10;

literals (constants)

--------

what is a literal?

--> a fixed value which can be assigned to the variable is called as

literal or constant.

ex: int a = 10;

here 10 is a literal or constant

types of literals

-----------------

--> there are 3 types of literals

1. numeric literals

i. integer literals

a. decimal literals

b. octal literals

c. hexadecimal literals

d. binary literals

ii. floating literals

2. non-numeric literals

i. character literals

ii. string literals

3. boolean literals

decimal literals

----------------

base - 10

range - 0 to 9

prefix - no prefix

ex:

1. int num = 3;

2. int num = 5;

3. int num = 9;

4. int num = 10;

5. int num = 123;

6. int num = 1000;

7. int num = 94562;

octal literals

--------------

base - 8

range - 0 to 7

prefix - 0

ex:

1. int num = 04;

2. int num = 07;

3. int num = 010;

4. int num = 0543;

5. int num = 07126;

6. int num = 038; //error

hexadecimal literals

--------------------

base - 16

range - 0 to 15 (0 to 9 a b c d e f)

prefix - 0x or 0X

ex:

1. int num = 0x10;

2. int num = 0x4f;

3. int num = 0xab3d;

4. int num = 0x7D3cF;

binary literals

---------------

base - 2

range - 0 and 1

prefix - 0b or 0B

ex:

1. int num = 0b10;

2. int num = 0b0101;

3. int num = 0b1010;

4. int num = 0B1011;

note:

1. by default every integer number or integer value is int type in java.

2. to represent an integer number or integer value as long type we use a

letter l or L as a suffix.

100 --> int type

100L or 100l --> long type

3. by default every floating number or value is double type in java.

4. to represent a floating number or value as double type we use a letter

f or F as a suffix.

2.5 --> double type

2.5f or 2.5F --> float type

5. java is strict or strong type checking language.

double a = 10;

System.out.println(a); //10.0

int b = 2.5; //error

System.out.println(b);

floating literals

-----------------

1. float a = 4.5; //error

2. float a = 4.5f;

3. double a = 4.5;

4. double a = 4.5f;

5. double a = 10; //10.0

6. double a = 13e3; //13000.0

character literals

------------------

--> a single alphabet or a single digit or a single special character or

a single white space character which is enclosed within single quotes

is called as character literal.

ex:

1. char ch = 'a';

2. char ch = '5';

3. char ch = '$';

4. char ch = ' ';

5. char ch = 'ab'; //error

6. char ch = '10'; //error

7. char ch = '@#'; //error

8. char ch = 97; //a

note: to represent unicode value sun microsystems has provided a unicode

character literal as '\uxxxx' where x can be any hexadecimal value.

9. char ch = '\u0061'; //a

escape sequences or escape sequence characters

----------------------------------------------

\n --> new line

\t --> tabspace

\r --> carriage return

\b --> backspace

\a --> alarm

\" --> "

\\ --> \

\' --> '

note: escape sequences or escape sequence characters can be used as

character literals in java.

10. char ch = '\n';

11. char ch = '\t';

12. char ch = '\b';

13. char ch = '\x'; //error

string literals

---------------

--> a group of characters which are enclosed (written) within double quotes

is called as string literal.

ex:

1. String str = "hyderabad";

2. String str = "welcome";

3. String str = "abc123$#@";

boolean literals

----------------

--> true and false are called as boolean literals in java.

note: literals are not keywords.

ex:

1. boolean b = true;

2. boolean b = false;

3. boolean b = 1; //error

4. boolean b = "true"; //error

5. boolean b = False; //error

--> boolean literals that is true and false are case sensitive.

-----------------------------------------------------------------------------

datatypes

---------

what are datatypes?

--> datatypes are keywords and datatypes are used to specify two things.

1. type of data

2. size of data

--> datatypes are used to specify the type of data a variable should hold.

--> datatypes are used to specify the size of memory should be allocated

for the data.

types of datatypes

------------------

--> there are 2 types of datatypes

1. primitive datatypes

--> these datatypes allow a variable to store single value only.

ex: byte, short, int, long, char, float, double, boolean

2. userdefined datatypes

--> these datatypes allow a variable to store multiple values which can

be homogeneous or heterogeneous.

ex: any class or class name

primitive datatypes

-------------------

--> these datatypes are divided into 4 types

1. integer datatypes --> ex: byte, short, int, long

2. floating datatypes --> ex: float, double

3. character datatype --> ex: char

4. boolean datatype --> ex: boolean

integer datatypes

-----------------

1. byte (lowest size integer datatype in java is byte)

----

size : 1 byte

7 7

range : -2 to 2 - 1 (-128 to 127)

ex: byte b = 10;

byte b = 127;

byte b = 128; //c.e

byte b = 2.5; //c.e

byte b = true; //c.e

byte b = "hello"; //c.e

2. short

-----

size : 2 bytes

15 15

range : -2 to 2 - 1 (-32768 to 32767)

ex: short s = 10;

short s = 128;

short s = 32767;

short s = 32768; //c.e

3. int

---

size : 4 bytes

31 31

range : -2 to 2 - 1

ex: int i = 10;

int i = 128;

int i = 32768;

int i = 493823;

4. long (highest size integer datatype in java is long)

----

size : 8 bytes

63 63

range : -2 to 2 - 1

ex: long a = 10;

long a = 100;

long a = 43829;

long a = 8237435;

floating datatypes

------------------

1) float

-----

size : 4 bytes

range : -3.4e-38 to 3.4e38

ex: float a = 2.5; //c.e

float a = 2.5f;

float a = 10; //10.0

float a = 011; //9.0

float a = 0x10; //16.0

float a = 0b10; //2.0

float a = true; //c.e

float a = "hello"; //c.e

float a = 'b'; //98.0

2) double

------

size : 8 bytes

range : -1.7e-308 to 1.7e308

ex: double a = 2.5;

double a = 2.5f;

double a = 10; //10.0

double a = 012; //10.0

double a = 0b101; //5.0

double a = true; //c.e

double a = "hello"; //c.e

--> float is 32 bit floating point datatype and double is 64 bit floating

point datatype.

--> float uses 6 digits as precision where as double uses 14 digits as

precision.

12.345

|

precision

character datatype

------------------

1) char

----

size : 2 bytes

range : 0 to 65535

ex: char ch = 'a';

char ch = 97; //a

char ch = '\u0061'; //a

char ch = '$';

char ch = '5';

char ch = '10'; //c.e

char ch = 'ab'; //c.e

boolean datatype

----------------

1) boolean

-------

size : 1 bit

range : true and false

ex: boolean b = true;

boolean b = false;

boolean b = 1; //c.e

boolean b = 0; //c.e

boolean b = "true"; //c.e

boolean b = True; //c.e

-----------------------------------------------------------------------------

operators in java

-----------------

what is an operator?

--> an operator is a symbol and an operator is used to perform some

operations on operands (variables).

ex: a + b

here + symbol is an operator and a, b are operands or variables

--> some operators are unary operators, some are binary operators and

some are ternary operators.

--> operators precedence table

|- operators, precedence, associativity

--> every operator is having a rank which is called as precedence and using

precedence we decide which operator should be given first preference or

priority.

--> associativity is the order in which an operator should be evaluated and

it can be either left to right or right to left.

precedence --> priority or preference

associativity --> order

ex:

1. int a;

a = 4 + 3 \* 2; --> a = 4 + 6; --> a = 10;

2. int a;

a = 2 + 4 + 3; //associativity of + is left to right

= 6 + 3

= 9

3. int a = 10; //associativity of = is right to left

new operator

------------

--> new is a keyword and operator and new operator is used to create object

to the class and when we create object to the class then memory will be

allocated for the members of the class and we can access the members of

the class.

syntax: ClassName objName = new ClassName();

class Employee

{

int id = 1;

String name = "ravi";

double salary = 1000.0;

public void work() {

System.out.println("working..");

}

}

----------Employee.java--------------

javac Employee.java

Employee.class (byte code)

java Employee //error: no main method

--> if we create the class memory will not be allocated for the members of the

class because a class is a blueprint.

--> if memory is not allocated for the members of the class then we cannot

access the members of the class.

--> we can access the members of the class when the memory is allocated for

the members of the class.

--> memory will be allocated for the members of the class when we create

object to the class.

--> to create object to the class we use new operator or new keyword.

member operator

---------------

--> . dot symbol is called as member operator.

--> member operator is used to access the members of the class or object.

uses of member operator

-----------------------

i. using member operator we can access instance variables of the class.

syntax: objName.instanceVariableName;

ii. using member operator we can access the instance methods of the class.

syntax: objName.instanceMethodName();

iii. using member operator we can access the static variables of the class.

syntax: ClassName.staticVariableName; //recommended

(or)

objName.staticVariableName;

iv. using member operator we can access the static methods of the class.

syntax: ClassName.staticMethodName(); //recommended

(or)

objName.staticMethodName();

v. using member operator we can access the sub package.

syntax: package-name.sub-package-name;

vi. using member operator we can access the class which is available

inside the package.

syntax: package-name.sub-package-name.ClassName;

increment and decrement operators :

---------------------------------

++ increment operator

-- decrement operator

--> both increment and decrement operators are unary operators.

--> increment operator increments the value of the variable by '1'

directly in the memory location.

--> decrement operator decrements the value of the variable by '1'

directly in the memory location.

arithmetic operators :

--------------------

+ - \* / %

Note: whenever we perform any arithmetic operations in java we must

have to follow a rule or formula :

MAX(int, type of first operand, type of second operand)

ex:

1. byte + byte = int

2. short + byte = int

3. char + short = int

4. char + char = int

5. byte + long = long

6. int + long = long

7. float + long = float

8. short + double = double

9. float + double = double

10. byte \* byte = int

11. short - byte = int

12. int / byte = int

13. long % int = long

division operator / (it gives quotient after division)

modulus operator % (it gives remainder after division)

---------------------------------------------------------------------------------

string concatenation operator (+ plus symbol)

-----------------------------

--> if atleast one operand or one value is string type then + acts as string

concatenation operator.

--> the result of string concatenation is string.

ex:

1. int a = 10, b = 20, c = 30;

String str = "hello";

System.out.println(a + b); //30

System.out.println(a + str); //10hello

System.out.println(str + a); //hello10

System.out.println(a + b + str); //30hello

System.out.println(a + b + c + str); //60hello

System.out.println(str + a + b + c); //hello102030

2. int a = 10;

output: a = 10

System.out.println("a = " + a); //a = 10

3. int a = 10, b = 20;

output: a = 10 and b = 20

System.out.println("a = " + a + " and b = " + b); //a = 10 and b = 20

4. int id = 1;

String name = "john";

char gender = 'm';

double salary = 1000.0;

output: id = 1, name = "john", gender = 'm', salary = 1000.0

Sop("id = " + id + ", name = \"" + name + "\", gender = '" + gender + "', salary = " + salary);

5. String str = "hyderabad";

output: "hyderabad"

System.out.println("\"" + str + "\"");

parenthesis ()

-----------

--> it is the highest precedence operator in java and it is used to control

the precedence of other operators.

ex: int a = (5 + 2) \* 7; //49

unary minus -

-----------

--> it changes the sign of the variable or value.

ex:

1. int a = -4;

System.out.println(-a); //4

2. int a = 4;

System.out.println(-a); //-4

unary plus +

----------

--> it will not change any sign but it represents positive value.

ex:

1. int a = 10;

System.out.println(+a); //10

2. int a = -10;

System.out.println(+a); //-10

3. int a = 3, b = -9, c = 4, d = -2, e = 0;

e = -a + -b + +c + +d;

System.out.println(e); //8

assignment operator =

-------------------

--> it is used to assign the value to the variable.

--> the associativity of assignment operator is right to left.

--> it can be used in 3 ways

i. simple assignment

ex: int a = 10; (or) int a;

a = 10;

ii. chained assignment

ex: int a, b, c;

a = b = c = 10;

iii. compound assignment

ex: a = a + b; --> a += b;

x = x \* y; --> x \*= y;

+= -= \*= /= %= &= |= ^= <<= >>= >>>=

conditional operator ?:

--------------------

--> it is also called as ternary operator because it is the only

ternary operator in java.

syntax: condition/expression?expression-1:expression-2;

ex: int a;

1. a = 2<5?10:20; //a = 10

2. a = 2>5?100:200; //a = 200

3. a = 2<5?10:20:30; //error

4. a = 2<5?10; //error

--> write the program to find the biggest of two numbers using conditional

operator.

------------------------------------------------------------------------------

relational operators :

--------------------

< > <= >=

note: relational operators return true if condition is satisfied or else

return false if condition is not satisfied.

ex: boolean res;

1. res = 2 < 5; //true

2. res = 5 > 7; //false

3. res = 4 <= 7; //true

4. res = 4 <= 4; //true

5. res = 4 <= 3; //false

6. res = 7 >= 4; //true

7. res = 7 >= 7; //true

8. res = 7 >= 9; //false

9. res = 2.5 < 4.5; //true

10. res = 'a' <= 'b'; //true

11. res = true >= false; //c.e: bad operand types

12. String s1 = new String("hello");

String s2 = new String("hello");

boolean res = s1 <= s2; //c.e: bad operand types

System.out.println(res);

note:

i. we can apply relational operators on all the primitive datatypes

except boolean.

ii. we cannot apply relational operators on objects or object types

(object type means class which is a userdefined datatype).

----------------------------------------------------------------------------------

equality operators :

------------------

== !=

--> equality operators are used for comparison.

note: equality operators return true if condition is satisfied or

else return false if condition is not satisfied.

ex: boolean res;

1. res = 2 == 2; //true

2. res = 5 != 4; //true

3. res = 5 == 4; //false

4. res = 5 != 5; //false

5. res = 10 == 10.0; //true

6. res = 'a' != 'b'; //true

7. res = true != false; //true

8. res = false == false; //true

8. String s1 = new String("hello");

String s2 = new String("hello");

System.out.println(s1 == s2); //false

System.out.println(s1 != s2); //true

note:

i. we can apply equality operators on all the primitive datatypes.

ii. we can also apply equality operators on objects and if we apply

equality operators on objects then equality operators compare

reference numbers but not data.

------------------------------------------------------------------------------

binary language

---------------

in binary, bi means two and nary means numbers

binary means two numbers that is 0 and 1

0 binary digit or bit

1 binary digit or bit

what is binary language?

--> the language which uses only 0's and 1's is called as binary language.

--> only machine understands binary language.

concepts of binary language

---------------------------

1. decimal to binary conversion

2. binary to decimal conversion

3. bcd code (8421)

4. msb and lsb

5. 1's complement and 2's complement

6. binary addition

7. negative decimal to binary conversion

8. negative binary to decimal conversion

bitwise operators

-----------------

--> bitwise operators work on binary.

& - bitwise and

| - bitwise or

^ - bitwise xor

~ - bitwise complement

<< - bitwise left shift

>> - bitwise right shift

>>> - bitwise zero fill right shift

--> bitwise complement is unary operator and remaining all are

binary operators.

--> &, |, ^ are binary operators and these operators can be applied on

integer and boolean types.

------------------------------------------------

a b a && b a || b a ^ b

------------------------------------------------

t f f t t

f t f t t

t t t t f

f f f f f

------------------------------------------------

ex: boolean b1 = true, b2 = false;

1. System.out.println(b1 & b2); //false

2. System.out.println(b1 | b2); //true

3. System.out.println(b1 ^ b2); //true

4. int a = 10, b = 6;

System.out.println(a & b); //2

1010

0110 &

------

0010 = 2

------

5. int a = 10, b = 6;

System.out.println(a | b); //14

1010

0110 |

------

1110 = 14

------

6. int a = 10, b = 6;

System.out.println(a ^ b); //12

1010

0110 ^

------

1100 = 12

------

bitwise complement ~ (tild symbol)

------------------

--> it is a unary category operator and it can be applied only on

integer type.

ex: int a = 5;

System.out.println(~a);

0000 0101 (binary of 5)

apply ~

1111 1010 =

converting negative binary number into decimal

----------------------------------------------

step-1: take the bits which are representing the value

111 1010

step-2: apply 2's complement

1

000 0101

+ 1

--------

000 0110

--------

step-3: convert the binary number into decimal and apply the sign

000 0110 =

------------------------------------------------------------------------

note: formula for bitwise complement is -(n+1)

int a = 5;

System.out.println(~a);

------------------------------------------------------------------------

bitwise left shift <<

------------------

--> it shifts the bits from left side.

--> it is a binary operator and it is applied only on integer type.

ex: int a = 10;

System.out.println(a << 1); //20

0000 1010

a << 1

\_000 1010

now all the bits start shifting towards left side

0001 010\_

the vacant place is filled with 0

0001 0100 = 20

bitwise right shift >>

-------------------

--> it shifts the bits from right side.

ex: int a = 10;

System.out.println(a >> 1); //5

0000 1010

a >> 1

0000 101\_

now all the bits start shifting towards right side

\_000 0101

the vacant place is msb and it is filled with either 0 or 1, in case of

positive number it is filled 0 so result is positive and in case of negative

number it is filled with 1 so result is negative

0000 0101 = 5

bitwise zero fill right shift >>>

-----------------------------

--> it is same as right shift operator except one difference i.e., in

case of bitwise zero fill right shift operator weather the number

is positive or negative always the msb is filled with 0 only so

result is always positive.

short circuit operators

-----------------------

&& logical and

|| logical or

--> short circuit operators are same as bitwise and, bitwise or operators

except two differences.

--------------------------------------------------------------------------------------

&, | &&, ||

--------------------------------------------------------------------------------------

1. bitwise operators can be 1. short circuit operators can be

applied on boolean and applied only on boolean type.

integer types.

2. second expression 2. second expression evaluation is

evaluation is always not always mandatory sometimes

mandatory in case of it is optional.

bitwise operators.

i. in case of && if first expression is false

then only second expression evaluation is

optional and if first expression is true then

second expression evaluation is mandatory.

ii. in case of || if first expression is true

then only second expression evaluation is

optional and if first expression is false

then second expresession evaluation is

mandatory.

-----------------------------------------------------------------------------------------

Note: it is always recommended to use short circuit operators over bitwise

operators because short circuit operators improve performance.

5s 5s 5s = 15s

expr-1 & expr-2

expr-1 | expr-2

5s = 5s

expr-1 && expr-2

expr-1 || expr-2

boolean complement operator !

---------------------------

--> it is a unary operator and it can be applied only on boolean type.

--> it changes true to false and false to true.

ex:

1. boolean b = true;

System.out.println(!b); //false

2. boolean b = false;

System.out.println(!b); //true

3. System.out.println(!(2<5)); //false

4. System.out.println(!(5>3 && 7<4)); //true

type cast operator

------------------

--> it is used to perform explicit type casting.

syntax: (type)

type casting

------------

what is type casting?

--> it is a process of converting one datatype value into another datatype

value.

types of type casting

---------------------

--> there are 2 types of type casting

1. implicit type casting

2. explicit type casting

--> the type casting which is performed by the compiler automatically is

called as implicit type casting.

ex: double a = 10; //upcasting or widening

System.out.println(a); //10.0

note: if we assign a lower datatype value to the higher datatype variable

then only compiler performs implicit type casting.

--> the type casting which is performed by the programmer manually is called

as explicit type casting.

ex: int a = 2.5; //compile time error

int a = (int)2.5; //downcasting or narrowing

System.out.println(a); //2

note: if we assign a higher datatype value to a lower datatype variable

then compiler will not perform implicit type casting, in this case

we have to perform explicit type casting.

what is upcasting or widening?

--> converting lower datatype value into higher datatype value is called as

upcasting or widening.

what is downcasting or narrowing?

--> converting higher datatype value into lower datatype value is called as

downcasting or narrowing.

-------------------------------------------------------------------------------------

what is inheritance?

--> acquiring all the properties of one class into another class is called

as inheritance.

--> inheritance establishes IS-A relationship (parent child relationship)

between the classes.

--> in java, to use inheritance we use extends keyword.

ex:

class A //parent class or super class

{

void m1() {

}

void m2() {

}

}

class B extends A //child class or sub class

{

void m3() {

}

void m4() {

}

}

public class Test

{

public static void main(String[] args)

{

B b = new B();

b.m3();

b.m4();

b.m1();

b.m2();

}

}

type casting using userdefined datatypes

----------------------------------------

ex:

//super class or parent class

class One //One type

{

public void methodOne() {

}

}

//sub class or child class

class Two extends One //Two type, One type

{

public void methodTwo() {

}

}

note:

1. a sub class is sub class type as well as super class type because sub class

has sub class properties and super class properties.

2. a super class is super class type only but not sub class type because super

class has super class properties only.

3. as a sub class is sub class type as well as super class type thats why we

can hold sub class object by using sub class reference variable as well as

by using super class reference variable also.

4. as a super class is super class type only thats why we can hold super class

object by using super class reference variable only but not using sub class

reference variable.

//super class

class One //class One is only super class type

{

public void methodOne() {

}

}

//sub class

class Two extends One //class Two is sub class type, super class type

{

public void methodTwo() {

}

}

case-1: Two t = new Two(); //valid

case-2: One o = new Two(); //valid

case-3: One o = new One(); //valid

case-4: Two t = new One(); //invalid (error)

------------------------------------------------------------------------------

class A //super class

{

void m1() {

}

}

class B extends A //sub class

{

void m2() {

}

}

B B = new B();

b.m1();

b.m2();

--> using sub class reference variable we can access both sub class members

and super class members.

class A //super class

{

void m1() {

}

}

class B extends A //sub class

{

void m2() {

}

}

A a = new B();

a.m1();

a.m2(); //error

--> using super class reference variable we can access only super class

members but not sub class members.

--> to access both super class members and sub class members we need sub

class object as well as sub class reference variable.

class A //super class

{

void m1() {

}

}

class B extends A //sub class

{

void m2() {

}

}

A a = new B();

B b = (B)a;

b.m1();

b.m2();

------------------------------------------------------------------------------

instanceof operator

-------------------

--> to test an object is an instance of a class type or interface type we use

instanceof operator.

syntax: boolean b = objName instanceof ClassName/InterfaceName;

note:

--> using instanceof operator we cannot test two different classes directly

which are not having parent child relationship that is inheritance.

--> we can test two different classes using instanceof operator if they are

having parent child relationship that is inheritance.

java.lang

|- Object : it is a class available in java.lang package, for every class

Object is the super class in java.

----------------------------------------------------------------------------------------------

program

-------

line-1

line-2

line-3

. linear order (top to bottom line by line)

.

.

.

.

line-100

--> by default the program is executed at runtime in linear order.

Control Flow Statements

-----------------------

--> control flow statements are the statements which are used in the program

to specify the order in which the program is executed at runtime.

types of control flow statements

--------------------------------

--> there are 3 types of control of statements

1. selection statements

ex: if statement or simple if

if else statement

if else ladder

switch case

2. iterative statements (loops)

ex: while loop

do while loop

for loop

for each loop

3. transfer statements

ex: break

continue

return

if statement

------------

syntax: if(condition/expression)

statement;

--> here condition or expression must be boolean type

--> for if statement body is optional

--> if we dont write body for if statement then if statement is applied

to only one statement, if we want to apply if statement for more than

one statement then body is mandatory.

ex:

1. if(2<5)

System.out.println("hello"); //hello

2. if(5<3)

System.out.println("hello");

3. if(2<5)

System.out.println("hello"); //hello bye

System.out.println("bye");

4. if(2>5)

System.out.println("hello"); //bye

System.out.println("bye");

5. if(2>5)

{

System.out.println("hello");

System.out.println("hi");

}

program-1: write the program to check the number is even or odd using if statement.

program-2: write the program to take some amount and check if amount is greater than

or equal to 10 thousand then give 25% discount using if statement.

nested if statement

-------------------

--> if statement inside another if statement is called as nested if statement.

ex: int x = 3, y = 9, z = 4;

if(x == 3)

{

if(y == 9)

{

if(z == 4)

{

System.out.println("hello");

}

System.out.println("welcome");

}

System.out.println("bye");

}

if else statement

-----------------

syntax: if(condition/expression)

statement-1;

else

statement-2;

--> here condition or expression must be boolean type.

--> for if-else statement body is optional.

--> if we dont write body for if-else statement then if-else is applied to only

one statement.

--> if we want to apply if-else statement to more than one statement then body

is mandatory.

--> if can exists without else but else cannot exist without if.

ex:

1. if(2<5)

System.out.println("hello"); //hello

else

System.out.println("bye");

2. if(2>5)

System.out.println("hello"); //bye

else

System.out.println("bye");

3. if(2<4)

System.out.println("hello"); //hello bye

else

System.out.println("hi");

System.out.println("bye");

4. if(2>4)

System.out.println("hello"); //hi bye

else

System.out.println("hi");

System.out.println("bye");

5. if(2<4)

System.out.println("hello"); //hello

else

{

System.out.println("hi);

System.out.println("bye");

}

6. if(2<5)

System.out.println("hello");

System.out.println("welcome"); //c.e: else without if

else

System.out.println("bye");

7. if(2>5)

{

System.out.println("hello");

System.out.println("welcome"); //bye

}

else

System.out.println("bye");

8. if(2<5)

{

System.out.println("hello"); //hello welcome bye

System.out.println("welcome");

}

else

System.out.println("hi");

System.out.println("bye");

9. if(2<5)

{

System.out.println("hello");

System.out.println("welcome"); //hello welcome

}

else

{

System.out.println("hi");

System.out.println("bye");

}

req-1: write the program to check age of the person, if the age is greater than

and equal to 18 then display eligible to vote and if age is less than 18

then display not eligible to vote using if else statement.

req-2: write the program to take some amount and check if amount is greater than

or equal to 10 thousand then give 20% discount or else 5% discount.

nested if else statement

------------------------

--> if else statement inside another if else statement is called as nested if else

statement.

req: write the program to find the biggest of 3 numbers using nested if else

statement.

if else ladder

--------------

syntax: if(condition-1)

statement-1;

else if(condition-2)

statement-2;

else if(condition-3)

statement-3;

.

.

.

else if(condition-n)

statement-n;

else

default-statement;

program-1: write the program to take 3 subjects marks, find total, average & result.

i. if average is greater than or equal to 75 --> Grade A

ii. if average is less than 75 and greater than equal to 65 --> Grade B

iii. if average is less than 65 and greater than equal to 55 --> Grade C

iv. if average is less than 55 and greater than equal to 45 --> Grade D

v. if average is less than 45 --> Fail

program-2: write the program to calculate the electricity bill amount based on units.

i. 1 to 100 --> 2.50

ii. 101 to 300 --> 3.50

iii. 301 to 500 --> 4.50

iv. 501 to 1000 --> 5.50

v. 1000+ --> 6.50

switch case :

-----------

syntax: switch(expression)

{

case label-1: statement(s);

break;

case label-2: statement(s);

break;

case label-3: statement(s);

break;

.

.

.

case label-n: statement(s);

break;

default: statement(s);

}

Note:

i. break is optional in switch.

ii. order of cases is not mandatory.

iii. cases are optional.

iv. switch expression and body are mandatory.

v. duplicate cases are not allowed.

vi. the types which are allowed for switch expression are :

byte, short, int, char, String, Wrapper class, Enum

vii. switch expression can be a variable or can be a constant.

viii. switch case label must be a constant.

-------------------------------------------------------------------------------------

iterative statements (loops)

--------------------

--> if we want to execute any statements repeatedly for some number of

times then use iterative statements.

--> iterative statements are also called as loops.

--> there are 4 iterative statements or loops

1. while loop

2. do while loop

3. for loop

4. for each loop

--> 4 loops are categorized into 2 types

1. pre-tested loops

--> in pre-tested loops first condition is checked then statements are

executed.

ex: while loop, for loop, for each loop

2. post-tested loops

--> in post-tested loops first statements are executed then condition is

checked.

ex: do while loop

---------------------------------------------------------------------------

while loop

----------

--> it is a pre-tested loop.

syntax:

while(condition/expression) //here condition or expression must be boolean type

{

statement(s);

}

prg-1: write the program to find the sum of 100 numbers using while loop.

prg-2: write the program to display all the even numbers between 1 to 100

using while loop.

prg-3: write the program to display multiplication table of a number.

ex: 9 x 1 = 9

9 x 2 = 18

9 x 3 = 27

.

.

9 x 10 = 90

prg-4: write the program to reverse a number.

ex: 1234

reverse number = 4321

do while loop

-------------

--> it is posted tested loop.

syntax: do

{

statement(s);

}

while(condition/expression);

program-1: write the program to display 1 to 10 numbers in reverse order.

program-2: write the program to display all the odd numbers between 1 to 100.

program-3: write the program to find the sum of all the even numbers between

1 to 100 and display.

for loop

--------

syntax: for(initialization;condition;updation)

{

statement(s);

}

program-1: write the program to find the factorial of a number.

ex: !5 = 120

program-2: write the program to check the number is prime or not.

nested loops

------------

--> loop inside another loop is called as nested loops.

1 2 3 4 5 i = 1, j = 1, 2, 3, 4, 5

1 2 3 4 5 i = 2, j = 1, 2, 3, 4, 5

1 2 3 4 5 i = 3, j = 1, 2, 3, 4, 5

1 2 3 4 5 i = 4, j = 1, 2, 3, 4, 5

1 2 3 4 5 i = 5, j = 1, 2, 3, 4, 5

1st loop --> rows (5 times)

2nd loop --> columns (25 times)

note: if we have rows and columns or 1 to many relation then use nested loops.

for(int i=1;i<=5;i++)

{

for(int j=1;j<=5;j++)

{

System.out.print(j + " ");

}

System.out.println();

}

------------------------------------------------------------------------------

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

for(int i=1;i<=5;i++)

{

for(int j=1;j<=i;j++)

{

System.out.print(j + " ");

}

System.out.println();

}

-----------------------------------------------------------------------------

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

for(int i=1;i<=5;i++)

{

for(int j=1;j<=i;j++)

{

System.out.print(" \* ");

}

System.out.println();

}

-----------------------------------------------------------------------------

5 4 3 2 1

5 4 3 2

5 4 3

5 4

5

for(int i=1;i<=5;i++)

{

for(int j=5;j>=i;j--)

{

System.out.print(j + " ");

}

System.out.println();

}

------------------------------------------------------------------------------

for each loop

-------------

--> for each loop is added in java from jdk 1.5v onwards.

--> it is a special loop and it is used to access one by one element of

an array or a collection.

syntax: for(datatype variableName : array/collection)

{

//statement(s);

}

transfer statements

-------------------

--> there are 3 transfer statements

1. break

2. continue

3. return

1. break

-----

--> it is a keyword and a transfer statement.

--> break can be used only in 2 places

i. inside the switch case

ii. inside the loops

--> if we use break outside the switch case or loops then compiler gives

error.

--> break is used inside the switch case to terminate the execution of

switch case and to come out of switch case.

--> break is used inside the loops to terminate the iterations of the loop

and to come out of the loop.

2. continue

--------

--> it is a keyword and a transfer statement.

--> it can be used only in 1 place that is inside the loops.

--> if we use continue outside the loops then compiler gives error.

--> to skip the iterations of the loop we use continue inside the loops.

program: write the program to skip first 5 iterations of while loop using

continue.

3. return

------

--> it is a keyword and a transfer statement.

--> there are 2 uses of return

i. return is used to return the value to the method whose return type is

other than void

ii. return is used to return the control to the caller of this method

note: if the return type of the method is void then we cannot use return

statement to return the value otherwise compiler gives error.

------------------------------------------------------------------------------

types of variables in java

--------------------------

--> a variable can hold single value or a variable can hold multiple values

so based on the value which is stored by a variable, all the variables

are divided into 2 types in java.

1. primitive variables

2. reference variables

primitive variables

-------------------

--> these variables hold single value.

ex: int a = 10;

double b = 2.5;

char ch = '$';

reference variables

-------------------

--> these variables hold object and object contains multiple values.

ex: String str = new String("welcome");

class Employee

{

int id = 1;

String name = "john";

double salary = 1000.0;

}

Employee emp = new Employee();

--> the datatype of primitive variables is primitive datatype.

--> the datatype of reference variable is class name which is a userdefined

datatype.

------------------------------------------------------------------------------

--> based on the purpose and position of declaration of the variable, all the

variables are divided into 3 types in java.

1. instance variables

2. static variables

3. local variables

instance variables

------------------

1. if the value of the variable has to be different from one object to another

object then use instance variables.

2. instance variables are declared directly inside the class but not inside

the method or block or constructor.

3. instance variables are created (memory is allocated) at the time of object

is created and instance variables are created inside the object.

note: creating object to the class means allocating the memory for instance

variables.

4. initialization of instance variables is optional, if we dont initialize the

instance variables then jvm takes the responsibility to initialize instance

variables with the default values.

-------------------------------------------------

datatype default-value

-------------------------------------------------

1. byte, short, int, long 0

2. float, double 0.0

3. char '\u0000'

4. boolean false

5. String null

6. reference variable null

-------------------------------------------------

5. instance variables are called as object variables or non-static variables.

6. instance variables are object members.

7. scope and lifetime

scope means accessibility and visibility

lifetime means creation and destruction

--> the scope of instance variables is same as object scope because when

object is created instance variables are created so we can access instance

variables and if object is not created then instance variables are also

not created so we cannot access instance variables.

--> the lifetime of instance variables is same as object life time because

instance variables are created when object is created and instance varia-

bles are destroyed when object is destroyed.

note: object creation and object destruction is called as object lifetime.

8. we cannot access instance variables of the same class directly from the

static methods and static blocks.

9. in every object seperate copy of instance variables are created and

available.

syntax: objName.instanceVariableName;

------------------------------------------------------------------------------

static variables

----------------

1. if the value of the variable has to be same from one object to another

object then go for static variables.

2. static variables are declared directly inside the class by using static

keyword but not inside the method or block or constructor.

3. static variables are created (memory is allocated) only once at the time

of classloading or loading the class in jvm's method area.

4. initialization of static variables is not mandatory, it is optional and

if we dont initialize static variables then jvm takes the responsibility

to initialize static variables with the default values.

5. static variables are also called as class variables.

6. static variables are class members.

7. scope and lifetime

i. the scope of static variables is same as class scope which means if class

is loaded static variables are created so we can access static variables,

if class is not loaded then static variables are not created so we cannot

access static variables.

ii. the lifetime of static variables is same as class lifetime which means

static variables are created when class is loaded and static variables

are destroyed when class is unloaded.

class lifetime means class loading and class unloading

8. static variables can be accessed directly from static methods and static

blocks.

9. every object in heap area shares a single copy of static variable which

is created in method area.

syntax: ClassName.staticVarName; //recommended

(or)

objName.staticVarName;

------------------------------------------------------------------------------------

local variables :

---------------

1. If the requirement of the program or the programmer is to use some

variables for temporary purpose then use local variables.

2. Local variables are declared directly inside the method or block or

constructor but not inside the class directly.

3. Local variables are created (memory is allocated) in java stacks

whenever method or block or constructor execution starts.

4. Initialization of local variable is mandatory at the time of

declaration and before accessing.

5. Local variables are also called as temporary variables or automatic

variables.

6. Local variables are method members or block members.

7. Scope and Lifetime

a. we can access local variables only within the method or block

or constructor where they are declared but not outside the

method or block or constructor so the scope of local variables

is method scope or block scope.

b. when method or block or constructor execution starts local

variables are created and when method or block or constructor

execution completes local variables are destroyed this is

called as lifetime of local variables.

8. for instance variables and static variables all the modifiers are

allowed but for local variables only final modifier is allowed.

Note: Instance variable can be primitive variable or reference

variable, static variable can be primitive variable or

reference variable and local variable can be primitive

variable or reference variable.

------------------------------------------------------------------------------

class One {

}

class Two

{

One o; //reference variable as instance variable

int i; //primitive variable as instance variable

static int j; //primitive variable as static variable

static One o; //reference variable as static variable

void methodOne()

{

int k; //primitive variable as local variable

One o; //reference variable as local variable

}

}

--------------------------------------------------------------------------------------------

types of methods in java

------------------------