Steps to write the code :

1. Open a notepad file
2. Write the code there
3. Save the notepad file with the same name as of class with an extension .java
4. Open command prompt
5. Move to the location where you have saved the file.
6. compile the file using **Compile** : javac file_name.java
7. Resolve all the errors
8. run the file using **Run** : java file_name.java

Java is simple, secure, robust, object-oriented, portable, multithreaded, Interpreted, distributed.

Class is a generalized representation of individual objects. Objects(entities) are independent and have the same structure. Every object will have its own properties(data members) and behaviors (methods).

Class is the template for the objects and Object is an instance of a class.

The object oriented paradigm is not to replace process oriented approach. There are certain class of problems which are solved by a procedure-oriented approach that would not be as good enough (efficient) as it would be when solved by an object -oriented approach. Object-oriented approach still involves procedure-oriented approach so they are not competing with each other indeed they are complementing each other

In procedure oriented programming we try to solve a problem as a collection of processes.With object oriented programming we can build more elegant sophisticated natural softwares and which directly connects to how we understand the real world.In object orientation we see the solution to a problem as a collection of objects.

Java is portable but java compiler, JVM are not portable:

.c file (source file) -> compile -> .exe file(machine executable code) -> HW1

In c programming only the source file is portable and requires a new .exe file for a new OS(Hardware).

.java file -> compiler -> bytecode ->JVM (interpreter - pseudo h/w env)

In Java both the compiler and JVM are bound to hardware. The portability is because of the bytecode. But, JVM's portability comes into picture because the JVM's are written in such a way that any JVM is capable of executing any bytecode produced by any compiler.

Java is Robust i.e strongly typed and has **automatic garbage collection** All the memory locations allocated will be moved to garbage implicitly if they are unused and the memory is reallocated for other purposes. In c++ there is no automatic garbage collection; the programmer should explicitly delete the memory allocated if there is no more use of the memory. If the allocated memory is not deleted, it continues to exist for the lifetime of the program and any further usage of memory may

not happen as it runs out of memory, this phenomenon is known as memory leakage which does not exist in JAVA.

In a Multiprocessing environment multiple processes can be executed simultaneously whereas in a **multithreading** environment the subparts of a single process can be executed simultaneously.

In **Interpretation** there is no compilation; rather we directly interpret that which is more efficient; java has a hybrid approach where it **first gets compiled and then it gets interpreted.**

Java is **distributed**, the objects to be used need not be on the same machine they can be at different locations.

What are three fundamental object - oriented Principles supported by JAVA?

1. **Encapsulation**
2. **Inheritance**
3. **Polymorphism**

Encapsulation is the mechanism that binds together code and the data it uses, and keeps both protected from outside interference and misuse.

Inheritance is the process by which one object acquires the properties of its parent.

Polymorphism-One interface for a general class of actions

★ **when assigning float values append the value with the suffix f**

Command-Line Arguments:

When I want to pass on something to the program that is to be executed. The arguments which go along with the execution command are known as command line Arguments.

When you compile the java file on the command prompt, **JVM** (Java virtual machine) accesses the main function as class_name.main().

Data is provided in the String args [] of the main function. At the run-time data is provided; if not by default the value will be stored as 0. In a command prompt values are entered in the run command line.

Any number of command-line arguments can be passed only in the form of Strings.

Ex: java file_name 1 2 3 4 5

Static method:
It is the method which can be called from any class using the class name without creating an instance of that class.

Main method is a static method.

AWT

import java.awt.*;

What for ?

- Applets : meant for running on the browser
- Standalone windows

- Building blocks(Objects) of GUI are known as components .
- Container is a component that contains other components.
- Placing the components in their appropriate coordinates is done by layout managers.

Source and listeners hold many to many relationship

Http client means browser. Browser knows how to display the content in html pages

Http servers means they are the applications which follow http server protocol and are capable of hosting any number of html pages
Based on url(ip address, directory, name and stuff) the web pages are distinguished.

APplet code : Java Code

We can embed the applet class into html file and then make it to be posted by web server

Split Formatting:
```
System.out.format("%3d", i+1);
System.out.format("%15.2f", rainfall[i]);
System.out.format("%15.2f\n", difference);
```

Constructors:

It is used to initialize the created objects. These once created can be reused; these are similar to methods. Mainly used for automatic initialization.

They don't have any return type. A constructor is automatically called when an object is created.

A method can be called at any point of the program but a constructor cannot.

We can have lots of constructors in a program provided that they have different parameters.

They are **3** types of Constructors:

1. No Arg constructor (constructors without parameters).
2. Parameterized constructor (constructors with parameters).
3. Default constructor

In presence of a Parameterized constructor Default constructor will not be implicitly available.

Q)Try to implement all three constructors in the same program and check whether it's compiled or not. If not compiled then explain the reason for the compilation error.

All the three constructors can't be compiled in the same program. Though we have constructor overloading [which happens with a difference in the signature of the constructor/no.of parameters in the constructor]. Here the default constructor and no argument constructor have the same signature and no arguments hence we get an compilation error.

Q)Try out all the possible combinations among all three types of constructors and list the combinations where there is no compilation error.

Default constructor, parameterized constructor
parameterized constructor, No argument constructor
No argument constructor, Default constructor

These are the possible combinations of all three types of constructors. Out of which these two :

Default constructor, parameterized constructor
parameterized constructor, No argument constructor

combinations have no compilation error

OS TUTs

1. Print Screen to take a screenshot of the desktop.
2. Alt + Print Screen to take a screenshot of a window.
3. Shift + Print Screen to take a screenshot of an area you select.

BASICS OF LINUX PROGRAMMING

cal : gives the current month

cal -m month_name : gives the required month

cal mm yyyy : gives the required month

cal -3 : gives the before,current, next month

date : gives the date along with time wrt timezone

date +%m : gives the month number

date +%h : gives the month name

date +%d : gives the month date

date +%H : gives the hour

date +%M : gives the minutes

date +%S : gives the seconds

date +%HMS : one of the three hour/minutes/seconds will be displayed.

date +%D : gives the date in complete notation

date +%T : gives the time in complete notation

man req_command : the details of the req command is displayed.

q : quits the command and moves to the next line

whoami :gives the username

who/ who -H/ who -Hu : gives the username and time he started using the terminal

passwd : command to change the password of login user account

cat > file_name : creates a file with given file name and asks to enter the contents

ctrl+d : quits the content entering space

cat file_name : displays the contents of the file

cat file_n2 file_n1: concatenates the contents in the given order

cat file_n2 file_n1 > file_n3: concatenates the contents in the given order and places them in the last file

cat >> file_name : retains the already present content and asks to enter the new content

mv oldfile_name newfile_name: Renaming the file

rm file_name : removes the file

Another method

ed: enter a in the next line and start entering content from the next line and . on new line when you finish entering contents and w **file_name** on the next line it returns total count of file contents and then **q** to quit the command

cd directory_name : moves to the required directory.

pwd : gives the path of the directory you are currently working

cd .. : moves to the parent directory of the current directory.

ls/ ls. : lists the items in the current directory; items in blue color are directories and items in white are textfiles

ls .. : items in the parent directory.

ls -F : gives more info about the items in the current directory; files with slash are directories and star are executable files.

ls -a : displays the hidden files; **.** - current directory, **..** - parent directory

ls -l : more details about the items in the current directory;

- files starting with **'-'** represent normal text files and starting with **'d'** are directories. **'rwx'** means the user has the read ,write, execute permission to the file/directory .

ls -lt : displays the newly created files at first.

ls-ali : displays the files with inode number

- inode number is an unique identity to a file similar to pid for a process

ls -lS: displays the items according to the file size.

wc -l **file_name:** displays the line count of the file
wc -w **file_name:** displays the word count of the file
wc -c **file_name:** displays the character count of the file
wc -lcw **file_name:** displays the word, line, character count of the file
wc -lcw **file_n1 file_n2 :** displays the word, line, character count of the files

Od -b file_name: displays the octal representation of contents of the file

Od -bc file_name: detailed display of octal representation of contents of the file

file file_name/directory_name: gives the type of file/directory along with filename.

file -b file_name/directory_name: gives the type of file/directory without filename.

mkdir directory_name : creates a directory with the given name

mkdir d_name1 d_name2 : 2 directories gets created simultaneously

mkdir absolute_path : creates the required file/directory; the absolute path must contain the new required file/directory

- **Absolute path** is the location given right from the top level of the file system. **Relative path** is the location given relative to the current directory we are working from.

rmdir directory_name : remove directory

ps : displays the list of process that are currently running

PID - process identifier

- We observe that the process id of bash process(shell) remains constant but it changes for the ps command when given again, because it treats the new ps command as a completely new process and the earlier ps command instance has been completely removed.

ps -e : displays all the list of process that are currently running

ps -e | more : displays more the list of processes that are occurring

ps -ef : displays more/all details of the list of processes that are occuring

ps -ef | grep process_name : displays the complete details of the process. (details in the sense who created the process at what time)

pstree: displays who created which process right from the root (family chart of all the processes).

kill -9 -1 : kills all the process it can kill

kill -9 -PID : kills the given process

kill L : lists all the available process into a table

df : gives the system disk space usage

echo content : returns the content // the content must be written in strings

echo content > filename : creates the file and writes the content into the file

sudo - executes the command as another user // mostly used in installation purposes

chmod xyz filename /- Changes the permissions to rwx of the file/directory.

chmod +x/r/w filename

Chmod 777 filename- gives permissions to anyone to access the file/directory.

- **First 7** indicates owner, **Second 7** indicates group and **Third 7** indicates anyone.
- 4 (2^2)-read, 2(2^1)- write, 1(2^0)-execute (7 = 4 + 2+ 1)

ln - makes links between files

ln sourcefile_name targetfile_name : creates the target file and a hard link between the two files.

ln -s sourcefile_name targetfile_name : creates the target file and a soft link between the two files.

readlink targetfile_name : gives the file_name of the soft link(if any) attached

Hard links and Soft Links/Symbolic links:

Generally, Links are created between two files

Hard Link :

1. When a Hard link is created we see that the target file becomes a replica to the source file. The inode number, file size of both the source file and target file remains the same in case of a hardlink.
2. Whatever changes we make for either source file or target file when they are hardlinked the changes apply to both the files. Hence we say the target file is a replica of source file when they are hardlinked
3. The moment the source file is killed (implies the link is killed) the target file starts behaving as an independent file.

Soft Link :

1. When a Soft link is created we see that the target file becomes a shortcut (it gives a link reference and shows the content of source file) to the source file. The inode number, file size of both the source file and target file are different in case of a soft link.
2. Since the target file points to the source file, if any changes are made to the source file, then the target file shows the source file with made changes.
3. The moment the source file is killed (implies the link is killed) the target file gets killed implicitly, but the link remains as it is unlike hardlink.

- The only difference between a hardlink and a softlink is that they differ in inode number, lifetime and file size.
- Once the file is renamed the link gets killed

cp **orginalfile_name** **newfile_name** : copies the content of original file into the new file

- Any changes made in original file or copy file will not affect the other

diff : **file_n1** **file_n2** : displays the content difference

top : displays the free space, buffer memory, % of memory used by the processes etc.

htop : similar to top with some extra features and effects.

gedit & **:** opens the texteditor

Running a c Program: Two ways

1.
    a. Open terminal and Use the vim editor
    b. Open file using **vim** **file_name.c**
    c. To Edit the file: Press i to go to insert mode,and type your program.
    d. To save the file: Press Esc button and then type : wq.
2.
    a. save the text file containing the c program as **filename**.c
    b. For compiling the file: gcc **filename**.c -o **filename.out**
    c. For running the file : ./**filename.out**
        i. ./**filename.out &** **:** It enables the terminal during execution.

Process Management

- Pid-1 is init.
- Systemd is the great grand parent of all processes on the linux system.

fork() creates a new process by duplicating the calling process. The new process is referred to as the **child process**. The calling process is referred to as the **parent process.**

The child process and the parent process run in **separate memory spaces**. At the time of fork() both memory spaces have the same content.

Child process gets the copy of local variables or any statements the parent process had.

Note : We don't know whether the child process or the parent process is executed first.

On success, the PID of the child process is returned in the parent, and 0 is returned in the child. On failure, -1 is returned in the parent, no child process is created.

- Pid_t is like a structure/container that contains PIDs
- getpid()- a function that gives the pid of the current process.
- getppid()- a function that gives the pid of the current process's parent.
- p- a function that gives the pid of the current process's child.

- wait(int *p) - parent can issue wait() to move out of the ready queue until the child is done , the process waits until it receives a signal from the child.

    - It requires a header file <sys/wait.h>
- Whenever we give the run command; the bash(parent of all the processes) forms a new process (forking itself) and the new process is the running the .c file
- Inside the code when are forking again which spuns itself having 2 process-1.out's

Orphan Process:

When the child process is sleeping the parent process gets terminated before the child could finish.

The child will no longer be associated with the parent because it was already terminated, now this child process has become an orphan, now someone (Systemd- process belonging to init's child) has to adopt(reap) and read this child process.

Zombie(half dead/defunct) Process:
The child gets started and finishes its execution but the parent hasn't completed so it waits for the parent (in other words still running - half dead).

While the parent is sleeping we can see a defunct process running which is known as zombie process

cat /proc/pid/status | grep State: It gives the current state of the process during execution.
The moment we run execl() after fork() it completely replaces the calling process with a different binary(different command) altogether.
    arguments:
1. different binary/command
2. Dummy Filename
3. Filename required by arguments
4. NULL

PIPE:

A unidirectional data channel that can be used for interprocess communication. The array pipefd is used to return two file descriptors referring to the ends of the pipe.

pipefd[0] refers to the read end of the pipe. pipefd[1] refers to the write end of the pipe. Data written to the write end of the pipe is buffered by the kernel until it is read from the read end of the pipe.

File descriptor values :
0 : standard input
1 : standard output
2 : error

read(pfd[0], message_array, size of the message) - reads the message from the pipe
write(pfd[1], message, size of the message) - writes the message into the pipe

It is better to close the read end of the pipe while writing , similarly closing the write end while reading facilitates better inter process communication. If it is not done then Intra process communication occurs if we have the read/write for write/read in the same process.

When we try to read from the pipe without anything being written it blocks for a while to receive from the sending process and if not it reads garbage values.

dup2(int oldfd, int newfd);
The dup2() system call creates a copy of the file descriptor oldfd, using the file descriptor number specified in newfd. If the file descriptor newfd was previously open, it is silently closed before being reused.

Whatever you are going to print at the standard output using pfds[1] is kept into the write end of the pipe.

Pipes facilitate Synchronous communication only.

Message Queue
With Message Queue we have an advantage of asynchronous communication as well, i.e we need not have the sender and the receiver to be active at the same time.

We will have a message queue where the messages sent by the sender are stored and are sent to the receiver at a later time.

1. ftok(pathname,proj_id);
2. msgget(key, msg_flg);
3. fgets(msq, msgsize, stream);
4. msgsnd(msqid, const void *msgp, size_t msgsz, int msgflg);
5. msgctl(int msqid, int cmd, struct msqid_ds *buf);
6. msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg);

1. The ftok() function uses the identity of the file named by the given pathname and the least significant 8 bits of proj_id (which must be nonzero) to generate a key_t type IPC key.

    a. On success, the generated key_t value is returned. On failure -1 is returned
2. The msgget() system call returns the message queue identifier associated with the value of the key argument.
    a. It is used either to obtain the identifier of a previously created message queue when msgflg is zero and the key does not have the value IPC_PRIVATE.
    b. A new message queue is created if
        i. the key has the value IPC_PRIVATE
        ii. key isn't IPC_PRIVATE,
        iii. no message queue with the given key exists
        iv. IPC_CREAT is specified in msgflg.
    c. If msgflg specifies both IPC_CREAT and IPC_EXCL and a message queue already exists for the key, then msgget() returns an error.

3. fgets() reads in at most one less than size characters from stream and stores them into the buffer
    a. Reading stops after an EOF(End Of the File) or a newline. If a newline is read, it is stored into the buffer.
    b. A terminating null byte ('\0') is stored after the last character in the buffer.

4. The msgsnd() and msgrcv() system calls are used to send messages to, and receive messages from, a message queue. The calling process must have write permission on the message queue in order to send a message and read permission to receive a message.

    a. The msgp argument is a pointer to a caller-defined structure of the following general form:

```
struct msgbuf
{
        long mtype;      /* message type, must be > 0 */
        char mtext[200];    /* message data *
};
```

    b. The mtext field is an array (or other structure) whose size is specified by msgsz, a nonnegative integer value. Messages of zero length (i.e., no mtext field) are permitted.
    c. The mtype field must have a strictly positive integer value. This value can be used by the receiving process for message selection.

This function is used to read its assigned share of values from the file, and stores the read data into shared memory. It takes the void casted pointer of “readdata” structure.

At first we declare some local variables which are further used in the function. Now, we will decide the interval a thread reads from the file. Startvaenterl and Endvalenter are the limits of the interval.

Suppose, there are 5 threads and partsize is 20. Thread0 will incorporate the characters from 0-20 i.e Startvalenter = 20 * 0; Endvalenter = 20 * (0+1). So the intervals would be like 0-20,20-40,40-60 and so on.

We use the fseek() function to open the file, here fp is the file pointer or file position indicator. fseek() function generally has 3 types of parameters

SEEK_SET : the file pointer points to the beginning of the file
SEEK_CUR :points to the current position of the file.
SEEK_END: points to the end of the file.

ftell() is used to obtain the current value of file pointer. Here the file pointer is positioned at the beginning of the file and its value is constantly checked until it reaches the end of the file.

Now the thread begins to read the values from the file until it reaches its end limit. As soon as it reaches its end limit it moves out of the loop.

Now, we’ll store the values into the global array, the reason we are using the global array is that more than one thread may run at a point of time which may lead to misinterpretation of data storage. Hence we used a global array

In the process of reading characters from the file empty spaces are also being read hence we kept a separate num_counter for counting the no.of values that are being read.

Now this global array is kept as shared memory. Hemanth will take us forward through the shared memory part.