Core Java for Everyone

A systematic incremental approach to become software engineer for any student or graduate & every enthusiast!

No familiarity of C/C++ or any programming language is required.

Madhusudan Mothe
M.Tech., IIT Bombay

Foreword by Pravin Rao, COO, Infosys.

Core Java

for

Everyone

Madhusudan Mothe B.E. (Govt. College of Engineering, Pune),
MTech. (IIT Bombay)

Core Java for Everyone

ISBN: 9781688558670
First Edition: September 2019
MRP: ₹ 699

Quality Knowledge Publisher.
qualityknowledgepublisher@gmail.com

Paperback copy of this book can be purchased from Amazon.
http://bit.ly/javaforall

Foreword

The first computer program for an electronic computer was written somewhere in 1950s. Since then the programming discipline has evolved so much that it is now referred to as the new literacy of 21st century – Programming is for everyone. As world-renowned theoretical physicist and cosmologist Stephen Hawking has put it - *"Whether you want to uncover the secrets of the universe, or you just want to pursue a career in the 21st century, basic computer programming is an essential skill to learn."*

In last seven decades, world has witnessed several programming languages from ABC to Z Shell, but Java is one of the most versatile and ubiquitous programming language. It has been used by most of the fortune 500 companies to build their enterprise apps.

There are several books out there for learning Java programming. However, to learn Java in a systematic and lucid manner, it is important to use the right reference. 'Core Java for Professional', authored by fellow Infoscion – Madhusudan Mothe, serves as an excellent handbook to help one build Java programming skills with pragmatic insights and guidance.

Whatever is your inspiration to learn Java programming, I hope this book will help you to guide and accomplish. Enjoy the reading!

U. B. Pravin Rao

Chief Operating Officer, Infosys Limited

Preface

When any beginner wants to study Java as his/her first programming language, many of us advice them to study C / C++ prior to start studying Java. This book provides smooth path to learn Java directly without studying C/C++.

Let's imagine that you are sitting in this plane to start the journey of Java programming world ☺

I welcome all of you to a tour of Java programming world! We are in the position to take off.
Please, buckle up your seat belts and get ready for take off! You are quite known with a fact that the speed of a plane gradually increases and attains higher speed of 1000km/hr or more. What will happen if the plane takes a take off with high speed like 1000 km/hr? It will simply crash. Similarly, if the complicated concepts are explained without building strong foundation of basic concepts, your confidence may crash. Care has been taken to illu strate fundamental concepts of Java programming in a systematic and lucid manner. **Explanation of complicated concepts in easy words is the major strength of this book**

Many people have done so much for me over the years and it is not possible to mention all these names here due to space constraint. Hence, I am mentioning few of them ☺ ***I do understand that the list is quiet lengthy, but, I take this opportunity to acknowledge them.***

I acknowledge Chandrakant Kunjir (Joint Director, Technical Education, Maharashtra), Chandrakant Khilari, Nana Nivangune, Shrikant Malegaonkar, Mahadeo Kadam, Santosh Gaikwad, Rajodhan Khalate, Suyog Raut, Soumyajit Sengupta , Sandip Chorage, Balasaheb Deshmukh, Sudhir Ghorpade, Satyendra Singh, Vikram Patil, Ramesh Pimple, Shyam Chavan, Vinod Pardeshi, Ashok Sahane, Bhagyashri Chandrakant Patil, Rekha Prabhakar Nhalade, Sidgonda Karigar, Vishnu Katkar, Pratik Dattatray Patil, Vijay Bhagwan Kolekar, Mrutyunjay Rajendra Ranage, Krantisinh Dilip Khandekar, Anjali Ranjeet Gavade, Murthy Shivaraj Patil, Ranjana Harischandra Gholve, Suresh Bhilawade, Navnath Shankar Lokare, Sumangala Pawar, Satish Kamble, Satish Karande, Shankar Sagare ,Ghanshyam Adsul, Nitin Raval, Shivaji Gavade, Gajanan Pathurde, Rupesh Pawar, Neeraj Thorat, Pramod Shinde, Pandurang Desai, Prakash Baburao Siddh,Vilas Khubannavar, Appasaheb Pujari, Dadaso Nargatte, Balasaheb Kushappa, Anna Awate, Anant Desai, Bajirao Hupare, Laxman Pujari, Dhondiram Mali, Sanjay Hukire, Jitendra Shirguppe, Namadeo Mane, Balasaheb Kamanna, Ajay Kumbhar, Yuvraj Mhakave, Nikhil More, Siddhu Devaba, Rajaram Birnje, Mahesh Vasudev, Sachin Sontakke, Hemant Sangaokar, Amit Kumbhojkar, Siddharth Patil, Ajay Jadhav, Shailesh Narvekar, Vishal Mehata, Rahul Mahajan, Rajendra Jadhav, Shivagonda Awate, Amar Chile, Vijay Pataskar, Vijay Naik, Amit Mahadar, Sejal Shah, Priyanka S., Delekar family. and many more for their support. Medicos Dr. Sanjay Desai, Dr .Suhas Kulkarni, Dr. Nilesh Puri, Dr. Sanjeev Dasarao, Dr. Sunil Bandgar, Dr. Parameshwar Kharat, and Ganesh Patil have been always helpful..

I am thankful to Infoscions Pravin Kulkarni (VP), Rajneesh Malviya (SVP) , C.Devarajan, Anand Sinha (Ex-AVP), Lt.Col.Yogesh Joshi (Retd), Prasanna Mulmule, Dakshinamurty Y., Tushar Bhalerao, Prasad Nazare, Vinod Sankaran Nair, Julie Karmakar, Prakash Somaiya, Suresh Sonawane, Shraddha Dhamangaonkar, Anant Sabane, Anand Pardeshi, Abhijit M.Naik , Sandeep Rahul, Kailas Tile , Sarabjeet Singh Vig etc. as well as carpool companions Sameer Chaugule, Pushakar Barve, Rajan Phatak, Madhura Velankar, Aditya Dhole, Chaitali Patil, Ajay Deokar, Amogh Pandharipande, Nitin Doiphode, Akshay Marane, Rahul Paranjape, Sandeep Bhavsar, Ganesh More, Gururaj Sagar, Shripad Dhole, Rohit Bhave, Chaitanya Khadilkar, Niranjan Phansalkar, Makarand Shete, etc. I also want to acknowledge tea group members Madhav Dabke, Durgesh Ghotagalikar, Suhas Pachore, Tushar Samanerkar, Hrishikesh Mhaiskar. I am also thankful to chairman Sachin Deshmukh, secretary Uttamrao Jagdhane and all the members of Runwal Society for their co-operation.

I take this opportunity to say thanks to academic professionals Dr.B.P. Bandgar (Ex.Vice Chancellor, Punyashlok Ahilyadevi Holkar Solapur Vidyapeeth),Dr.Sudhakar Nhalade, Dr.Dattatraya Parle, Dr. Bhakti Raghavendra Umarji, Dr. Dhananjay Talange, Dr. Balasaheb Patre, Dr. Aniket Khandekar, Dr. Ashok Gaikwad, Dr. H.T. Jadhav, Dr. Sandip Narote, Dr.Yashwant Kolekar, Dr.Rohit Ramteke, Dr.Sudhir Agashe, Dr. Sanjaykumar Patil, Dr.Uttam Chaskar, Dr.Sanjay Shendge, Dr. Sandeep Meshram, Dr.Sanjay Koli, Dr. Sanjay Patil and Bharati Narute for their encouragement.

I must express my gratitude to the tremendous co-operation given by Ashok Kumar Ratnagiri, Sujatha Yakisari, Santhosh K.B, Deepali Tannu, Vivek Mhaswade and all members of Edgeverve Security & Performance Engineering Team. All my colleagues have been very understanding & supportive.

It gives me pleasure whenever I interact with my COEP friends Abhijit Deshpande, Jayant Mhetar, Anil Kulkarni, Amit Lele, Shantanu Joshi, Paresh Deshpande, Shailesh Kamble, Suyog Bhokare, Vivek Joshi and Ranjit Hogade mainly on social media. The same is true with all my Private High School & V. S. Khandekar schoolmates.

I am also thankful to Kartik Dutt, Dipal Shah, Dhruvi Shah,, Tejaswi Mahajn, Shailesh Suresh Patil, Ishani Chakradeo ,Sayali Kadam,Varsha Gharu, Sohail Khan, Lovely Gangwar, Seema Chavan, Mohini Thor, Nikhil Patange, Purva Loharkar, Pravin Mohature, Ayushi Sharma, Ketaki Bakshi, Himanshree Nagavekar, Ankit Ranjan, Siddhi Jadhav because of whom I could make this book better.

Thanks to my former colleagues Sachin Daftary, Gurunath Kulkarni, Kiran Laturkar, Atul Kahate, Manish Inamdar, Prashant Khisti, Manoj Madhav Apte, Atul Edlabadkar, Nitin Sapre, Dattaprasad Vaidya, Abhijit Mahale, Amarendra Gokhale, Pundlik Jumbad, Sanjay Phanshilkar, Balgonda Hulyalkar, Ganesh Ramakrishnan, Milind Bhimarao Joshi, Shailendra Kumar, Abhijit Bhalerao, Tushar Surve, Mandar Gore, Ashwinikumar Buche, Sameer Kajale & Bipin Kumar.

This book will remain incomplete if I do not acknowledge to my parents, wife, sister Sharvari, younger brother Kulbhushan, all members of Mothe family, relatives & native of Pattankodoli who have been standing behind me like a wall. I would like to mention my niece Skyla Wade and nephew Ayush Mothe who always makes me happy whenever I play with them ☺

The explanation of complicated concepts is provided in a lucid language which becomes the major strength of the book. I hope that readers will benefit from this book in maximum possible way. Readers can email me on **mothemadhusudan@gmail.com**. I hope that readers will benefit from this book in a maximum possible way.

Madhusudan Mothe
http://bit.ly/mothemadhusudan

Guide to Readers:

Q. I have never done programming as of now. Can I read this book?

Yes, you can read this book from chapter 1. If you are familiar with basics related to computer, you can even directly start studying from chapter 2.

Q. I have done programming in C/C++ etc. Is this book useful?

Anyone who already have exposure of little bit programming can directly read 7th chapter.

Q. I am already Java programmer. Is this book
useful to me?

Anyone who is already Java programmer can read “Dynamic Polymorphism” chapter & see if there is any value addition in his/her knowledge.

If you like book content, you may click below Amazon link & appreciate this book.

https://www.amazon.in/Everyone-MTech-Bombay-Madhusudan-Mothe/dp/B07XBL4SVL/ref=sr_1_2?qid=1567584411&refinements=p_27%3AMadhusudan+Mothe&s=books&sr=1-2

Or

http://bit.ly/javaforall

Reader can skip this chapter if he/she is familiar with basics of computer and start chapter 2

Chapter 1

WELCOME TO COMPUTER AND PROGRAMMING WORLD!

1.1 Why Computers?

Imagine that you are at Mumbai V.T. station at 2 o'clock in the midnight. Sleeper class tickets for Delhi express are fully booked and only first class tickets are available. You need Rs. 2,000 more to purchase a first class ticket. Where can you get these 2000 rupees at midnight? Of course! ATM. Thanks to a computer technology which has made human life easier and luxurious!

Computers have become an integral part of the human life now-a-days. **Automated Railway Reservation System** is a well-known practical example of applications of a computer. You can book railway ticket from internet at anytime from anywhere. Computers are used in **hospitals** to monitor and track patients, doctors, other staffs etc. You can play various **games** on a computer. Application software like **Microsoft word** can be used to create various types of documents. In a nut shell, computer technology is a boon to human life and hence study of computers is important for everyone.

Let's add two numbers, say, 5 and 12. Your brain can do this calculation very easily. Now, tell me what is the addition of 13579 and 2468? It's difficult for a normal brain to process it quickly. However, computer can perform this addition in fraction of a second. Further, suppose you have to add at least such 1000 numbers, this repetitive calculation job will be tedious and there can be mistakes. One can definitely rely on a computer to do this job in fraction of seconds and that too without any mistakes.

The great scientist, Blaise Pascal's father was working in a tax department. He had to spend nights in doing calculations. He faced similar tedious situation of repetitively adding tax values and hence invented the calculator in 1643. It was used for calculation purpose till 1799 in Europe. Then, Charles Babbage, famously known as a **Father of a computer**, developed analytical Engine performing multiplication and division.

1.1.1 What is a Computer?

Computer = compute + er

Computer is a general purpose machine (electronic device) that does accurate (repetitive) computations million or even billion times faster than normal human being. The calculation can be of arithmetic or logical[1] type.

[1] Logical calculations will be discussed in the next chapter.

Atanasoff-Berry invented First Electronic Digital Computing Device in 1937 that was capable of solving up to 29 simultaneous linear equations.

We have just read that computer does computations. It can add any 2 numbers like 13579 and 2468 in a fraction of a second. Being a beginner***, let's try to utilize a computer to add 2 simple numbers and gradually gain the knowledge to become a best programmer***. We need to give set of instructions in a logical order to perform this task of addition.

Program is a set of instructions arranged in a logical sequence to perform a certain task.

Before using a computer, let us quickly understand the manual process[2]♣ of the same which everyone knows.

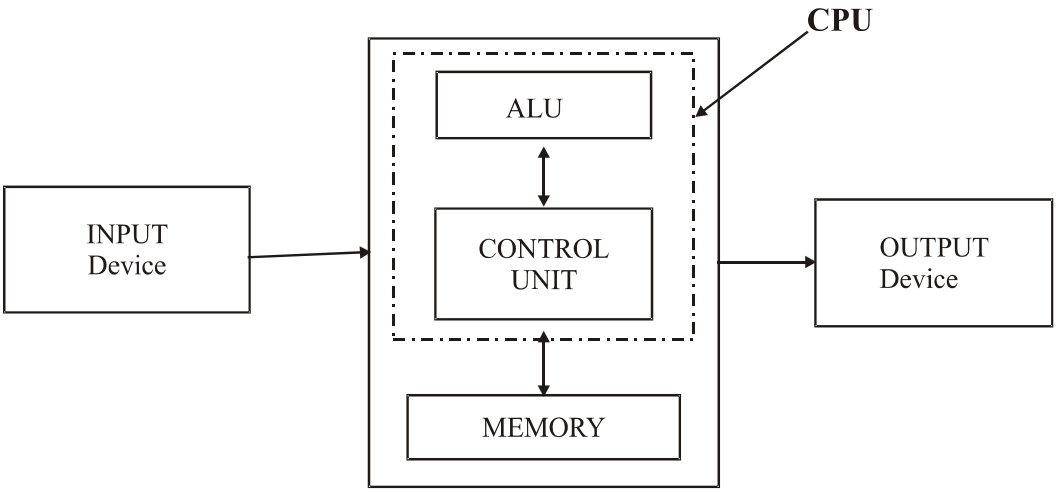
Input Data[3] : 2 Numbers
Output Data[4] : Number which is the addition of these 2 numbers

If you want to ask the addition of 2 numbers to your friend, following steps are involved.

- You will tell 2 numbers to your friend which he/she will hear it by an ear.
- Friend's brain will store and add these 2 numbers.
- He/she will communicate the addition of these 2 numbers by speaking to you.

Ear, mouth and brain acts as input, output and processing unit of a human being respectively.
As we want to get this execution done from a computer, let's understand input, output and other units of a computer.

1.1.2 Block Diagram of a Computer



Computer can be logically divided into following four parts.

1.1.2.1. Input Unit

We need to input two numbers to a computer to get the sum of these two numbers. What are the parts of a computer that can receive this input data? Computer receives information through input devices like

[2] Manual process is the process executed by a human being.
[3] Data received by a computer from user is referred as an input data.
[4] Data communicated (displayed) by a computer on a monitor, printer etc. is referred as an output data.

- Keyboard
- Mouse
- Speaking to a computer
- Joystick
- Track Ball
- Scanner
- Network (One can access a file on the network located on the other computer.)

For the time being, beginner may consider keyboard, mouse as i/p devices.

1.1.2.2. Memory Unit

Memory is a unit of a computer where data is stored.
In case of the above addition of two numbers program[5], user enters two numbers as input, let us say through keyboard. When this data is successfully received by a computer, it is saved in a memory.

Computer memories can be categorized as
I. Primary Memory
II. Secondary Memory

I) Primary Memory

Primary memory is directly accessed by a CPU. It is volatile in nature means data stored in it is lost if the power is switched off (Exception: ROM). Primary memories can be further classified as

i. RAM (Random Access Memory)

It provides random access to data stored in it. Data stored on RAM can be changed.

The main feature of RAM is that the time required to retrieve data from any memory location is constant irrespective of the memory location address. That's why it is known as RAM.

ii. ROM (Read Only Memory)

It is a read only memory. Once information is stored on ROM, it cannot be easily modified or erased.

II) Secondary Memory

It is not directly accessed by a CPU. It is a permanent storage of the data that is nonvolatile in nature. It is also cheaper and lesser in speed than primary memory.

Secondary memory devices are magnetic tape, diskette, hard disk, CD ROM etc.

When you ask mobile number to someone, can you note it or just store it in your brain forever? It is not possible for a normal brain to store all the numbers. Hence, we note the data in a diary. **Here, human brain acts as a primary memory, while, diary plays a role of secondary memory.** We can retrieve the data from the brain very quickly as compared to the data written in the diary. Diary, notebook etc. acts as secondary memory for the human being.

When someone calls you, you hear it by ear and your brain takes certain action based on the message it receives. There must be certain programs existing in human memory by birth which contains instructions to recognize voice, to produce certain voice by mouth etc. These programs are always alive for normal human being and must be residing in read only memory of human being. Similarly, ROM stores the data that cannot be erased. ROM contains the programs that are required for the functioning of a computer. e.g. BIOS program starts a computer.

[5] Program is a set of instructions arranged in a logical sequence to perform certain task. e.g. We need to write set of instructions in a logical sequence to add the numbers 13579 and 2468.

FAQ[6]

Q. It is true that we cannot forget the image of our best friend even if we try our best to forget it. Can't it be an example of read only memory of human being?

When we revise certain study material, we won't forget it. However, if we do not revise it, we forget it slowly. Similarly, you will slowly forget the image of your friend if you do not remember or memories him/her. Practically, it is unavoidable to remember the close friend and his/her image gets revised in our memory. Hence, we cannot forget it. Therefore, it is not an example of ROM of human being.

1.1.2.3. Central Processing Unit (CPU)

It is the overall administrator (co-coordinator) of a computer.

CPU has two main parts.
1. Control Unit
2. ALU

I. Control Unit

The control unit (CU) reads instructions from memory, decodes[7] and executes.
When the instruction requires to store data into a memory, it receives the data from input device like a keyboard and stores it into a memory. When the instruction requires calculation, it passes the required information to the ALU.
It is also referred as a brain within brain because based on the result of the execution of the instruction by control unit, operations of other parts of a computer takes place.

CPU speed is a feature that tells how quick CPU can execute an instruction.

The speed of the CPU is 2.0 GHz. It means the CPU can execute 2×10^9 instructions per second.
Many today's computers have multiple CPU's.

II. Arithmetic and logic unit (ALU)

It can perform arithmetic and logical calculations on data. For example, it can add two numbers together (in binary).

1.1.2.4. Output unit

Once this calculation is done by a computer, the calculated sum (output) can be sent to the following output devices.

- Monitor/VDU
- Printer
- To control other devices
- Network like internet
- Soundcard
- Film
- Video
- Robot arms etc.

[6] **FAQ** stands for general/frequently asked question/questions. This questions generally pops up in reader's mind while reading or understanding a program. **This approach is used to effectively deliver the knowledge.**
[7] Convert the instruction into binary language.

For the time being, beginner may consider monitor, printer as output devices.

Let's compare Primary and secondary memory.

Primary Memory	**Secondary Memory**
CPU can access it directly.	CPU cannot access it directly.
It is volatile (Exception: ROM).	It is permanent.
The speed is high.	The speed of accessing the data stored in secondary memory is slow.
The cost is more.	The cost is less.
RAM, ROM is an examples of primary memory.	Secondary memory devices are diskette, hard disk, CD ROM etc.

Let's compare RAM & ROM.

RAM	**ROM**
The data stored on RAM can be modified.	The data stored on ROM cannot be modified easily.
It stores data temporarily. When the machine is switched off, the data stored on RAM vanishes.	It stores data permanently.

1.2 Machine Language

Hope you have understood the basic concepts explained in previous sections.

Let's go one step ahead and try to use computer practically to add two numbers. It is necessary to give right instructions to a computer to add two numbers. Oh, but, can a computer directly understand the instruction "Add two numbers and show the result?" No. We must have to give this instruction in the language which computer can understand. Then, what is the language of a computer?

Computer can only understand binary language known as a machine language. Binary language has only 2 digits i.e. 0 and 1.

Hence, the following instructions need to be written in a machine language to add two numbers.

Do not try to understand the following instructions; just see that how complicated and messy instructions (code) are written in a machine language to add two numbers.

Binary Instruction	**Meaning**
010 0 001101	Store the value of the Accumulator in memory location 13.
001 1 000101	Load the value 5 into the Accumulator.
010 0 001110	Store the value of the Accumulator in memory location 14.
001 0 001101	Load the value of memory location 13 into the Accumulator.
011 0 001110	Add the value of memory location 14 to the Accumulator.
010 0 001111	Store the value of the Accumulator in memory location 15.
111 0 000000	Stop execution.

Machine language is the basic language of a computer which is directly understood by it.
It consists of binary numbers that tells computer to execute basic operations like printing, adding 2 numbers etc.

Advantages of Machine Language:

1. Though it is tough to write programs in machine language, the execution is fast.
2. It is directly understood by a computer. Other languages are not directly understood by a computer.

Disadvantages of Machine Language:

1. As one has to write the instructions in the number format, chances are always there of doing mistakes in the program.
2. Machine languages are specific to machine. It means the program executed on one machine cannot be executed on the other machine because the binary representation for various instructions are different for different machines.

1.3 Assembly Language

Human being is inventive and innovative by nature. Machine language instructions are in the form of the numbers (that too only 0 and 1) and difficult to read. People started to use English like abbreviations (Mnemonics) to give instructions to a computer to perform a particular task.

Binary instruction or set of instructions are mapped to English abbreviations known as mnemonics.

You can just have a glance on below equivalent assembly language instructions for the above machine language instructions.

```
mov     a1,0Ah
mov     b1,14h
mov     eax, a1
add     eax, b1
mov     c1, eax
```

Assembly language is a machine-orientated language in which mnemonics (symbolic names) are used to represent each machine-language instruction. It is a programming language that is one step above machine language. Each CPU has its own specific assembly language. Hence, assembly language program executed on one machine cannot get executed on the other machine of different processor (CPU).

Advantages:

1. Assembly language is easy to write and also chances of making mistakes in writing instructions are less as compared to machine language programs.

Disadvantages:

1. Assembly language instructions (code) is not directly understood by a computer. It needs to be converted into machine language instructions by another program called as an assembler.
2. Though easier than machine language, writing programs in assembly language is complicated and messy.

Machine and assembly languages are low level languages.

FAQ. Does computer directly understand assembly language instructions?

No. Computer can only understand binary language. Hence, the assembly language instruction needs to be converted into a machine language instruction.

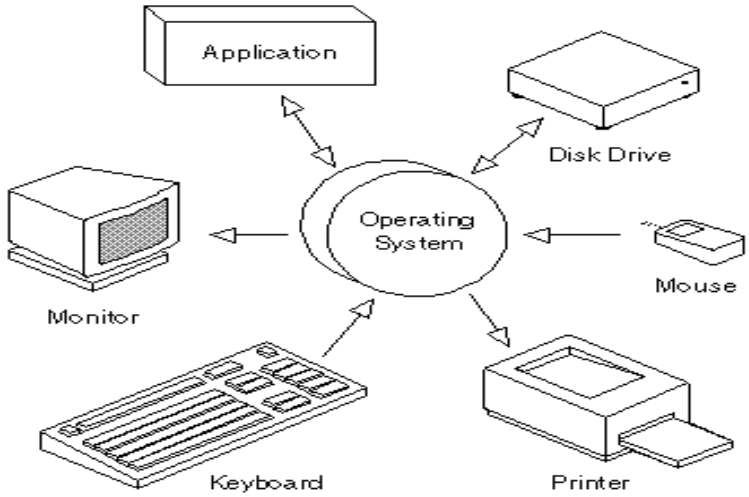
Assembler is a program that converts assembly language instructions into machine language instructions.

As seen above, one has to write so many instructions even to write a simple program. Further, middle and high level languages are developed to speed up the programming.

1.4 Operating System

An operating system (OS) is a set of computer programs that manages the hardware and software resources of a computer. It is heart of the computer.
Operating systems perform basic tasks, such as recognizing input from the keyboard, sending an output to the display screen, keeping track of files and directories on the disk, controlling peripheral devices such as disk drives and printers etc.

Windows, Unix, Linux, Macintosh etc. are the operating systems.
Unix is a multiuser system. Many users can work on the same machine by connecting through terminals. It means all the users share the CPU and memory of one machine at the same time.



1.5 Points to Remember

1. Computer performs arithmetic and logical calculations.
2. Primary memory is directly accessed by CPU. It is volatile (Exception:ROM) in nature.
3. The source code program (e.g. add.java) is stored in a secondary memory. i.e. inside C:\ or any other drive. Hard disk is an example of secondary memory.
4. When executable file (e.g. add.exe) is executed (just click on it), it gets copied into the RAM.

   The program that copies the executable file from secondary memory to a primary memory (RAM) is called as a loader.

5. An operating system (OS) is a set of computer programs that manages the hardware and software resources of a computer

FAQ:

Q. Can computer directly understand the above add.java program and display the addition of 2 numbers as output?

Basic language of a computer is a binary language and it can directly understand binary instructions. Computer cannot directly understand Java language instruction. Hence, we need to convert these instructions in binary language.

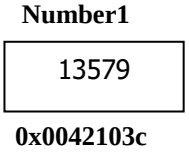
Q. Why computer can directly understand 0 and 1 and not any other numbers or alphabets?

Computer is made up of transistors which are used in on-off mode. When transistor is on, it is treated as 1 and when transistor is off, it is treated as 0. That's why computer can only understand 0 and 1.

Q. Who finally executes binary instructions inside executable file?

CPU reads every instruction from the instruction register and executes it one by one.

Q. Computer understand only 0 and 1. Then, how Number1 is stored in a decimal format as shown in below figure?



The above representation is only for understanding purpose at a beginner level.

Computer understand only 0 and 1. Data is stored in a memory. Memory is divided into units referred as memory locations. Each memory location can store maximum 8 bits. Every memory location is identified by a unique address. e.g. 0x0042103c is an address. Address is a unique integer number.

When any data is stored in a computer, memory should be allocated. In case of an integer, 4 consecutive memory locations are allocated. i.e. the size of an integer data type is 4. The integer value 13579 is converted into binary form. i.e.00110101 00001011. Hence, it is actually stored as

00000000	00000000	00110101	00001011
0x0042103f	0x0042103e	0x0042103d	0x0042103c

Little Endian Address Representation of 13579

Integer value can be positive or negative. Hence, the last bit i.e. 32th bit indicates the sign of the integer value stored. If 32th bit is 0, the integer value is positive, else it is negative.

1.6 Deep Knowledge Section[8]

Q 1. Why RAM is faster than hard disk memory?

RAM is IC based data storage. Hence, it allows to access the stored data in any order- means randomly and **without any physical movement of the storage medium / reading head.** In case of secondary memory like hard disk, data is accessed sequentially. Hence, it takes time to move the read/write head from one location to other memory location from where data needs to be read. Physical movement of head and physical location of data causes more access time for secondary memory data access.

Q 2. How computer starts?

i. When you turn on your computer, the **microprocessor** passes control to the **BIOS** program which is always located at the same place on (Erasable Programmable Read-only Memory) EPROM.
ii. Bios does power-on self-test (POST) to make sure all computer's components are operational.
iii. BIOS looks for the system files at a specific place on your hard drive and copies information from it into specific locations in RAM. This information is known as the **boot record** or Master Boot Record.
iv. The boot sector on a disk is always the first sector on the first track on the first head.

[8] Deep knowledge Section should be read after developing average level in Java. Beginner may skip this section in first reading.

v. It then loads the boot record into a specific place (hexadecimal address 7C00) in RAM.
vi. The boot record contains a program that BIOS now branches to, giving the boot record control of a computer.
vii. The boot record loads the **initial system file** (for example, for DOS systems, IO.SYS) into RAM from the diskette or hard disk.
viii. The initial file (for example, IO.SYS, which includes a program called SYSINIT) then loads the rest of the operating system into RAM.
ix. The initial file (for example, SYSINIT) loads a system file (for example, MSDOS.SYS) that knows how to work with the BIOS.
x. One of the first operating system files that is loaded is a system configuration file (for DOS, it's called CONFIG.SYS).
xi. Another special file that is loaded is one that tells which specific applications or commands the user wants to have included or performed as part of the boot process. In DOS, this file is named AUTOEXEC.BAT. In Windows, it's called WIN.INI.
xii. After all, operating system files have been loaded, the operating system is given control of a computer and performs requested initial commands and then waits for the first interactive user input.

Questions

Q.1 State True or False.

i. Computer cannot think.
ii. Computer can make a mistake.
iii. Data stored on a RAM can be changed.
iv. Information stored on ROM cannot be erased.
v. Data on secondary memory is volatile in nature.
vi. ALU performs arithmetic and logical calculations on data.
vii. Program is a set of logical instructions used to perform a certain task.
viii. Computer can only understand binary language known as a machine language.
ix. Machine and assembly languages are low level languages.
x. Computer can directly understand the assembly language code.
xi. Assembler is a program that converts assembly language code into machine language instructions.
xii. An operating system (OS) is a set of computer programs that manage the hardware and software resources of a computer.

Q.2 What is primary memory? What is RAM and ROM ?
Q.3 What is secondary memory?
Q.4 What is ALU?
Q.5 What is program? What language computer can understand directly?
Q.6 What is an assembler? What it does?
Q.7 What is an operating system?

Answers:

Q1. i. True ii. False iii. True iv. True
v. False vi. True vii. True viii. True
ix. True x. False xi. True xii. True

General Information

A. Brief History of Computer Technology

A complete history of computing would include a multitude of diverse devices such as the **ancient Chinese abacus**. Abacus was the first device known to carry out the calculations. It works by sliding beads back and forth on a frame with the beads on the top of the frame representing fives and on the bottom ones.

Charles Babbage's analytical engine (1834). This machine can perform all the four operations that are addition, subtraction, multiplication and division.

In 1960s, analog computers were routinely used to solve systems of finite difference equations arising in oil reservoir modeling. In the end, digital computing devices proved to have the power, economics and scalability necessary to deal with large scale computations. Digital computers now dominate the computing world in all areas ranging from the hand calculator to the supercomputer and are pervasive throughout society.

The advent of **the first electromechanical computers** was an exciting time, because these scamps could actually provide significant amounts of computational power. Development of electromechanical computers continued into the 1960's. The Minivac 601 was a model introduced in 1961 with its name derived from the Sperry Rand Univac computer.

The **evolution of digital computing** is often divided into generations. Each generation is characterized by dramatic improvements over the previous generation in the technology used to build computers, the internal organization of computer systems, and programming languages. Although not usually associated with computer generations, there has been a steady improvement in algorithms, including algorithms used in computational science.

B. History of computers

ABACUS(2400 B.C.E.)	**Anonymous Chinese**	**It is the earliest known tool for use in computation.**
Calculating Clock (1623)	**Wilhelm Schickard**	**First Automatic Calculator. Performing Addition and Subtraction.**
Pascal's Calculator (1643)	**Blaise Pascal**	**Used for Taxes in France Until 1799. Performing Addition, Subtraction.**
Analytical Engine Difference Engine.	**Charles Babbage**	**First to conceptualize and design a fully programmable computer early as 1820**
Hollrith and tabulating Machine.(1890)	**Harman Hollrith**	**Device could automatically read census information which had been punched onto card.**
First Electronic Digital Computing Device(1937)	**Atanasoff-Berry**	**The machine, conceived in 1937, was capable of solving up to 29 simultaneous linear equations and was successfully tested.**

Chapter

2

WELCOME TO JAVA PROGRAMMING WORLD!

2.1 History

Java programming language was originally developed by a company called as Sun Microsystems. Java development project was initiated by James Gosling in June 1991 for use in one of his many set-top box projects. It was released in 1996 as core component of Sun Microsystems's Java platform (Java 1.0 [J2SE]). Current Java version is 12.

The language, initially called Oak after an oak tree that stood outside Gosling's office, also went by the name Green and ended up later renamed as Java, from a list of random words.

Sun released the first public implementation as Java 1.0 in 1996. It promised **Write Once, Run Anywhere** (WORA[1]), providing no-cost run-times on popular platforms.
On 13th November 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL).

On 8th May 2007, Sun finished the process, making all of Java's core code free and open-source, aside from a small portion of code to which Sun did not hold the copyright.

On 27th May 2010, Oracle bought Sun Microsystems & ownership of Java is with Oracle till then. The latest Version of java is Java SE 12.

There were five goals[2] in the creation of the java language.

1. It should be" **Simple**, **Object Oriented**".
2. It should be "**Robust and Secure**".
3. It should be "**Architectural- neutral and Portable**"
4. It should be "**High Performance**".
5. It should be "**Interpreted Multi-threaded"**.

[1] You will able to understand meaning of WORA later after studying Java. If you do not understand open source etc. in this "History" section, you can ignore at this stage.

[2] It is just mentioned for your reference; you will understand meaning of the above mentioned five goals after studying Java.

Version:

1. Jdk 1.0(January 21 ,1996)
- Jdk 1.1(February 19, 1997)
- J2SE 1.2(December 8, 1998)
- J2SE 1.3(May 8, 2000)
- J2SE 1.4(February 6, 2002)
- J2SE 5.0(September 30, 2004)
- Java SE 6(December 11, 2006)
- Java SE 7(July 28, 2011)
- Java SE 8(March 18, 2014)
- Java SE 9(September 21, 2017)
- Java SE 10(March 20 ,2018)
- Java SE 11(September 25,2018)
- Java SE 12(March 2019)

James Gosling

2.2 Embarking Java Programming: First Java Program

I know you are eager to write first java program, aren't you? Welcome to the world of java programming!

Let's write a java program that performs addition of 2 numbers (i.e. 13579 and 2468) and display it. Though it is a simple program, it will give you exposure to the various aspects of java programming.

Before writing any program, it's better to understand how the same task can be done manually.

You will do this calculation manually as below.

1. You will remember value 13579 as number1 in your memory.
2. You will remember value 2468 as number2 in your memory.
3. You will add number1 and number2.
4. You will say "The sum is = 16047".

In a programmer's language, the above mentioned steps can be written as below.

- Reserve a memory to store a value 13579.
- Store a value 13579 in the above reserved memory.
- Reserve a memory to store a value 2468.
- Store a value 2468 in the above reserved memory.
- Reserve memory to store the addition of 13579 and 2468.
- Add these 2 numbers and store the sum in the above reserved memory.
- Display the addition of these 2 numbers (sum).

Don't get scared with the word "Algorithm". The above 7 steps are collectively known as an "Algorithm" for a program that adds 13579, 2468 and displays the result. History of algorithms is given in the General Information section at the end of this chapter (Refer Appendix).

Algorithm is a set of steps arranged in a logical sequence to perform a certain task.

It's important to know following 2 thumb rules before writing any java program.

Rule 1:

Remember forever, every java program starts with the main method block (command) as below.

```
 public static void main (String args[])
{

}
```

It's not possible to write a java program without main method block. You can think it like a command to a computer at this beginner's learning stage. You will understand the details of this once you read method chapter.

Rule 2:

Every Java program has a driver class as shown below. Driver class always contains main method as shown in Rule 1 above. At a beginner stage, you can just assume that it is the format by which any Java program can be written. Let us name driver class as Test as shown below. Again, you should think it is a command to the computer instead of trying to understand what is class or driver class at this stage. You will understand it in class chapter.

```
class Test
{
      public static void main (String args [])
    {

     }
}
```

Even if you write "public class Test" instead of writing "class Test" as shown above, it is fine. You will understand when to use word "public class" instead of "public" after studying package in 11.8 section of this book. You should not think about it at this beginning stage and can ignore.
Henceforth, we will be using above Test class for writing Java programs in this book.

Step 1: Reserve a memory to store a value 13579.

As 13579 is an integer, we need to reserve a memory that can hold (store) an integer value. Following instruction reserves a memory to hold an integer value.

```
int Number1;
```

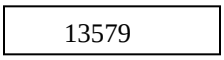
int is a keyword. Every keyword has a special meaning. int keyword specifies to allocate a memory to hold an integer value.

Number1 is referred as an integer variable. Integer variable can store one integer value at a time.

In English, every statement is terminated by a full stop. Similarly, in java, every instruction is terminated by a semicolon (;).

Step 2: Store a value 13579 in the above reserved memory.

```
Number1=13579;
```



0x0042103c

Here, the value 13579 is stored in a memory location 0x0042103c[3] (reserved memory in step 1). Number1 is referred as a variable name.

[3] This number is presented in the hexadecimal system format.

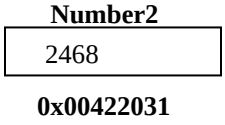
Step 3: Reserve memory to store a value 2468.

Similarly, Number2 is declared as below.

int Number2;

Step 4: Store a value 2468 in the above reserved memory.

Similarly,
Number2=2468;

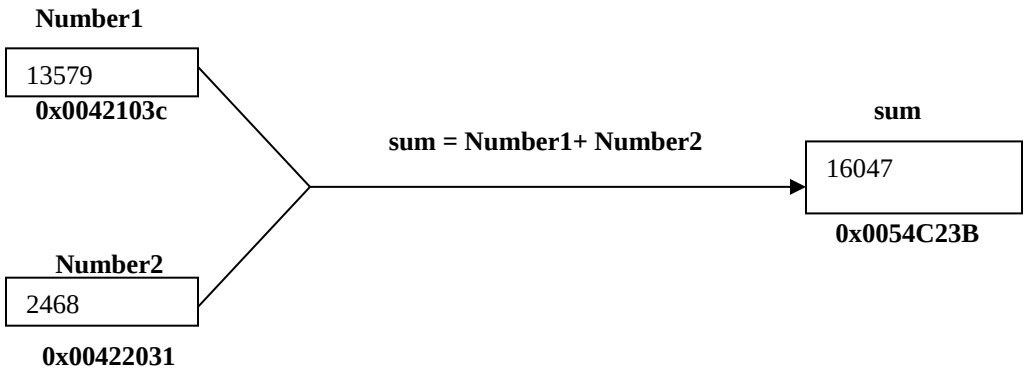


Step 5: Reserve memory to store the addition of 13579 and 2468.

Similar to step 1 and step 3, variable sum is declared as below.
int sum;

Step 6: Add these 2 numbers and store the sum in the above reserved memory.

sum = Number1 + Number2;



Step 7: Display addition of these 2 numbers (sum).

System.out.println(variablename) is a standard command (method[4] or instruction) that displays the value stored at the memory location variablename. on the output device like a monitor. As we want to display the contents of a variable sum, the instruction is

System.out.println (sum);

It is mandatory to declare a method (command) before we use it in a java program. Declaration of a System.out.println method is given in the package[5] java.lang. Following is the format to include any standard package

java.lang.*;

When compiler compiles the instruction containing System.out.println method, it tries to find the meaning of the symbol System.out.println. As the declaration of System.out.println is given in java.lang.*, it compiles this instruction successfully. You will understand compilation process in the later part of this chapter. Java compiler has java.lang package by default, hence, it is not necessary to mention it in the program.

The complete program of addition of 13579, 2468 and displaying the result can be written as below.

[4] Method will be discussed in the chapter 5.
[5] You will understand this in the later part of this book.

Program 2.1:

```
class Test
{
    public static void main(String args[])
    {
        int Number1;                    // step 1 Variable Declaration
        Number1=13579;                  // step 2
        int Number2;                    // step 3 Variable Declaration
        Number2=2468;                   // step 4
        int sum;                        // step 5 Variable Declaration
        sum = Number1 + Number2;        // step 6
        System.out.println(sum);        // step 7
    }
}
```

Output:

16047

FAQ:

Q. Every memory address is uniquely identified by a number like 0x0042103c. Then, why we are referring allocated memory by a name?

As it's difficult to refer memory locations by a number, variable names are used to refer memory locations where data is stored. As the program size increases, it becomes difficult to refer memory locations by an address. Hence, names are used. It makes easier to read and write a program.

Q. What is a keyword? What is an int keyword? Enlist the keywords in Java.

Keywords are identifiers (names) reserved by the language for special use. Every keyword has a special meaning. Java has 50 keywords as given in the below chart.

abstract	**break**	**boolean**	**byte**	**switch**
continue	**assert**	**catch**	**class**	**double**
extends	**enum**	**finally**	**final**	**float**
interface	**if**	**implements**	**import**	**new**
long	**native**	**case**	**char**	**throw**
protected	**public**	**return**	**else**	**short**
synchronized	**for**	**void**	**volatile**	**while**
try	**do**	**int**	**instanceof**	**this**
throws	**transient**	**native**	**strictfp**	**super**
import	**package**	**long**	**goto**	**default**
true	**false**	**null**		

true, false and null are not reserved words but cannot be used as identifiers, because it is literals of built-in data types like integer, float which we will discuss in the section 2.4.

Q. What are literals?

A constant value in Java is created by using a literal representation of it. E.g. 100, 98.2, 'x'.

Literals in Java are a **sequence of characters** (digits, letters, and other characters) that represent constant values to be stored in variables.

Q. Can I use variable names a, b instead of Number1, Number2?

Yes, you can use it. However, as the program size increases, it becomes difficult to understand the meaning of the data the variable is containing. It's better programming practice to use relevant variable names to improve the readability of a program.

Q. What is a method?

Method is a set of instructions arranged in a sequence to perform certain task.
Method is component of a program that provides particular functionality. e.g. System.out.println method used is used to display the value of the variable sum on the screen in above program. Every method has input and gives output accordingly.

Q. What is the meaning of the "// step 1 Variable Declaration"?

It is a comment.

Q. What is a comment? What are the different types of comments?

It is information written in a source code that is ignored by a preprocessor[6] when the source file is pre-processed. Comments improve the readability of the program. It is easy to read and understand a program with comments. Especially, comments help when a new programmer to work on the source code. Comments help at the time of maintenance[7]/enhancement of the application.

There are 2 types of comments.

i. **Single line comment:**
Any text written after // is considered as a single line comment.

ii. **Multi line comment:**
Multi line comments begin with /* and ends with */ Multiline comment can be using to comment single or more lines.

It is possible to assign a value to a variable at the time of declaration itself. Hence, it is possible to declare variable Number1 and assign value in a single instruction.

```
int Number1=16047;
```

The program 2.1 can be written by assigning values to the variables at the time of declaration itself.

[6] It will be discussed in the next section.
[7] Maintenance is a process in which defects in the program are found and fixed. Concept of maintenance of a vehicle is similar to a maintenance of a program.

Program 2.2:

```
class Test
{
    public static void main(String args[])
    {

        int Number1=13579;                  // step 1 & 2
        int Number2=2468;                   // step  3 & 4

        int sum = Number1 + Number2;   // step 5 & 6
        System.out.println(sum);
    }
}
```

Output:
16047

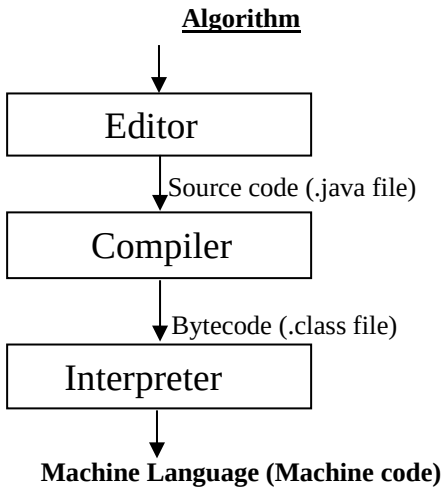
2.3 Typical Java Language Environment

Following are the steps involved in the development of an executable file from writing a Java language program in an editor.

Editor is a program (tool) where one can write instruction/ instructions.

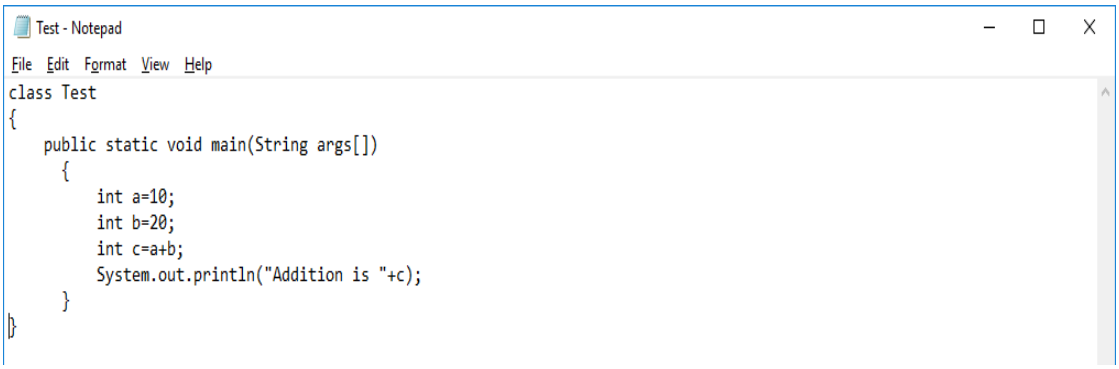
Compiler checks the grammar of every instruction, produces corresponding byte code, creates corresponding class file. e.g. Compiler compiles Test.java into Test.class.

Interpreter interprets the class file (e.g. Test.class in this case) generated by Java compiler and converts into executable code (machine code).



2.3.1 Editor

Editor is a standard program that allows you to edit your source program code. Program is written in a file in an editor. There are many editors available, however, we will use notepad as it is good for beginners.



2.3.2 Compiler

Compiler checks the grammar of every instruction, and produces corresponding bytecodes. It **checks** whether every instruction in a program is written as per the Java language standard **grammar**.

Java has compiler named javac. It compiles java source file (. java file) into bytecodes (.class file). E.g. Test.java is compiled to Test.class. Test.class contains bytecodes.

2.3.3 Interpreter

As you know, computer only understands machine language. After compilation, java program gets converted into .class file. e.g. Test.java is compiled into Test.class. Compiled java .class file contains bytecodes which are not understood directly by a computer. Hence, it is necessary to convert these bytecodes into machine code. Conversion of bytecodes into machine code is done by Java Virtual Machine(JVM). It acts as interpreter of bytecodes into machine code.

Table No:2.1

Program	Input	Output	Functionality
Editor	Algorithm	Source Program	Instructions in Java language are written in a file using an editor. A file where a program is written is known as a **source program**. e.g. Test.java
Compiler	Source Program	Bytecode	Compiler **checks** whether every instruction in a program is written as per the Java language standard **grammar**. It generates compile time error/warnings in case of any discrepancy. Java compiler produces an intermediate code known as bytecode. e.g. Java compiler compiles Test.java into Test.class
Interpreter	Bytecode	Executable File	Java has interpreter called as JVM (Java Virtual Machine). It converts the bytecodes generated by Java compiler (.class file e.g. Test.class) into machine code.

Assume that you have saved this program in a file Test.java in a directory, say "programs" in a C drive. The complete path of the program is "C:\ programs \Test.java".

Directory is a location on a computer where files and other directories (sub-directories) can be stored.

A directory helps to manage the data systematically on a computer.

If you keep your books and other belongings all over the house, it will be tough time for you to locate it later. Similarly, if you save your java programs at different places on a computer, you will mess up and will not be able to locate it when you need it. Better way is to save all java programs in the same directory and retrieve them as and when required.

Every Java source program file name ends with .java extension. E.g. Test. java

There are 3 types of error that may occur during writing a java program.

I. Syntax Error

When any instruction in the program is not as per the Java language grammar rule, compile time error occurs. e.g.

- Misspelled variable and function(method) names
- Missing semicolons
- Improperly matches parentheses, square brackets, and curly braces
- Incorrect format in selection and loop statements

There are many ways by which this type of error can occur. You can observe this error by removing semicolon from an instruction or by erasing one of the curly brace in the above Test.java program.

II. Runtime Error

Runtime error occurs while executing any instruction in the executable file. e.g.

- Dividing a number by zero. i.e. int a=0; int k=90/a;
- Trying to open a file that doesn't exist. You will study this in File chapter.

Execution of this instruction will give you runtime error.

III. Logical Error

The aim of the above program 1.1 is to display the sum of the 2 numbers. i.e. 13579 and 2468. If we write the instruction System.out.println(Number1) instead of System.out.println(sum); you will get incorrect result. If you use – (minus) symbol instead of +(plus) symbol by mistake, it will give you wrong results which is also a logical error.

Common examples where logical error may occur are:

i. Multiplying when you should be dividing
ii. Adding when you should be subtracting
iii. Opening and using data from the wrong file
iv. Displaying the wrong message

Do not get discouraged while compiling your first program. You will learn more if you come across many errors initially.

You must be familiar with the name **Thomas Edison** who invented the bulb. He failed 10,000 times to produce an electric bulb and then successfully invented first electric bulb. When someone asked him about his failure, he said" Now, I know 10,000 different ways by which bulb cannot be produced." **Take motivation from him and do not get nervous even if you initially fail many times to compile & execute a Java program.** You will understand different types of errors that become hurdle to compile a Java program. This experience will help you to develop big sized programs with ease.

2.4 Java Data Type

We have already used integer data type in the programs 2.1 & 2.2. In these programs, we have created Number1 variable of integer data type to store value 13579. Similarly, we have created integer variable Number2 to store value 2468. Further, we have created integer variable sum to store addition of these two numbers. In a nut shell, we are familiar with integer data type.

Now, can we store decimal values like "11.31" or characters like 'x' by creating an integer variable? No, Reason is simple. In practical life, we store books in the shelf. Do we store liquid in the shelf? No, it's not possible. We require container to store liquid. Similarly, Java supports various data types so that programmer can store various type of values depending upon the need by creating variables of those data types.

Following table shows various data types that Java supports. It specifies range which indicates the maximum and minimum value that variables of that data type can store.

Data Type	Size	Range
byte	1 byte(8 bits)	-128 to 127
char	2	0 to 65,536
short	2	-32,768 to 32,767
int	4	-2,147,483,648 to 2,147,483,647
long	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4	1.4e^-045 to 3.4e^038
double	8	4.9e^-324 to 1.8e^308
Boolean	1 bit	true/false

Table: Primitive Data Types

Size mentioned in the above table specifies memory allocated in terms of bytes for variable created of a specific data type. e.g. When an integer variable is created, 4 bytes are allocated inside a memory.
Let us understand how to create variables of these data types depending upon the need through practical examples.

2.4.1 float

What if someone wants to store decimal values like 11.23? Is there any data type in Java whose variables can store decimal values? Yes, Java supports float data types whose variables are used to store decimal values.

Let us understand below program 2.3 which stores decimal value 11.23 in float variable Number1.

Program 2.3:

```
class Test
{
    public static void main (String args [])
    {
        float Number1;                    // step 1 Variable Declaration
        Number1=11.23f;                   // suffix f is must
        System.out.println(Number1);
     }
}
```

Output:
11.23

It is necessary to put suffix f at the end of the floating point number (decimal value).

Java allows to store decimal values in a range 1.4×10^{-45} to 3.4×10^{38} in variables of float data type. Size of the float data type is 4 bytes.

Data Type	Size	Range
float	4	1.4×10^{-45} to 3.4×10^{38}

2.4.2 char

Now, what if you want to store any character like 'x'? Is there any data type in Java whose variables can store a character? Yes, Java supports char data types whose variables are used to store character values.

Let us understand below program 2.4 which stores decimal value 11.23.

Program 2.4:

```
class Test
{
    public static void main(String args[])
    {
        char x;                    // step 1 Variable Declaration
        x='A';
        System.out.println(x);
    }
}
```

Output:
A

2.4.3 long

We have studied int, float and char data types so far.

Now, let's try to guess the output of the following program.

Program 2.5:

```
class Test
{
    public static void main (String args [])
    {
        int Number1;                    // step 1 Variable Declaration
        Number1=2147483647;
        System.out.println(Number1);
    }
}
```

Output:
2147483647

Yes, the output is 2147483647. Let's make a minor change in this program 2.5 & guess the output.

Program 2.6:

```
class Test
{
    public static void main (String args[])
    {
        int Number1;                    // step 1 Variable Declaration
        Number1=2147483649;
        System.out.println(Number1);
    }
}
```

Output: Exception in thread "main" java.lang.Error: Unresolved compilation problem:
The literal 2147483649 of type int is out of range.

This program 2.6 does not compile. Reason is that integer variable can hold any integer value upto 2147483647. If we try to store integer value greater than 2147483647, compilation error occurs. It is similar to a practical example e.g. when a big guy tries to sit on small chair, it is not possible. Big guy requires big chair to sit or we cannot store 1000 litre water in 1 liter container.

Variables of integer data type can hold maximum value upto 2147483647. This is the upper limit. Immediately, question will pop up in your mind "Is there any lower limit as well?" Yes, variables of integer data type can hold values upto -2,147,483,648 in negative side. Variables of integer data type can not hold the values beyond -2,147,483,648. i.e. integer variables cannot hold values e.g.
- 2147483649, -2147483650 & beyond it.

Hence, we can summarise that variables of integer data types i.e. integer variables can hold values in a range -2,147,483,648 to 2,147,483,647.

Data Type	Size	Range
int	4	-2,147,483,648 to 2,147,483,647

As a big guy requires a big chair, similarly, values larger than 2147483647 requires variable of data type which can hold integer values greater than 2147483647. Is there any data type in Java whose variables can hold values above 2147483647? Yes, Java has long data type whose variables can hold integer values upto 9,223,372,036,854,775,807.

Data Type	Size	Range
long	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Long variable requires 8 bytes to store any value as specified in the above table.

Let's try to guess the output of the following program.

Program 2.7:

```
class Test
{
    public static void main (String args [])
    {
        long Number1;                    // step 1 Variable Declaration
        Number1=214748364;
        System.out.println(Number1);
     }
}
```

Output:
214748364

We have studied integer data type that allows to create integer variables. Similarly, long data type allows to create long variables that can hold values greater than variables of integer data type can hold.
Then, you might have thought what is the need of long and integer data types as variables of long data type can hold all the values that variables of integer data type can hold? Reason is that integer variables requires 4 bytes, while, long variables requires 8 bytes. It means it is wastage of space to use long variables where we can use integer variables. You may say that saving of 4 bytes does not have that much negative effect on a program. Yes, you are right. It is true for a small program. However, in a particular program, if we use 1 lakh long variables instead of integer variables, there is wastage of 4 × 1 lakh=4 lakh bytes which is costly affair.

2.4.4 double

Let's try to guess the output of the following program.

Program 2.7:

```
class Test
{
    public static void main(String args[])
    {
        float Number1;                    // step 1 Variable Declaration
        Number1=3.4E38f;                 // 3.4E38 means 3.4x10^38
        System.out.println(Number1);
     }
}
```

Output:
3.4E38

As per the range specified, float variable can hold decimal value upto 3.4×10^{38} f. Hence, the program 2.7 successfully compiles and displays the output 3.4E38f.

Let's make a minor change in the above program 2.7. You can try to guess the output now.

Program 2.8:

```
class Test
{
    public static void main(String args[])
    {
        float Number1;                          // step 1 Variable Declaration
        Number1=3.4E39f;
        System.out.println(Number1);
    }
}
```

Output:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
The literal 3.4E39f of type float is out of range
```

This program does not compile as we are trying to store a value greater than 3.4×10^{38}. If we try to store value greater than 3.4×10^{38}, compilation error occurs.

Is there any data type in Java whose variables can store decial values greater than 3.4×10^{38}? Yes, Java supports double data type whose variables can store values greater than upto 1.8×10^{308}

Program 2.9:

```
class Test
{
    public static void main(String args[])
    {
        double Number1;                         // step 1 Variable Declaration
        Number1=7E39;                           // 7E39 means 7x10^39
        System.out.println(Number1);
     }
}
```

Output:

```
7.0E39
```

double stores floating point numbers without specifying f at the end of the floating point number (11.21f) because every decimal value is treated as double if nothing is specified.

Let us try to guess the output of the following program.

Program 2.10:

```
class Test
{
    public static void main(String args[])
     {
        float Number1;                          // step 1 Variable Declaration
        Number1=11.23;
        System.out.println(Number1);
    }
}
```

Output:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
Type mismatch: cannot convert from double to Float
```

In Java, decimal values are treated as double by default and not as a float. Hence, compiler does not compile above program 2.10. Hence, it is necessary to specify f at the end of the floating point number. The other way to write Number1=11.23f; is Number1=(float)11.23;

Data Type	Size	Range
long	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Apart from int, long data types, Java also supports byte and short data types whose variables holds integer values.

Data Type	Size	Range
byte	1 byte(8 bits)	-128 to 127
short	2	-32,768 to 32,767

2.4.5 byte

Let us assume that you want to create a variable that can store value 11. We can use byte,short, int or long as variables of these data types can store any integer value like 11. Hence, we can use variables of any data type, but, we should create a variable of data type which will take less space in memory. We use bottle to store 1 liter water and not 1000 liter tank as there is a wastage.

e.g. if we want to store any value between -128 to 127, it is wise idea to use byte and not an int or float or long. Reason is that variables of byte data types occupies only 1 byte, whereas, short,int and long requires 2,4,8 bytes.

Program 2.13:

```
class Test
{
    public static void main(String args[])
    {
        byte num;
        num = 126;
        System.out.println(num);
    }
}
```

Output:
126

2.4.6 short

If we want to store integer up to 32768, it is better to create variable of short data type.

Program 2.12:

```
class Test
{
    public static void main(String args[])
    {
        short num;
        num = 32767;
        System.out.println(num);
    }
}
```

Output:
32767

2.4.7 boolean

Apart from arithmetic evaluations like addition, subtraction, computer evaluates logical operations.
e.g. if number a is greater than b or not i.e. a>b, it evaluates it to either true or false.

Program 2.12:

```
class Test
{
    public static void main(String args[])
    {
        int Number1=1987;
        int Number2=98232;
        boolean result= Number1> Number2;
        System.out.println(" Result is "+result);
    }
}
```

Output:
false

You can execute above program by manually changing values of Number1 and Number2 and see the result. You will get output as either true or false.

Q Guess the output of the following programs.

I.

```
class Test
{
    public static void main(String args[])
    {
        float Number1;                    // step 1 Variable Declaration
        Number1=1.4E-45f;
        System.out.println(Number1);
    }
}
```

Output:
1.4E-45

II.

```
class Test
{
    public static void main(String args[])
    {
        double Number1;                   // step 1 Variable Declaration
        Number1=11.23;
        System.out.println(Number1);
    }
}
```

Output:
11.23

III.

```
class Test
{
    public static void main(String[] args)
    {
        int x;
    }
}
```

Output:
(Blank Output)

IV.

```
class Test
{
    public static void main(String[] args)
    {
        int x;
        System.out.println(x);
    }
}
```

Output:
Compilation Error

Variable x is not initialized, hence, it is compilation error. It is necessary to initialise variables in Java before you use it.

You can try to guess what variables one should create to store below values.

I. 32
II. '#'
III. 22751
IV. 258,325,325
V. 68948974908483
VI. 325.59
VII. 1.2E-300

Value	**Data Type**
32	byte
'#'	char
22751	short
258,325,325	int
68948974908483	long
325.59	float
1.2E-300	double

As byte, short, int, long, char stores basic values, these data types are known as primitive data types.

Java supports two categories of data types.

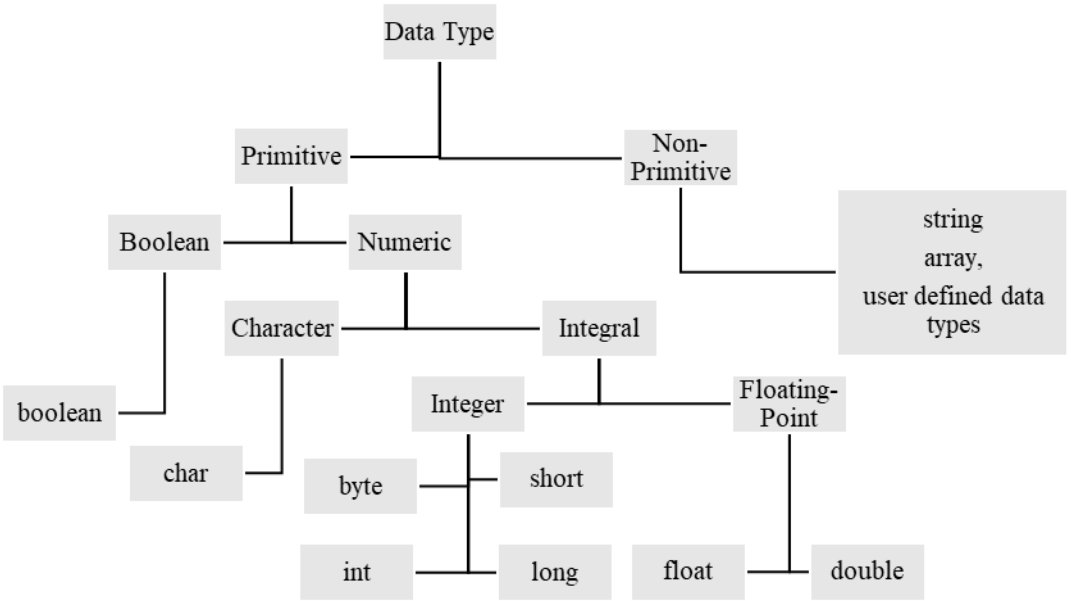
- Primitive data types.
- Non-primitive data types.

The data types that are derived from primary data types are known as non-Primitive data types. These datatypes are used to store one or more values. We will be studying non-primitive data types later part of this book.

Variables of primitive data types are used to store values like ‘z’,1121.12 etc. e.g. variables of integrate data type is used to store integer values in the Java programs.

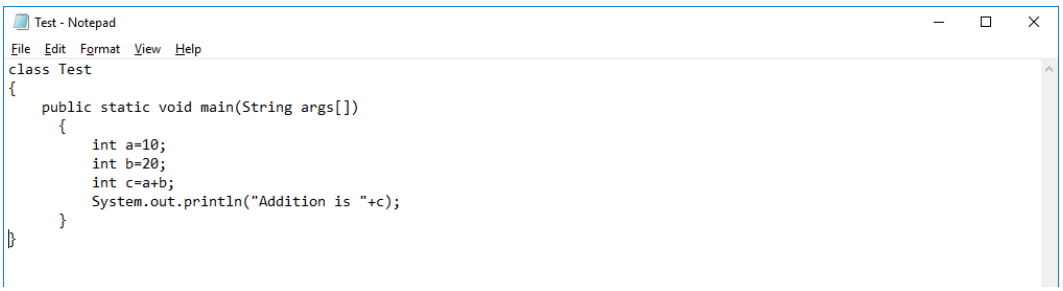
Non-primitive data types use primitive data type. e.g. String is set of characters like “Amit”

Diagram: 2.1



2.5 Java Program Development & Execution

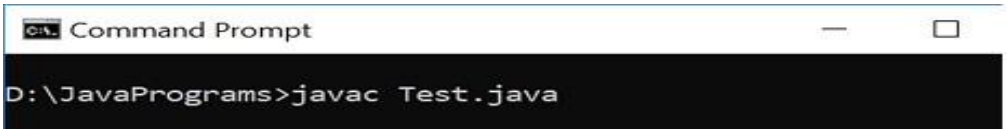
1. Write the simple Addition program using notepad.



Save this program as **.java** file. Example **Test.java.**

2. Open the command prompt. Change the directory on the command prompt where Test.java is saved.

3. Compile the program by running the command “javac <program _name.java>” on the command prompt shown as below.



4. To execute the Java class file, run the command "java <program _name>". Note that, it does not include .java or .class extension.

2.6 Points to Remember

1. We cannot define multiline comment inside other multiline comment. However, multiline comments can contain single line comment/comments.

2. Java language is a case sensitive. Following instruction won't compile.
   INT Number1=13579;
   This is because compiler won't understand the meaning of the word INT and it will generate a compile time error.

3. You can always declare and assign the value of a variable in the instruction itself.
   Program 2.15:

```
class Test
{
   public static void main(String args[])
   {
        int Number1=13579;        // Definition of Number 1
        int Number2=2468;         // Definition of Number 2
        int sum= Number1+ Number2;
        System.out.println(sum);
   }
}
```

   You compile the above program and execute the same for better understanding.
   Further, you can combine three declarations in one line as below.

   Program 2.16:

```
class Test
{
    public static void main(String args[])
    {
        int Number1, Number2, sum;
        Number1=13579;
        Number2=2468;
        sum = Number1+ Number2;
        System.out.println(sum);
    }
}
```

Even, you can write the same program as shown in 2.17.

Program2.17:

```
class Test
{
    public static void main(String args[])
    {
        int Number1=13579, Number2=2468, sum;
        sum = Number1+ Number2;
        System.out.println(sum);
    }
}
```

Output:
16047

4. Variable names correspond to locations in the computer's memory. In memory, every variable has assigned a particular address. Every variable has a name, a type, a size and a value.

<type> <variable_name>;

Whenever a new value is placed into a variable, it replaces (and destroys) the previous value.

5. To make the program add.java more user friendly, you can change the System.out.println instruction as below.

System.out.println ("The sum is" +sum);

Whatever characters we write inside the quotes of the first parameter of the System.out.println method gets printed as output

System.out.println ("Enthusiasm is a key to success.") displays "Enthusiasm is a key to success" as output. We will discuss System.out.println method in Input Output chapter in detail.

Q. Guess the output of the following programs.

I.

```
class Test
{
    public static void main(String args[])
    {
        int Number1;
        System.out.println(sum);
    }
}
```

Output: Compilation error
Variable Number1 might not have been initialized.

When a variable is declared inside a method (in this case main method), memory is allocated by a compiler. If the variable is not initialized & accessed, compiler gives error.

II.

```
class Test
{
    public static void main(String args[])
    {
        int Number1=13579;
        int Number2=2468;
        int Number1=4517;
        int sum= Number1+ Number2;
        System.out.println(sum);
    }
}
```

Output:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Duplicate local variable Number1
```

The output is a compile time error because the variable Number1 is declared twice in the same method (main in this case) which is not allowed in Java as per grammar rule.

III.

```
class Test
{
    public static void main(String args[])
    {
        int Number1=13579;
        int Number2=2468;
        int sum= Number1+ Number2;
        sum=156;
        System.out.println(sum);
    }
}
```

Output:

```
156
```

The addition of 13579 and 2468 is first stored in a variable sum.

However, the next instruction sum=156; changes the contents of the variable sum to 156. Hence, the value stored in the variable sum i.e. 156 is displayed as output.

IV.

```
class Test
{
    public static void main(String args[])
    {
        int Number1;
        int Number2;
        int sum;
        Number1=13579;
        Number2=2468;
        sum= Number1+ Number2;
        System.out.println(sum);
    }
}
```

Output:

```
16047
```

In case of the above program variables Number1, Number2 are declared at the beginning of the main method block. The values 16047 and 2468 are stored later which is fine.

V.

```
class Test
{
    public static void main(String args[])
    {
        int Number1;
        int Number2;
        Number1=13579;
        Number2=2468;
        int sum= Number1+ Number2;
        System.out.println(sum);
    }
}
```

Output:
16047

VI.

```
  /* This is the first program written by me in java language.
  /* Test.java */
  I have learned so many fundamentals because of this program.
  */
  class Test
  {
      public static void main(String args[])
      {
          int Number1;
          int Number2;
          int sum;
          Number1=13579;
          Number2=2468;
          sum= Number1+ Number2;
          System.out.println(sum);
      }
}
```

Output:
Exception in thread "main" java.lang.Error: Unresolved compilation problem

As a multiline comment is declared inside other multiline comment, hence, it is an error

FAQ:

Q. Why should we use Java language for writing this program, let us say, addition of 2 numbers as compared to machine and assembly language?

If you use machine language to write this program, you had to write instructions in machine language which is difficult and error prone.

If you use assembly language to write this code, you would have to write instructions in assembly language using mnemonics. Number of the instructions in assembly language are less than machine language. However, still, it is tedious job.

If you see Java language program, it is hardly few lines of code as compared to programs written in machine and assembly language instructions. This reduces the chances of doing mistakes while writing a program because the size of the program is small. The biggest advantage of Java language is platform independent. It means it can be compiled on different processors and executed. Machine and assembly languages are specific to the machine.

Q. Why Java is a middle level language?

Java is often called a middle level computer language, because it combines the best elements of high level languages with the control and flexibility of assembly language.

Q. What are the features of the Java?

Object Oriented: In java everything is an Object. Java can be easily extended since it is based on the Object model.

Platform independent: Unlike many other programming languages including C and C++ when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by virtual Machine (JVM) on whichever platform it is being run.

Simple: Java is designed to be easy to learn. If you understand the basic concept of OOP java would be easy to master.

Secure: With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

Architectural- neutral: Java compiler generates an architecture-neutral object file format which makes the compiled code to be executable on many processors, with the presence Java runtime system.

Portable: Being architectural neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler and Java is written in ANSI C with a clean portability boundary which is a POSIX subset.

Robust: Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.

Multi-threaded: With Java's multi-threaded feature it is possible to write programs that can do many tasks simultaneously. This design feature allows developers to construct smoothly running interactive applications.

Interpreted: Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light weight process.

High Performance: With the use of Just-In-Time compilers Java enables high performance.

Distributed: Java is designed for the distributed environment of the internet.

Dynamic: Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time

2.7 Deep Knowledge Section[8]

Q 1. Guess the output of the following program?

I

```
class Test
{
    public static void main(String args[])
    {
        byte a1=127;
        byte b1=125;
        byte c1=0;
        c1=a1+b1;
        System.out.println(c1);
    }
}
```

Output:

-2

This program gave us compilation error because variable c1 is of type byte so it cannot hold the values greater than 128,but sum of a1 and b1 will exceed the limit.

II

```
class Test
{
    public static void main(String args[])
    {
        int a1=2147483647;
        int b1=2147483647;
        int c1=0;
        c1=a1+b1;
        System.out.println(c1);
     }
}
```

Output:

Exception in thread "main" java.lang.Error: Unresolved compilation problem:
Type mismatch: cannot convert from int to byte

The output Is -2 because it integer data type can hold maximum value 2147483647. Hence, the output overflows and produces -2.

III

```
class Test
{
    public static void main(String args[])
    {
        float a1=3.4E38f;
        float b1=3.4E38f;
        float c1=0;
        c1=a1+b1;
        System.out.println(c1);
    }
}
```

Infinity

Anything above 1.79E308 is considered as INFINITY. Sum of a1 and b1 has exceeded the limit.

---

[8] Deep knowledge Section should be read after developing average level in Java. Beginner may skip this section in first reading.

Q 3. What is the correct declaration of a main() method?

public static void main(String args[])

public- Access specifier, visible to all classes.
static- main to be called without instantiating a particular instance of class.
void-main should not return any value to operating system.
String args[]- parameter name args, which is instance array of class String.

Q 3. Is there any other way to declare main() method?

Yes. There are different ways in which we can write the main method in JAVA. For Example:

1. public static void main(String[] args)
2. public static void main(String... args)
3. public static void main(String args[])

Q 4. What is meaning of System.out.println.

System is a **class** in the java.lang package.
out is a **static member** of the System class, and is an instance of java.io.PrintStream.
println is a **method** of java.io.PrintStream.

Questions

Q.1	**State true or false.**
i.	java was developed at AT&T Bell Laboratories in 1972.
ii.	Algorithm is a set of logical steps to perform a certain task.
iii.	It is possible to write a java program without a main method.
iv.	System.out.println is a standard command that displays the value stored at that memory location.
v.	Method is a group of set of instructions arranged in a sequence to perform certain tasks.
vi.	Comment is information written inside the source code that is ignored by a preprocessor when the source file is preprocessed.
vii.	Directory is the location on a computer where files and other directories can be stored.
viii.	Java language is not a case sensitive language.
ix.	Every Java program source file ends with an extension .java
x.	Every instruction in Java should be terminated by a semicolon (;).
xi.	In Java language, it is mandatory to declare variables at the beginning of a block, otherwise, compile time error is generated.

Q.2 Guess the output of the following programs.

I.
```
class Test
{
    public static void main(String args[])
    {
        int one=1947;
        int two=191;
        int subtraction = one- two;
        System.out.println(subtraction);
    }
}
```

II.
```
class Test
{
    public static void main(String args[])
    {
        int add()
        {
            int one=1980;
            int two=191;
            int subtraction = one- two;
            System.out.println(subtraction);
        }
    }
}
```

III.
```
class Test
{
    public static void main(String args[])
    {
        int one=1947;
        float two=191.01f;
        int subtraction = one- two;
        System.out.println(subtraction);
    }
}
```

IV.
```
class Test
{
    public static void main(String args[])
    {
        int one=1947;
        int two=191;
        int subtraction = one- two,
        System.out.println(subtraction);
    }
}
```

V.
```
class Test
{
    public static void main(String args[])
    {
        int Number1;
        System.out.println(Number1);
    }
}
```

VI.
```
class Test
{
    public static void main(String args[])
    {
        int Number1;
        int Number2;
        Number1=13579;
        Number2=2468;
        int sum= Number1+ Number2;
        System.out.println(sum);
    }
}
```

VII.
```
class Test
{

    public static void main(String args[])
    {
        float Number1;                    // step 1 Variable Declaration
        Number1=1.4E-46f;
        System.out.println(Number1);
    }
}
```

Q.3 What is a data type? Give example.
Q.4 Create 3 integer variables that stores 10, 20 and 30 respectively.
Q.5 Explain Java language environment in detail.
Q.6 Write a program that does multiplication of 128 and 3876?
Q.7 What is the basic language of a computer? Why?
Q.8 Who invented Java? Discuss the history of Java.
Q.9 Who is a father of a computer?
Q.10 Why one should use Java language for programming?

Answers:

Q.1 i. True ii. True iii. False iv. True v. True vi. True vii. True viii. False ix. True x. True xi. False

Q2. I 1756 II. Compilation Error III. Compilation Error IV. Compilation Error V. Compilation Error VI.16047 VII. Compilation Error

General Information[9]

A. History of Algorithms

The word algorithm comes from the name of the 9th century Persian mathematician Abu Abdullah Muhammad bin Musa al-Khwarizmi. The word algorithm was only originally referred to the rules of performing arithmetic calculations using Hindu-Arabic numerals. The first case of an algorithm written for a computer was Ada Byron's notes on the analytical engine written in 1842. The lack of mathematical rigor in the "well-defined procedure" definition of algorithms posed some difficulties for mathematicians and logicians. This problem was largely solved with the description of the Turing machine, an abstract model of a computer formulated by Alan Turing.

B. Appendix

Before we proceed further, it is important that we should set up the java environment correctly. This section guides you on how to download and set up Java on your machine. Please follow the following steps to set up the environment.

Java SE is freely available from the link Download Java. So you download a version based on your operating system.

https://www.oracle.com/technetwork/java/javase/downloads/jdk11-downloads-5066655.html

Popular Java Editors:

To write your java programs you will need a text editor. There are even more sophisticated IDE available in the market. But for now, you can consider one of the following:

- **Notepad:** On Windows machine you can use any simple text editor like Notepad (Recommended for this tutorial), TextPad.
- **Netbeans:** is a Java IDE that is open source and free which can be downloaded fromhttp://www.netbeans.org/index.html.

Eclipse: is also a java IDE developed by the eclipse open source community and can be downloaded from http://www.eclipse.org/.

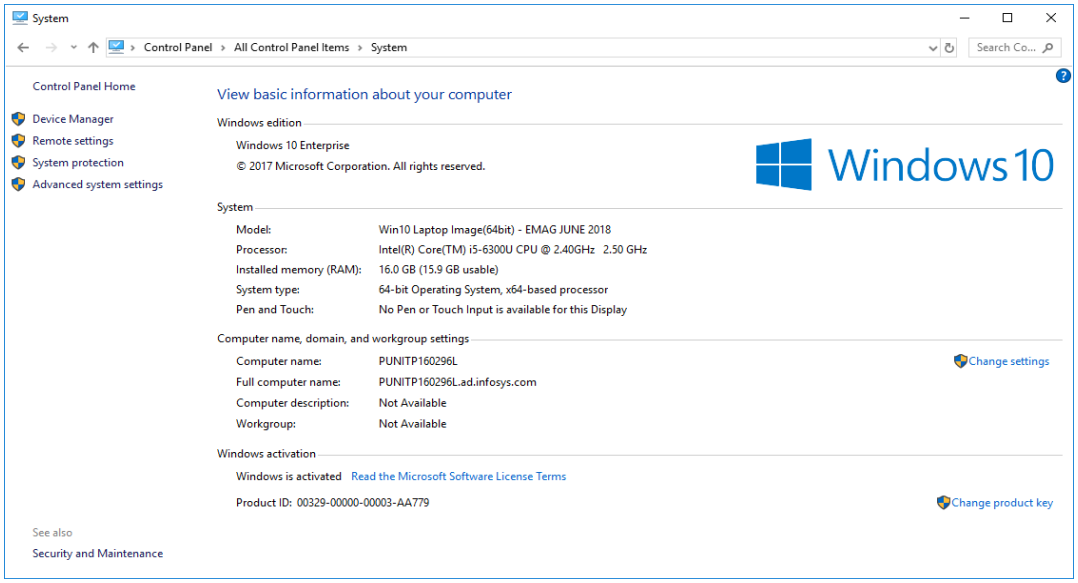
Setting up the path for windows 10:

Assuming you have downloaded Java 12 at D:\jdk-12_windows-x64_bin directory. It is applicable for Windows 10 and 64 bit machine.

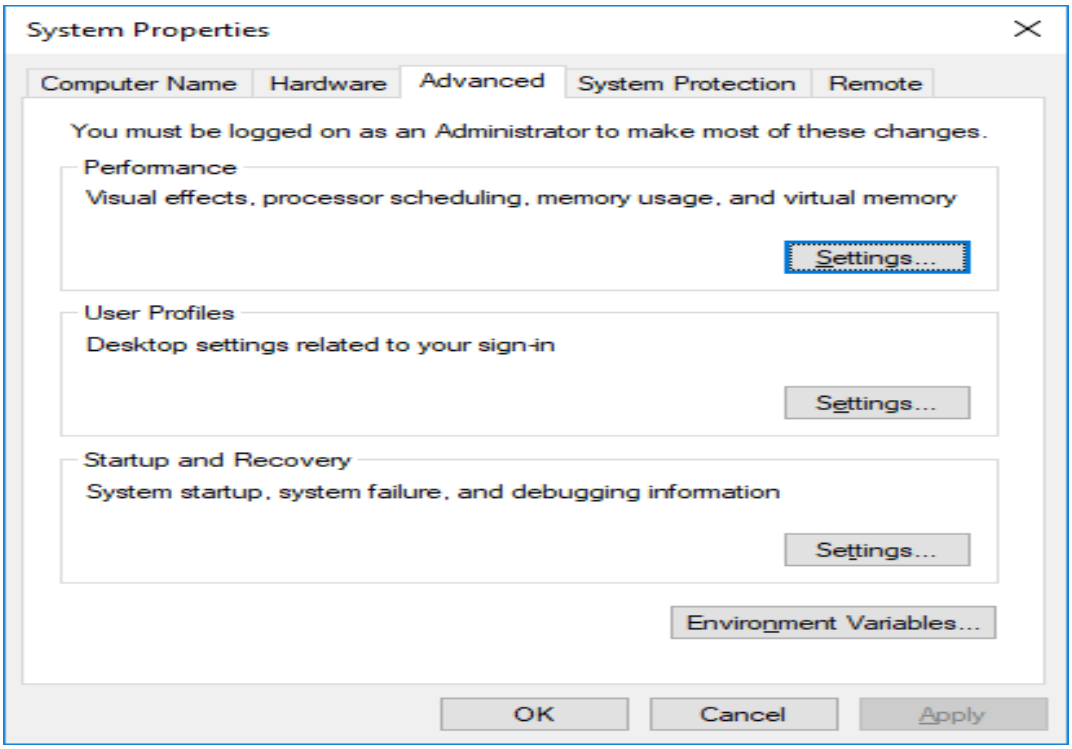
2. Click on 'Start' and select 'Control Panel'.

3. Click on the ‘System’.

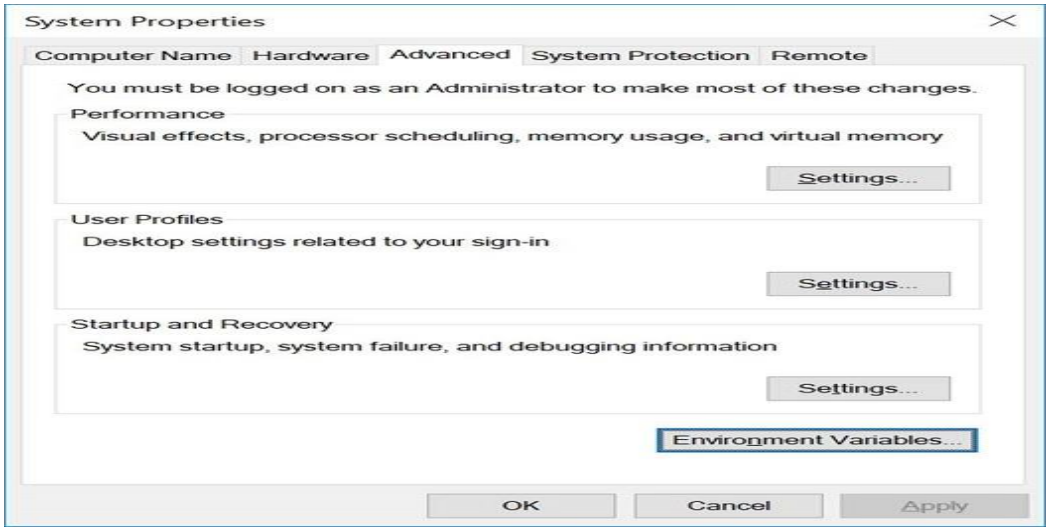
[9] Reader may skip general information section.



4. Click on the ‘Advanced system setting’.

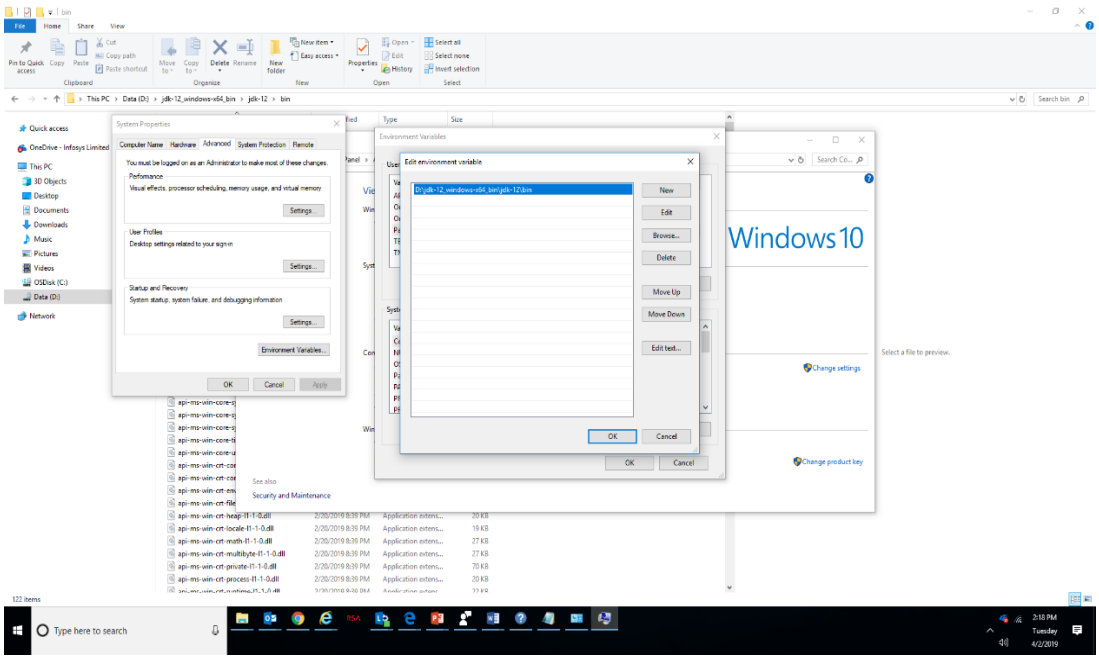


5. Click on the 'Environment variables' button under the 'Advanced' tab.



6. To change the variable name and value, click on the ‘Edit’.

7. Now, change the 'Path' variable so that it also contains the path of the directory where JDK is downloaded. Typically, it is bin directory where JDK (i.e. Java compiler (javac), Java etc) programs are present. Example, if the path is currently set to 'C:\WINDOWS\SYSTEM32', then change your path to ‘C:\WINDOWS\SYSTEM32; In our case , it is present at D:\jdk-12_windows-x64_bin\jdk-12\bin”



Chapter

3

STRUCTURED PROGRAMMING: SEQUENTIAL AND SELECTION CONTROL FLOW

Many of you might have wondered to see structured programming chapter in this book because we have heard that Java is object oriented language[1]. Yes, Java is object oriented programming language, but, Java has also structured programming features. Without these structured programming features, it is highly difficult to write any Java program. Let us study structured programming features like sequential control flow, selection control flow, iteration control flow etc.

3.1 Sequential Control Flow

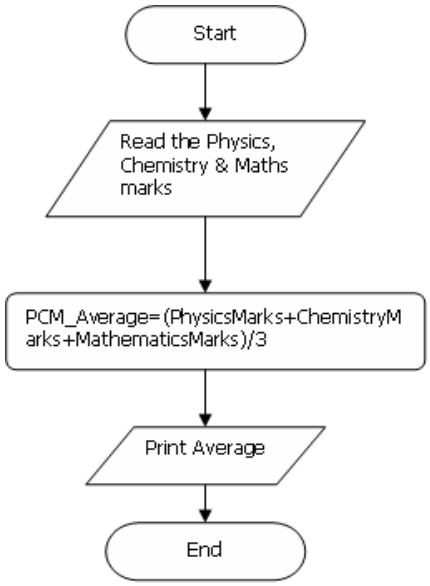
Imagine a hypothetical situation! Your lecturer friend approaches your one fine evening. He can use a computer but cannot do programming. He wants to find average of Physics, Chemistry and Mathematics marks of around 48 students accurate up to 2 decimals. Further, he has to do this task many times and hence request you to write a Java program. The algorithm and flow chart for this program is given as below.

Let's convert every step in the above algorithm into equivalent Java language instruction to write a complete Java program.

[1] Object oriented programming will be discussed later part of this book.

Algorithm:

1. Computer should display the message. "Enter Physics Marks?" on the output device.
2. Once the user enters the Physics marks through keyboard, computer should store it in a memory.
3. Computer should display the message. "Enter Chemistry Marks?" on the output device.
4. Once the user enters the Chemistry marks through keyboard, computer should store it in a memory.
5. Computer should display the message. "Enter Mathematics Marks?" on the output device.
6. Once the user enters the Mathematics marks through keyboard, computer should store it in a memory.
7. Averages of stored marks can be calculated as PCM Average= (Physics Marks+ Chemistry Marks+ Mathematics Marks) /3;
8. After calculating average marks, computer should display a message. "Average of your marks in Physics, Chemistry and Mathematics is ….."



Step 1: Computer should display the message "Enter Physics Marks?" on the output device.

We have studied in the second chapter that System.out.println is a standard command (method) that displays the data on the output device. The instruction to display "Enter Physics Marks?" is as below.

```
System.out.println ("Enter Physics Marks?");
```

Step 2: Once the user enters the Physics marks through keyboard, computer should store it in a memory.

Computer receives input data through input devices like a keyboard. Therefore, you have to enter the Physics Marks (input data) through a keyboard. But, what is the command/instruction (method) that can read data entered through keyboard and store it in computer memory? In order to read integer value entered through the keyboard, it is necessary to use nextInt() command (method) present in Scanner class which is already available in Java language. For using any command available in any class, it is necessary to create object of that class. Here, we have to create object of Scanner class and then use nextInt command to read the integer value entered through the keyboard. Instead of understanding this at this beginning stage of programming, you can think that below two commands (instructions) are used to receive integer value from a keyboard.

```
Scanner sc = new Scanner (System.in);
int physicsMarks = sc.nextInt()
```

Step 3-6: Similarly. we can write following instructions for receiving Chemistry marks

```
System.out.println("Please tell me your Chemistry Marks?");
Scanner sc = new Scanner (System.in);
int chemistryMarks = sc.nextInt();
```

Step 3-6: Similarly, we can write following instructions for receiving Mathematics marks.

```
System.out.println("Please tell me your Mathematics Marks?");
```

```
Scanner sc = new Scanner (System.in);
int MathematicsMarks = sc.nextInt();
```

It is seen that Scanner sc = new Scanner (System.in); instruction is written three times. There is no need of rewriting the same instruction, we can write it once. Hence, he complete program till step 6 is as below.

```
import java.util.Scanner;
class Test
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner (System.in);

        System.out.println("Please tell me your Physics Marks?");
        int physicsMarks = sc.nextInt();

        System.out.println("Please tell me your Chemistry Marks?");
        int chemistryMarks = sc.nextInt();

        System.out.println("Please tell me your Mathematics Marks?");
        int mathematicsMarks = sc.nextInt();
    }
}
```

Step 7: Averages of stored marks can be calculated as PCM Average = (Physics Marks + Chemistry Marks + Mathematics Marks) /3;

As average marks can be a decimal value, we require one more variable to store this decimal value. int cannot be used to store a decimal value.

float is a data type used to create a float variable. float variable stores a decimal value.

7th statement in the algorithm can be

```
int  PCM_Average = (physicsMarks + chemistryMarks + mathematicsMarks) / 3;
```

When we use computer, multiplication and division is denoted by * and / symbol respectively instead of × and ÷ .

Step 8: After calculating average marks, computer should display a message "Average of your marks in Physics, Chemistry and Mathematics is …"

We can use standard System.out.println method to display the average of PCM.

```
System.out.println("Average of Physics, Chemistry and Mathematics mark are= "+ PCM_Average)
```

The program can be written as shown in program 3.1.

Program 3.1:

```
import java.util.Scanner;
class Test
{
    public static void main(String [] args)
    {
        Scanner sc = new Scanner (System.in);

        System.out.println("Please tell me your Physics Marks?");
        int PhysicsMarks = sc.nextInt();

        System.out.println("Please tell me your Chemistry Marks?");
        int ChemistryMarks = sc.nextInt();

        System.out.println("Please tell me your Mathematics Marks?");
        int MathematicsMarks = sc.nextInt();

        float PCM_Average = (PhysicsMarks + ChemistryMarks + MathematicsMarks)/3;
        System.out.println("Average of Physics, Chemistry and Mathematics mark are="
                           +PCM_Average);
    }
}
```

Q. What is the meaning of the

import java.util.Scanner;

As we require to crate object of Scanner class as well as we need to use nextInt() command (method), it is necessary to write this instruction at the beginning of the program. You will understand it later part of this book.

Q. What will be the output of the above program if Physics, Chemistry and Mathematics marks are 97, 99,100 respectively? Is there any error in the above program?

The output would be 99.00 which is as expected.

Q. What will be the output of the above program if Physics, Chemistry and Mathematics marks are 98, 98,99 respectively? Is there any error in the above program?

The output would be 98.00 which is not as expected. It should be 98.33.
It is a logical error as the expected output is 98.33%.

When numerator and denominator are integer values, the result of the division is always an integer.

(PhysicsMarks + ChemistryMarks + MathematicsMarks) / 3 evaluates to 295/3=98.

Hence, 0.33 is truncated and it gives result as 98.00 which is incorrect output of the program. We need to modify the statement 7 to get the correct output as below.

PCM_Average= **(float)** (PhysicsMarks + ChemistryMarks + MathematicsMarks) /3;

(float) tells compiler to treat the evaluation of (PhysicsMarks + ChemistryMarks + MathematicsMarks) as a floating value.

When instructions in a program are executed one after another i.e. in a sequential manner, it is known as a sequential control flow.

It is one of the feature of a structured programming. The program 3.1 is an example of a sequential control flow.

Q. What is a control flow? What are the different control flows used in a program?

Control flow is a flow of the execution of the instructions in a program.

OR

It is the order in which the instructions in a program are executed.

In this chapter, you will understand, the instructions are not always executed in the order they are written in a program. By default, the flow of instructions is sequential. A control statement disrupts the sequential flow. It means certain instruction or instructions are skipped during the execution of a program.

Based on the order of execution of instructions, Java programs are mainly written in 4 different structure formats.

1) **Sequential:** Statements in a program are executed one after another i.e. in a sequential manner.
2) **Selection:** Statements in a program are executed based on certain condition/conditions defined in a program. e.g. if; if-else; switch-case statement.
3) **Iteration:** Statements in a program are executed repeatedly. e.g. while, do-while, for
4) **Jump:** Different set of instructions in the program are executed as per the jump defined in a program. e.g. break, return, continue.

Do not get confused after reading these 4 different types of control flows. We are going to study it in this chapter in a simplified and understandable language.

3.2 Selection Control Flow

3.2.1 if-else statement

Assume that one of your friend says that he has got 91 marks in Physics and 99 marks in Mathematics. Instantly, if you think in which subject your friend has got highest marks? Your brain will compare 91 marks to 99 marks and conclude that Mathematics marks are greater than Physics marks. Will it again compare 99 marks to 91? No. It will do only 1 comparison instead of 2 comparisons because based on the result of only one comparison, final result can be concluded in this case.

When one statement out of given 2 statements is executed based on the result of a condition, if-else statement is used. If the condition is true, statement associated with if-part is evaluated, otherwise, statement associated with else part is evaluated.

The general form of if-else statement

```
if(expression)
   Statement1; //Statement associated with if part
else
   Statement2; //Statement associated with else part
```

If expression is true, statement1 is executed. Otherwise, statement2 is executed.

Q. What is an expression?

Expression is a combination of an operator and operand that evaluates to a constant value.

OR

Expression can be any combination of variable, operator and / or constant that results into a constant value after its evaluation.

Operand can be a constant value like 6, 17 etc. or a variable x, i etc. The evaluation of an expression results into a constant value.

Few examples of expressions are a+b; x*y+3; 8; c etc. where x, y, a, b are variables.

Let's understand concept of if-else statement by writing a below small program that accepts Physics and Mathematics marks as input from a keyboard and checks in which subject the student has obtained more marks as compared to the other subjects. It is assumed that student has not obtained same marks in Physics and Mathematics.

Following are the steps involved to do this task.

1. Computer should display the message "Enter Physics Marks?" on the output device.
2. Once the user enters the Physics marks through keyboard, computer should store it in a memory.
3. Computer should display the message "Enter Chemistry Marks?" on the output device.
4. Once the user enters the Chemistry marks through keyboard, computer should store it in a memory.
5. if Physics marks are greater than Mathematics mark
   Display "Physics marks are greater than Mathematics marks."
   else
   Display "Mathematics marks are greater than Physics marks."

You can easily write instructions for step 1 to step 4. If you require to understand step 1- step 4, you can refer program 3.1. The main logic to write this program is to write step 5.

In step 5, only one condition needs to be evaluated out of 2 conditions, hence, we can use if-else statement.

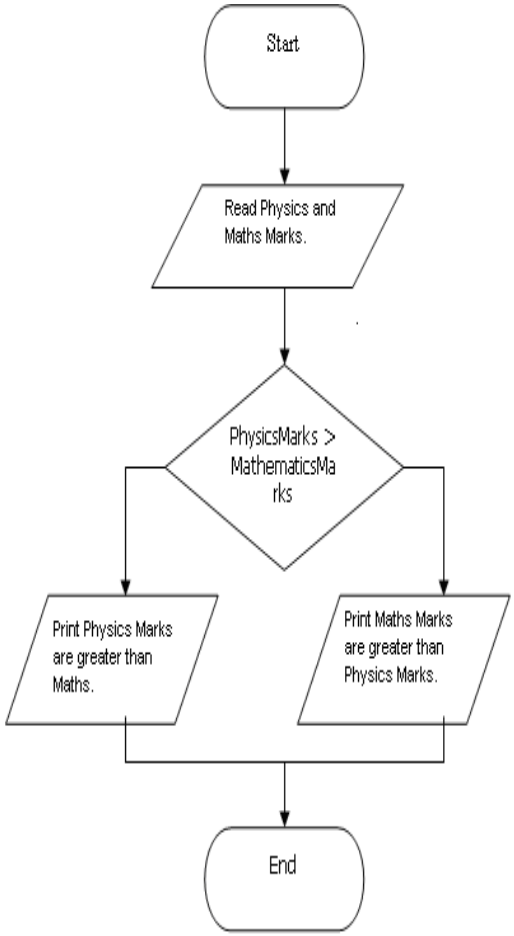
if-else statement for step 5 can be written as below.

```
if ( PhysicsMarks > MathematicsMarks )
      System.out.println ("Physics marks are greater than Mathematics marks.");
else
     System.out.println ("Mathematics marks are greater than Physics marks.");
```

The complete program can be written as below.

Program 3.2:

```
import java.util.Scanner;
class Test
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner (System.in);

        System.out.println("Please tell me your    Physics Marks?");
        int PhysicsMarks = sc.nextInt();

        System.out.println("Please tell me your Chemistry Marks?");
        int ChemistryMarks = sc.nextInt();

        if (PhysicsMarks > ChemistryMarks)
            System.out.println ("Physics marks are greater than Mathematics marks.");
        else
            System.out.println ("Chemistry Marks marks are greater than Physics marks.");
    }
}
```



3.2.2 if statement

What if someone tells you equal marks in both these subjects? Here, we need to check one condition i.e. whether the marks received in both the subjects are equal.

When given statement is executed based on the result of a condition, if statement is used.
If the condition is true, statement associated with if statement gets executed. If the condition is false, statement associated with if is not executed.

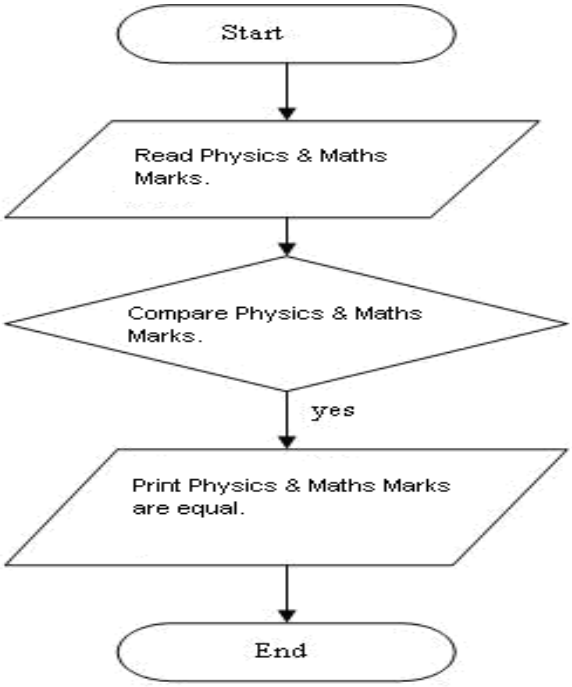
The general form of if statement is as below.

```
if (expression)
    statement;
```

Let's write a program that accepts Physics and Mathematics marks as input, checks whether the Physics marks are equal to Mathematics marks and display the below message if this condition is true.

"Physics marks are exactly equal to Mathematics marks."

If this condition is false, it should not display anything.



Program 3.3:

```
import java.util.Scanner;
class MyClass
{
    public static void main (String [] args)
    {
        Scanner sc = new Scanner (System.in);

        System.out.println("Please tell me your Physics Marks?");
        int PhysicsMarks = sc.nextInt();

        System.out.println("Please tell me your Mathematics Marks?");
        int MathematicsMarks = sc.nextInt();

        if (PhysicsMarks == MathematicsMarks)
            System.out.println ("Physics marks are exactly equal to Mathematics marks.");
    }
}
```

Let's write a program that compares Physics and Mathematics marks and displays following message

If Physics marks are exactly equal to Mathematics marks, display the message
"Physics marks are exactly equal to Mathematics marks."

If Physics marks are greater than Mathematics marks, display the message
"Physics marks are greater than Mathematics marks."

If Mathematics marks are greater than Physics marks, display the message
"Mathematics marks are greater than Physics marks."

Program 3.4:

```
import java.util.Scanner;
class MyClass
{
    public static void main (String [] args)
    {
        Scanner sc = new Scanner (System.in);

        int PhysicsMarks;
        int MathematicsMarks;

        System.out.println("Please tell me your Physics Marks?");
        PhysicsMarks = sc.nextInt();

        System.out.println("Please tell me your Mathematics Marks?");
        MathematicsMarks = sc.nextInt();

        if (PhysicsMarks==MathematicsMarks)
            System.out.println ("Physics marks are exactly equal to Mathematics marks.");

        if (PhysicsMarks> MathematicsMarks)
            System.out.println ("Physics marks are greater than Mathematics marks.");

        else
            System.out.println ("Mathematics Marks marks are greater than Physics marks.");
    }
}
```

Can you guess the output of the program 3.4 if the marks of Physics and Mathematics are 98? The output will be

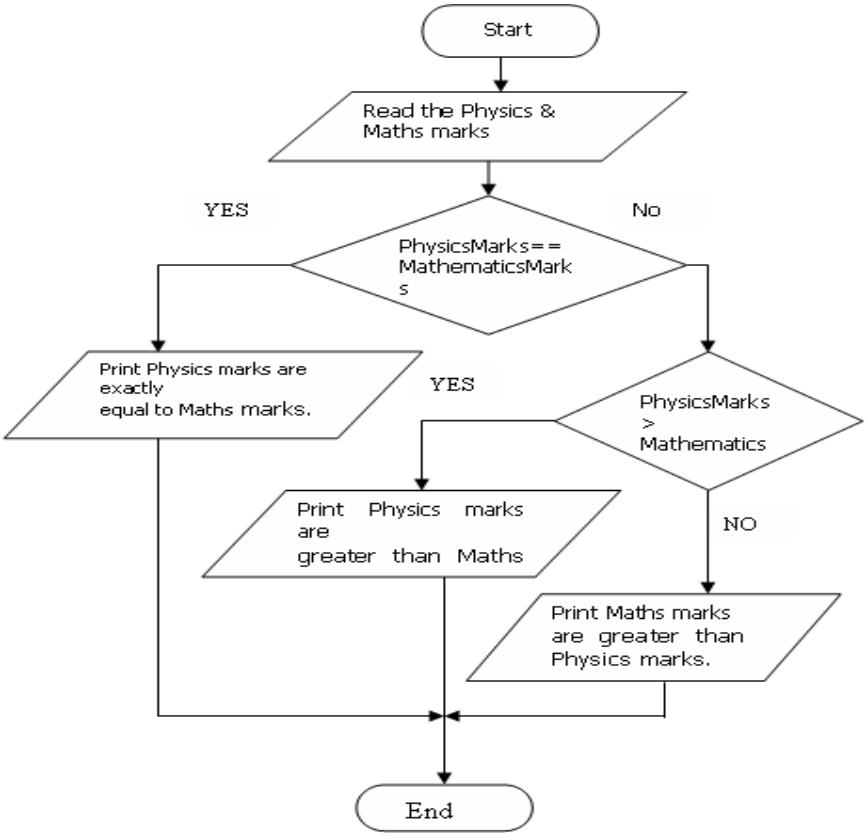
Physics marks are exactly equal to Mathematics marks.
Mathematics marks are greater than Physics marks.

It means there is a logical error in the above program. What is the Java language construct used when there are multiple conditions need to be evaluated and the result is only 1?

Java language provides **if *-else if* - else** construct when one condition is true from given set of conditions.

The general form of if-else-if ladder is as below.

```
if(expression1)
        Statement1;
else if(expression2)
       Statement2;
…
…
…
else
      Statements;
```



The above program can be written like this

Program 3.5:

```
import java.util.Scanner;
class Tester
{
    public static void main (String [] args)
    {
        Scanner sc = new Scanner (System.in);

        System.out.println("Please tell me your Physics Marks?");
        int PhysicsMarks = sc.nextInt();

        System.out.println("Please tell me your Mathematics Marks?");
        int MathematicsMarks = sc.nextInt();

        if (PhysicsMarks==MathematicsMarks)
            System.out.println ("Physics marks are exactly equal to Mathematics marks.");

        else if (PhysicsMarks > MathematicsMarks)
            System.out.println ("Physics marks are greater than MathematicsMarks marks.");

        else
            System.out.println ("MathematicsMarks marks are greater than Physics marks.");
    }
}
```

if can be used only at the beginning and else only at the end. if is mandatory, while, else is optional.

Q. Guess the output of the following program.

I

```
class Test
{
    public static void main (String [] args)
    {
        int k=90;
        if( k>=0)
            System.out.println (“Kolhapur");
        else
            System.out.println ("Pune");
    }
}
```

Output:
Kolhapur

II

```
class Test
{
    public static void main (String [] args)
    {
        int k=5;
        if(k =5 )
            System.out.println (“A”);
        else
            System.out.println (“B”);
     }
}
```

Output:
Compilation error

Condition must evaluate to Boolean value i.e. true or false. Hence, compilation error.

- **Remember = is an assignment operator, while, == is an equality operator.**

  Equality operator compares values of its operands. It returns true if the values are equal, otherwise false. If you forget to write one = symbol in equality operator, it becomes assignment operator.

III

```
class Test
{
    public static void main (String [] args)
    {
          int k=5;
          if(k ==5 )
            System.out.println (“A”);
        else
            System.out.println (“B”);
     }
}
```

Output:
A
As k is equal to 5, condition k==5 results into true. Hence, the output.

IV

```
class Test
{
    public static void main (String [] args)
    {
        int p=10, c=10, m=10;

        if ((p == c) == false)
            System.out.println ("M");
        else
            System.out.println ("N");
    }
}
```

Ourput:

N

(p==c) results into true which is compared to false. Hence, the condition results into false.

Pseudo code and Flowchart:

Pseudo code is an artificial, informal, and similar to English like language that helps us to develop a program. It is structured English for describing algorithms. It breaks the bigger tasks into smaller tasks. It is actually not executed on the computers. Pseudo code helps to write actual program.

Flowchart is a graphical representation of an algorithm. It has single-entry and single-exit. It is drawn using certain special-purpose symbols connected by arrows called flow lines.

Rectangular symbol	Processing block like initialization, assignment and expression evaluation
Arrow symbol	Connects one symbol to others or describe the flow.
Oval symbol	Indicates the beginning or end of a program or a section of code.
Diamond symbol	Used to take decisions (condition evaluation)
Parallelogram symbol:	Used to display input, output.

Let's develop a flow chart for a program that accepts Physics, Chemistry and Mathematics marks as input and displays the maximum marks obtained as output. Assume that marks received as input are different in each subject.

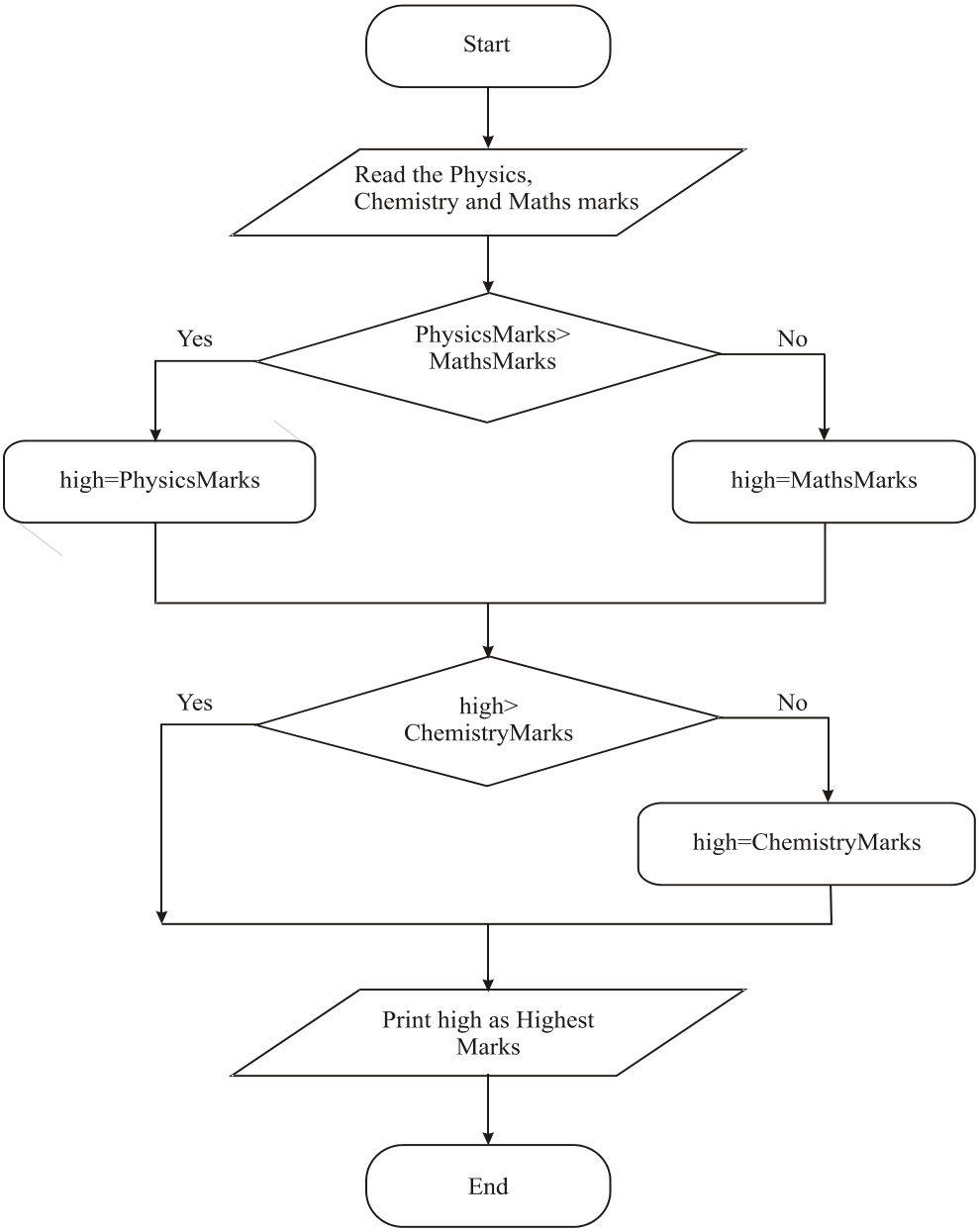


Fig. Flowchart that finds highest marks among Physics, Chemistry and Mathematics.

3.2.3 switch-case Construct

The general form of the switch statement is as below.

```
switch (integralExpression)
{
    case constant1:
            statement1;
            break:
    case constant2:
            statement2;
            break;
    case constant3:
           statement3;
           break:
    case constant4:
           statement4;
           break:
    .
    .
    .
    .
    .
    default: statement
}
```

Oh, but what is an integral expression? Integral expression is an expression that evaluates to a constant integer value. e.g. 5, a+5,7*6+a*3 etc are integer expressions. a is an integer variable in above expressions. switch, case, default, break are the keywords in Java. break and default statements are optional.

switch is a multi-way decision statement that compares value of the switch expression with constant value of every case expression sequentially.

It seems it is going top of the head of you. Let me illustrate switch statement with the following program.

Let's write a program that displays
"Hi" when the user enters input value as 1.
"Hello" when the user enters input value as 2.
"Hey" when the user enters input value as 3.
"Excuse me" when the user enters input value other than 1, 2 or 3.

Program 3.6:

```
import java.util.Scanner;
class Test
{
    public static void main (String [] args)
    {
        Scanner sc = new Scanner (System.in);

        System.out.println("Enter your choice:");
        int ch = sc.nextInt();
```

```
        switch(ch)
        {
            case 1: System.out.println ("Hi");
                    break;
            case 2: System.out.println ("Hello");
                     break;
            case 3: System.out.println ("Hey");
                     break;
            default: System.out.println ("Excuse me ");
        }
        System.out.println ("End of the program.");
    }
}
```

Scenario 1:

When the user enters choice as 2, the switch statement will be executed as below.

The value of the integral expression (ch=2) is compared with a constant value specified in the first case. i.e. 2 is compared with a value 1. These values are not equal.

The value of the integral expression (ch=2) is compared with constant value specified in the second case. i.e. 2 is compared with a value 2. These values are equal. Hence, System.out.println ("Hello"); statement will be executed.

As break statement is present after this statement, the execution control will be transferred to the next instruction after the switch statement. i.e. System.out.println ("End of the program"); is executed.

Scenario 2:

When the user enters choice as 1, the switch statement will be executed as below.

The value of the integral expression (ch=1) is compared with a constant value specified in the first case. i.e. 1 is compared with a value 1. These values are equal. Hence, System.out.println ("Hi"); statement will be executed.

As the break statement is present after this statement, the execution control will be transferred to the next instruction after the switch statement. i.e. System.out.println ("End of the program"); is executed.

Scenario 3:

When the user enters choice as 7, switch statement will be executed as below.

The value of the integral expression (ch=7) is compared with constant value specified in the **first case**. i.e. 7 is compared with a value 1. These values are not equal.

The value of the integral expression (ch=7) is compared with constant value specified in the **second case**. i.e. 7 is compared with a value 2. These values are also not equal.

The value of the integral expression (ch=7) is compared with constant value specified in the **third case**. i.e. 7 is compared with a value 3. These values are also not equal.

As the value of the integral expression does not match with any of the case's constant value, the default statement (if present) will be executed. Hence, System.out.println ("Excuse me"); will be executed.

The execution control will be transferred to the next instruction after the switch statement.

You can try to guess the output of the above program when the user enters 3,5 etc.

Let's write one more program using if-else and switch statement. Following program accepts 2 integer numbers as input and add, subtract, multiply or divide them based on the user choice. User enters following number as a choice for the corresponding operation.

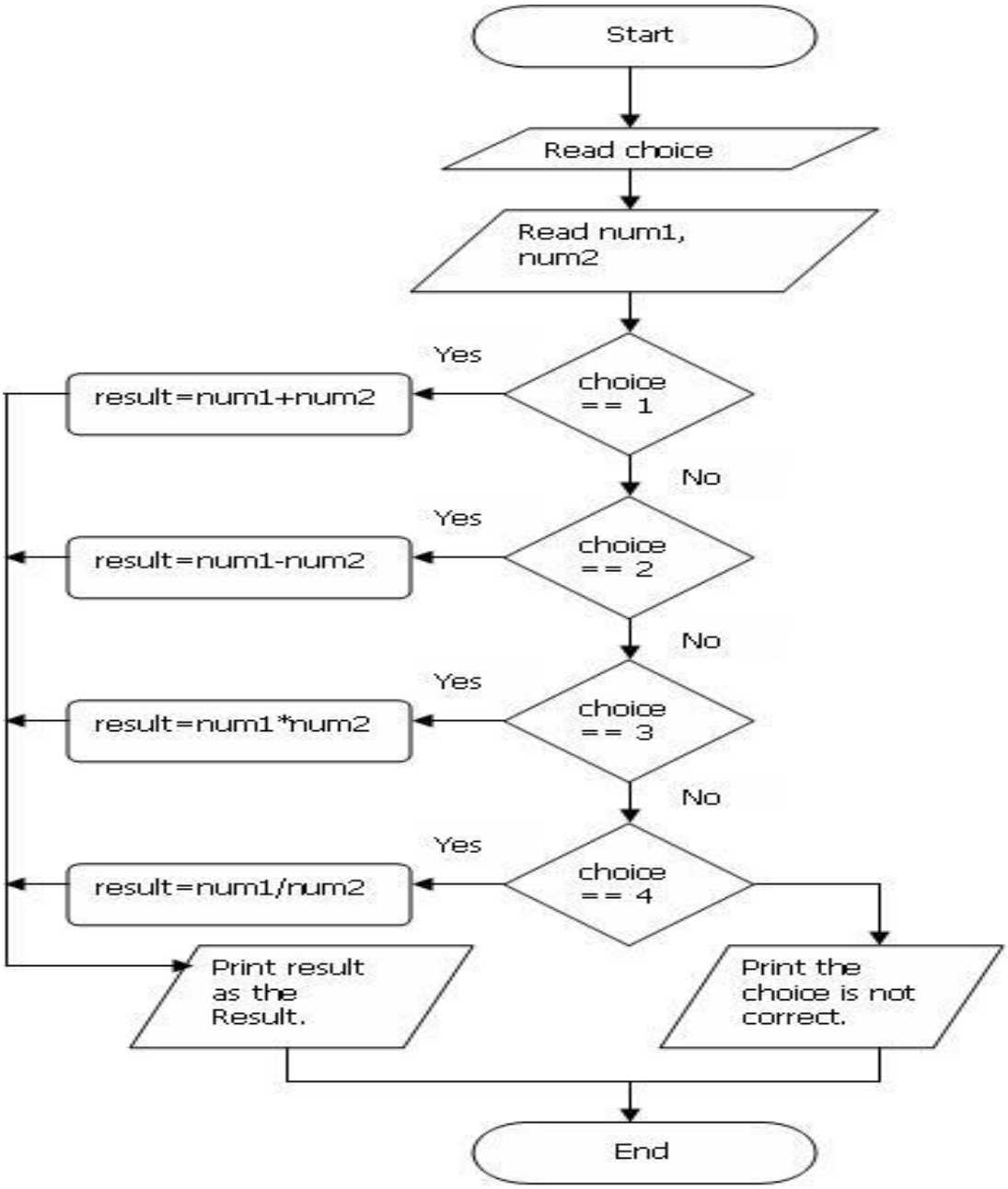
User Choice	Operation Performed
1	addition
2	subtraction
3	multiplication
4	division

The above program can be written using if-else statement as below.

Program 3.7:

```
import java.util.Scanner;
class Test
{
    public static void main (String [] args)
    {
        Scanner sc = new Scanner (System.in);
        int choice;
        int result=0, num1, num2;
        int addition=1, subtraction=2, multiplication=3, division=4;

        System.out.println (" Enter the choice “);
        System.out.println (" 1. Addition ");
        System.out.println (" 2. Subtraction ");
        System.out.println (" 3. Multiplication ");
        System.out.println (" 4. Division ");
        System.out.println (" Enter Your Choice ");
        choice = sc.nextInt();

        System.out.println ("Enter the No.1: ");
        num1 = sc.nextInt();

        System.out.println ("Enter the No.2: ");
        num2 = sc.nextInt();

        if (choice == addition)
            result=num1+ num2;

        else if (choice == subtraction)
             result=num1- num2;

        else if (choice == multiplication)
            result = num1*num2;

        else if(choice==division)
            result=num1/num2;

        System.out.println (" Result is ="+ result);
    }
}
```

It is seen that the user choice is compared with a number 1,2,3,4 individually and addition, subtraction, multiplication and division is performed. Java provides switch construct that provides more readability and understanding of the above program.



The program 3.7 can be written using switch statement as shown below in program 3.8.

Program 3.8:

```
import java.util.Scanner;
class Test
{
    public static void main (String [] args)
    {
        Scanner sc = new Scanner (System.in);
        int choice;
        int result=0, num1=0, num2=0;
        int addition=1, subtraction=2, multiplication=3, division=4;

        System.out.println (" Enter the choice");
        System.out.println (" 1. Addition ");
        System.out.println (" 2. Subtraction ");
        System.out.println (" 3. Multiplication ");
        System.out.println (" 4. Division ");

        System.out.println (" Enter Your Choice ");
        choice = sc.nextInt();

        System.out.println ("Enter the No.1: ");
        num1 = sc.nextInt();

        System.out.println ("Enter the No.2: ");
        num2 = sc.nextInt();

        switch (choice)
        {
            case 1: result=num1+num2;
                    System.out.println (" Result is ="+ result);
                    break;
            case 2: result=num1-num2;
                    System.out.println ("Result is ="+ result);
                    break;
            case 3: result=num1*num2;
                    System.out.println (" Result is ="+ result);
                    break;
            case 4: if(num2==0)
                    break;
                    result=num1/num2;
                    System.out.println (" Result is ="+ result);
                    break;
            default:
                     System.out.println (" The choice entered is not correct ");
        }
    }
}
```

Advantages of switch statement:

1. It improves the readability, hence, easy to maintain the program.
2. Suppose, by mistake, if you write 3 instead of 4 in the above 2 programs, what will happen? You will get compile time error in case of switch-case statement as constant value of the case cannot be duplicate. In case of if-else statement, there won't be any compile time error.
3. If the same condition is evaluated for different case values, switch is better.

Program 3.9:

```
import java.util.Scanner;
class Test
{
    public static void main(String [] args)
    {
        Scanner sc = new Scanner (System.in);
        int choice;

        System.out.println ("\n Enter Your Choice ");
        choice = sc.nextInt();
        if(choice==13)
            System.out.println ("Inspiration");

        else if (choice==19)
            System.out.println ("Inspiration ");

        else if (choice==27)
            System.out.println ("Inspiration ");

        else if(choice==89)
            System.out.println ("Inspiration ");

        else if(choice==14)
            System.out.println ("Silence");

        else if(choice==1)
            System.out.println ("Silence");

        else if(choice==21)
            System.out.println ("Silence ");

        else if(choice==93)
            System.out.println ("Silence ");

        else
            System.out.println ("No Luck ");
    }
}
```

The above program 3.9 can be written easily using switch statement as below.

Program 3.10:

```
import java.util.Scanner;
class Test
{
    public static void main(String [] args)
    {
        Scanner sc = new Scanner (System.in);
        int choice=0;

        System.out.println ("\n Enter Your Choice ");
        choice = sc.nextInt();
        switch(choice)
        {
            case 13:
            case 19:
            case 27:
            case 89:
                    System.out.println ("Inspiration");
                    break;
            case 14:
            case 1:
            case 21:
            case 93:
                    System.out.println ("Silence ");
                    break;
            default:
                   System.out.println ("No Luck ");
        }
    }
}
```

In case of lottery, assume that 100 tickets won 1 lakh rupees' prize. Lottery system programmer has to write 100 if statements to show that particular ticket number has got the prize of a 1 lakh rupees. Switch statement makes it easy.

```
switch(choice)
{
    case 121233: case 234234: case 131233: case 235234:  ……
        System.out.println ("Congrats! If your ticket number is"
        + choice "you have won 1 lakh rupees.",choice);
  …….
  ……..
}
```

Limitations:

1. Switch statement can only use integer expressions whose value is compared with case constant values. We cannot use float, double as a constant value of a case.
2. Execution of statements begins with a matched case, but will not end until the keyword break/return/exit is encountered. Otherwise, it will execute all the statements starting from matched case to the end of switch statement.
3. Testing of ranges is not possible. We can test the range of values by using if statement easily.

```
if(i<10 && i>20)
        System.out.println (“Determination”);
 else
        System.out.println (“Concentration”);
```

You need to write following instructions to check a range from 11 to 19 which is practically not convenient.

```
import java.util.Scanner;
class Test
{
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);
        int value=0;
        System.out.println("Enter your value ");
        value= sc.nextInt();
        switch(value)
        {
            case 11:
            case 12:
            case 13:
            case 14:
            case 15:
            case 16:
            case 17:
            case 18:
            case 19:
                    System.out.println("Determination");
            default:
            System.out.println("Concentration");
        }
    }
}
```

We were writing break as a last instruction in the case statement so that other case statements including default below that case statement should not get executed. A new feature is introduced in Java 12. It is not necessary to write break if you want to execute only that particular case. Program 3.8 can be written in Java 12 as below.

I.

```
class Test
{
    public static void main(String args[])
    {
        int i=4;
        switch(i)
        {
            case 1:
                    System.out.println (“A”);
                    break;
            case 2:
                    System.out.println (“ B ”);
                    break;
            case 3:
                    System.out.println (“C ”);
                    break;
```

```
            case 4:
                    System.out.println ("D ");
                      break ;
            default :
                      System.out.println ("E");
        }
        System.out.println ("F");
    }
}
```

Output:
D F

II.

```
class Test
{
    public static void main(String args[])
    {
    int i=4;
    switch(i)
    {
        case 1:
                System.out.println ("A");
                break;
        case 2:
                System.out.println ("B");
                break;
        case 3:
               System.out.println ("C");
               break;
        case 2*2:
                  System.out.println ("D");
                  return ;

        default :
                 System.out.println ("E");
        }
        System.out.println ("F");
    }
}
```

Output:
D F
2*2 evaluates to 4 which is a constant integer expression. Hence, switch integral value 4 exactly matches with case 4 and the statements corresponding to it gets executed.

III

```
class Test
{
    public static void main(String args[])
    {
        int i=1;
        int k=2;
        switch(i)
        {
            int  j=10;
            k=3;
```

```
            case 1:
                    System.out.println (" j= ”+j +“k= "+k);
                    break;
            case 2:
                    System.out.println (" j=” +j+100” k=" + k+1000);
                    break;
        }
    }
}
```

Output:
Compilation error . It won’t execute.

The instructions written inside switch statement without case/default label never gets executed. Hence, j is not initialized and contains a junk value. k is already initialized with a value 2.

IV
```
class Test
{
    public static void main(String args[])
    {
        int i=1;
        int a=1,b=2;
        switch(i)
        {
            case a:
                    System.out.println ("X");
                    break;
            case b:
                    System.out.println ("Y");
                    break;
            default:
                     System.out.println ("Z");
        }
    }
}
```

Output:
Compile time error

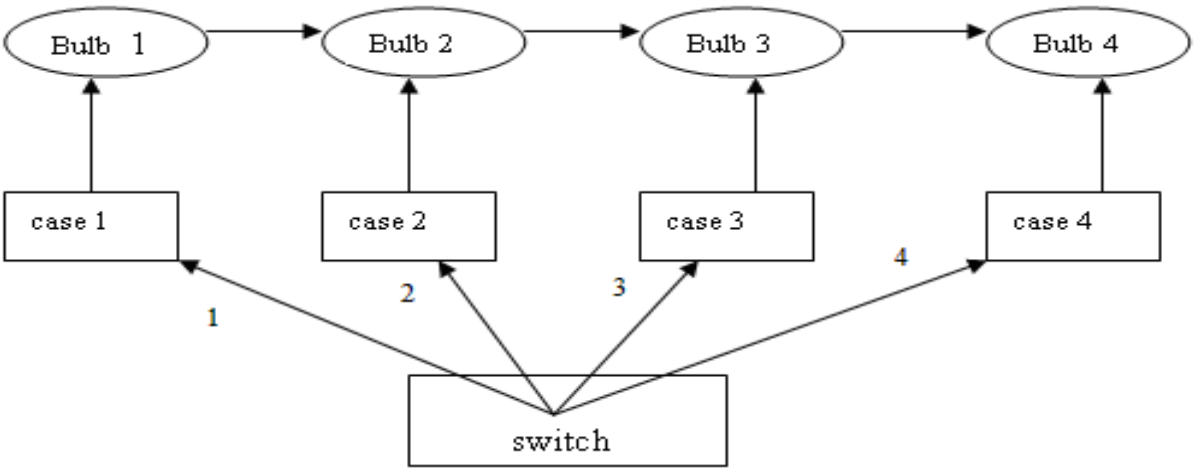
a, b are not constant expressions. case requires constant expression. Hence, it gives compile time error.

Let’s write a program that calculates allowance of various employees based on his/her eligibility. Program will accept input as integer number 1,2,3,4 for software engineer, Team leader, Project Leader and Project Manager respectively.

1. Software Engineer is eligible for car allowance.
2. Team leader is eligible for Car and house rent allowance.
3. Project leader is eligible for Car, house rent and medical allowance.
4. Project Manager is eligible for provident fund, house rent, medical and conveyance allowance.

Allowances are as below.

Car=10,000; House Rent Allowance=20,000;
Medical Allowance=30,000; Conveyance Allowance=40,000;



Program 3.11:

```
import java.util.Scanner;
class Test
{
     public static void main(String[] args)
    {
        int choice;
        int Total_Allowance=0;
        Scanner sc = new Scanner(System.in);
        System.out.println (“Enter the choice “);
        System.out.println ("1. Software Engg ");
        System.out.println (" 2. Team Leader ");
        System.out.println (" 3. Project Leader ");
        System.out.println (" 4. Project Manager ");
        System.out.println (" Enter Your Choice ");
        choice= sc.nextInt();

        switch(choice)
        {
            case 4:
                    Total_Allowance=40000;
            case 3:
                    Total_Allowance+=30000;
            case 2:
                    Total_Allowance+=20000;
            case 1:
                    Total_Allowance+=1000;
                    System.out.println ("Total Allowance is" +Total_Allowance);
            default:
                     System.out.println (" Choice is not valid. ");
        }
    }
}
```

The above example can be compared to the following scenario. Assume that there are 4 bulbs connected in the series as shown in the diagram. If the switch of the bulb1 is turned on, Bulb1, Bulb2, Bulb3, Bulb4 gets switched on. Similarly, if the switch of the Bulb2 is turned on, Bulb1, Bulb3, Bulb4 gets switched on and so on.

3.2.4 Conditional Operator

Let's discuss operators before we discuss conditional operator (ternary operator).

Operator operates on the data. Unary, binary, ternary operator have 1,2 and 3 operands respectively. e.g. & is an unary operator = is a binary operator and? is a ternary operator. The complete list of operators is given in the appendix.

Note that + is an unary as well as binary operator. int k=+10; Here, + acts as a unary operator as it indicates the sign of 10 is positive. int k=10+20; Here, + adds 2 operands 10,20 and acts as a binary operator.

The general form of ternary (conditional) operator is as below.

expression1? expression2: expression3

expression1, expression2 and expression3 are the 3 operands of a ternary operator. It is the abbreviated form (shorthand) to write if-else conditional statement. If expression1 is true (non-zero value), then only expression2 is evaluated. Keep in mind that in this case, expression3 is not evaluated.

Example 1:

Let's write a program 3.2 using conditional operator as below.

Following program accepts Physics and Mathematics marks as input from a keyboard and checks in which subject the student has obtained more marks as compared to the other subject. It is assumed that student has not obtained same marks in Physics and Mathematics.

Program 3.12:

```
import java.util.Scanner;
class Test
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);

        System.out.println ("Enter Physics marks ");
        int PhysicsMarks= sc.nextInt();

        System.out.println ("Enter Mathematics marks ");
        int MathematicsMarks= sc.nextInt();

        int c = (PhysicsMarks > MathematicsMarks)? 1: 2;

        if (c == 1)
            System.out.println("Physics marks are greater than Mathematics marks");
        else
            System.out.println ("Mathematics marks are greater than Physics marks");
    }
}
```

Example 2:

If expression1 is false (non-zero value), then expression3 is evaluated. Remember, expression2 is not evaluated in this scenario.

Program 3.13:

```
class Test
{
    public static void main(String[] args)
    {
        int iVal1=30, iVal2 = 100;
        int c = iVal1 > iVal2 ? 1:2;

         if(c == 1)
            System.out.println ("ABC");
        else
            System.out.println ("XYZ");
    }
}
```

Output:

```
XYZ
```

Exact conditional-expression General Form:

logical-OR-expression? expression: conditional-expression;

i.e. expression1? expression2: expression3

Logical-OR-expression results into true or false i.e. It should result into non-zero(true) and zero(false) value.

The beauty of conditional operator is that we are able to write the convertible if-else statements in a single line instead of 4 lines as shown in the above examples. However, generally conditional operator (ternary operator) is not used because it's difficult to understand and maintain the code.

3.3 Popular Postfix/Prefix operator

Postfix/Prefix operators are unary operators as it operates only on a single value.

++ is known as postfix or prefix increment operator depending upon its presence before or after the operand (variable). If ++ is present before the operand, it is known as prefix increment operator. e.g. ++a. Here, ++ acts as prefix increment operator. If ++ is present after the operand, it is known as postfix increment operator. e.g. a++

Variable	Operator
i++	postfix increment operator
++i	prefix increment operator
i--	postfix decrement operator
--i	prefix decrement operator

Let's understand how postfix/prefix operators are evaluated in expressions.

Program 3.14:

```
class Test
{
    public static void main (String args[])
    {
        int i=5;
        int j;
        j= i++;                    // Statement 3
        System.out.println ("i=" +i+  "j=" +j);
    }
}
```

Output:

```
i=6 j=5
```

In case of postfix ++ operator, the increment of the variable occurs after the execution of the statement. When the statement 3 is executed, the value of a variable i is stored in a variable j. Hence, the value of j is 5. However, after the execution of the statement where postfix increment operator is used, the value of the operand is incremented by 1. Hence, the value of i is 6.

Program 3.15:

```
class Test
{
    public static void main(String args[])
    {
        int i=5;
        int j;
        j=++i;
        System.out.println ("j=" +j);
        System.out.println ("i=" +i);
    }
}
```

Output:

```
j=6
i=6
```

In case of prefix ++ operator, the increment of the variable occurs during the execution of the statement where it is used. Hence, the value of j is 6. After the execution of the statement the value of the operand is same as in the statement where it is used with prefix operator. Hence, the value of i is 6.

Let me explain you the simplest trick to solve prefix/postfix expressions.

Whenever postfix/prefix increment expression occurs in a program, first write the original statement by removing the postfix/prefix notation. Then, convert i++ into an expression i=i+1. If ++ indicates postfix operator, write i=i+1 before the statement in which it is specified.

The above program 3.16 can be rewritten as shown in program 3.17.

```
class Test
{
    public static void main(String args[])
    {
        int i=5;
        int j;
        j= i;                 // Statement int j=i++ is written after removing postfix and prefix notations.
        i=i+1;              // As ++ is a postfix operator, i=i+1 should be written after the original statement.

        System.out.println ("i=" +i+ "j=" +j);
    }
}
```

You can now easily evaluate the values of i and j which is a straightforward calculation.

Similarly, the program 3.17 can be written as below.

```
class Test
{

  public static void main(String args[])
    {
        int i=5;
        int j;
        i=i+1 ;    // As ++ is a prefix operator, i=i+1 should be written before the original statement.
        j= i ;       // Statement int j=i++ is written after removing postfix and prefix notations.

        System.out.println (" i=" + i +  "j=" +j);
    }
}
```

Q. Guess the output of the following program.

```
class Test
{
    public static void main(String args[])
    {
        int i=5;
        int j;
        int k=6;
        j=++ i +k--;
        k=j--;
        System.out.println ("i=" +i+  " j=" +j + "k=" +k);
    }
}
```

Output:

i=6 j=11 k=12

Let’s rewrite the above program in a simple form.

```
class Test
{
    public static void main(String args[])
    {
        int i=5;
        int j;
        int k=6;

        i=i+1;          // ++i is a prefix expression. Hence, i=i+1 is written before original statement.
        j=i+k;          // original statement without any prefix/postfix notation
        k=k-1;          // k-- is a postfix expression. Hence, k=k-1 is written after original statement.

        k=j;            // original statement without any prefix/postfix notation
        j=j-1;          // j-- is a postfix expression. Hence, j=j-1 is written after original statement.
        System.out.println ("i=" +i+  "j=" +j+  "k=" +k);
    }
}
```

Output:
i=6 j=11 k=12

Q. Guess the output of the following program.

I.
```
class Test
{
    public static void main(String args[])
    {
        int i=0;
        if ( --i )
            System.out.println (”A”);
        else
            System.out.println (”B”);
    }
}
```

Output:
Unresolved compilation problem:
Type mismatch: cannot convert from int to boolean

II.
```
class Test
{
    public static void main(String args[])
    {
        boolean i=true;

        if ( i-- )
            System.out.println (”A”);
        else
            System.out.println (”B”);
    }
}
```

Output:
Compilation error

Boolean variable cannot be incremented or decremented. Prefix/postfix operators cannot be applied on it.. The expressions 5++ or (i+j)++ are illegal. Prefix/postfix operators can be applied to only a single variable. It cannot be applied to a constant or expression

3.4 Operator Precedence

Can you guess the output of the following program?

```
class Test
{
    public static void main(String args[])
    {
        int m;
        int i=7; int j=3; int k=8; int n=5;
        m=i*j + k + n;
        System.out.println ("m=" +m);
    }
}
```

You may feel the output is (7*3 + 8) + 5= 34 or 7 * (3 + 8 + 5) = 112. Oh, then, what is the correct output? In the first evaluation 7 and 3 are identified as operands of *, while, in the second evaluation, 7 and (3+8+5) are identified as operands of operator *. Hence, there should be certain rules to determine the operands of operator to avoid calculation ambiguity.

Certain rules are used while computing such expressions in C to solve such ambiguities. Multiplication operator has higher priority than +. i.e. Operands of * operator are determined before determining operands of +. Hence, we can rewrite the above expression as

```
m=(i*j ) + k + n;
```

Operator Precedence Table given in the appendix shows the priority of operators by which its operands are determined. A parenthesis has highest priority.

Operator precedence does not decide the order of the evaluation. It determines the operands of an operator according Operator Precedence Table.

Many people feel that i* j is evaluated first and the result of the evaluation is added to k and l to get the final value of m which is incorrect. It depends upon the compiler whether to add the value of (i*j) into k or the value of k into (i*j). It is not possible to add (i*j) into l without evaluating it. Hence, as a causality, the (i*j) gets evaluated and the resultant expression becomes.

```
m=21+8+5;
```

There is no order of the evaluation of binary + operator. Hence, 21 can be added to 8 or 8 can be added to 21. The conclusion is that

Operator precedence decides the operands of an operator and not the evaluation order.

C does not define the order of the evaluation of operands except 4 operators (logical And, logical Or, conditional operator, comma operator)[1]. e.g. int c=a+b; C does not define whether operand a should be evaluated first and added to b or vice versa.

Q. Guess the output of the following Program.
Assume Physics, Chemistry and Mathematics marks are 95,99 and 99 respectively.

```
import java.util.Scanner;
class Test
{
    public static void main(String args[])
    {
        int PhysicsMarks;
        int ChemistryMarks;
        int MathematicsMarks;
        float PCM_Average;

        Scanner sc = new Scanner(System.in);

        System.out.println ("Please tell me your Physics Marks?\n");
        PhysicsMarks= sc.nextInt();

        System.out.println ("Please tell me your Chemistry Marks?\n");
        ChemistryMarks = sc.nextInt();

        System.out.println ("Please tell me your Mathematics Marks?\n");
        MathematicsMarks = sc.nextInt();

        PCM_Average= (float)(PhysicsMarks + ChemistryMarks + MathematicsMarks /3.0);
        System.out.println ("Average of Physics, Chemistry and Mathematics mark are "
                            +PCM_Average);
    }
}
```

Output:
227.00
As division operation has higher priority than addition, operands close to division operator are bound tightly in the parentheses as shown below. It means the operands of the operator division / are determined first. Hence, the correct output is

PCM_Average=Mathematics+ Chemistry+MathematicsMarks/3;
=95+99+**(99/3.0)**
=95+99+33.00
=227.00

If you use parentheses as shown below, you will get the output 97.666%.

PCM_Average= (Mathematics+ Chemistry+ Mathematics Marks) / 3.0;
= **(95+99+99)/3.0**
=**293**/3.0
=**97.666%**

Let's try to guess the output of the following program.

```
import java.lang.Math;
import java.io.*;
 class Test
{
    float x=3.0*3/4;
    System.out.println("x=" +x);
}
```

You will be in a dilemma as the operator precedence of the * and / is same. So, which operators' operand should be determined first? If you decide the operands of * first then the expression becomes float x=(2.0*3)/4; which results into a value 1.5. And if you decide the operands of / first then the expression becomes float x=2.0*(3/4); which results into a value 0 as 3/4 is 0. Hence, there is ambiguity when the operators of same precedence present in the expression. How to resolve it? Yes, every operator is not only recognized by its precedence but also associativity.

Associativity of operator decides which operator in the expression should be given higher priority when two or more operators of same priority are present in an expression.

Operators * and / have same precedence and their associativity is from left to right. It means the operands of the operator which is present at the left of the expression are determined first and so on. Hence, the above expression is evaluated as float x= (2.0*3)/4; which results into 1.5.

Q. Guess the output of the following program.

I.
```
class Test
{
    public static void main(String args[])
    {
        int x =2*3+4/5-6*7;
        System.out.println ("x=" +x);
    }
}
```

Output:

x= -36

Hope, you understand the following evaluation very easily.

```
int x =2*3+4/5-6*7;
        = (2*3) + 4/5-6*7;        // Left hand side * operator
        = (2*3) + (4/5)-6*;       // / operator
        = (2*3) + (4/5) - (6*7);  // Right hand side * operator
         = (6+0)- 42;             // + operator
         = -36
```

II.
```
class Test
{

    public static void main(String args[])
    {
        int a=1, b=2, c=3, d=4;
        a=b=d=10-c;
        System.out.println ("a=" +a);
    }
}
```

Output:

a=7

a=b=d=(10-c);

a=b=(d=(10-c));

a=(b=(d=(10-c)));

III.
```
class Test
{
    public static void main(String args[])
     {
         int a=1,b=2,c=3,d=4;
         a =-b*c;
         d=10;
         System.out.println ("a=" +a);
     }
}
```

Output:
a= -6

Unary operator – has higher precedence than * and =. Hence, the expression becomes a =((-b)*c)=d=10; As a causality, ((-b)*c) needs to be evaluated. Then, it becomes, a=-6=c=d; We cannot have left operator of assignment operator other than variable name. Hence, it gives compilation error.

3.5 Logical Operators

Logical expression is that which contains logical operators/operators. Logical expression results into a true or false value. Any non-zero value is treated as true, while other values are treated as false. && (AND), || (OR) ,! (NOT) are logical operators.

Let's understand these operators one by one.

```
int k= (1<9) && (51>12) ;  // k is true.
```

In case of && operator, both the operands must be true. Operand 1 is evaluated first. If it is true, then only operand 2 is evaluated.

```
int k= (7==7) && (108>98); // true
int k=false && (6>2);  // false
int k=(6>2) &&  (0 ==1);  // false
int k= (7>7) && (7!=7);  // false
```

Let's write a program that accepts 3 integers (say num1, num2 and num3) from user and displays the highest number. The logic of this program is as below.

1. Accept 3 integer numbers from user.
2. If (num1 is greater than num2 AND num1 is greater than num3)
   Display num1 as highest number
   else if (num2 is greater than num1 AND num2 is greater than num3)
   Display num2 as highest number
   else
   Display num3 as highest number.

Program 3.16:

```
import java.util.Scanner;
class Test
{
    public static void main(String args[])
    {
        int num1, num2, num3;
        Scanner sc = new Scanner(System.in);
        System.out.println ("Enter num1.");
        num1= sc.nextInt();
        System.out.println ("Enter num2.");
        num2= sc.nextInt();
        System.out.println ("Enter num3.\n");
        num3= sc.nextInt();

        if((num1 >num2) && (num1>num3))
            System.out.println ("Highest number= " + num1);
        else if((num2 >num1) && (num2>num3))
            System.out.println ("Highest number= " + num2);
        else
            System.out.println ("Highest number= " + num3);
    }
}
```

In case of logical AND operator, both the operands must be true to get the evaluation of expression true.

Let's write a program that accepts input number from user and prints the message.
"Number entered is either divisible by 3 or 5."

The algorithm is simple

1. Accept input number from user.
2. if ((number is divisible by 3) OR (number is divisible by 5))
   Display "Number entered is either divisible by 3 or 5."

In order to expression condition in the if statement, we need to use logical OR operator. The evaluation of logical expression is true if either of the operand of logical OR operator is true.

```
int k=  ( 1 = = 1) || ( 6 > 3 ) ;  // true
int k= (1 ! = 1 ) || ( 6  > 3 );  ; // true
int k= ( 6 > 3);   ||  (1 ! = 1 ); // true
int k=  ( 1 ! = 1 ) || ( 1 ! = 1 ); // false
```

The program that accepts input integer number and displays "Number entered is either divisible by 3 or 5." can be written as below

Program3.17:

```
import java.util.Scanner;
class Test
{
    public static void main(String args[])
    {
        int num;
        Scanner sc = new Scanner(System.in);

        System.out.println ("Enter input number.");
        num= sc.nextInt();
```

```
        if((num%3==0) || (num%5==0))
            System.out.println( "Number entered is either divisible by 3 or 5.");
    }
}
```

Let's modify program 3.17 so that it should print following message.

Number entered is either not divisible by 3 or 5."

It means we need to check the number entered is either not divisible by 3 or 5. (num%3==0) expression is true when number is divisible by 3. When number is not divisible by 3, we need to invert the result of this expression. NOT (!) operator operates on a single value. If the value is true, it inverts it to false and vice versa.

Following program accepts integer from user and prints the following message.
"Number entered is either not divisible by 3 or 5."

Program 3.18:

```
import java.util.Scanner;
class Test
{
    public static void main(String args[])
    {
        int num;
        Scanner sc = new Scanner(System.in);
        System.out.println ("Enter input number.");
        num= sc.nextInt();
        if(!(num%3==0) || !(num%5==0) )
            System.out.println ("Number entered is either not divisible by 3 or 5.");
    }
}
```

3.6 Points to Remember

1. Conditional operator can be nested.
   int a=10, y=30;
   (a>y)? a=180: ((a>y)? a=180:1);
2. Conditional operator evaluates either expression2 or expression3. It won't evaluate expression2 or expression3 at the same time.
3. Every if-else statement can't be converted into conditional operator because expression3 must be a conditional expression.
   Expression3 must be a conditional expression, otherwise, compile time error will occur. e.g.
   int a=10, y=30;
   (a>y)? a=180: a=1;
   It won't compile as a=1 is not a conditional expression. Expression3 can't be an assignment expression.
4. No expression is optional in case of a conditional operator.
5. If you want to convert if statement into conditional operator, simply put some numeric value in expression3 which doesn't produces any effect on a program.
6. The expression
   operand1? operand2: operand3? operand4: operand5? Operand6: operand7
   can be rewritten for easier understanding like this:
   operand1? operand2:(operand3? operand4:(operand5? operand6: operand7))

Q. Guess the output of the following program.

```
class Test
{

    public static void main(String args[])
    {
        int a, c;
        double b, d;
        a=10;
        b=23.4;
        c=a < 3? a:6;
        System.out.println ("c="+c);
     }
}
```

Output:

6

As per the operator precedence rule, assignment operator has lowest priority. Hence, c=a<3?a:6; becomes c= (a<3?a:6); a<3 is false. Hence, 6 is the output of the conditional operator that is assigned to c.

3.7 Deep Knowledge Section

Q.1 Guess the output of the following program

I.

```
class Test
{
    public static void main(String args[])
    {
        int num1=11 ,num2=98, num3=78;
        if(num1>num2)
            System.out.println ("One");
        if(num2>num3)
            System.out.println ("Two");
        else
           System.out.println ("Three");
    }
}
```

Output:

Two

The ambiguity to evaluate statements in the above program is resolved as follows.

Associate else with the closest previous else-less if.

else is attached with the closest else-less if. Hence,

```
if(num2>num3)
  System.out,println ("Two");
else
  System.out,println ("Three");
```

is treated as separately.

II.

```
class Test
{
    public static void main(String args[])
    {
        int num1=11, num2=98, num3=78;
        if(num1<num2)
        {  //start of block statement
            System.out.println ("One");
            if(num2>num3)
                System.out.println ("Two");
            else
                System.out.println ("Three");
        } // End of block statement
    }
}
```

Output:
One
Two

Block statement is a statement that contain none or many statements. The above used block statement contains System.out.println method call as first statement and if-else statement as a second statement.1.

Questions

Q.1 State true or false.

i. In sequential control flow, instructions are executed one after another.
ii. When numerator and denominator are integer values the result is always integer.
iii. In selection control flow Based on certain condition defined in the program, statements in a program are executed.
iv. Expression can be any combination of variable, operator and/or constant
v. Any non-zero value in Java is treated as true, while, 0 is treated as false.
vi. Flowchart is a graphical representation of an algorithm
vii. No expression is optional in conditional operator.
viii. Operator precedence decides the operands of an operator and not the evaluation order.
ix. Negative value in Java is treated as false.

Q.2 Guess the output of the following program.

i

```
class Test
{
    public static void main(String args[])
    {
        int k=0;
        if(k>=0)
            System.out.println("Mr. Sangram Kendre is very active and smart.");
        else
            System.out.println("Mr. Govind and Atul are ever smiling.");
    }
}
```

ii.
```
class Test
{
    public static void main(String args[])
    {
        int k=2;
        if(k-2==k-2)
            System.out.println (“Balasaheb Kushappa”);
        else
             System.out.println (“Nashik”);
    }
}
```

iii.
```
class Test
{
    public static void main(String args[])
    {
        int k=2;
        if(k=k-2)
            System.out.println (“Calcutta”);
        else
            System.out.println (“Madras”);
    }
}
```

iv.
```
class Test
{
    public static void main(String args[])
    {
        int k=2;
        if(k==k-2)
            System.out.println (“Shrihari”);
        else
            System.out.println (“Akshay”);
    }
}
```

v.
```
class Test
{
    public static void main(String args[])
    {
        int k=2;
        if(k-k)
            System.out.println (“Govinda.”);
        else
            System.out.println (“Vittal.”);
    }
}
```

vi
```
class Test
{
    public static void main(String args[])
    {
        int k=0;
```

```
        if(k++)
            System.out.println (“Ajay Kumbhar.”);
        else
            System.out.println (“Nikhil Delekar”);
    }
}
```

vii

```
class Test
{
    public static void main(String args[])
    {
        int k=0;
        if(++k)
            System.out.println (“Gayatri Bahirat.”);

        else
            System.out.println (“Shravani Bahirat.”);
    }
}
```

viii.

```
class Test
{
    public static void main(String args[])
    {
        int iVal1=30, iVal2=100, c;
        c=iVal1>iVal2?1:2;
        if(c==1)
            System.out.println ("Vishal Delekar");
        else
            System.out.println ("Pooja Delekar");
    }
}
```

ix.

```
class Test
{
    public static void main(String args[])
    {
        int i=5;
        int j;
        System.out.println ("i=" + i++);
        j=i;
        System.out.println ("j=" +j);
    }
}
```

x.

```
import java.util.Scanner;
class Test
{
    public static void main(String args[])
    {
        try
        {
            int PhysicsMarks;
            int MathematicsMarks, c;
```

```
            Scanner sc = new Scanner(System.in);
            System.out.println ("Enter Physics marks ");
            PhysicsMarks= sc.nextInt();

            System.out.println ("Enter Mathematics marks ");
            MathematicsMarks = sc.nextInt();

            c=(PhysicsMarks > MathematicsMarks)?1:2;


            if(c==1)
                System.out.println ("Physics marks are greater than Mathematics marks");
            else
                System.out.println ("Mathematics marks are greater than Physics marks");
        }
    }
}
```

xi.

```
class Test
{
    public static void main(String args[])
    {
        int i=5;
        int j;
        j= i++;
        System.out.println (”i=” +i  ”j=” +j);
    }
}
```

xii.

```
class Test
{
    public static void main(String args[])
    {
        int i=5;
        int j;
        j= ++i;
        System.out.println (”i=” +i  ”j=” +j);
    }
}
```

xiii.

```
class Test
{
    public static void main(String args[])
    {
        int i=5;
        int j;
        int k=6;
        j=++i+k--;
        k=j--;
        System.out.println (”i=” +i  ”j=” +j ” k=” +k);
    }
}
```

xiv.
```
class Test
{
    public static void main(String args[])
    {
        int m;
        int i=7; int j=3; int k=8; int n=5;
        m=i*j +  k + n;
        System.out.println ("m=" +m);
    }
}
```

Q.3 What is a control flow? Discuss various types of control flows with examples?

Q.4 Which control flows are features of structured programming?

Q.5 Which control flow is not a feature of structured programming?

Q.6 What is a side effect in case of expression evaluation? Which unary operator has side effects?

Q.7 What is a sequence point?

Q.8 Compare the followings.

i. structured and unstructured programming
ii. post-increment and pre-increment operator
iii. if-else if and switch statements

Q.9 Which expression is mandatory in case of while and do-while loop?

Q.10 Write a program that finds the average of 4 decimal values supplied by a user?

Q.11 Write a program that determines number supplied by a user is even or odd?

Q.12 Write a program using **if-else if** construct that checks the number supplied by a user is completely divisible by 2,4 and 16?

Answers:

Q1.

i. True **ii.** True **iii.** True
iv. True **v.** True **vi.** True
vii. True **viii.** True **ix.** False

Q2.

i. Mr. Sangram Kendre is very active and smart. **ii.** Balasaheb Kushappa **iii.** Madras
iv. Akshay **v.** Compilation Error **vi. .** Compilation Error
vii. Compilation Error **iii.** Pooja Delekar **ix.** i=5 j=6
x. Compilation Error **xi.** Compilation Error **xii.** Compilation Error
xiii. Compilation Error **xiv.** m=34

Operator Precedence Table

Precedence	Operator	Type	Associativity
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Right to left
13	++ -- + - ! ~ (*type*)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	* / %	Multiplication Division Modulus	Left to right
11	+ -	Addition Subtraction	Left to right
10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to right
9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
8	== !=	Relational is equal to Relational is not equal to	Left to right
7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5	|	Bitwise inclusive OR	Left to right
4	&&	Logical AND	Left to right
3	||	Logical OR	Left to right
2	?:	Ternary	Right to left
1	= += -= *= /= %	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

Chapter

4

STRUCTURED PROGRAMMING: ITERATION CONTROL FLOW

If someone asks you the summation of numbers starting from 0 to 1000, how much time will you require to calculate it? Computer can do this task of summation in fraction of a second.

Human beings have certain limitations where computer's use is highly beneficial. One of the main benefit of a computer is to perform repetitive task without doing any mistakes. This chapter discusses how one can write JAVA language instructions to perform repetitive tasks.

4.1 for loop

Let's write a program that displays five consecutive numbers (series of numbers) starting from 12 and the difference between two consecutive numbers should be 5. It should not include the starting number 12. i.e. Program should print output as 17,22,27,32,37.

The steps involved to do this task manually are as below.

1. Note the starting number as 12 and the difference between two consecutive numbers is 5.
   num = 12; diff = 5;
2. Add the difference number "diff" to "num", store it into "num" and display the new number.
3. Add the difference number "diff" to "num", store it into "num" and display the new number.
4. Add the difference number "diff" to "num", store it into "num" and display the new number.
5. Add the difference number "diff" to "num", store it into "num" and display the new number.
6. Add the difference number "diff" to "num", store it into "num" and display the new number.

The complete program can be written as shown below.

Program 4.1:

```
class Test
{
    public static void main(String[] args)
    {

        int num=12;                                  // step 1
        int diff=5;                                  //  step 1

        num=num+diff;                                // step 2  Repetitive Instructions
        System.out.println("Number=" +num);         // step 2

        num=num+diff;                                // step 3
        System.out.println("Number=" +num);         // step 3

        num=num+diff;                                // step 4
        System.out.println("Number=" +num);          // step 4

        num=num+diff;                                // step 5
        System.out.println("Number=" +num);          // step 5

        num=num+diff;                                // step 6
        System.out.println("Number=" +num);         // step 6
    }
}
```

Output:

```
17
22
27
32
37
```

Q. What is the meaning of the instruction num= num + diff?

Actually in mathematics, num=num+diff is not correct. In case of programming, right hand expression of = (equal to) operator is evaluated first and the result of the evaluation is stored at the left hand side variable.

Left hand side of assignment operator (=) must have variable name. sum+3=sum*4; or 4=sum; is not allowed as left side of = must have unique variable name. Java being a concise language, it allows to express num = num +diff; like num+=diff; += is known as "addition assignment" operator.

The program 4.1 displays five consecutive numbers starting from 12 (excluding 12) and the difference between two consecutive numbers is 5.
Suppose your friend asks you to display 1000 consecutive numbers in this series, how can you enhance the above program?

You can observe that there are repetitive instructions as shown by marked lines. You have to repeat this repetitive instruction to calculate next number in the series of these numbers. As you want to display 1000 consecutive numbers in this series, you have to copy and paste the below instruction 995 times in the above program.

```
num=num+diff;
System.out.println("Number=" +num)
```

But, is it a practical solution? There can be a mistake in writing such lengthy programs. Instead of copying these repetitive instructions 995 times, you may copy it less or more than 995 times by mistake.

Is there any way by which one can write repetitive instruction/instructions once and execute it specified number of times? Can I write following repetitive instructions once and execute it 1000 times?

```
num=num+diff;                          // step 2
System.out.println("Number=" +num)      // step 2
```

Yes, JAVA provides loop programming construct that executes repetitive instruction or instructions for the specified number of times. It helps to write the above program in a simpler manner.

Loop is a programming construct by which a computer can execute set of instructions repetitively for a specified number of times.

We should know following information to write a loop.

- Repetitive set of instructions/instruction
- Number of times the loop is executed
- Initial values of the variables used in the repetitive set of instructions

In case of the program 4.1, information is as below.

Repetitive instructions	num=num+diff; System.out.println("Number=" +num);
Number of times the loop is executed	5
Initial values of the variables used in the repetitive set of instructions	num=12,diff=5

There should be a variable in a program that should count the number of repetitions of the repetitive instruction or instructions. This variable is known as a loop counter i.e. loop counter counts the repetition of the loop.

In case of the program 4.1, we can initialize the value of the loop counter to 0 at the start of the loop. When the repetitive set of instruction gets executed once, loop_counter should get incremented by 1. When the value of the loop counter becomes 5, the loop should stop executing and the next statement in the program should get executed.

General form of a for loop:

```
for (expression1; expression2; expression3)
        Repetitive instruction/ instructions;
```

where

expression1: It is an expression that is used to initialize loop counter and other variables in for loop that needs to be initialized. This expression is executed only once at the beginning of the execution of the loop.

expression2: It is an expression that is used to specify the condition that is tested before the execution of the repetitive instruction/instructions given in the general form of for loop. This statement is executed before the execution of the repetitive statement.

expression3: It is an expression that is used to increment / decrement the loop counter or other variables if required. This expression is executed after the execution of the repetitive statement.

For the above program,

expression1: loop_counter =0
expression2: loop_counter <5
expression3: loop_counter=loop_counter+1

Repetitive Statement:

```
{
    num=num + diff;
    System.out.println("Number=" +num);
}
```

The complete program can be written as below by using for loop construct.

Program 4.2:

```
class Test
{
    public static void main(String[] args)
    {
        int num=12, diff=5;
        int loop_counter;

        for (loop_counter=0;loop_counter<5;loop_counter=loop_counter+1)
        {
            num=num+diff;
            System.out.println("Number=" +num);
        }
    }
}
```

Loop is an example of iteration control flow. Loop counter variable is also referred as a control variable as it controls the execution of the loop.

Let's understand simpler example for the better understanding of for loop.

Practice Example 1: Calculate the sum of the numbers starting from 0 to 12.

Let us store the summation of numbers starting from 0 1to 12 in a variable sum. Before beginning to add the numbers starting from 0, we have to initialize variable sum with value 0.

sum=0;

sum=sum+0
sum= sum+1
sum= sum+2
sum= sum+3
sum= sum+4
sum= sum+5
sum= sum+6
sum= sum+7
sum= sum+8
sum= sum+9
sum= sum+10
sum= sum+11
sum= sum+12

It is seen that left hand operand of + is a variable name i.e. sum is a variable present at the left side of + operator. Right side operand is a constant value which is varying from 0 to 12. Hence, we can write it as sum=sum+ loop_counter where loop_counter is varying from 0 to 12. As loop_counter is varying from 0 to 12, we can

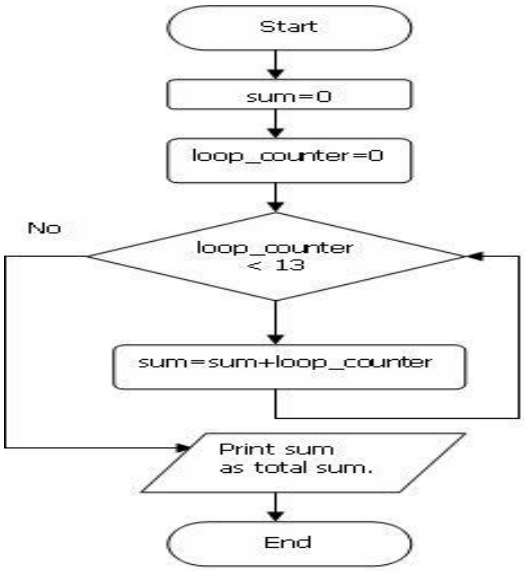
initialize loop_counter to 0, execute the repetitive instructions and increment loop_counter by 1 until it's value is greater than 12.

For the above program,

expression1: loop_counter =0
expression2: loop_counter <13 // 13 because we want to add numbers till 12
expression3: loop_counter++

Repetitive Statement:

```
{
    sum=sum+ loop_counter;
}
```



The complete program can be written as below.

Program 4.3:

```
class Test
{
    public static void main(String[ ] args)
    {
        int sum=0;
        int loop_counter;

        for(loop_counter=0;loop_counter<13;loop_counter=loop_counter+1)
        {
            sum=sum+loop_counter;
        }
        System.out.println("Number=" +sum);
    }
}
```

Q. Is it mandatory to choose a variable name of a controlled variable / loop counter as loop_counter?

No. You can use any Java valid identifier. e.g. count, cnt or i etc. can be used. loop_counter is used for understanding and readability of the program

Practice Example 2: Write a program that calculates a factorial of a given number.

The steps involved are as follows.

1. Ask the user to enter a number for which he or she wants to calculate the factorial.

2. Receive an input number for which you want to calculate the factorial

3. As we know factorial of a number
   Fact=number*(number-1) * (number-2) *…….*1
   e.g. For a number 5
   Fact=5*4*3*2*1
   which can be written as

   Fact=Fact*5;
   Fact=Fact*4;
   Fact=Fact*3;
   Fact=Fact*2;
   Fact=Fact*1;

It means above operation is repetitive where num is varying from num to 1. The decrement is by 1. Hence, we can write

```
expression1: loop_counter =num
expression2: loop_counter >0
expression3: loop_counter--     // Right hand operator of * is varying from num to 1 in a decreasing
                                 // order.
Repetitive Statement:
{
    Fact=Fact*loop_counter;
}
```

4. Display the factorial value.

The complete program can be written as

Program 4.4:

```
import java.util.Scanner;
class Test
{
    public static void main (String args [])
    {
        int loop_counter=0;
        int fact=1;
        Scanner in=new Scanner(System.in);
        System.out.println("Enter the number to find the factorial of a given number.");
        int num=in.nextInt( );
        for(loop_counter=num;loop_counter>0;loop_counter--)
        {
            fact=fact*loop_counter;
        }
        System.out.println ("The factorial of"+num+" is "+fact);
    }
}
```

Advantages of loops:

1) The instructions written by using loop construct are compact (small in size).
2) It's easy to modify, enhance and maintain the source code that is written by using loop construct.
3) Chances of making mistakes in writing a repetitive set of instructions repetitively like a sequential control flow are more. In case of loops, repetitive set of instructions are written only once.
4) Suppose if the number of times of repetition of the instruction set is changed, you have to add repetitive instruction set or delete already written repetitive instruction set / sets in case of a sequential control flow. In case of loops, you have to change the conditional expression.
   e.g. If tomorrow your friends want such 10000 numbers, one has to do underlined change like this

```
for(loop_counter =0; loop_counter <10000; loop_counter++)
{
    num=num+diff;
    System.out.println("Number=" +num);
}
```

Chapter

7

CLASS

7.1 What, Why and When Class Data Type?

Think for a moment that you are staying in a hostel where let's say there are 53 students staying in 53 rooms. Conceive for a while that no room is allocated to a particular student and anyone can stay in any room and also keep his books and other items in any room.

- Will it be easy task for you to locate your particular book after say 2 weeks?
- If any visitor comes to meet you, will it be easy for him/her to locate you from 53 Rooms?

Practically this problem is solved by allocating a particular room to a particular student where he/she can keep his/her books and other items. It means **your books and items are bound to a particular room now** and can be easily located.

Similarly, in programming, it's better idea to group attributes related to a particular entity in a single unit. Oh, but, what is an entity?

Entity is that which is sensed (or known or inferred) to have its own physical existence (living or non-living) e.g. person, car, book etc.

We will learn soon how this (grouping of attributes related to an entity) approach enables systematic organization of data, reduces number of variable names (identifiers) in a program which automatically improves readability and hence maintenance becomes easy.

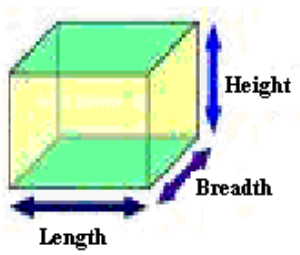
Let me define class at primary level as below. We will improve its definition eventually as your knowledge enhances ☺

Class is a data type that provides a convenient means of grouping attributes related to a particular entity under a single name. It helps for easier handling and identification of data. These attributes can be of similar or dissimilar (int or float or char etc.) data types.

Entity time is associated with 3 attributes. i.e. seconds, minutes and hours. Entity **tile** is associated with attributes: type of tile (like ceramic, marble etc.), length, breadth, price, weight etc.
It seems that it's going from top of your head. Let's try to understand the concept of a class practically by understanding following program.

Following program that checks and displays whether individual metal brick purchased is pure or impure. Assume that the shape of the metal brick is cuboid as shown in the figure.

Metal Name	Length (cm)	Breadth (cm)	Height (cm)	Mass (gm)	Accurate Density gm/cm^3
Silver	10	9.5	2.8	2793	10.5
Gold	7.1	5.8	9.1	89	19.3
Platinum	3.5	2.7	1.8	189	21.4



Hint: i. First, calculate the volume of a cuboid=length*breadth*height;
ii. Then, calculate density=mass/volume;
iii. Check whether calculated and accurate density of a metal are equal.

- **If the calculated density and accurate density of metal are equal, metal is said to be pure.**

Every metal has a unique density. It means density of silver is always 10.5. If the calculated density of a brick is exactly equal to accurate density, the metal is pure.

Program 7.1:

```
public class Test
{
    public static void main(String args[])
    {
        Test t=new Test ();    //See FAQ given below this program

        // Silver Metal Plate
        String s_name="Silver";
        float s_length=10.00f;
        float s_breadth=9.5f;
        float s_height=2.8f;
        float s_mass=2793f;
        float s_accurate_Density=10.5f;

       // Gold Metal Plate
        String g_name ="Gold";
        float g_length=7.1f;
        float g_breadth=5.8f;
        float g_height=9.1f;
        float g_mass=89f;
        float g_accurate_Density=19.3f;
        // Platinum Metal Plate
        String p_name="Platinum";
        float p_length=3.5f;
        float p_breadth=2.7f;
        float p_height=1.8f;
        float p_mass=189f;
        float p_accurate_Density=21.4f;

        t.determineMetalPurity(s_name,s_length,s_breadth,s_height,s_mass,s_accurate_Density);
        t.determineMetalPurity(g_name,g_length,g_breadth,g_height,g_mass,g_accurate_Density);
        t.determineMetalPurity(p_name, p_length,p_breadth,p_height,p_mass,p_accurate_Density);
    }
```

```
    void determineMetalPurity(String name, float length, float breadth, float height, float mass,
                                    float accurate_density)
    {
        float volume, density;
        volume=length*breadth*height;
        density=mass/volume;

        if(density==accurate_density)
            System.out.println("Metal" +name + "is pure.");
        else
            System.out.println("Metal" +name + "is not pure.");
    }
}
```

Output:
Metal Silver is pure.
Metal Gold is not pure.
Metal Platinum is not pure

Q. What is the meaning of Test t=new Test();?

At this stage, just assume that it is a command that is required to call any method defined inside that class. e.g. In this case, Test class is calling a method void determineMetalPurity. Hence, above instruction is written. This will be explained in the later part of this chapter.

In the above program 7.1, there are 6 (data types) variables associated with every metal brick i.e. Metal name, length, breadth, height, mass, accurate density. Hence, total variable names associated with 3 metal plates are 6 $\times$ 3=18.

If we want to check purity of such 100 metal bricks, we require 600 (Number of variable names per metal brick $\times$ Number of bricks=6$\times$100=600) variables (variable names). As the number of the variables increases, the program 7.1 becomes unreadable. Consequently, it becomes difficult at the time of debugging[1] the instructions and subsequently also in the maintenance[2]/ enhancement of an application.

Is there any data type in Java which allows user to define all attributes associated with an entity?

Class is a data type by which attributes associated with an entity can be grouped together and referred as a single unit.

As metal brick is associated with name of the metal, length, breadth, height, mass and accurate density, it is possible to define a class for it.

Class facilitates systematic organization of the data in a program, improves readability and maintenance becomes easy.

[1] Debugging is a process of finding and reducing the number of bugs/defects in a program.
[2] Software Maintenance is the process of enhancing the application or remedying defects.

7.2 Class and Object

7.2.1 Class

7.2.1.1 Declaration/Definition

Like int is a keyword used to declare integer data type, class is a keyword used to declare class data type.

General form of a class declaration/definition is given as below.

```
class class_name
{
    attribute1;
    attribute2;
    ……..
    ……..
    attributen;
}
```

In program 7.1, one can easily identify entity MetalBrick is associated with attributes i.e. name of the metal, length, breadth, height, mass and accurate density. Hence, we can define MetalBrick class as below

```
class MetalBrick
{
    String name;
    float length;
    float breadth;
    float height;
    float mass;
    float accurate_Density;
}
```

Further, it is seen that values of the attributes related to class MetalBrick are initialized as below.

```
// Silver Metal Plate
        String s_name="Silver";
        float s_length=10.00f;
        float s_breadth=9.5f;
        float s_height=2.8f;
        float s_mass=2793f;
        float s_accurate_Density=10.5f;
```

This has been repeated 3 times to initialize values of attributes associate with Silver, Gold & Platinum metal plates. The code is repetitive. So, is it possible to write a method and use it? Yes.

```
void CreateMetalBrick(String ob_name,float ob_mass, float ob_length, float ob_breadth,
                                float ob_height, float ob_accurate_Density)
    {
        name= ob_name;
        length=ob_length;
        breadth=ob_breadth;
        height=ob_height;
        mass=ob_mass;
        accurate_Density=ob_accurate_Density;
    }
```

Is there any data type in Java which allows user to define all the methods associated with an entity?

Class is a data type by which attributes & methods associated with a single entity can be grouped together and referred as a single unit.

In case of metal, CreateMetalBrick and void determineMetalPurity methods are associated with class MetalBrick. The above program 7.1 can be rewritten using class MetalBrick as below.

Program 7.2:

```
class MetalBrick
{
    String name;
    float length;
    float breadth;
    float height;
    float mass;
    float accurate_Density;

   void CreateMetalBrick(String ob_name,float ob_mass, float ob_length, float ob_breadth,
                                float ob_height, float ob_accurate_Density)
    {
        name= ob_name;
        length=ob_length;
        breadth=ob_breadth;
        height=ob_height;
        mass=ob_mass;
        accurate_Density=ob_accurate_Density;
    }

    void determineMetalPurity()
    {
        float volume, density;
        volume=length*breadth*height;
        density=mass/volume;

         if(density==accurate_Density)
            System.out.println("Metal" +name + "is pure.");
        else
            System.out.println("Metal"   +name + "is not pure.");
    }
}

class Test
{
    public static void main(String args[])
    {
        MetalBrick s=new MetalBrick();
        MetalBrick g=new MetalBrick();
        MetalBrick p=new MetalBrick();

        s. CreateMetalBrick("Silver", 2793f, 10.00f, 9.5f, 2.8f, 10.5f);
        g. CreateMetalBrick("Gold", 189f, 7.1f, 5.8f, 9.1f, 19.3f);
        p. CreateMetalBrick("Platinum", 189f, 73.5f, 52.7f, 1.8f, 21.4f);

        s.determineMetalPurity();
        g.determineMetalPurity();
        p.determineMetalPurity();
    }
}
```

Q. What is the meaning of the instructions?

```
MetalBrick s=newMetalBrick();
MetalBrick g=newMetalBrick();
MetalBrick p=new MetalBrick();
```

s, g, p are objects of class MetalBrick. We would be studying objects concept in the next section.

Q. Why there are 2 classes in the above program i.e. MetalBrick & Test?

Metalshape associates attributes and methods related to metal plates, while, Test is driver class. As we know, driver class contains public static void main(String args[]) method which is the starting execution point of any Java program.

Q. What is the meaning of the s.determineMetalPurity();?

s associates with following values of the attributes.

String name;	"Silver",
float length;	2793f
float breadth;	10.00f
float height;	9.5f
float mass;	2.8f
float accurate_Density;	10.5f

These values are used inside determineMetalPurity() method. Hence, there is no need to pass any value separately as a parameter list.

Aim of the object oriented programming languages is to create variables of class in the same way we create variables of built in data types.
e.g. int a=10; Similarly, can we create variable of class as below.

MetalShape t=new MetalShape ("Silver", 2793f, 10.00f, 9.5f, 2.8f, 10.5f) // Variable of a class

Method that is used to create variable of a class is a special method and known as constructor. Name of the class and name of the constructor is the same. Constructor does not have any return data type like other methods. As constructor does not have any return data type, it does not return any value.

Constructor for class MetalShape can be written as below.

```
MetalBrick(String ob_name,float ob_mass, float ob_length, float ob_breadth, float ob_height,
            float ob_accurate_Density)
{
   name= ob_name;
   length=ob_length;
   breadth=ob_breadth;
   height=ob_height;
   mass=ob_mass;
   accurate_Density=ob_accurate_Density;
}
```

7.2.2 Object Declaration

A variable of integer data type is known as an integer variable. Though, it seems easy to think that variable of integer data type is integer variable, variable of a class is as a class variable, But, it is referred as an object and not as a class variable.

Object Definition:

A variable of class is known as an Object.

General Form:

Class_name class_object= new class_name(value of variable1, value of variable2, …..,value of last variable)

Class MetalBrick can be defined as below.

```
MetalBrick s = new MetalBrick ("Silver",10,9.5,2.8,2793,10.5);
```

s is a class object of class MetalBrick. Definition of s indicates

```
s.name="Silver";
s.length =10;
s.breadth =9.5;
s.height =2.8;
s.mass =2793;
s.accurate_Density=10.5;
```

Object is also known as instance of a class.

Following table gives better picture of these concepts.

Table 7.1: Class concepts.

Class Declaration/ Definition	class MetalBrick
	{
	String name;
	float length;
	float breadth;
	float height;
	float mass;
	float accurate_Density;
	}

Object Declaration/Definition	MetalBrick s=new MetalBrick ("Silver",10,9.5,2.8,2793,10.5);

Q: How can we create objects?

In the below code, we are creating three objects, each with its own reference variable s1, s2, s3.

```
MetalBrick s1= new MetalBrick ("Silver",10,9.5,2.8,2793,10.5);
MetalBrick s2= new MetalBrick ("Gold",10,9.5,2.8,2793,10.5);
MetalBrick s3= new MetalBrick ("Silver",10,9,2,2798,10);
```

Q: How can we access objects?

Just like we need variables to access and use values, we need variables to access and reuse the objects that we create. Such variables that are used to access objects are called reference variables. In above example s1, s2, s3 are reference variables.

Q: Can we have method in a class which has the same name of a class?

Yes. As studied, the methods with the name same as the name of class are called constructors of that class. These methods are used to create the objects of the class and assign values to them.

Q: What is new MetalBrick ("Silver",10,9.5,2.8,2793,10.5)?

MetalBrick ("Silver",10,9.5,2.8,2793,10.5) is the constructor of the class MetalBrick. It creates object of the MetalBrick class with the help of **new** operator. new is a keyword.

It is seen from the class definition given in the above table 7.1, attributes are associated with given entity as of now. As constructor of the class is used to create object of that class, can't it be included in class definition? Yes,

The modified class definition with constructor is as below

```
class MetalBrick
{
   String name;
   float length;
   float breadth;
   float height;
   float mass;
   float accurate_Density;

    //Constructor
    MetalBrick(String ob_name,float ob_mass, float ob_length, float ob_breadth, float ob_height,
                float ob_accurate_Density)
    {
       name= ob_name;
       length=ob_length;
       breadth=ob_breadth;
       height=ob_height;
       mass=ob_mass;
       accurate_Density=ob_accurate_Density;
    }

}
```

Naturally, your logical brain will ask you whether it is possible to add method or methods? Yes, methods related to entity for which class is defined are part of the class definition.

The modified class definition with constructor & method is as below

```
 class MetalBrick
{
  String name;
  float length;
  float breadth;
  float height;
  float mass;
  float accurate_Density;
```

```
    MetalBrick(String ob_name,float ob_mass, float ob_length, float ob_breadth, float ob_height,
                    float ob_accurate_Density)                  // Constructor
    {
        name= ob_name;
        length=ob_length;
        breadth=ob_breadth;
        height=ob_height;
        mass=ob_mass;
        accurate_Density=ob_accurate_Density;
    }

    void determineMetalPurity()                 // Method
    {
        float volume, density;
        volume=length*breadth*height;
        density=mass/volume;
        if(density==accurate_Density)
            System.out.println("Metal" +name + "is pure.");
        else
            System.out.println("Metal"   +name + "is not pure.");
    }
}
```

The above program 7.1 can be rewritten using class MetalBrick as below.

Program 7.2:

```
class MetalBrick
{
    String name;
    float length;
    float breadth;
    float height;
    float mass;
    float accurate_Density;
    void CreateMetalBrick(String ob_name,float ob_mass, float ob_length, float ob_breadth,
                                float ob_height, float ob_accurate_Density)
    {
        name= ob_name;
        length=ob_length;
        breadth=ob_breadth;
        height=ob_height;
        mass=ob_mass;
        accurate_Density=ob_accurate_Density;
    }
    void determineMetalPurity()
    {
        float volume, density;
        volume=length*breadth*height;
        density=mass/volume;
        if(density==accurate_Density)
            System.out.println("Metal" +name + "is pure.");
        else
            System.out.println("Metal" +name + "is not pure.");
    }
}
```

```
public class Test
{
    public static void main(String args[])
    {
        MetalBrick s=new MetalBrick();
        MetalBrick g=new MetalBrick();
        MetalBrick p=new MetalBrick();

        s. CreateMetalBrick("Silver", 2793f , 10.00f, 9.5f, 2.8f, 10.5f);
        g. CreateMetalBrick("Gold", 189f, 7.1f, 5.8f, 9.1f, 19.3f");
        p. CreateMetalBrick("Platinum", 189f, 73.5f, 52.7f, 1.8f, 21.4f);

        s.determineMetalPurity();
        g.determineMetalPurity();
        p.determineMetalPurity();
    }
}
```

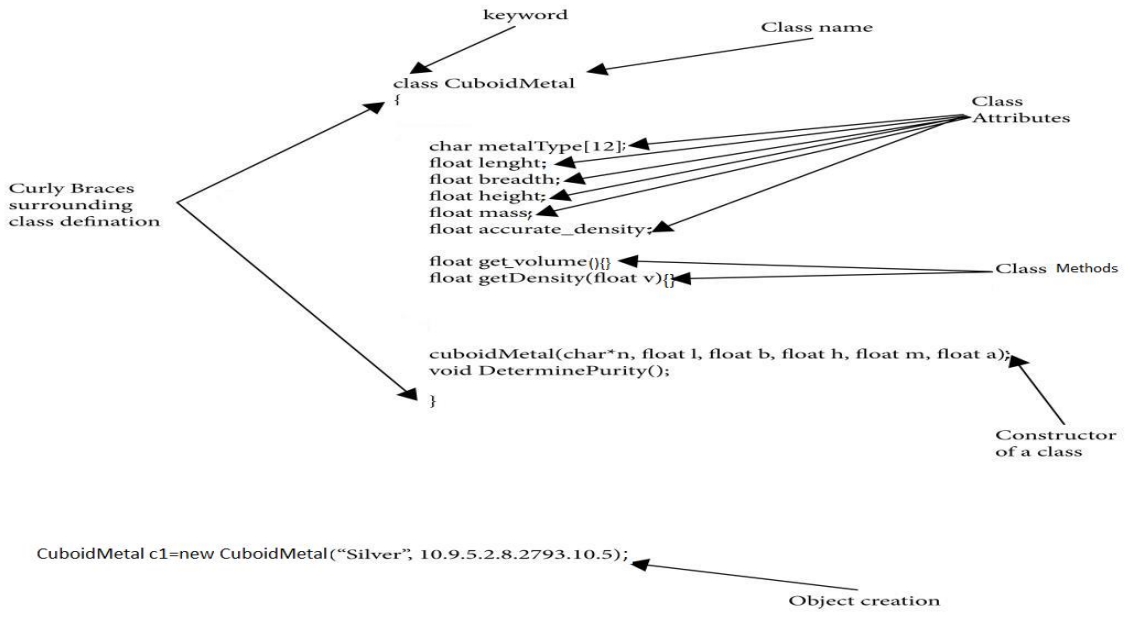
Output:

Metal Silver is pure.
Metal Gold is not pure.
Metal Platinum is not pure

The enhanced complete definition of a class is as below.

Class is a data type that provides a convenient means of grouping attributes & methods related to a particular entity under a single name. It helps for easier handling and identification of data. These attributes can be of similar or dissimilar (int or float or char etc.) data types.

Don't you think class definition is analogous to an architectural plan of a building? Based on the plan, many similar type of buildings can be constructed. Similarly, based on class definition, many objects can be created. Moreover, building's plan is made on the paper/computer and does not require any physical space, but, space is required to construct a building.



Memory representation of object of class MetalBrick is as shown below.

String	length	breadth	height	mass	accurate_Density
String	float	float	float	float	float
1245040	**1245050**	**1245054**	**1245058**	**1245062**	**1245066**

Memory representation of object of class MetalBrick

FAQ:

Q. How many classes are defined in the above program 7.2? Enlist them.

There are 2 classes in the program 7.2:

1. class MetalBricks
2. Driver class Test

Q. How many class objects are defined inside the main Method? Enlist them.

Three class objects defined are s, g and p are defined inside the main method.

Q. Is semicolon mandatory after the definition of a class?

No.

Q. How one can access attributes of a class?

Object of a class can be used to access every attribute of a class. Attributes of a class can be accessed in the following manner.

```
s.name;
```

In the above example, s is an object of a MetalBrick class.

Q. Can I define class and also define object in the same instruction?

Yes.

```
class MetalBrick
{
    String name;
    float length;
    float breadth;
    float height;
    float mass;
    float accurate_Density;
}
MetalBrick s = new  MetalBrick ('Silver', 10, 9.5, 2.8, 2793, 10.5);
```

Q. Is it possible to define multiple objects of same class?

Yes.

```
class MetalBrick
{
   String name;
   float length;
   float breadth;
   float height;
   float mass;
   float accurate_Density;
}
```

```
MetalBrick s = new MetalBrick ("Silver", 10, 9.5, 2.8, 2793, 10.5);
MetalBrick g =new MetalBrick ("Gold",7.1, 5.8, 9.1, 7103, 19.3);        // Another object
MetalBrick p = new MetalBrick ("Platinum",10, 9.5, 2.8, 5400, 21.4); // Another object
```

Q. What are the differences that you have observed in Program 7.1 and Program 7.2?

Sr. No.	Program 7.1 (Without Class Concept)	Program 7.2 (Class Concept)
1	Program 7.1 uses variables of built in data type to store actual data.	Program 7.2 defines class to associate attributes and methods associated with metal brick. Variables of this class (termed as objects) stores actual data.
2	Number of variable names are 18 to store data related to 3 metal bricks.	Numbers of variable names are only 6+3=9 to store data related to 3 metal bricks.

3	6 separate instructions are used to initialize variables associated with a single metal brick.
	String s_name="Silver";
	float s_length=10.00;
	float s_breadth=9.5;
	float s_height=2.8;
	float s_mass=2793;
	float s_accurate_Density=10.5;
		Constructor is used to initialize variables associated with a single metal brick.
		MetalBrick s =new MetalBrick ("Silver",10,9.5,2.8,2793,10.5);

4	Size of the source code (instructions) is more in case of this approach.	Size of the source code (instructions) is less in case of this approach.
5	Method signature requires to pass every attribute associated with an object separately. Hence, the method signature becomes lengthy. void determineMetalPurity(char name[10], float length, float breadth, float height, float mass, float accurate_density);	Irrespective of number of attributes associated with an object, only a single class object can be passed. void determineMetalPurity(MetalBrick t);
6	Program 7.1 is less readable as compared to program 7.2.	Program 7.2 is more readable as compared to program 7.1.
7	Program 7.1 is little difficult to understand as compared to program 7.2. Hence, enhancement and maintenance is comparatively little difficult.	Program 7.2 is easy to understand as compared to program 7.1. Hence, enhancement and maintenance is comparatively easier.

Q. Does program 7.2 saves memory as compared to program 7.1?

No. Memory allocated for variable storage is same in both the cases.

Alternate Definition of a Class:
When we want to declare a, b, c as variables of integer data type, we can declare it as below.

```
int a, b, c;
```

Similarly, if the consecutive attributes are of same data type in class definition, we can combine declarations in a single instruction as below.

```
class MetalBrick
{
    String name;
    float length, breadth, height, mass, accurate_Density;
}
```

Q. Is there any difference between class and object? If yes, compare class with an object?

Sr. No.	Class	Object
1	Class is a blue print (Master copy) to create objects.	It is a variable of class.

2	class MetalBrick
	{
	String name;
	float length;
	float breadth;
	float height;
	float mass;
	float accurate_Density;
	}
		MetalBrick is a class and any variable of it is an object.

3	No memory is allocated when class is defined.	When we define an object, memory is allocated corresponding to the size of the class.
4	Class can be defined only once in a particular scope. e.g. Once MetalBrick class is defined, it cannot be redefined in the program in the same scope.	Many class objects of a particular class can be defined.

Q. Is it possible to assign values of one class object to another object of the same class?

Yes, it is possible and implemented like this

```
MetalBrick s = new MetalBrick (“Silver”,10.00,9.5,2.8,2793,10.5);
MetalBrick p= new MetalBrick ("Platinum",10,9.5,2.8,5400,21.4);
s=p;
```

Q. What is the difference in copying and assigning an Object?

The difference is that in case of copying an object, a new object is created, while, in case of assignment, new copy of the object is not created.

If you have liked this book content, pl, write your review @Amazon.

Book has 17 chapters. You may buy printed copy of this book on Amazon.

https://www.amazon.in/Everyone-MTech-Bombay-Madhusudan-%20Mothe/dp/B07XBL4SVL/ref=sr_1_2?qid=1567584411&refinements=p_27%3AMadhusudan%20+Mothe&s=books&sr=1-2

Or

http://bit.ly/javaforall

Chapter

13

DYNAMIC POLYMORPHISM

13.1 Introduction

My friend Narayana from Andhra and I went for lunch. We took coupons of Andhra meal and Maharashtrian meal for Narayana and me respectively. We handed over coupons to the waiter at the counter and got Andhra and Maharastrian meal for Narayana and me respectively. Based on the type of the coupon, waiter served either Andhra or Maharastrian meal.

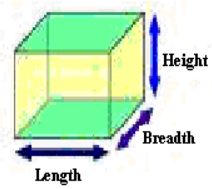
```
Meal    ServeMeal();      // Method
coupon1. ServeMeal();  // Returns Andhra meal as the coupon is for Andhra meal.
coupon2. ServeMeal(); // Returns Maharashtrian meal as the coupon is for Maharashtrian meal.
```

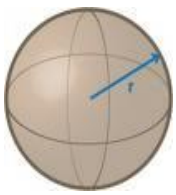
It means though the same method is called by different entities (coupons here), methods associated with that coupon is called. It may go on top of your head. Have some patience and understand the magic of dynamic polymorphism in programming.

13.2 Dynamic Polymorphism: What, When and Why?

Let's write a program that displays information of cuboid metals/ spherical metals as shown in the below table.

Metal Name	**Shape**	**Length (cm)**	**Breadth (cm)**	**Height (cm)**	**Mass (gm)**	**Accurate Density gm/cm^3**
Silver	Cuboid	10	9.5	2.8	2793	10.5
Gold	Cuboid	7.1	5.8	9.1	89	19.3
Platinum	Cuboid	3.5	2.7	1.8	189	21.4
Silver	Spherical	21 (Radius)	Not Applicable	Not Applicable	38808	10.5
Gold	Spherical	42 (Radius)	Not Applicable	Not Applicable	5991955.2	19.3
Platinum	Spherical	63 (Radius)	Not Applicable	Not Applicable	189	21.4





Program 13.1:

```
class MetalShape
{
	protected String metalType;
	protected float mass;
	protected float accurate_density;

	public MetalShape(String n, float m, float a)
	{
		metalType = n;
		mass = m;
		accurate_density = a;
	}

	public void displayData()
	{
		System.out.println("Metal Type: "+metalType);
		System.out.println("Mass: "+mass);
		System.out.println("Accurate Density: "+accurate_density);
		System.out.println();
	}
}

class CuboidMetal extends MetalShape
{
	float length;
	float breadth;
	float height;

	public CuboidMetal(String n, float l, float b, float h, float m, float a)
	{
		super(n, m, a);
		length = l;
		breadth = b;
		height = h;
	}

	public void displayData()
	{
		System.out.println("Metal Type: "+metalType);
		System.out.println("Length: "+length);
		System.out.println("Breadth: "+breadth);
		System.out.println("Height: "+height);
		System.out.println("Mass: "+mass);
		System.out.println("Accurate Density: "+accurate_density);
		System.out.println();
	}
}

class SphericalMetal extends MetalShape
{
	private float radius;
```

```
        public SphericalMetal(String n, float r, float m, float a)
        {
                super(n, m, a);
                radius = r;
        }

        public void displayData()
        {
                System.out.println("Metal Type: "+metalType);
                System.out.println("Mass: "+mass);
                System.out.println("Accurate Density: "+accurate_density);
                System.out.println("Radius: "+radius);
                System.out.println();
        }
}

public class Test
{
        public static void main(String [] args)
        {
            CuboidMetal c[]=new CuboidMetal[3];
            c[0]=new CuboidMetal("Silver",10,9.5f,2.8f,2793,10.5f);      //object c[0] of CuboidMetal Class
            c[1]=new CuboidMetal("Gold",7.1f,5.8f,9.1f,7103,19.3f);      //object c[1] of CuboidMetal Class
            c[2]=new CuboidMetal("Platinum",10,9.5f,2.8f,5400,21.4f) ;  //object c[2] of CuboidMetal Class

            SphericalMetal s[]=new SphericalMetal[3];
            s[0]=new SphericalMetal("Silver",21,407484,10.5f);      //object s[0] of SphericalMetal Class
            s[1]=new SphericalMetal("Gold",42,5991955.2f,19.3f);      //object s[1] of SphericalMetal Class
           s[2]=new SphericalMetal("Platinum",63,22423262.4f,21.4f);      //object s[2] of SphericalMetal   //
                    Class

           for (int i=0; i<3; i++)
           {
                    c[i].displayData();
           }

           for (int i=0; i<3; i++)
           {
                    s[i].displayData();
            }
    }
}
```

Output:

```
Metal Type: Silver
Length: 10.0
Breadth: 9.5
Height: 2.8
Mass: 2793.0
Accurate Density: 10.5

Metal Type: Gold
Length: 7.1
Breadth: 5.8
Height: 9.1
Mass: 7103.0
```

```
Accurate Density: 19.3

Metal Type: Platinum
Length: 10.0
Breadth: 9.5
Height: 2.8
Mass: 5400.0
Accurate Density: 21.4

Metal Type: Silver
Mass: 407484.0
Accurate Density: 10.5
Radius: 21.0

Metal Type: Gold
Mass: 5991955.0
Accurate Density: 19.3
Radius: 42.0

Metal Type: Platinum
Mass: 2.2423262E7
Accurate Density: 21.4
Radius: 63.0
```

The previous program 13.1 displays data for 3 cuboid metals and 3 spherical metals. The number of instructions written in the main method for each cuboid metal or spherical metal are 2. i.e.

```
CuboidMetal c1("Silver", 10, 9.5, 2.8, 2793, 10.5f);
c1.DisplayData();
```

If we want to display data of such 1000 cuboid metals or 1000 spherical metals, we need to write 1000 × 2=2000 instructions for cuboid metal and spherical metal respectively. Is it practicable? No. One can use array to store the objects of cuboid metal and spherical metal as discussed in chapter 2.

The modified main method is as below.

```
public static void main(String args[] )
{
    CuboidMetal    c[]={
                          new CuboidMetal("Silver",10,9.5,2.8,2793,10.5),
                          new CuboidMetal("Gold",7.1,5.8,9.1,7103,19.3),
                          new CuboidMetal ("Platinum",10,9.5,2.8,5400,21.4)
                        };

    SphericalMetal s[]={
                          new SphericalMetal ("Silver",21,407484,10.5),
                          new SphericalMetal ("Gold",42,5991955.2,19.3),
                          new SphericalMetal ("Platinum",63,22423262.4,21.4)
                       };
    for(int i=0;i<3;i++)
       c[i].displayData( );

    for(int i=0;i<3;i++)
       s[i]. displayData( );
}
```

Number of instructions in the main method are less as compared to previous program 13.1. But, this is not the optimized solution because it has *for* loop for every class i.e. cuboid metal, spherical metal in this scenario. If we have 100 classes, then we will require such 100 loops which is not practical. Can we have a single *for* loop that can perform this job?

Yes. Base class reference can contain reference of a derived class object. Hence, instead of creating array of individual objects, let's create an array of base class object which will contain references to the child class. The modified program can be written as below.

Program 13.2:

```
// Classes remains same as defined in program 13.1
public static void main(String[] args)
{
        CuboidMetal c0=new CuboidMetal("Silver",10,9.5f,2.8f,2793,10.5f);
        CuboidMetal c1=new CuboidMetal("Gold",7.1f,5.8f,9.1f,7103,19.3f);
        CuboidMetal c2=new CuboidMetal("Platinum",10,9.5f,2.8f,5400,21.4f) ;

        SphericalMetal s0=new SphericalMetal("Silver",21,407484,10.5f);
        SphericalMetal s1=new SphericalMetal("Gold",42,5991955.2f,19.3f);
        SphericalMetal s2=new SphericalMetal("Platinum",63,22423262.4f,21.4f) ;

        MetalShape m[]= {c0, c1, c2, s0, s1, s2};  //Super class can refer to child class object

        for (int i = 0; i < 6; i++)
        {
                m[i].displayData();
        }
}
```

Output:

```
Metal Type: Silver
Length: 10.0
Breadth: 9.5
Height: 2.8
Mass: 2793.0
Accurate Density: 10.5

Metal Type: Gold
Length: 7.1
Breadth: 5.8
Height: 9.1
Mass: 7103.0
Accurate Density: 19.3

Metal Type: Platinum
Length: 10.0
Breadth: 9.5
Height: 2.8
Mass: 5400.0
Accurate Density: 21.4

Metal Type: Silver
Mass: 407484.0
Accurate Density: 10.5
Radius: 21.0
```

```
Metal Type: Gold
Mass: 5991955.0
Accurate Density: 19.3
Radius: 42.0

Metal Type: Platinum
Mass: 2.2423262E7
Accurate Density: 21.4
Radius: 63.0
```

m[0].DisplayData(); // Calls method of CuboidMetal class as m[0] is address of CuboidMetal object.
m[1].DisplayData(); // Calls method of CuboidMetal class as m[1] is address of CuboidMetal object.
m[2].DisplayData(); // Calls method of CuboidMetal class as m[2] is address of CuboidMetal object.
m[3].DisplayData(); // Calls method of SphericalMetal class as m[3] is address of SphericalMetal object.
m[4].DisplayData(); // Calls method of SphericalMetal class as m[4] is address of SphericalMetal object.
m[5].DisplayData(); // Calls method of SphericalMetal class as m[5] is address of SphericalMetal object.

DisplayData method is known as overridden method because it is implemented in the base class (MetalShape) as well as in the derived class (CuboidMetal). When a base class object reference calls overridden method, overridden method of respective class is called based on the object that base class object reference refers instead of always calling base class method. This process is also known as late binding or runtime polymorphism.

In the above program, m is the reference of class MetalShape calls overridden method i.e. DisplayData(), But the object stored at m[i] is of either of the derived classes. Hence, method of respective derived class is called.

In Java we make use of super class reference to call methods of child class. This is how concept of Late binding or Runtime polymorphism is achieved.

The program 13.2 can be compacted by calling DisplayData method of base class (MetalShape) from DisplayData method defined in CuboidMetal or SphericalMetal as shown below.

```
void displayData( )
{
    System.out.println("Length"+length);
    System.out.println("Breadth"+breadth);
    System.out.println("Height"+height);
}
```

You can modify displayData method of SphericalMetal class and execute program 13.2.

Polymorphism refers to the concept of defining single method in different ways.

FAQ:

Q. How does dynamic polymorphism is achieved in java?

Method overriding is the way to achieve dynamic (run time) polymorphism in java.

Q. What is method overriding in java?

In method overriding both parent and child classes have same method name. At run-time, it depends on the type of the object being referred to (not the type of the reference variable) that determines which class's overridden method will be executed.

This is already explained in the above section for CuboidMetal

```
class Test
{
    public void print()          //overriden method
    {
        System.out.println("print in superclass.");
    }
}

class subclass extends Test
{
    @Override
    public void print()
    {
            System.out.println("print in subclass.");
    }

     public static void main(String[] args)
    {
            Test A = new Test();           //object of superclass
            Test B = new subclass();       //object of subclass
            A.print();
            B.print();
    }
}
```

Output:
print in superclass.
print in subclass.

Q. Is there any special method which cannot be overridden?

Yes, Final method cannot be overridden. When a method is declared as final, then that method can be overloaded but can't be overridden in the sub classes.

```
class Test1
{
    final void print()          //overridden method
    {
            System.out.println("print in superclass Test.");
    }
}
class subclass extends Test1
{
    @Override
    void print(String Hi)
    {
            System.out.println("print in subclass.");
    }
}
public class Test
{
    public static void main(String[] args)
    {
```

```
        subclass s=new subclass();
        s.print();
    }
}
```

Output:
print in superclass Test.

13.3 Abstract Method

Let's implement abstraction for program 10.7. Here, DeterminePurity method of derived classes is called. Hence, we should declare DeterminePurity method as abstract in base class and create an array of base class object references. But, base class i.e. MetalShape does not contain DeterminePurity method. Hence, we need to define empty DeterminePurity method as shown in the below program and declare it as abstract.

Program: 13.3

```
abstract class MetalShape
{
    private String metalType;
    private float mass;
    private float accurate_density;

    public MetalShape(String n, float m, float a)
    {
            metalType = n;
            mass = m;
            accurate_density = a;

  abstract void determinePurity();                    // abstract method

    protected String getMetalType()
    {
            return metalType;
    }
   protected float getMass()
    {
            return mass;
    }

    protected float getAccurate_density()
    {
            return accurate_density;
    }

    public float getDensity(float vol)
    {
            float d=mass/vol;
            return d;
    }
}
```

```
class CuboidMetal extends MetalShape
{
    float length;
    float breadth;
    float height;

    float getVolume()
    {
            float v=length*breadth*height;
            return v;
    }

    public CuboidMetal(String n, float l, float b, float h, float m, float a)
    {
            super(n, m, a);
            length = l;
            breadth = b;
            height = h;
    }

    public void determinePurity()
    {
            float v=getVolume();
            float d=getDensity(v);

            if(d==getAccurate_density())
               System.out.println("Metal "+getMetalType()+" is pure.");
            else
               System.out.println("Metal "+getMetalType()+" is not pure.");
    }
}
class SphericalMetal extends MetalShape
{
    private float radius;

    private float getVolume()
    {
            float v=4.0f/3.0f*22.0f/7.0f*radius*radius*radius;
            return v;
    }

    public SphericalMetal(String n, float r, float m, float a)
    {
            super(n, m, a);
            radius = r;
    }

    public void determinePurity()
    {
            float v=getVolume();
            float d=getDensity(v);

            if(d==getAccurate_density())
               System.out.println("Metal "+getMetalType()+" is pure.");
```

```
            else
                System.out.println("Metal "+getMetalType()+" is not pure.");
    }
}

public class Test
{
    public static void main(String[] args)
    {
        CuboidMetal c0=new CuboidMetal("Silver",10,9.5f,2.8f,2793,10.5f);
        CuboidMetal c1=new CuboidMetal("Gold",7.1f,5.8f,9.1f,7103,19.3f);
        CuboidMetal c2=new CuboidMetal("Platinum",10,9.5f,2.8f,5400,21.4f) ;

        SphericalMetal s0=new SphericalMetal("Silver",21,407484,10.5f);
        SphericalMetal s1=new SphericalMetal("Gold",42,5991955.2f,19.3f);
        SphericalMetal s2=new SphericalMetal("Platinum",63,22423262.4f,21.4f) ;

        MetalShape m[]= { c0, c1, c2, s0, s1, s2 };  //Super class can refer to child class object

        for (int i = 0; i < 6; i++)
        {
                m[i]. determinePurity();
        }
    }
}
```

Output:

```
Metal Silver is pure.
Metal Gold is not pure.
Metal Platinum is not pure.
Metal Silver is not pure.
Metal Gold is not pure.
Metal Platinum is pure.
```

Previous program 13.3 works successfully. But, one thought may linger in your mind: Why one should provide determinePurity method in base class MetalShape with empty body though practically object of MetalShape class does not exist? Yes, you are right. It is not a good idea to create an object of MetalShape because there is nothing like metal physically in existence. MetalShape represents abstraction of family of classes like CuboidMetal, SphericalMetal etc. If someone asks you to show metal shape physically, you will show cuboid, spherical, cube metals etc., but, you won't be able to show metal shape. MetalShape represents common features of such derived classes like CuboidMetal, SphericalMetal etc. It is true that every subclass of MetalShape should implement methods like determinePurity().

Is there any way by which base class can force derived classes to implement certain method/methods?

Yes. This can be achieved by declaring Abstract methods. The syntax of abstract method is as below.

abstract void determinePurity ();

Class that contains at least one abstract method is called as an abstract class.

MetalShape class is an abstract class. One cannot create object of an abstract class. But references can be created.

The modified version of previous program is as below.

Program: 13.4

```
abstract class MetalShape
{
    private String metalType;
    private float mass;
    private float accurate_density;

    public MetalShape(String n, float m, float a)
    {
            metalType = n;
            mass = m;
            accurate_density = a;
    }

    public abstract void determinePurity();  //abstract method

    protected String getMetalType()
    {
            return metalType;
    }

    protected float getMass()
    {
            return mass;
    }

    protected float getAccurate_density()
    {
            return accurate_density;
    }

    public float getDensity(float vol)
    {
            float d=mass/vol;
            return d;
    }
}

class CuboidMetal extends MetalShape
{
    float length;
    float breadth;
    float height;

    float getVolume()
    {
            float v=length*breadth*height;
            return v;
    }

    public CuboidMetal(String n, float l, float b, float h, float m, float a)
    {
        super(n, m, a);
        length = l;
        breadth = b;
```

```
        height = h;
    }

    public void determinePurity()
    {
        float v=getVolume();
        float d=getDensity(v);


        if(d==getAccurate_density())
            System.out.println("Metal "+getMetalType()+" is pure.");
        else
            System.out.println("Metal "+getMetalType()+" is not pure.");
    }
}

class SphericalMetal extends MetalShape
{
    private float radius;

    private float getVolume()
    {
        float v=4.0f/3.0f*22.0f/7.0f*radius*radius*radius;
        return v;
    }

    public SphericalMetal(String n, float r, float m, float a)
    {
        super(n, m, a);
        radius = r;
    }

    public void determinePurity()
    {
        float v=getVolume();
        float d=getDensity(v);
        if(d==getAccurate_density())
            System.out.println("Metal "+getMetalType()+" is pure.");
        else
            System.out.println("Metal "+getMetalType()+" is not pure.");
    }
}
public class Test
{
    public static void main(String[] args)
    {
        CuboidMetal c0=new CuboidMetal("Silver",10,9.5f,2.8f,2793,10.5f);
        CuboidMetal c1=new CuboidMetal("Gold",7.1f,5.8f,9.1f,7103,19.3f);
        CuboidMetal c2=new CuboidMetal("Platinum",10,9.5f,2.8f,5400,21.4f) ;

        SphericalMetal s0=new SphericalMetal("Silver",21,407484,10.5f);
        SphericalMetal s1=new SphericalMetal("Gold",42,5991955.2f,19.3f);
        SphericalMetal s2=new SphericalMetal("Platinum",63,22423262.4f,21.4f) ;
```

```
        MetalShape m[]= {c0, c1, c2, s0, s1, s2};  //Super class can refer to child class object

        for (int i = 0; i < 6; i++)
        {
                m[i]. determinePurity();
        }
    }
}
```

Output:

```
Metal Silver is pure.
Metal Gold is not pure.
Metal Platinum is not pure.
Metal Silver is not pure.
Metal Gold is not pure.
Metal Platinum is pure.
```

Q. Is it mandatory to implement determinePurity method of MetalShape class if one wants to derive new classes like a CubicalMetal from MetalShape class?

Yes, it's mandatory. Otherwise, you will get compilation error while compiling that newly derived class.

Q. Is it possible to create object of an abstract class?

No. It is a compilation error. Reason is that metal does not exist physically, but, various metallic shapes exist. But you can create references.

Q. Constructor is used to create an object. One cannot instantiate (create object) of an abstract class. Then, why abstract class have constructor as shown in MetalShape class?

When derived class object is created, it's necessary to call base class constructor in order to initialize attributes in base class. MetalShape class initializes mass, metalType, accurate_density for all its derived classes.

Q. xyz method is declared abstract in base class. Will it get inherited in its derived classes?

Yes. Abstract methods are not implemented where they are declared. They get inherited in immediate derived classes and implemented there.

13.4 Upcasting, Downcasting and Object Slicing

13.4.1 Upcasting

We have studied that runtime polymorphism is achieved by storing reference of derived class object into base class object and calling the overridden method. E.g. In case of classes MetalShape, CuboidMetal defined in the program 13.3, we can implement runtime polymorphism as below.

```
MetalShape ref=new CuboidMetal();       // Storing reference of derived class object into base class reference is called as upcasting.
ref.determinePurity();                  // Call to overridden method
```

When reference of derived class object is stored into base class reference, it is known as upcasting. There's no need for programmer to upcast manually. First, you must understand, that by casting you are not actually changing the object itself, you are just labeling it differently. For example, if you create a CubicalMetal and upcast it to MetalShape, then the object doesn't stop from being a CubicalMetal. It's still a CubicalMetal, but it's just treated

as any other MetalShape and it's CubicalMetal properties are hidden until it's downcasted to a CubicalMetal again. Let's see an example.

```
class MetalShape {          }

class CubicalMetal  extends MetalShape { }
class SphericalMetal  extends MetalShape { }

public class Test
{
    public static void main(String[] args)
   {
       CubicalMetal c = new CubicalMetal();
       System.out.println(c);
       MetalShape m = c; // upcasting
       System.out.println(m);
    }
}
```

Output:

```
CubicalMetal@70dea4e
CubicalMetal@70dea4e
```

As you can see, CubicalMetal is still exactly the same CubicalMetal after upcasting, it didn't change to a MetalShape, it's just being labeled MetalShape right now. This is allowed, because CubicalMetal is a MetalShape. Note that, even though they are both MetalShape, CubicalMetal cannot be cast to a SphericalMetal.

13.4.2 Downcasting

When reference of base class object is assigned to derived class reference, it is known as downcasting. If we perform it directly, compiler gives Compilation error.

```
CuboidMetal obj=new MetalShape();        // This won't compile.
```

Downcasting is performed manually. Now you will think,Why is that so, that upcasting is automatic, but downcasting must be manual? Well, you see, upcasting can never fail. But if you have a group of different MetalShape and want to downcast them all to a CubicalMetal, then there's a chance, that some of these MetalShape are actually SphericalMetal, and process fails, by throwing ClassCastException. Let's see an example.

```
class MetalShape {}

class SphericalMetal extends MetalShape
{
  public static void method(MetalShape m)
   {
          SphericalMetal s=(SphericalMetal)m;//downcasting
          System.out.println("ok downcasting performed");
    }
}


class Test
{
```

```
    public static void main (String [] args)
    {
        MetalShape a=new SphericalMetal();
        SphericalMetal.method(a);
    }
}
```

Output:
ok downcasting performed

Note, that casting can't always be done in both ways. If you are creating a MetalShape, by calling "new MetalShape ()", you are creating an Object that is a MetalShape, but it cannot be downcasted to CubicalMetal or SphericalMetal, because it's neither of them.

For example:

```
MetalShape m = new MetalShape ();
SphericalMetal s = (SphericalMetal)m;
```

Such code passes compiling, but throws "java.lang.ClassCastException: MetalShape cannot be cast to SphericalMetal "exception during running, because I am trying to cast a MetalShape, which is not a SphericalMetal.
General idea behind casting, is that, which object is which. You should ask, is SphericalMetal a MetalShape? Yes, it is - that means, it can be cast. Is MetalShape a SphericalMetal? No it isn't - it cannot be cast. Is SphericalMetal a CubicalMetal? No, it cannot be cast.

13.4.3 Object Slicing

When derived class object is assigned to base class object, base class portion of derived class object is copied into base class object and derived class portion of derived class object is sliced off. This is known as object slicing. You can simply say object slicing occurs when derived class object is assigned to base class object, derived class portion of the object is sliced off. e.g.

```
MetalShape ms=new MetalShape ("Platinum",5400f,21.4f); //As MetalShape class in 13.1 program is not abstract, we can create object.
CuboidMetal c=new CuboidMetal ("Silver",10f,9.5f,2.8f,2793f,10.5f);
ms=c; // Object slicing happens here
```

The contents metalType (“Silver”), mass (10), accurate_density(9.5) of object c are copied as attributes of object m, while, the remaining part of object c i.e. length(2.8), breadth(2793) and height(10.5) are sliced off.

Guess the output of the following program.

Q1.

```
class A
{
    public void display()
    {
        System.out.println("A");
    }

    void show()
    {
        System.out.println("Show A");
    }
}

class B extends A
{
```

```
    public void display()
    {
        System.out.println("B");
    }
    void show()
    {
        System.out.println("Show B");
    }
}

class C extends B
{
    public void display()
    {
        System.out.println("C");
    }

    void show()
    {
        System.out.println("Show C");
    }
}

public class Test
{
    public static void main(String arg[])
    {
        A a1=new C();
        a1.show( );
    }
}
```

Output:
Show C

Q2.

```
//Refer class A, B and C defined in the above Q1.
public class Test
{
        public static void main(String arg[])
      {
           B b1=new C();
           b1.show();
    }
}
```

Output:
Show C

Q3.

```
class A
{
    public void display( )
    {
        System.out.println("A ");
     }
```

```
}

class B extends A
{
    public B ()
    {
        display();
    }

    void display()
    {
        System.out.println("B");
    }
}

public class Test
{
    public static void main(String arg[])
    {
        B b1=new B();
        b1.display();
    }
}
```

Output:
Compilation Error: overridden method should have same signature

Q4.

```
class A
{
     public void display()
    {
        System.out.println("A ");
     }

     public void display(int a)
    {
        System.out.println("A: "+a);
     }

}

class B extends A
{
   public void display( )
    {
        System.out.println("B ");
     }
     public void display(int a)
    {
        System.out.println("B: "+a);
     }
}
public class Test
{
```

```
    public static void main(String args[])
    {
        A a1=new  B( );
        a1.display( 9 );
    }
}
```

Output:
B: 9
It is possible to overload and override the same method. i.e. Display method is overloaded in class A and class B and overridden in a class hierarchy.

13.5 Static Binding and Dynamic Binding

Connecting a method call to the method body is known as binding.
There are two types of binding.

1. Static Binding / Early Binding/ Method overloading
2. Dynamic Binding / Late Binding/ Method overriding

13.5.1 Static binding/ Early Binding/ Method overloading

When type of the object is determined at compiled time (by the compiler), it is known as static binding. It is also known as Early Binding or Method overloading. If there is any private, final or static method in a class, there is static binding.

Let's write a program that finds maximum number from the following set of numbers.

a) 11,123
b) 121,22,55
c) 34.34,22.11,55.01

Program 13.5:

```
public class Test
{
    public static int max1(int a,int b)
    {
        if (a>b)
            return a;
        else
            return b;
    }
    public static int max2(int a, int b, int c)
    {
         int highest;
         if (a>b)
             highest= a;
         else
             highest=b;
         if(c>highest)
             highest=c;
          return highest;
    }
```

```
    public static float max3(float d, float e, float f)
    {
        float highest;
        if (d>e)
           highest= d;
        else
           highest=e;

        if(f>highest)
           highest=f;
       return highest;
    }

    public static void main(String [] args)
    {
         System.out.println("The maximum number between 11 and 123 is "+ max1(11,123));

         System.out.println("The maximum number between 121,22 and 55 is "+max2(121,22,55));

          System.out.println("The maximum number between 34.34,22.11 and 55.01 is
          "+max3(34.34f,22.11f,55.01f));
    }
}
```

Output:

```
The maximum number between 11 and 123  is 123
The maximum number between 121,22 and 55  is 121
The maximum number between 34.34,22.11 and 55.01 is 55.01
```

It is seen that the max1, max2 and max3 methods are used to find maximum number from two integers, three integers, and three float respectively. These methods perform similar operations of finding highest number.

Is there any way by which one can define more than one method with the same name? i.e. Can we use the same method name max to define two or more methods that can have different number or type or order of parameters (arguments)?

In Java, it is possible to define methods that can have same name. This concept is known as method overloading. Method overloading is a programming concept that allows programmers to define two or more methods with the same name. Though the method name is same, compiler differentiates these methods by number of parameters, type of each parameter and order of parameters specified in the method signature. The return types of methods need not be the same.

Let's go through the below examples of method overloading. Though the method names are same, compiler differentiates them based on method overloading mechanism: number of parameters, type of each parameter and order of parameters.

Number of parameters:

```
 int max(int a,int b);
 int max(int a,int b,int c);
```

Parameters' type:

```
 int max(float a,int b);
 int max(int a,int b);
```

Parameters' order:

```
int max(float a,int b);
 int max(int a, float b);
```

Following program defines max method that accepts two, three parameters and find the maximum value. These methods are defined for integer and float types.

```
int   max(int a,int b);
int   max(int a,int b,int c);
float max(float a,float b);
float max(float a,float b,float c);
```

Method overloading is a mechanism by which one can define more than one method by same name that are differentiated by a compiler either by their parameter list, number of parameters or order of parameters. All these set of methods defined by same name should perform similar methodality.

Program 13.6:

```
public class Test
{
    public static int max(int a,int b)
    {
        if (a>b)
           return a;
        else
           return b;
    }

    public static int max(int a,int b,int c)
    {
        int highest;

        if (a>b)
           highest= a;
        else
           highest=b;

        if(c>highest)
           highest=c;
         return highest;
    }
    public static float max(float d,float e,float f)
    {
        float  highest;
        if (d>e)
           highest= d;
         else
        highest=e;

        if(f>highest)
            highest=f;
        return highest;
    }
    public static void main(String [] args)
    {
        System.out.println("The maximum number between 11 and 123 is "+ max(11,123));

        System.out.println("\nThe maximum number between 121,22 and 55 is "+max(121,22,55));

         System.out.println("\nThe maximum number between 34.34,22.11 and 55.01 is
         "+max(34.34f,22.11f,55.01f));
```

```
    }
}
```

Output:

```
The maximum number between 11 and 123 is 123
The maximum number between 121,22 and 55 is 121
The maximum number between 34.34,22.11 and 55.01 is 55.0
```

13.5.2 Dynamic binding/ Late Binding/ Method overriding

When type of the object is determined at run-time, it is known as dynamic binding. It is also known as Late Binding or Method overriding. Other than private method, static and final methods can be overridden. Please refer dynamic polymorphism topic where method overriding is explained.

13.6 Points to Remember

1. Abstract method must be a member of a class. It cannot be defined at global level.
2. Abstract method cannot be static.
3. Abstract methods are accessed by using object references or even the object can access it directly.
4. An abstract method in a base class must be defined, even though it may not be used.
5. The prototype of the base class version of an abstract method and all the derived class versions must be identical. If method with the same name have different prototype, Java considers them as overloaded methods and the method abstraction is ignored.
6. A reference to base class can refer to any type of the derived object, the reverse is not true. i.e. We cannot directly use a reference to a derived class to access an object of the base type. It can be achieved by downcasting.
7. It is not possible to declare a constructor as abstract.
8. It is possible to use switch case statements instead of dynamic polymorphism. One can understand the type of the object and make appropriate method call. But, use of dynamic polymorphism helps to write more compact code.

```
// Classes remains same as defined in program 13.1
public class Test
{
        public static void main(String [] args)
        {

            CuboidMetal c=new CuboidMetal("Silver",10,9.5f,2.8f,2793,10.5f);
            SphericalMetal s=new SphericalMetal("Platinum2",21,407484,10.5f);

             Scanner scanner = new Scanner(System.in);
             System.out.print("Enter choice :");
             String  ch = scanner.nextLine();

            switch(ch)
            {
                case "Cuboidal":
                        c.displayData();
                        break;
                case "Spherical":
                        s.displayData();
                        break;
                default:
                        System.out.println("Invalid operator!");
                  break;
              }
        }
}
Output:
Enter choice: Cuboidal

Metal Type: Silver
Length: 10.0
Breadth: 9.5
Height: 2.8
Mass: 2793.0
Accurate Density: 10.5
```

9 Disadvantage of using abstract method is that they require more instructions for execution and hence slower. Another disadvantage is that it is difficult to understand the method call that is made which makes maintenance difficult.

10 It is possible to force derived classes to implement particular methods by declaring those methods as abstract in a base class.

11 Abstract class can have non abstract methods apart from abstract method or methods.

12 A class which allows to create its object is called as a concrete class.

13.7 Deep Knowledge Section

Q1. Does an abstract method have implementation?

Yes.

As shown in the below program, it is possible to define abstract outside the class definition.

```
abstract class A
{
    public void display()
    {
        System.out.println("A");
    }

    abstract void show();
}

class B extends A
{
    public void display()
    {
        System.out.println("B");
    }


   void show()
   {
        System.out.println("show A");                    // Implementation of abstract method.
        System.out.println("show B");
   }
}
public class Test
{
    public static void main(String args[])
    {
        B b1=new B();
        b1.show();
    }
}
```

Output:

```
show A
show B
```

Q 2. Guess the output of the following program.

```
class A
{
    public void display()
    {
        System.out.println(“A”);
    }
}
class B extends A
{
    private void display()
    {
        System.out.println(“B”);
    }
}
class Test
{
    public static void main(String arg[])
    {
        A ptr=new B();
        ptr.display();
    }
}
```

Output:
Compilation Error

In java, signature of the method should be same while overriding it.

Q 4. Guess the output of the following program.

```
class A
{
     public void display( )
    {
        System.out.println("A ");
    }
}

abstract class B extends A
{
    public void display()
    {
        System.out.println("B ");
    }

    public abstract void show();
}
class C extends B
{
    public void display()
    {
        System.out.println("B ");
    }
```

```
    public void show()
    {
            System.out.println("Showing C");
        }
}
public class Test
{
    public static void main(String args[])
    {
        B b=new C();
        b.show();
    }
}
```

Output:
Showing C

It means it is possible that base class can be a concrete class, while, derived class is abstract class. In the above program, class A is concrete class, while, class B is abstract class.

Questions

Q.1 State true(T) or false(F).

i. Polymorphism is nothing but one name, multiple forms.
iii. Dynamic binding means late binding.
iii. We can create object of an abstract class.
iv. Static binding and dynamic polymorphism is same.
v. Abstract methods must be static.
vi. Abstract class can contain constructor.
vii. Abstract methods call up is maintained by compiler via lookup tables.
viii. Abstract methods can have definition.
ix. Abstract methods can have empty implementation.
x. Assigning address of derived class object to base class reference is downcasting.
xi. Assigning base class object to derived class object is object slicing.

Q2. Guess the output of the following programs.

i.
```
class A
{
    private void display()
    {
        System.out.print(”A”);
    }
}

class B extends A
{
    public void display()
    {
        System.out.print(”B”);
    }
}
```

```
public class Test
{
    public static void main(String args[] )
    {
        A a=new B();
        a.display() ;
     }
}
```

ii.

```
class A
{
    public void display()
    {
        System.out.print(”A”);
    }
}

class B extends A
{
    private void display()
    {
        System.out.print(”B”);
    }
}
public class Test
{
    public static void main(String args[] )
    {
        A a=new B();
        a.display() ;
    }
}
```

iii.

```
class A
{
    public void display()
    {
        System.out.print(”A”);
    }
}

class B extends A
{
    public void display()
    {
        System.out.print(”B”);
    }
}

public class Test
{
    public static void main(String args[])
```

```
    {
        A a=new A();
        a.display() ;
    }
}
```

iv.

```
class A
{
   public A()
   {
      System.out.println(“A Constructor");
   }
   void Greet()
   {
      System.out.println(“A says hi");
   }

   void Talk()
   {
      System.out.println(“A says to discuss");
   }
}
class B extends A
{
   public B()
   {
     System.out.println(“B constructor");
   }

   void Talk()
   {
     System.out.println(“B says to discuss");
   }
}

public class Test
{
    public static void main(String args[])
    {
        B b1=new B();
        b1.Talk();
        b1.Greet();
    }
}
```

v.

```
// Refer class A and class B from above question iv.
public class Test
{
    public static void main(String args[])
    {
        A a1=new B();
        a1.Talk();
        a1.Greet();
```

```
    }
}
```

Q3. What is mean by Dynamic Polymorphism? Explain with suitable example.

Q4. Why abstract method is used?

Q5. Define the following terms.

i. Dynamic Polymorphism
ii. Abstract class
iii. Abstract method
iv. Object slicing
v. Upcasting
vi. Downcasting
vii. Concrete class

Answers:

Q.1.

i. True	ii. True	iii. False	iv. False
v. False	vi. True	vii. True	viii. True
ix. True	x. False	xi. False	

Q.2

i. Compilation error as display method is declared private in class A.
ii. B. Refer deep knowledge section Q.2.
iii. A. Address of object of class A is stored in a reference of class A. Hence, it calls method defined in base class A.
iv. A Constructor
A says to discuss
A says hi
v. A Constructor
B constructor
B says to discuss
A says hi

"JAVA is ubiquitous in the technology world and finds its usage and application in multiple domains. Madhusudan has harnessed his deep practical experience to enable new learners pick up JAVA easily through his new book. Want to come to the terms with JAVA quickly enabled by a practitioner perspective? Pick up this book!

Gururaj Deshpande,
Senior Vice President and Head of Banglore Developement Centre, Infosys.

It is not any exaggeration to say that Java has become a quintessential part of a programmer's work life. However, for a serious practitioner it is not enough just to be acquainted with this very popular programming language, but to develop proficiency in the same. In many of my interactions with aspiring software engineers and even seasoned programmers and architects, I have often felt that a well-structured guide can help hone the skills to the next level. Madhusudan's well-crafted book precisely helps in this journey of a programmer from being a novice to becoming an adept. The 'Points to Remember' and FAQ/Questions in each chapter makes the subject very easy to revise and remember. This book can serve as a text book to students as well as a hand book to practitioners. The programmer community can do well to pick a copy of this book!

Ashok Kumar Ratnagiri,
Director & Head, Systems Engineering, Infosys

About the Author

Mr. Madhusudan Mothe is Bachelor of Engineering from Government College of Engineering, Pune (COEP) and M. Tech. from Indian Institute of Technology (IIT), Bombay. His all India GATE rank was 46. His books on C & C++ are part of the syllabus of various universities. In all his books, he explains complicated concepts in easily understandable language. He is currently working as a Senior Technical Manager in Edgeverve, a subsidiary of Infosys. He has had the distinction of being interviewed by the Pune campus of Infosys as "The Rising Star of Pune Development Centre".

His webpage is http://bit.ly/mothemadhusudan