OBJECT ORIENTED PROGRAMMING USING JAVA
UNIT 1
INTRODUCTION TO JAVA

CHAPTER 1: INTERNET

INTERNET:

The internet is a global system of interconnected computer networks that use the standard Internet Protocol Suit (TCP / IP) to serve billions of users worldwide.

PROTOCOL:

When a computer communicates with each other there need to be common set of rules and instructions that each computer follows. A specific set of communication rules is called a protocol.
Example: HTTP, SMTP, TCP/IP, FTP, Internet uses TCP/IP, WWW uses HTTP, Email uses SMTP

HISTORY OF INTERNET/ PURPOSE OF INTERNET / EVALUTION OF INTERNET:

- In 1957 USSR launches Sputniks Satellite.The United States believed it to be a great threat to its security.
- So in 1958 February the United States created Advance Research Project Agency (ARPA or DARPA) to regain a technical lead and strengthen its military and academic initiative to promote research in the area of mathematics, pure science and engineering.
- ARPA created the information processing technology office (IPTO) to further the research of Semi-Automatic Ground Environment (SAGE) program, which had networked country – wide radar system together for the first time.
- In 1969, the U S Defense Department funded a project to develop a network, which can withstand the bombing. The idea was to develop a much secured network which can work even after a nuclear attack. This project was known as ARPANET.
- The proposed network was not supposed to have a central control which would be an obvious target.
- After ten year of research brought Local Area Network (LAN) and workstations were developed to get connected to LAN. These workstation and LAN then connected to ARPANET.
- Computer connected to ARPANET used a standard or rule to communicate with each other. This standard used by ARPANET is known as NCP(National Control Protocol)
- But the rapid change in the information technology suppressed NCP and brought TCP / IP (Transmission Control Protocol / Internet Protocol) in the world of networking.
- TCP converts messages into streams of packets at the source, and they reassembled back into messages at the destination.
- Internet protocol handles the dispatch of these packets. IP handles the addressing and make sure that a packet reaches its destination through multiple nodes even across multiple network with multiple standards.
- Slowly the ARPANET became a massive network of networks and now it is known as INTERNT.
- Nobody controls the INTERNET and in principle, any computer can speak to any other computer as long as it obeys the technical rules of the TCP / IP protocol.
- This freedom of internet helped it to move out of its original base in military and research institutions, into elementary and high school, college, public libraries, commercial sectors, shops.
- In 1971 there were 15 nodes in ARPANET. In 1972 there were 37 nodes
- In October 1972 the first demonstration of the ARPANET was made public at the international computer communication conference.
- The initial “hot” application, electronic mail was introduced.

- Basic email messages were written, sent and read, filed, forwarded and answered. From there email took off as the largest application.
- In 1990's it became possible for people to post information via Internet sites.
- Using the Gopher system people could look at a list of links. Each link led either to another set of links or to a document.
- In 1994 there was a development of World Wide Web(WWW) with this user can jump from any point in one document to any point in another document , known as hypertext , in WWW tsupports the use of pictures , movies, videos, sound files etc.

DIFFERENCE BETWEEN INTERNET AND WEB:

	INTERNET	WWW
1	Internet is a massive network of networks, a networking infrastructure.	WWW is way of accessing information over the medium of the internet.
2	It connects millions of computers together forming a network in which any computer can communicate to internet.	It is an information sharing model that is built on the top of internet.
3	Information that travels over the internet does so via variety of language know as protocol.	The web uses HTTP protocol, to transmit data.
4	It does not uses web browser. It is the structure on which WWW is based.	It is also utilizes browsers such as internet explorer of firefox to access web documents called web pages that are linked via hyperlinked.
5	No creator	It was created by TIM BERNERS LEE in 1992
6	Internet provides the structure	It provides the dynamic network via a variety of different methodologies and protocol.

INTERNET ARCHITECTURE:

Internet infrastructure is endless miles of telephone lines and fiber optic cables. These cables connect millions of individual users and business to other parties, transmitting data at varying speed, depending on the types of cabling used; it includes telephone modems, high speed connection methods like cable modems ISDN, DSL, T1 Lines and company network.

There are ISPs (internet service provider) which maintains racks of modems; users connect to these modems in order to gain access to the ISP's network, which can vary in reach depending upon the ISp's size. Once connected to ISP user than attempt to communicate by sending emails message to other internet users or by requesting Web pages or downloadable files from any number of servers located across world.

In order to send and receive information a user on One ISp's network must be able to connect to users on another ISP's network, which may be located across the nation or across the globe. If two users were located on the same ISP's network ISp's connect to one another at NAP's , also called as Internet exchanges(IX).

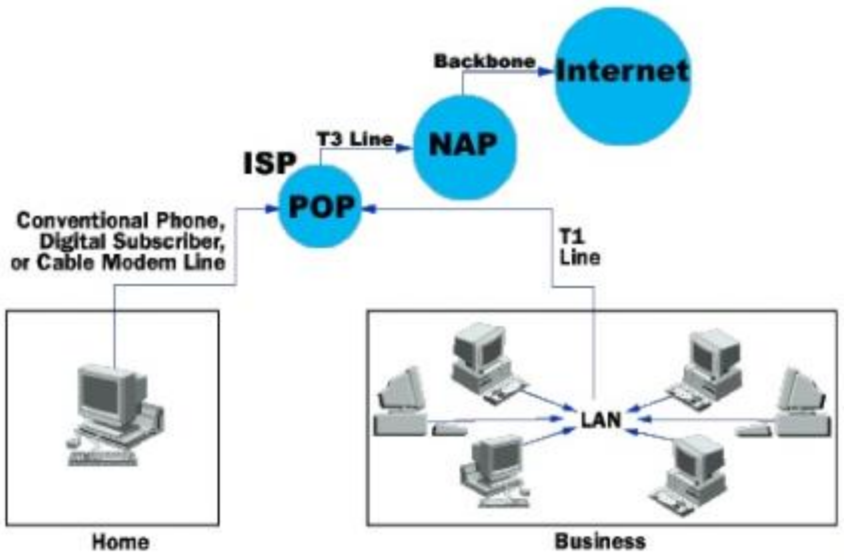
Devices knows as routers make sure that the packets of data sent from a computer on one ISP's network are sent to the intended machine on another local or wide area network via quickest most efficient route. Routers need to know the address of the device to which information is being sent via the internet.

All devices communicating on the internet , including servers to host web sites have unique Internet Protocol address, which are four sets of numbers separated by decimal corresponding to numeric IP addresses are domain names, which are easier for humans to remembers than long sequences of number. In the web site address www.yahoo.com , .Com is called the top level domain and the word yahoo is called as second level domain. As the internet is evolved a distributed database called Domain Name System was created which contains all the domain names and IP addresses with registered entities. Domain name Server located across the internet are responsible for finding registered domain names and converting them to IP addresses so a connection can occur.

<u>IMPORTANT ELEMENTS OF INTERNET ARCHITECTURES:</u>

- Modem
- Backbone
- ISP
- TCP/IP
- Routers
- Internet
- Network access points or NAPs
- Domain Name Service (DNS) and DNS Servers.

HOW INTERNET WORKS:



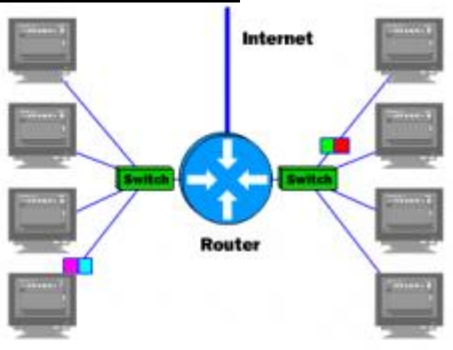
Every computer that is connected to the internet is part of a network. We may use modem and dial a local number to connect to an Internet Service Provider. When you connect to ISP you become the part of their network. The ISP may then connect to a larger network and become part of their network. The internet is simply a network of networks.

Most large communication companies have their own dedicated backbones connecting various regions. In each region the company has a Point of Presence (POP). The POP is a place for local users to access the company's network often through a local phone number or dedicated line. There are several high level networks connecting to each other through Network Access Point or Naps

POINT OF PRESENCE:

POP is access point from one place to the rest of the internet. A POP necessarily has a unique Internet Protocol address. The internet service provider has a point of presence on the internet and probably more than one.

THE FUNCTION OF INTERNET ROUTERS:



All the network relays on NAPs, backbones and routers to talk to each other. The router determines where to send information from one computer to another. Routers are specialized computers that send our message.

Routers join the two networks, passing information from one to the other. It also protects the networks from one another preventing the traffic on one from necessarily spilling over the other.

USE OF ROUTER:
The router determines where to send information from one computer to another. A router has two jobs,
- It ensures that information does not go where it's not needed.
- It makes sure that information does not make it to the intended destination.

INTERNET BACKBONES:
Def: Internet backbone is the physical network usually relying on fiber optic cables that carries internet traffic between different networks and is measured in megabits per second.

IP ADDRESS:

Def: Every machine on the internet has a unique identifying number, called an IP Address. The IP stands for Internet Protocol, which is a language, that computer use to communicate over the Internet.
A typical IP address looks like this: 256.27.119.89.

The four numbers in an IP address are called as Octets. Each Octet can contain any value between 0 to 255. Combine the 4 octet we get 4,294,967,296 unique values. The IP address 0.0.0.0 is reserved for the default network and 255.255.255.255 is used for broadcasts.

DOMAIN NAME SYSTEM:

Def: The DNS is a distributed database which keeps track of computers names and their corresponding IP addresses on the internet. It is like software and this software will be running in DNS Server.
No DNS server contains the entire database; they only contain a subset of it. If a DNS server does not contain the domain name requested by another computer, the DNS servers re directs the requesting computer to another DNS server.

URL: UNIFORM RESOURCE LOCATOR
When we use the Web or Send an email message, we use a domain name to do it.

INTERNET PROTOCOL:
IP is the primary network protocol used on the internet, developed in the 1970s. On the internet and many other networks, IP is often used together with TCP/IP.

CHAPTER – 1:

2 Marks:

1. **Differentiate between Internet and WWW.**
2. **What are the basic elements of Internet Architecture?**
3. **Define Internet?**
4. **What is DNS?(2010)**
5. **What is Unicode(2011)**
6. What is WWW?
7. What is protocol? Which protocol is used by Internet, WWW and Email
8. What is URL? Give an example?
9. Write the importance of Internet Protocol?

5Marks:

1. Write a brief history of World Wide Web.

2. Explain the history of an internet?
3. How internet is evolved?
4. Explain the architecture of Internet?
5. How internet works?
6. Explain the role of ISP , DNS and Backbone
7. What is the use of routers?

CHAPTER 2

WHAT IS JAVA?

Java is a simple, object oriented, distributed, compiled and interpreted, robust, secure, architecture neutral, multithreaded, and dynamic programming language.

HISTROY OF JAVA:

1990→ Sun Microsystems decided to develop special software that could be used to manipulate consumer electronic devices. A team of Sun Microsystems programmers headed by James Gosling was formed to undertake this task.
1991→ After exploring the possibility of most Object Oriented Programming Language C++, the team announced a new language named “Oak”.
1992→ The team, known as a Green Project team by Sun, demonstrated the application of their new language to control a list of home appliances using a hand-held device with a tiny touch sensitive screen.
1993→ The World Wide Web (WWW) appeared on the internet and transformed the text-based Internet into a Graphical-rich environment. The green Project team came up with the idea of developing Web Applets (tiny programs) using the new language that could run on all types of computers connected to Internet.
1994→ The team developed a web browser called “Hot Java” to locate and run applet programs on Internet. Hot Java demonstrated the power of the new language, thus making it instantly popular among the Internet users.

1995→ Oak was named “Java”, due to some legal snags. Java is just a name and is not an acronym. Many popular companies including Netscape and Microsoft announce to their support to Java.
1996→ Java established itself not only a leader for Internet Programming but also asa general-purpose, object oriented programming language. Java found its home
1997→Sun release Java Development Kit 1.1(JDK 1.1)
1998→Sun releases the Java2 with version 1.2 of the software Development kit(SDK 1.2)
1999→Sun releases Java 2 Platform, Standard Edition(J2SE) and Enterprises Edition(J2EE).
2000→J2SE with SDK 1.3 was released.
2002→J2SE with SDK 1.4 was released.
2004→J2SE with JDK 5.0 was released. This is known as J2SE 5.0.

FEATURES / CHARACTERISTICS OF JAVA PROGRAMMING:

- Java is **Simple, small and familiar.**
- Java is **Compiled and Interpreted.**
- Java is **Platform-Independent**
- Java is **Portable**
- Java is **Object-Oriented**
- Java is **Robust**
- Java is **Secure**
- Java is **Distributed**
- Java is **Multithreaded and Interactive**
- Java is **High Performance**
- Java is **Dynamic and Extensible.**

JAVA IS SIMPLE:

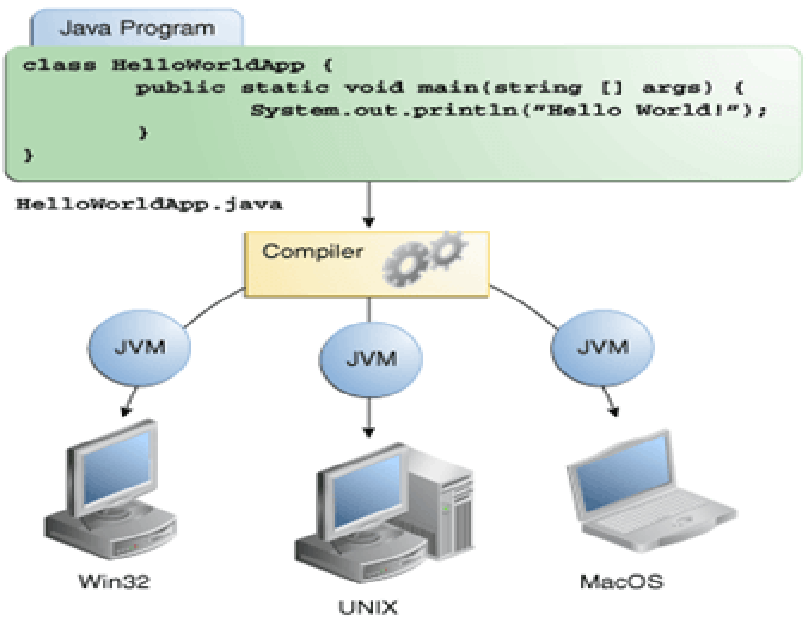
Easy to learn. Because java inherits the C/C++ syntax and many of the objects oriented features of C++.
Java uses automatic memory allocation and garbage collection.
Java does not use operator overloading, multiple inheritance, pointers, header files and goto statements.

JAVA IS SECURE:

Java provides a “firewall” between a networked application and user’s computer. Java system verifies all memory access and also ensures that no viruses are communicated with an applet.
The absence of pointer in java ensures that program cannot gain access to memory locations without proper authorization.

JAVA IS PORTABLE and JAVA IS PLATFORM-INDEPENDENT :

Platform independent means "write once run anywhere”. Java is called so because with the java virtual machine you can write one program that will run on any platform.
Java is potable language which can be moved from one computer to another anywhere and anytime.
Java programs can be run on any platform without being recompiled.
Translating a java program into byte code makes it much easier to run a program in a wide variety of environments.JVM differs from platform to platform, all interpret the same java byte code.



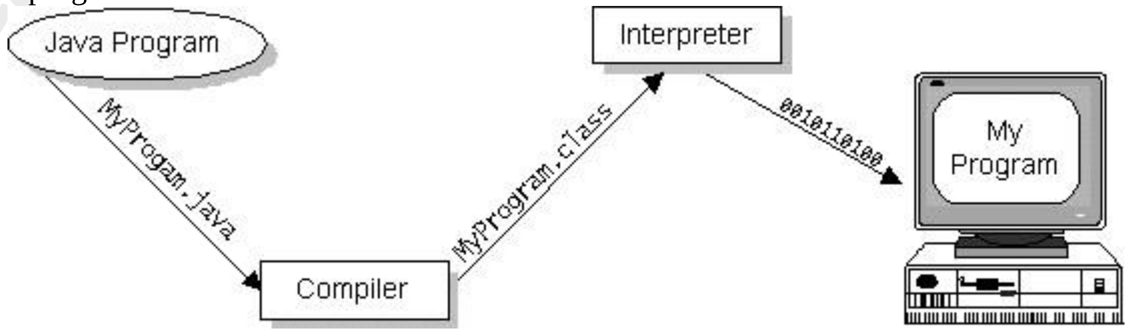
```
class HelloWorldApp {
        public static void main(string [] args) {
                System.out.println("Hello World!");
        }
}
```

JAVA IS OBJECT ORIENTED LANGUAGE.
Java is true object oriented language. Everything in java is an object. All programs code and data resides within object and classes. These classes are arranged in packages which can be used in our program by inheritance. The object model in Java is simple and easy to extend, while simple types such as are kept as high performance non objects.

JAVA IS ROBUST:
Java is robust (robust means strong).Java has strict compile time and run time checking for data types.
Java has strong memory allocation and automatic garbage collection mechanism. Java also includes concept of exception handling which captures series of errors and eliminates the risk of crashing the system.

JAVA IS A MULTITHREADED:
Multithreaded means handling multiple tasks simultaneously. Java supports multithreaded programs this means that there is no need to wait for one application to complete before beginning the other. Multithreaded programming which allows you to write programs that do many things simultaneously.
E.g:-we can listen to audio clips and at the same time scroll pages and also download applet from distant computer. This improves interactivity of graphics application.

JAVA IS COMPILED AND INTERPRETED.
A computer language is compiled and interpreted. Java combines both the approaches, these making it two stage system.
First-Java compiler translates the source code into byte code instruction which is a not a machine language.
Second stage-java interpreter generates machine code that can be directly executed by machine that is running java program.



Thus Java is compiled and interpreted.

JAVA IS ARCHITECTURE NEUTRAL:
Java programs can be executed on any processors like Pentium, Celeron, dual core, core 2 duo, AMD and so on. So java is architectural natural.
It is essential that the application must be able to migrate easily to different computer system and a wide variety of hardware architecture and operating system architecture. The java compiler does this by generating byte code instructions to be easily interpreted on any machine and to be easily translated into native machine code. The compiler generates an architecture neutral objects file format to enable a java application to execute anywhere on the network and then the compiled code is executed on many processors given the presence of run time system.

JAVA IS DISTRIBUTED LANGUAGE:
Java is designed as distributed language for creating applications on networks. Java supports TCP / IP protocol. It has the ability to share both data and program. Java application can easily access remote objects on internet as easily as they can do in a local system.

JAVA IS HIGH PERFORMANCE:
Java performance is impressive for an interpreted language mainly due to use of intermediate byte code.
Java is designed to reduce the overheads during runtimes.
Due to incorporation of multithreading the execution speed of java program is enhanced.

JAVA IS DYNAMIC AND EXTENSIBLE:
Java is capable of dynamically linking in new class libraries, methods and objects. Java programs support function written in other language such as C and C++. These functions are known as native methods. Native methods are linked dynamically at runtime.

MULTIMEDIA SUPPORT:
Most programming languages do not have built in multimedia capabilities. Java through the packages of classes that are an integral part of the java programming world provides extensive multimedia facilities that will enable a programmer to start developing powerful multimedia application.

DIFEERENCE BETWEEN JAVA AND C++:

	C++	JAVA
1.	C++ is not pure object oriented programming language as we can write C++ programs without using a class or not object.	Java is pure object oriented programming language everything in java is part of a class only.
2.	Pointers in C++	No pointer in java
3.	Allocating and deallocating memory is the responsibility of the programmer	JVM takes care of allocating and deallocating memory.
4.	C++ has goto statements	No goto statements in java
5.	Multiple inheritance feature is available in C++	No multiple inheritances in java, but there is ways to achieve it.
6.	Automatic casting available in C++	Implicit casting available in java. But it is good to use casting whenever required.
7.	Operator overloading available in C++	No operator overloading in java
8.	# define, typedef and header files are available in C++	# define, typedef and header files are not available in Java
9.	There are 3 accesss specifier in C++: private, protected, public	Java supports 4 access specifiers: private , protected , default and public.
10	There are constructor and destructor in C++	Only constructors are available in Java. No destructors in Java
11	Supports Global variable	No global variable in Java. Everything is part of

		Java
12	Supports Structure and Union	No Unsafe Structure and Unions in java
13	Supports template classes	Java does not support templates.

DIFFERENCE BETWEEN C AND JAVA:

	C	JAVA
1.	C is structured and procedure oriented programming language.	Java is pure object oriented programming language everything in Java is part of a class only
2.	Pointers in C	No pointers in Java
3.	Allocating and deallocating memory is the responsibility of the programmer	JVM takes care of allocating and deallocating memory.
4.	C has goto statements	No goto statements in java
5.	Automatic casting available in C	Implicit casting available in java. But it is good to use casting whenever required.
6	# define, typedef and header files are available in C	# define, typedef and header files are not available in Java
7	Supports Global variable	No global variable in Java. Everything is part of Java
8	C supports auto,extern,register , signed and unsigned modifiers	Java does not supports auto,extern,register , signed and unsigned modifiers
9	C does support labeled break and continue	Java does not support labeled break and continue
10	C supports sizeof operator	Java does not support sizeof operator
11	Supports structure and union	No unsafe structure and unions in java

BYTECODE:
Java byte codes are a special set of machine instructions that are not specific to any processor or computer system. A platform form specific byte code interpreter executes the java byte codes. The byte code interpreter is also called the java virtual machine or the java runtime interpreter.
javac translate java source code into an intermediate code called as BYTECODE.

JAVA AND INTERNET:
Java is a general purpose language that had many features to support it as the internal language.

- **Applets:**
  Java's ability to create applets makes java as important.
- **Cross Platform compatibility:**
  The byte code once generated can execute on any machine having JVM. The byte code of a java program can be distributed to any machine and any platform through internet.
- **Supports of internet protocol:**
  Java has rich variety of classes that abstract the internet protocols like HTTP, FTP, IP, TCP/IP, SMTP etc.
- **Supports to HTML:**
  Most of the Programming languages that are used for web application use the HTML pages as a view /GUI to interact with the user. Java programming language provides its support to html.
- **Support to Web Services:** Java has a rich variety of APIs to use xml technology in diverse applications that supports N-Tiered enterprise application over the internet
- **Supports to java enabled Mobile device:**
  Java programming language is made is such a way so that it is compatible with mobile device also.
- **Supports to personal digital assistants: Java language is compatible with PDA's.**
- **Supports to XML parsing:**

Java has built in APIs to read the xml data and create the xml document using different xml parsers like DOM and SAX. These APIs provides mechanism to share data among different application over the internet.

JAVA AND WWW:

How Java is associated with WWW or Web:

Java is intimately associated with the internet and the World Wide Web. Before java, WWW was limited to display only static content like images and text. The java program "applet" are meant to be transmitted over the internet and displayed on web pages. A Web Server transmits a Java applet just as it would transmit any other type of information. A web Browser that understands Java includes an interpreter for the java virtual machine can run the applet right on the web page. Since applets are programs they can do almost anything including complex interaction with the user. With Java a web page becomes more than just passive display information.

JAVA COMMMUNICATION WITH WEB:

Step 1: User selects the HTML pages from the web browser like internet explorer .The HTML pages is residing in the remote computer's web server.
Step 2: The client request goes to web browser and web server process the request .the response will be sent back to user.
Step 3: The response HTML documents contains the <APPLET> tag. The corresponding applet byte code also returned to the user.
Step 4: The java enabled web browser on the user computer interprets the byte codes and displays the output.

USE OF WEB SERVER:

A web server responds to requests by sending Hypertext markup Language(HTML) documents back to the browser which is then displayed for the user.

WEB BROWSERS:

Web browsers are basically software programs to search for and view various kinds of information on the Web.

- Microsoft's internet explorer.
- Mozilla's Firefox.
- Google chrome.
- Mac safari.
- Opera

JAVA ENVIRONMENT

The Java environment contains both development tools and class libraries.

- The development tools are part of **JAVA DEVELOPMENT KIT (JDK).**
- Class libraries are part of **JAVA APPLICATION PROGRAMMING INTERFACE (API).**

JAVA DEVELOPMENT KIT:

A Java development Kit is a program development environment for writing java applets and application.The JDK include:

- The Java Runtime Environment (JRE) that "sits on top" of the operating system layer.
- Command Line Development tools, such as compilers and debuggers.

Standard JDK Tools and Utilities:

- **BASIC TOOLS:**

Tool	Description
javac	The compiler for java programming language. Using this tool we can compile java programs and generates its byte code.
java	The launcher for java application .Using this tool we can execute the java byte code.
Javadoc	API documentation generator. Using this tool we can generate documentation for our java program.
appletviewer	Using this tool we can run and debug applets without a web browser.
jar	Create and manage java Archive (jar) files.
jbd	The java debugger. Using this tool we can debug our java programs to find out any errors.
javah	C header and stub generator. Used to write native method.
javap	Class file disassemble. Using this tool we can convert byte codes to a program description.

- Security Tools(Keytool, jarsigner, policytool , kinit , klist , ktab)
- Internationalization Tools(native2ascii)
- Remote method invocation(RMI) Tools(rmic, rmiregistry,rmid,serialver)
- Java IDL and RMI-IIOP Tools(tnameserver,idlj,ordb,servertool)
- Java Deployment Tool(pack200,unpack200)
- Java plug-in Tools(htmlconverter)
- Java Web Start Tools(javaws)

<u>JAVA APPLICATION PROGRAMMING INTERFACE (JAVA API):</u>

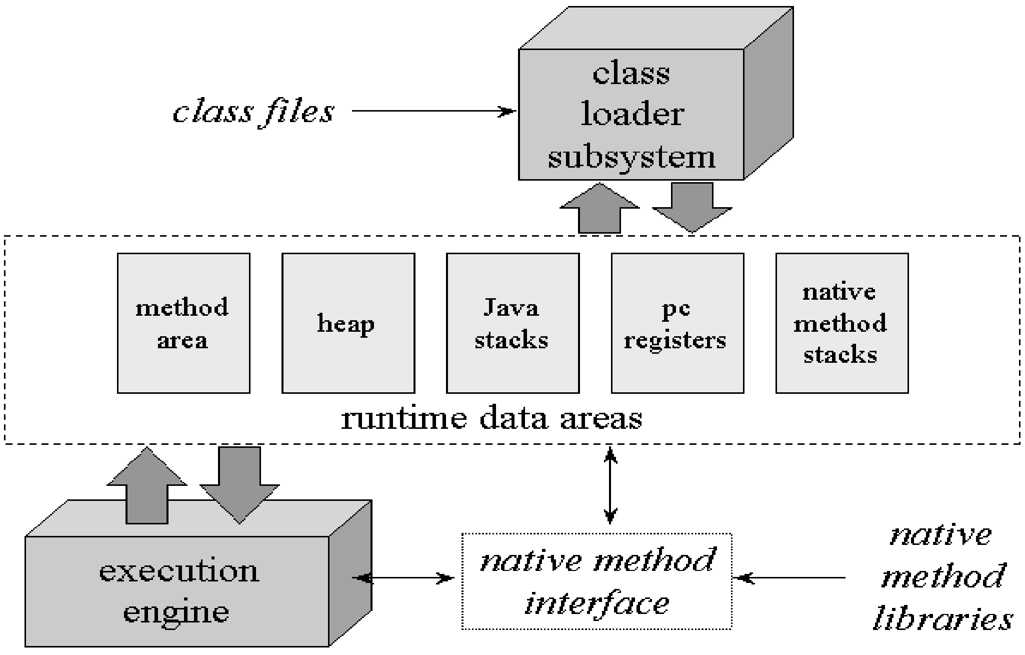
The Application programming interface are pre written java code that can be used by other programmers to create java application.
The java API is the set of huge number of classes and methods grouped into packages and it is included with the Java Development Environment.

Purpose	Package Name	Description
Language support package	java. Lang	These classes support the basic language features and the handling of arrays and strings. Classes in this package are always available directly in our programs by default
Input/output package	java.io	Classes for a data input and output operation
Utilities package	java.util	A collection of classes to provide utility functions such as date and time functions.
Networking package	java.net	A collection of classes for communicating with other computer via internet.
AWT	java.awt	The Abstract Window Tool kit package contains classes that implements platform independent graphical user interface.
Applet	java.applet	This includes a set of classes that allows us to create java applets.

<u>JAVA VIRTUAL MACHINE:</u>

- All languages compilers translate source code into machine code for a specific computer java compiler.
- Java compiler produces an intermediate code known as byte code for a machine that does not exist.
- This machine is called JAVA VIRTUAL MACHINE.
- JVM is a heart of Java Program execution process.

- It is responsible for loading .class file and converting each byte code instruction into the machine language instruction.



Class Loader Subsystem:

The class loader subsystem performs the following tasks:

- Loads .class file into memory(RAM)
- It verifies whether all bytes code instructions are proper or not. If it finds any instruction suspicious then execution is terminated.
- If the byte code instructions are proper then it allocates necessary memory to execute the program.
- It allocates memory to static variables and sets the default value to static variables.

Run Time Data Areas:

The memory is divided into 5 parts, called run time data areas, which contains the data and the result. While running the program:

- **Method Area:** It is logical memory components of JVM. This memory holds information about classes and interface. Memory for static variables and method codes are allocated in this area.
- **Heap:** This is the area where objects are created. The JVM allocates memory in heap for an object which is created using 'new' operator.
- **Java Stacks:** Methods codes are stored in methods area. For execution, a method need memory because of the local variables and the arguments it has taken. So java stacks are memory areas where Java methods are executed.
- **Program counter Registers:** it keeps track of the sequence of the execution of the program. PC registers hold the addresses of the instructions to be executed next. If there are three methods three PC resisters are used to track the instruction of the methods.
- **Native method stacks:** native methods which are written in C/C++ are executed inside the native methods stack. The libraries required for the execution of native methods are available to the JVM through **Java Native Interface.**

Execution Engine:

Execution engine contains both interpreter and Just In Time(JIT) compiler, which are responsible for converting the byte code instruction into machine code so that the processor will execute them.

JDK, JVM, JRE, JIT:

JAVA DEVELOPMENT KIT (JDK)

- Java Development Kit is the core component of Java Environment and provides all the tools, executable and binaries required to compile, debug and execute a Java Program.
- JDK is platform specific software and that's why we have separate installers for Windows, Mac and UNIX systems.
- We can say that JDK is superset of JRE since it contains JRE with Java compiler, debugger and core classes. Current version of JDK is 1.7 also known as Java 7.
- JDK=Java Development Tools + JRE

JAVA VIRTUAL MACHINE (JVM)

- JVM is the heart of java programming language.
- When we run a program, JVM is responsible to converting Byte code to the machine specific code.
- JVM is also platform dependent and provides core java functions like memory management, garbage collection, security etc.
- JVM is customizable and we can use java options to customize it, for example allocating minimum and maximum memory to JVM.
- JVM is called *virtual* because it provides an interface that does not depend on the underlying operating system and machine hardware. This independence from hardware and operating system is what makes java program write-once run-anywhere.

JAVA RUNTIME ENVIRONMENT (JRE)

- JRE is the implementation of JVM, it provides platform to execute java programs.
- JRE consists of JVM and java binaries and other classes to execute any program successfully.
- JRE doesn't contain any development tools like java compiler, debugger etc.
- If you want to execute any java program, you should have JRE installed but we don't need JDK for running any java program.
- JRE=JVM + Java Packages(like io,lang,awt,swing etc) + runtime libraries

JUST-IN-TIME COMPILER (JIT)

- JIT is part of JVM that optimizes byte code to machine specific language compilation by compiling similar byte codes at same time, hence reducing overall time taken for compilation of byte code to machine specific language.

JDK	JRE
It is bundle of software that you can use to develop Java Based Application	It is an implementation of the java virtual machine which is actually executed java program.
Java development kit is needed for developing java application	Java run time environment in a plug in needed for runtime java programs
JDK needs more disk space as it contains JRE along with various development tools	JRE is smaller than JDK so it needs less disk space
JDK can be downloaded/supported freely from java.sun.com	JRE can be downloaded / supported freely from java.com
It includes JRE , set of API classes, Java compiler, webstart and additional files needed to write java applets and application	It includes JVM. Core libraries and other additional components to run application and applets in java.

PARTS OF JAVA:

Java is divided into 3 parts – Java SE , Java EE and Java ME

Java SE: It is the java standard edition that contains basic core java classes. This edition is used to develop standard applets and application.
Ex: payroll application

Java EE: It is the java enterprises edition and it contains classes that are beyond Java SE. Java EE is mainly used to develop enterprises business solution on a network. We need Java SE in order to use many of the classes in Java EE.
Ex: Developing Internet banking Application

Java ME: It is the java macro edition. Java Me is for developers who develop code for portable device such as PDA or mobile phone.
Ex: Developing game for mobile application.

JAVA SUPPORTS SYSTEM:

The systems that are required to support Java for developing application on the internet are

- Java
- Java enabled web browsers.
- Web server.
- HTML
- Java code
- Byte code
- Applet tag and
- Internet connection.

ADVANTAGES OF JAVA:

Simple: Java was designed to be easy to use, write, compile, debug, and learn than other programming languages. Java is much simpler than C++ because Java uses automatic memory allocation and garbage collection.

Object-Oriented: Allows you to create modular programs and reusable code.

Platform-Independent: Ability to move easily from one computer system to another

Distributed: Designed to make distributed computing easy with the networking capability that is inherently integrated into it.

Secure: The Java language, compiler, interpreter, and runtime environment were each developed with security in mind.

Allocation: Java has the feature of Stack allocation system. It helps the data to be stored and can be restored easily.

Multithreaded: The capability for a program to perform several tasks simultaneously within a program.

DISADVANTAGES OF JAVA:

Performance: Significantly slower and more memory-consuming than natively compiled languages such as C or C++.

Look and feel: The default look and feel of GUI applications written in Java using the Swing toolkit is very different from native applications.

Single-paradigm language: The addition of static imports in Java 5.0 the procedural paradigm is better accommodated than in earlier versions of Java.

ADVANTAGES OF OBJECT ORIENTED PROGRAMMING LANGUAGE:

The object oriented programming language directly represents the real life objects like Car, Dog, Account and Customer etc. The features of the OOPL like polymorphism. Inheritance and encapsulation make it powerful. Polymorphism, inheritance and encapsulation are the pillars of OOPL.

ABSTRACTION: Abstraction refers to the act of representing essential features without including the background details of explanation.

ENCAPSULATION: Encapsulation is a technique used for hiding the properties and behavior of an object and allowing outside access only as appropriate. It prevents the objects from directly altering or accessing the properties or methods of the encapsulated objects.

INHERITANCE: inheritance is the process by which objects of one class acquire the properties of objects of another class. The class that does the inheriting is called a subclass. Inheritance is done by using the keyword extends. The two most common reason to use inheritance is: to promote code reuse and to use polymorphism.

POLYMORPHISM: Polymorphism is briefly described as “one interface many implementations”. Polymorphism is a characteristic of being able to assign a different meaning or usage to something in different contexts- specifically, to allow an entity such as variable, a function or an object to have more than one form.

CHAPTER – 2

2 MARKS:

1. What is the difference between JDK, JRE and JVM?
2. What is the difference between Java and C++?(2010)(2011)
3. What is platform independent and Architecture Neutral?(2012)
4. When was java developed and what was it initial names?
5. Why java is called portable language?
6. What is Byte code in java?(2011)
7. How java is secured language?
8. How can we say java is platform independent?(2010)
9. How java byte code different from other low level computer languages coding.(2012)
10. Why java is simple. Mention any two reason (2016).
11. What is java API?(2012)(2016)
12. What is java?
13. How can we say java is robust and secure?

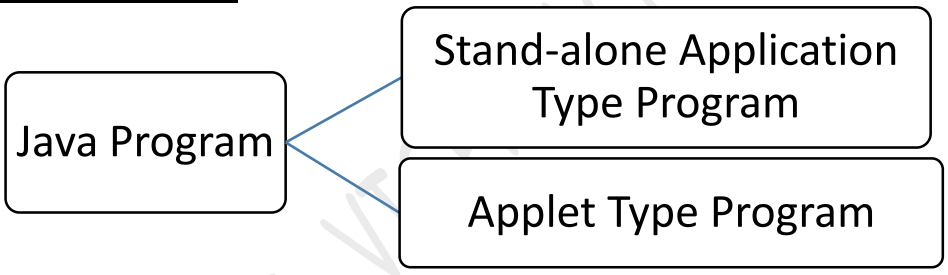
5 MARKS:

1. Difference between Java and C++
2. Difference between Java and C
3. Explain web browser
4. How java strongly associated with internet(2011)
5. What is Java API?(2010)

6. What is JVM? Explain the various parts of JVM?(2012)
7. Write a note on W3C
8. Explain command line arguments
9. Explain the features of java(2015)
10. Explain the history of java.(2016)
11. Explain any four object oriented concepts.
12. What are the advantages of OOPL?
13. Java is platform independent? Explain with a neat diagram?
14. Java is architectural neutral justify?
15. How java and WWW are associated
16. How Java communicates with WWW.
17. What is web browser? Explain the purpose of it.
18. What is JDK? List the important tools of JDK?
19. What is the difference between heap and stack memory?
20. What are the different parts of java?
21. Explain the advantages and disadvantages of java

CHAPTER 3:

TYPES OF JAVA PROGRAM:



- Stand alone application type programs are written to carry out specific tasks on a standalone local computer.
- Applet type programs are used in internet applications. The java applet will be placed on the web by embedding into a HTML file. Any internet user can click the applet and run the applet on web browser.
- Stand alone application programs are compiled using javac.exe and executed using java.exe interpreter.
- Applet programs are compiled using javac.exe and executed using java enabled web browser.

PHASES OF JAVA PROGRAM DEVELOPMENT

Phase 1: Creating a program

We create the program on editor, after that it stored in the disk with the name's ending .java

Phase 2: Compile the java program into the byte code

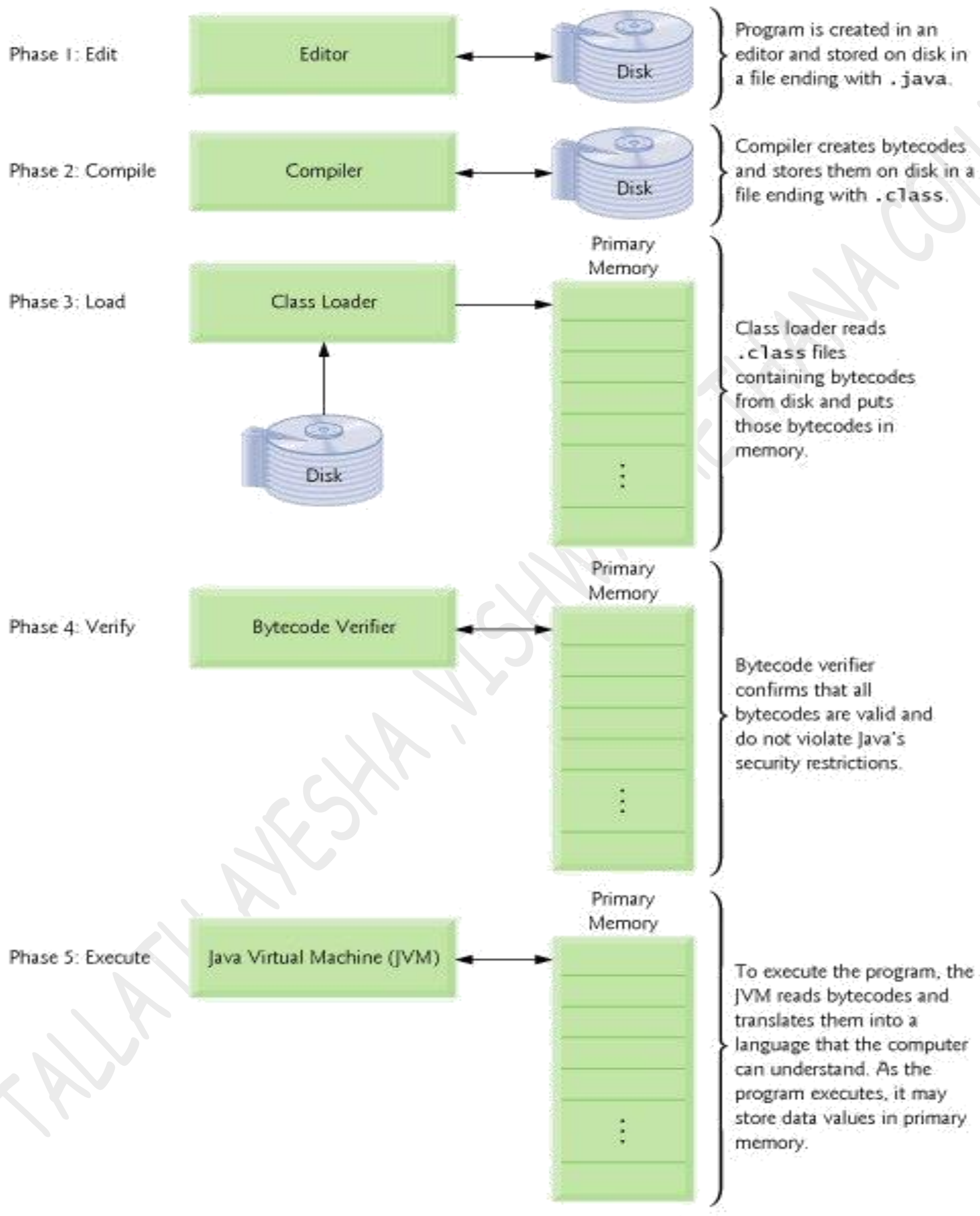
Compiler translate from high-level language program to byte codes and store it in disk with the ending name .class. For example, to compile a program called MyProgram.java we would type

```
javac MyProgram.java
```

Phase 3: Loading a Program into memory

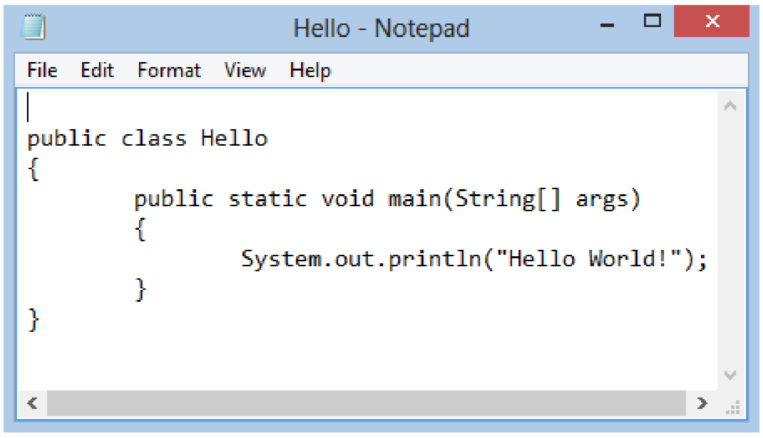
the program must be placed in memory before it can execute a process known as loading. The class loader takes the .class files containing the program's byte code from hard drive and transfers them to primary memory RAM.

Phase 4: **Byte code Verification**
the classes are loaded the byte code verifier examines their byte codes to ensure that they are valid and do not violate java security restriction.
Phase 5: **Execute**
Java Virtual Machine (JVM) read and translates those byte codes to language that computer can understand (Machine Language). Then execute the program, store it values in primary memory.


Phase 1: Edit
Editor
Disk
Program is created in an editor and stored on disk in a file ending with .java.
Phase 2: Compile
Compiler
Disk
Compiler creates bytecodes and stores them on disk in a file ending with .class.
Phase 3: Load
Class Loader
Disk
Primary Memory
Class loader reads .class files containing bytecodes from disk and puts those bytecodes in memory.
Phase 4: Verify
Bytecode Verifier
Primary Memory
Bytecode verifier confirms that all bytecodes are valid and do not violate Java's security restrictions.
Phase 5: Execute
Java Virtual Machine (JVM)
Primary Memory
To execute the program, the JVM reads bytecodes and translates them into a language that the computer can understand. As the program executes, it may store data values in primary memory.

First java program:

Step 1: Open a notepad and type the below program



```
public class Hello
{
        public static void main(String[] args)
        {
                System.out.println("Hello World!");
        }
}
```

Step 2: The name of the class is Hello and hence the file name should be Hello.java
Step 3: Save the file a Hello.java in some working directory of your hard disk.
Step 4: Compile the program using the below command. Javac Hello.java
Step 5: The byte code Hello.class is generated.
Step 6: Execute the program using java Hello.

UNDERSTANDING THE PROGRAM:

Class declaration: The first line "class Hello" declares a class, **class** is keyword and declares that a new class definition. **Hello** is a java identifier that specifies the name of the class to be defined.

Opening braces: every class definition in java begins with an opening brace "{" and ends with a matching closing brace "}" .

The Main line: the third line **public static void main(String args[ ])** defines a method named main. Every java application program must include the main( ) method. A java application can have any number of classes but only one of them must include a main method to initiate the execution. This line contains a number of keywords, public, static and void.
public : the keyword public is an access specifier that declares the main method as unprotected and therefore making it accessible to all other classes.
static: the static keyword which declares method as one that belongs to the entire class and not a part of any objects of the class. The main must be always declared as static since the interpreter uses this method before any objects are created.
void : the type modifier void states that the main method does not return any value .

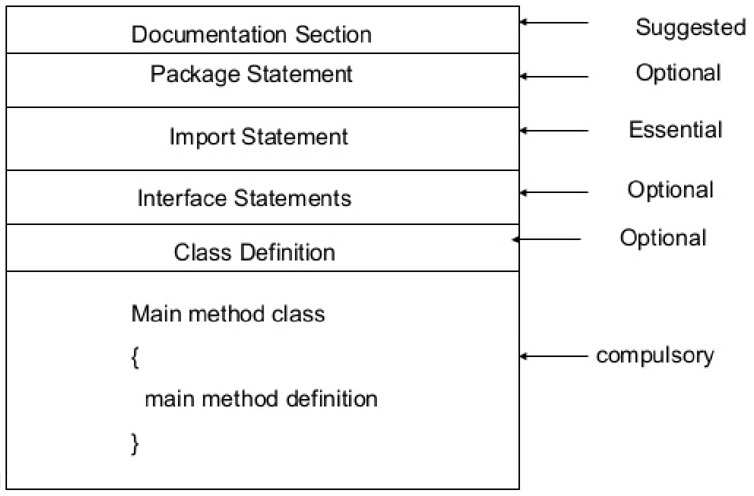
String args[ ] : main ( ) method accepts arguments. It accepts a group of strings, which is also called as string type array. This array is String args[ ] , where String is java's predefined class and args[] is the name of an array. if we pass two strings like " RAMA" and "SITA" , then args[0] contains "RAMA" and args[1] contains "SITA"

The output line: System.out.println("Java is better than C++"); since Java is a true object oriented language , every method must be part of an object. The println method is a member of the out object, which is a static data member of System class.

JAVA PROGRAM WITH MULTIPLE CLASSES:

```
class DemoA
{
        static void functionA( )
        {
                System.out . println(" DemoA – functionA( )");
        }
}
class DemoB
{
        static void functionB( )
        {
                System.out . println(" DemoB – functionB( )");
        }
}
class DemoC
{
        static void functionC( )
        {
                System.out . println(" DemoC – functionC( )");
        }
}
public static void main ( String args [ ])
{
        DemoA.functionA();
        DemoA.functionA();
        DemoA.functionA();
}
}
```

JAVA PROGRAM STRUCTURE:



DOCUMENT SECTION:
This section contains the set of comments lines giving the name of the program, the author and other details which the programmer would like to refer to at a later stage.
PACKAGE STATEMENT:
The first statement allowed in a java file is package statement. This statement declares a package name and informs the compiler that the classes defined here belong to the package.
e.g.: package student;
IMPORT SECTION:

The next thing after a package statement (but before any class definition) may a number of import statements .This is similar to #include statement in c.
E.g. Import java.io.*;

INTERFACE STATEMENT:

An interface is like a class but includes a group of methods declaration. Used only when multiple inheritance features in the program.

CLASS DECLARATION:

A java program may contain multiple class definitions. Classes are the primary and essential features of a java program.

MAIN METHOD CLASS:

Every java stand-alone program requires a main method as its starting point. The main method creates object of various classes and establish communication between them. On reaching end of the main the program terminates and the control passes to back operating system.

EX:

```
// Java program
Class Program
{
        public static void main (String args [])
        {
                System.out.println (“hello java”);
        }
}
```

CHAPTER – 3

2 MARKS:

1. What does static keyword do in class(2016)
2. What is the documentation section?
3. What is the use of package section?
4. What is the order of usage of package, class, and import statements in Java?
5. What is System.out.println ()? Explain what is System.out.println

5 MARKS

1. Write a program to demonstrate multiple classes in single source file.
2. What is public static void main (String args [])? Explain each word.(2010)(2016)
3. Explain java program structure?(2012)(2015)
4. What are the different types of java program?
5. What are the different phases of java programming development?
6. Write a simple java program to print the message and explain.

CHAPTER 4:

JAVA LANGUAGE FUNDAMENTALS:

JAVA TOKENS: The tokens are the building blocks of the java language. All the characters of a java program are grouped into symbols called tokens.

Types of tokens:

- Identifiers
- Keywords
- Operators
- Separators
- Constants
- Comments

JAVA IDENTIFIERS: The identifiers are case sensitive names given to variables, methods, classes, objects, packages etc.

Rules for identifiers:

- All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (_).
- After the first character identifiers can have any combination of characters.
- A keyword cannot be used as an identifier.
- Most importantly identifiers are case sensitive.
- Examples of legal identifiers: age, $salary, _value, __1_value
- Examples of illegal identifiers : 123abc, -salary

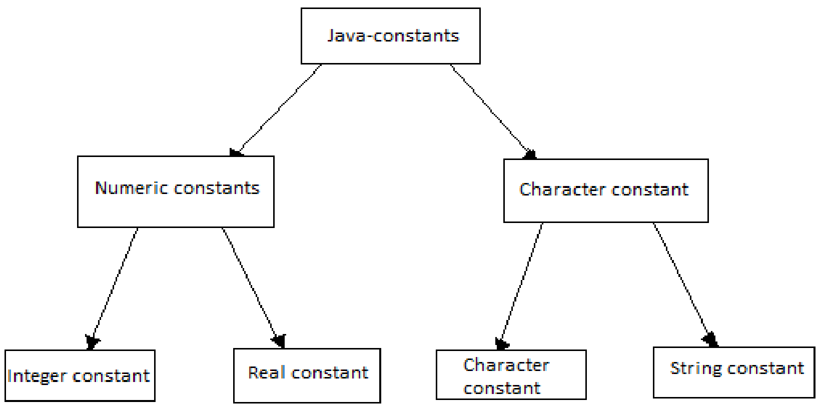
JAVA KEYWORDS:
Keywords are the reserved words and it cannot be used as an identifier. Each keyword has special meaning to the java compiler.

The following list shows the reserved words in Java.

Data Declaration	Modifiers and Access	Looping	Conditional	Exception	Structure	Miscellaneous
Boolean	final	break	case	catch	abstract	import
byte	new	continue	else	finally	class	package
char	private	do	if	throw	default	return
double	protected	for	switch	throws	extends	super
float	public	while		try	implements	this
int	static				instance of	enum
long	mod				interface	strip
shot						

LITERALS/CONSTANTS:

A constant value in program is denoted by a literal. Literals represents numerical, character, Boolean or String values. Constants in Java refer to fixed values that do not change during the execution of a program.



INTEGER CONSTANTS:
Refers to a sequence of digits. There are three types of Integers, namely, decimal, octal and hexadecimal integer.
Decimal integer:-consist of a set of digits, 0 through 9, preceded by an optional minus sign.
Octal integer constant: - consists of any combination of digits from the set 0 through 7, with a leading 0.Ex: 012,0123,08999
Hexadecimal integer constant:-A sequence of digits preceded by ox or OX is considered as hexadecimal integer. They may also include alphabets A through F. Ex: 0x2 ,0x7A

FLOATING POINT CONSTANS:

Floating point Constants: Integer constant are inadequate to represent quantities that vary continuously, such as distance, heights, temperature, prices and so on. These quantities are represented by numbers containing fractional parts like 17.546. Such numbers are called real.
Floating point data consist of float and double types. The default data type of floating point is double
Example: double-0.0, 0.0D, 0.7d…
float -0.0f, 9.0f…

BOOLEAN LITERALS:
True and false are reserved literals representing the truth values true and false respectively. Boolean literals fall under the primitive data type Boolean.

SINGLE CHARACTER CONSTANTS:
A single character constant (or simply character constant) contains a single character enclosed within a pair of single quote marks. Characters in java represented by 16 bit Unicode character set.
Examples of character constants are: '5' 'X' ';'

STRING CONSTANTS:
A string constant is a sequence of characters enclosed between double quotes. The characters may be alphabets, digits, special characters and blank spaces.
Examples are: "Hello Java" "1997".

SEPARATORS: The separators are used by the java compiler to divide a program into segments. Also known as punctuation characters and paired-delimiters. Separators define the structure of a program.

Separator	**Name**	**Use**	**Exmaple**
.	Period	It is used to separate the package name from sub-package name & class name. It is also used to separate variable or method from its object or instance.	Object.method(10,20,30);
,	Comma	It is used to separate the consecutive parameters in the method definition. It is also used to separate the consecutive variables of same type while declaration.	Display(10,20,30);
;	Semicolon	Semicolon are delimiters that mark the end of statements.	Int a,b,c;

()	Parenthesis	This holds the list of parameters in method definition which are being called are declared. Also used in control statements & type casting.	public void calc()
			{
			int a =3*(2+1);
			return a;
			}

{}	Braces	This is used to define the block/scope of code, class, methods. Braces begin and end a block of code statements.	public void display()
			{
			System.out.println("Hello World");
			}

[]	Brackets	It is used in array declaration. Brackets represent the index of an array. Java uses brackets instead of parentheses for array so that they not confused with methods.	int[ ] myarray=new int[10];

COMMENTS IN JAVA:
Comment is a not executable portion of a program that is used to document the program. Because comments are not executable instructions, they are ignored by the compiler. Their sole purpose is to make the program easier for the programmer to read and understand.

Java supports single line and multi-line comments very similar to c and c++. All characters available inside any comment are ignored by Java compiler.

Single line comment-// : Double slashes (//) are used in the C++ programming language and tell the compiler to treat everything from slashes to the end of the line as comment.

Syntax:

```
//This is single line comment
```

Example:

```
public class CommentExample1
 {
        public static void main(String[] args)
        {
                i=10;//Here, i is a variable
               System.out.println(i);
         }
}
```

Multiline comment-/*.........*/ : Instead of double slashes you can use C style comments(/* */)to enclose one or more lines of codes to be treated as comments.

Syntax:

```
/*
This is
multi line
comment
*/
```

Example:

```
public class CommentExample2
{
    public static void main(String[] args)
    {
            /* Let's declare and
            print variable in java. */
            int i=10;
            System.out.println(i);
    }
}
```

Javadoc comment-/…………*/:** To generate documentation for our program, we should use the doc comments (/** */) to enclose lines of text for the Javadoc tool to find. The Javadoc tool locates the doc comment embedded in source files and uses those comments to generate API documentation.

Syntax:

```
/**
 *This is documentation comment
 */
```

Example:

```
/**
 *The Calculator class provides methods to get addition and subtraction of given 2 numbers.
 */
public class Calculator
{
    public static void main(String args[])
    {
            System.out.println("I m a simple program");
    }
}
```

Example:

```
public class MyFirstJavaProgram
{
/* This is my first java program.
* This will print 'Hello World' as the output
* This is an example of multi-line comments.
*/
        public static void main (String []args)
        {
        // This is an example of single line comment
        /* This is also an example of single line comment. */
        System.out.println ("Hello World");
        }
}
```

JAVA STATEMENTS:

A statement is a single executable command and it contains group of java tokens. The sequence of execution of program is controlled by statements. One statement can cover many lines of code but the compiler reads the whole thing as one command. Individual statements end in a semicolon (;) and group statements end in a closing curly braces (}).Multiple line statements are generally called code blocks.

Statements	Description	Example
Package statements	To make the class to be part of any package	Package animal;
Import statements	To import the classes defined in java standard library	Import java.util.Date;
Empty statements	An empty statement does nothing. Execution of an empty statements always completes normally	;

Labeled statements	A label provides a statement with an identifier which will be referred elsewhere in the program. For example, we can use a label to identify a loop, and then use the break or continue statements to indicate whether a program should interrupt the loop or continue its execution.	Outer:
		While(i<10)
		{
		Do something();
		}
		Here , the label outer identifies a while loop.

Expression statements	An expression statement is executed by evaluating the expression.	Int a=3+(8/2*3)/2

Selection Statements	Java has three kinds of statements that executes based on the value of a boolean expression. These statements are the if statement, the if else statements and the switch statements.	If(a==1)
		{ System.out.println("true");
		}
		else
		{
		System.out.println("false");
		}

Iteration statements	Loop statements allow a nested statement to be executed repetitively. Java has three kinds of loop statements : while loop, for loop and do while loop	while(i<10)
		{
		System.out.println(i);
		}

Jump statements	The jump statements are used to transfer the control out of enclosing statement. Example: return, break and continue are jump statements.	while(i<10)
		{
		System.out.println(i);
		if(i = =5)

		break;
		}

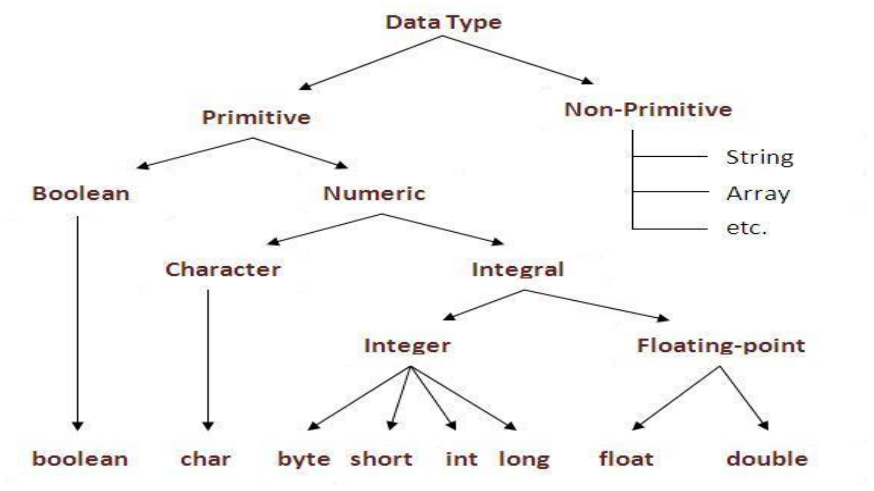
Synchronization statements	These are used to developing multi threaded programs	synchronized(this)
		{
		   statements;
		}

Exception handling statements	These statements are used to handle exception or errors occurred in the program .example: try, catch, finally and throw.	try
		{
		    throw new exception();
		}
		catch
		{
		}
		finally
		{
		}

Assignment statements	These statements are used to assign the value to variables	a=10;b=120;

DATA TYPES:

A data type in a programming language is a set of data with values having predefined characteristics.



Primitive Data Types: There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named by a key word. Let us now look into detail about the eight primitive data types.

BYTE:

- Byte data type is a 8-bit signed two's complement integer.
- Minimum value is -128 (-2^7)
- Maximum value is 127 (inclusive)(2^7 -1)
- Default value is 0
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an int.
- Example : byte a = 100 , byte b = -50

SHORT:

- Short data type is a 16-bit signed two's complement integer.
- Minimum value is -32,768 (-2^15)
- Maximum value is 32,767(inclusive) (2^15 -1)
- Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an int

- Default value is 0.
- Example : short s= 10000 , short r = -20000

INT:

- int data type is a 32-bit signed two's complement integer.
- Minimum value is - 2,147,483,648.(-2^31)
- Maximum value is 2,147,483,647(inclusive).(2^31 -1)
- Int is generally used as the default data type for integral values unless there is a concern about memory.
- The default value is 0.
- Example : int a = 100000, int b = -200000

LONG:

- Long data type is a 64-bit signed two's complement integer.
- Minimum value is -9,223,372,036,854,775,808.(-2^63)
- Maximum value is 9,223,372,036,854,775,807 (inclusive). (2^63 -1)
- This type is used when a wider range than int is needed.
- Default value is 0L.
- Example : int a = 100000L, int b = -200000L

FLOAT:

- float data type is a single-precision 32-bit i.e. 754 floating point.
- float is mainly used to save memory in large arrays of floating point numbers.
- default value is 0.0f.
- float data type is never used for precise values such as currency.
- example : float f1 = 234.5f

DOUBLE:

- double data type is a double-precision 64-bit ieee 754 floating point.
- this data type is generally used as the default data type for decimal values. Generally the default choice.
- double data type should never be used for precise values such as currency.
- default value is 0.0d.
- example : double d1 = 123.4

BOOLEAN:

- boolean data type represents one bit of information.
- There are only two possible values: true and false.
- This data type is used for simple flags that track true/false conditions.
- Default value is false.
- Example : boolean one = true

CHAR:

- char data type is a single 16-bit Unicode character.
- Minimum value is '\u0000' (or 0).
- Maximum value is '\uffff' (or 65,535 inclusive).
- Char data type is used to store any character.
- Example. char letterA ='A'

PROGRAM:

```
public class DataTypeDemo
{
        public static void main(String args[])
        {
                System.out.println("Min Byte Value = " + Byte.MIN_VALUE);
                System.out.println("Min Byte Value = " + Byte.MAX_VALUE);
                System.out.println("Min Int Value = " + Integer.MIN_VALUE);
                System.out.println("Min Int Value = " + Integer.MAX_VALUE);
                System.out.println("Min long Value = " + Long.MIN_VALUE);
                System.out.println("Min Long Value = " + Long.MAX_VALUE);
                System.out.println("Min Float Value = " + Float.MIN_VALUE);
                System.out.println("Min Float Value = " + Float.MAX_VALUE);
        }
}
```

REFERENCE DATA TYPES:

- Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, Employee, Puppy etc.
- Class objects, and various type of array variables come under reference data type.
- Default value of any reference variable is null.
- A reference variable can be used to refer to any object of the declared type or any compatible type.
- Example : Animal animal = new Animal("giraffe");

VARIABLE:

A variable is a container that holds values that are used in a program. The variable provides mechanism to hold data inside a program. Unlike constants that remain unchanged during the execution of program.
Examples of variables: average, height, total height.
Variable name may consist of alphabets, digits, the underscore (_) and dollar characters.

Rules to write Variable/Identifier in Java:

- They must not begin with digit
- Upper and lowercase are distinct. This means that the variable Total is not the same as total or TOTAL.
- It should not be a keyword.
- White space is not allowed.
- Variable names can be of any length.

Declaring Variable:

data type variablename;

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

Initializing Variable:

```
        datatype variablename=value;
```

Example:

```
        public class variable
        {
                public static void main(String args[])
        {
                int myinteger;
                myinteger=5;
                String mystring=”Hello”;
        }
}
```

TYPES/SCOPE OF VARIABLE:

- Local variables
- Instance variables
- Class/static variables

LOCAL VARIABLES:

- Local variables are declared in methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor or block.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor or block.
- Local variables are implemented at stack level internally.
- There is no default value for local variables so local variables should be declared and an initial value should be assigned before the first use.
  Example:

```
public class LocalVaribale
{
        void display()
        {
                int a=10;
                System.out.println(“a=”+a);
        }
        public static void main(String args[])
        {
                display();
        }
}
```

INSTANCE VARIABLES:

- Instance variables are declared in a class, but outside a method, constructor or any block.
- Instance variables are created when an object is created with the use of the key word 'new' and destroyed when the object is destroyed.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.

```
public class InstanceVaribale
{
    int a,b,c;
    float f1.f2;
    double d1,d2;
    public static void main(String args[])
    {
            int x=10;y=20;z=30;
            System.out.println(“Instance variable scope demo”);
    }
}
```

CLASS/STATIC VARIABLES:

- Class variables also known as static variables are declared with the *static* keyword in a class, but outside a method, constructor or a block.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- Static variables are stored in static memory. It is rare to use static variables other than declared final and used as either public or private constants.
- Static variables are created when the program starts and destroyed when the program stops.
- Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.
- Static variables can be accessed by calling with the class name. *ClassName.VariableName*.

```
public class Variable
{
        int a,b,c;
        float f1,f2;                    } instance variable
        double d1,d2;

        static int s1,s2;
        static float r1,r2;         }static variable

        public static void main(String args[])
        {
                int x=10,y=20;──────────► local variable
                System.out.println ("Instance variable scope demo");
        }
}
```

SYMBOLIC CONSTANT:

- We can define symbolic constant in java by using Keyword "final"
- This is similar to the keyword const c/c++;
- The final keyword specifies that the value of a variable will not change throughout the program.
  Syntax:
  ```
  Final datatype variablename =value;
  final double PI=3.14;
  ```

TYPE CASTING:

Casting: Casting is the process of converting one variable of certain data type into another data type.

There are two types of casting. Implicit casting which is also called as automatic conversion or widening conversion and Explicit casting is also called narrowing conversion.

IMPLICIT CASTING/AUTOMATIC CONVERSION/WIDENING CONVERSION:

Automatic casting done by the java compiler internally is called implicit casting. Implicit casting is done to convert lower data into a higher data type.

$$\begin{tikzcd}
\boxed{\text{byte}} \arrow[r] & \boxed{\text{short}} \arrow[r] & \boxed{\text{int}} \arrow[r] & \boxed{\text{long}} \arrow[r] & \boxed{\text{float}} \arrow[r] & \boxed{\text{double}} \\
 & & \boxed{\text{char}} \arrow[u] & & &
\end{tikzcd}$$

The following conversion takes places implicitly:

byte ->short->int->ling->float->double.
short->int->ling->float->double.
char->int->ling->float->double.
int->long->float->double
long->float->double
float->double.

Boolean is not compatible with any data type and hence it cannot be converted to other data types.

If we to convert double to float, then it is not possible through implicit casting .we need to do it explicitly.
Example:

1. int i=65;
   long l=i;
   Both int and long are compatible types. long takes 8 bytes and int takes 4 bytes. Long can hold int value without any loss of data.

2. short s= 65;
   float f=s;
   both short and float are compatible types. They both belongs to numeric types. Though short is integer type an d float is Real type. Float takes 8bytes and short takes 2 bytes.

3. float f=3.14f;
   double d=f;
   both float and double are compatible types. Double takes 8 bytes and float takes 2 bytes. Double can hold float value without any loss of data.

4. char ch='A';
   int i=ch;
   characters are represented in memory as Numeric. Hence both int and char are compatible. Int takes 4 bytes and char takes 2 bytes. int can hold char value without any loss of data.

EXPLICIT CASTING:

The casting done by the java programmer or developer is explicit casting. Explicit casting is must while converting from higher data type to lower data type.
Syntax:

(typename)value:

Examples:

1. int i=65;
   short s(short)i;
   Short is 2 bytes and int is 4 bytes.4 bytes cannot fit in 2 bytes. Convert i=65 to short and then assign the value to s. No loss of data.

2. double d=89.567;
   int i=(int)d;
   First double is converted into int and then assign to i. Loss of data as i cannot store prision.

WHAT IS WIDENING AND NARROWING:

Converting lower data type into higher data type is called widening.
Converting a higher data type into lower data type is called narrowing.

Widening is safe, because even if the programmer does not use cast operator, the java compiler takes care of it. Narrowing is unsafe, because the programmer should explicitly use cast operator in narrowing.

READING INPUT FROM KEYBOARD.

- A stream is required to accept input from the keyboard.A stream represents flow of data from one place to another place.A stream can carry data from keyboard to memory or memory to monitor or from memory to printer.
- There are two types of stream INPUT AND OUTPUT.
- Input stream are those stream which receives or read data from some other place.
- Output stream are those streams which send or write data to some other place.
- The console is represented by a field, called "out" in System class. We can write output to console using "System.out".
- The keyboard is presented by a field called "in" in System class. We can read java input from "System.in".

There are 4 common ways to read input from console.

1. InputStreamReader wrapped in BufferedReader.
2. Scanner class in JDK1.5.
3. Console
4. DataInputStream

INPUTSTREAMREADER CLASS:

InputStreamReader class can be used to read data from keyboard. It performs two tasks:

- Connects to input stream of keyboard
- Converts the byte-oriented stream into character-oriented stream

To accept data from keyword, we need to connect it to an input stream. The following steps are required in this process.

Step 1: Connect the keyword (System.in)to an input stream object. We can use InputStreamReader class that can read data from the keyboard.

```
InputStreamReader is = new InputStreamReader(System.in);
```

In the above statement, we have created InputStreamReader object and connecting the keyboard(System.in) to it.
Step 2: Connect InputStreamReader to BufferedReader, which is another input stream. We are using BufferedReader as it has got methods to read data properly from the input stream.

```
BufferedReader br= new BufferedReader(is);
```

Here, we have created BufferedReader object and connected to InputStreamReader object is. We can achieve Step1 and step2 in single statement.

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

Step 3: Now we can read the data from the keyboard using read() and readLine() methods, of BufferedReader class.
Step 4: To read entire line from the keyboard.

```
String str=br.readLine();
```

Step 5: To read a Single character from the keyboard , we can use read( ) method.

```
char ch = (char)br.read();
```

Step 6: To read an integer from the keyboard.

```
int age = Interger.parseInt(br.readLine());
```

Remember what ever enter in the console is treated as string and hence we should convert it our required data type.

Example:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class InputDemo

{
        public static void main(String args[])
        {
                BufferedReader br= new BufferedReader(new InputStreamReader(System.in));
                System.out.println('Enter your name");
                String str=br.readLine();
                System.out.println("Enter your age");
                int age= Interger.parseInt(br.readLine());
                System.out.println("Are you Male or Female");
                char ch=(char)br.read();
                System.out.println("Name is =" +str);
                System.out.println("Age is ="+age);
                System.out.println("Sex is ="+ch);
        }
}
```

USING SCANNER CLASS:

The scanner is a class in java.util package. The scanner class is introdueced in java 1.5 . it will work only if we have JDK1.5 onwards. The Scanner class is a class used for scanning primitive and string. It can be used to get input from the InputStream(System.in), to parse through a String of text or to read from a file. The usage of Scanner class is easy way to read input from file or console and code is cleaner.
Step 1: import Scanner class from java.util package

```
import java.util.Scanner;
```

Step 2: create Scanner object and attach it to input stream(System.in)

```
Scanner sc=new Scanner (System.in);
```

Step 3: Now, if user has given the integer value from the keyboard, it is stored into the Scanner object(sc) as a token. To receive that token, we can use the method: sc.nextInt( ). To receive string use sc.next( ) and to receive float use sc.nextFloat methods.

```
String str=sc.next( );
int age=sc.nextInt();
float avg = sc.nextFloat();
double amount=sc.nextDouble( );
long bignumber=sc.nextLong( );
byte smallnumber=sc.nextByte( );
```

```
import java.util.Scanner;
```

```
public class ScannerDemo
{
        public static void main (String args[])
        {
                Scanner sc= new Scanner(System.in);
                System.out.println(‘Enter your name”);
                String str=sc.next( );
                System.out.println(“Enter your Age”);
                int age=sc.nextInt();
                System.out.println(“Enter the Amount”);
                float amount=sc.nextFloat();
                System.out.println(“Name is =” +str);
                System.out.println(“Age is =”+age);
                System.out.println(“Amount is =”+amount);
        }
}
```

WRITING DATA TO CONSOLE:

To display a value

```
class Test
{
        Public static void main(String args[])
        System.out.print(‘A’);
}OUTPUT: A
```

To display a word or sentence:

```
class Test
{
        Public static void main(String args[])
        System.out.print(“Skyward Publisher’);
}OUTPUT: Skyward Publisher
```

To display a constant number:

```
class Test
{
        Public static void main(String args[])
        System.out.print(199);
}OUTPUT: 199
```

To display a variable:

```
class Test
{
        Public static void main(String args[])
        int a=100;
        int b=200;
        System.out.print(“ the value of a =” + a);
        System.out.print(“ the value of b =” + b);
}OUTPUT: A
```

What is the use of System.out.printf():

System.out.printf() can be used to send formatted numerical output to the console. It uses a java.util.Formatter object internally and it is similar to printf() function in C . it is easy and best way to format the output. It included from java 1.5.

Ex: System.out.printf(“ the value of a is %d and b is %d”,a,b);

Format character	Decription
%s	String
%c	Character
%d	Decimal Integer
%f	Float
%o	Octal Number
%b	Boolean
%x	Hexa Decimal Number
%e	Number in scientific notation
%n	New Line Character.

To display the output we have to use either System.out.println () or System.out.println ().

Unicode System: Unicode system is an encoding standard that provides a unique numner for every character, no matter what the platform, program or language is.Unicode is standard to include the alphabet from all human language. Java supports Unicode system.

CHAPTER – 4

2 MARKS:

1. What are the different types of comments in java?
2. What is explicit casting?
3. What is type casting?
4. Which are the reserved literals in java
5. List the basic data types supported in java
6. What is System.out and System.in?
7. What is empty statement?(2011)
8. What are symbolic constants?
9. What are separators?(2011)
10. What are transient variables?
11. How do you input user input in java.
12. What is meant by scope of a variable.(2012)
13. What are the data types used in java(2012).
14. What are the two ways of giving values of to a variable(2015).
15. Write down the default values of byte and char datatypes.(2015)
16. What are string literals(2016)
17. What is an identifier? What are the rules for identifier?
18. What is keyword? List some of the keywords in java?
19. What is null, true and false in java?
20. Explain the different types of literals in java with example?
21. List the different separators used in Java with an example:
22. Mention the size of primitive data types in java
23. What are the default values of primitives in java
24. What is Unicode character
25. What is variable? How to declare and initialize a variable
26. What are the class variables
27. What is the use of final keyboard
28. What is scanner class
29. What is the difference between System.out.println() and System.out.print()

30. What is use of System.out.printf()
31. Which is standard input device and output device in java
32. What is casting? Why is it required? Mention the different types of casting
33. Explain implicit casting and explicit casting with example
34. What is widening and narrowing

5 MARKS:

1. Explain various java statements with an example.
2. Explain the classification of java data bytes and give an example for each
3. How to read a value from the keyboard? explain with an example
4. Explain the scope of variable(2015)
5. How to define constants in java
6. What are java tokens? Explain with examples?
7. Define variables. what are the different kinds of variables Java defined
8. What are the different types of Literals in java(2010)

OPERATORS & EXPRESSION:

Operators are special symbols for things

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Conditional Operators
- Logical operator
- Short hand assignment operator.

ARITHMETIC OPERATOR:

The arithmetic operators are the operators that perform any basic form of mathematics. These includes : addition, subtraction, multiplication and division and modulus**.**
Assume integer variable A holds 10 and variable B holds 20 then:

op	**Description**	**Example**
+	Addition - Adds values on either side of the operator	A + B will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	A - B will give -10
*	Multiplication - Multiplies values on either side of the operator	A * B will give 200
/	Division - Divides left hand operand by right hand operand	B / A will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	B % A will give 0

```
public class Test
{
        public static void main(String args[])
        {
                int a = 10;
                int b = 20;
                int c = 25;
                int d = 25;
                System.out.println("a + b = " + (a + b) );
                System.out.println ("a - b = " + (a - b) );
                System.out.println ("a * b = " + (a * b) );
                System.out.println ("b / a = " + (b / a) );
                System.out.println("b % a = " + (b % a) );
                System.out.println("c % a = " + (c % a) );
        }
}
```

UNARY OPERATOR:

Unary operators are operators that only require one operand. The unary operators used without an assignment operator. They simply perform their operations on a single variable, changing its value appropriately.

Op	Description	Example
+	The "+" operator refers to the positive value.	Int a=+10;
-	The "-" operator refers to the positive value	Int a=-10;

++	Increment operator - to increase the value of operand by 1	Int a=10;
		A++;
		++a;

--	Decrement operator – to decrease the value of operand by 1	Int a=10; a--; --a;

RELATION OPERATOR:

Relational operators are used to compare two values and determine the relationship between them.

Assume variable A holds 10 and variable B holds 20 then:

Op	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks-- if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

```
public class Test
{
        public static void main(String args[])
        {
                int a = 10;
                int b = 20;
                System.out.println("a == b = " + (a == b) );
                System.out.println("a != b = " + (a != b) );
                System.out.println("a > b = " + (a > b) );
                System.out.println("a < b = " + (a < b) );
                System.out.println("b >= a = " + (b >= a) );
                System.out.println("b <= a = " + (b <= a) );
        }
}
```

THE LOGICAL OPERATORS:

The logical operator returns a true or false value based on the state of the variable . there are 6 logical operators.

Short Circuit Operator: && and ||

Assume boolean variables A holds true and variable B holds false then

Op	**Description**	**Example**
&&	Called Logical AND operator. If both the operands are non-zero then then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands are non-zero then then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

```
public class Test
 {
        public static void main(String args[])
        {
                boolean a = true;
                boolean b = false;
                System.out.println("a && b = " + (a&&b));
                System.out.println("a || b = " + (a||b) );
                System.out.println("!(a && b) = " + !(a && b));
        }
}
```

INCREMENT AND DECREMENT OPERATOR:

Assume integer variable A holds 10 and variable B holds 20 then:

++	Increment - Increase the value of operand by 1	B++ gives 21
--	Decrement - Decrease the value of operand by 1	B-- gives 19

THE ASSIGNMENT OPERATORS:

The assignment operator is the single equal sign =. The behavior os assignment operator is similar to other programming languages like C and C++.

The syntax: var = expression

```
Example: int x=10;
           int x,y,z;
           x=y=z=100;
```

THE SHORTHAND ASSIGNMENT OPERATOR:

The shorthand assignment operator performs shortcut in common programming operation. It is also called as Compound Assignment operators.

Syntax: var1 operator = var2.

Op	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assigne value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the	C *= A is equivalent to

	left operand and assign the result to left operand	C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

```
public class Test
{
        public static void main(String args[])
        {
                int a = 10;
                int b = 20;
                int c = 0;
                c = a + b;
                System.out.println("c = a + b = " + c );
                c += a ;
                System.out.println("c += a = " + c );
                c - = a ;
                System.out.println("c -= a = " + c );
                c *= a ;
                System.out.println("c *= a = " + c );
                a = 10;
                c = 15;
                c /= a ;
                System.out.println("c /= a = " + c );
         }
}
```

CONDITIONAL OPERATOR (?:):

A conditional operator is also known as ternary operators. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide which value should be assigned to the variable.

Syntax : variable x = (expression) ?value if true : value if false

Example:

```
public class Test
{
        public static void main(String args[])
        {
                int a , b;
                a = 10;
                b = (a == 1) ? 20: 30;
                System.out.println( "Value of b is : " + b );
                b = (a == 10) ? 20: 30;
```

```
                    System.out.println( "Value of b is : " + b );
          }
}
```

THE BITWISE OPERATORS:

Assume integer variable A holds 60 and variable B holds 13 then:

Op	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in eather operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the efect of 'flipping' bits.	(~A) will give -60 which is 1100 0011
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>>	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

```
public class Test
{
          public static void main(String args[])
          {
                    int a = 60; /* 60 = 0011 1100 */
                    int b = 13; /* 13 = 0000 1101 */
                    int c = 0;
                    c = a & b; /* 12 = 0000 1100 */
                    System.out.println("a & b = " + c );
                    c = a | b; /* 61 = 0011 1101 */
                    System.out.println("a | b = " + c );
                    c = a ^ b; /* 49 = 0011 0001 */
                    System.out.println("a ^ b = " + c );
                    c = ~a; /*-61 = 1100 0011 */
                    System.out.println("~a = " + c );
                    c = a << 2; /* 240 = 1111 0000 */
                    System.out.println("a << 2 = " + c );
                    c = a >> 2; /* 215 = 1111 */
                    System.out.println("a >> 2 = " + c );
                    c = a >>> 2; /* 215 = 0000 1111 */
                    System.out.println("a >>> 2 = " + c );
          }
}
```

INSTANCE OF OPERATOR:

This operator is used only for object reference variable. This operator checks whether the object is of particular type (class type or interface type)

```
Syntax: (Object reference variable) instanceof (class/interface type)
Example: String name=”Srikanth”;
```

```
boolean result=name instanceof String; // this will return true since name is of type String.
```

WHAT IS OPERATOR PRECEDENCE?

Operator precedence determines the order in which expression are evaluated.

WHAT IS ASSOCIATIVITY?

When two operands have the same precedence, associativity determines evaluation order.

If an expression contains both * and / . It has same precedence. In this case associativity determines the evaluation order. Asssociativity can be either left to right or right to left.

PRECEDENCE OF JAVA OPERATORS:

Category	Operator	Associativity
Postfix	() [] . (dot operator)	Left to right
Unary	++ - - ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>>>>>><<	Left to right
Relational	>>= <<=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	“:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

ESCAPE SEQUENCE:

Java language supports few special escape sequences for String and char literals as well. They are:

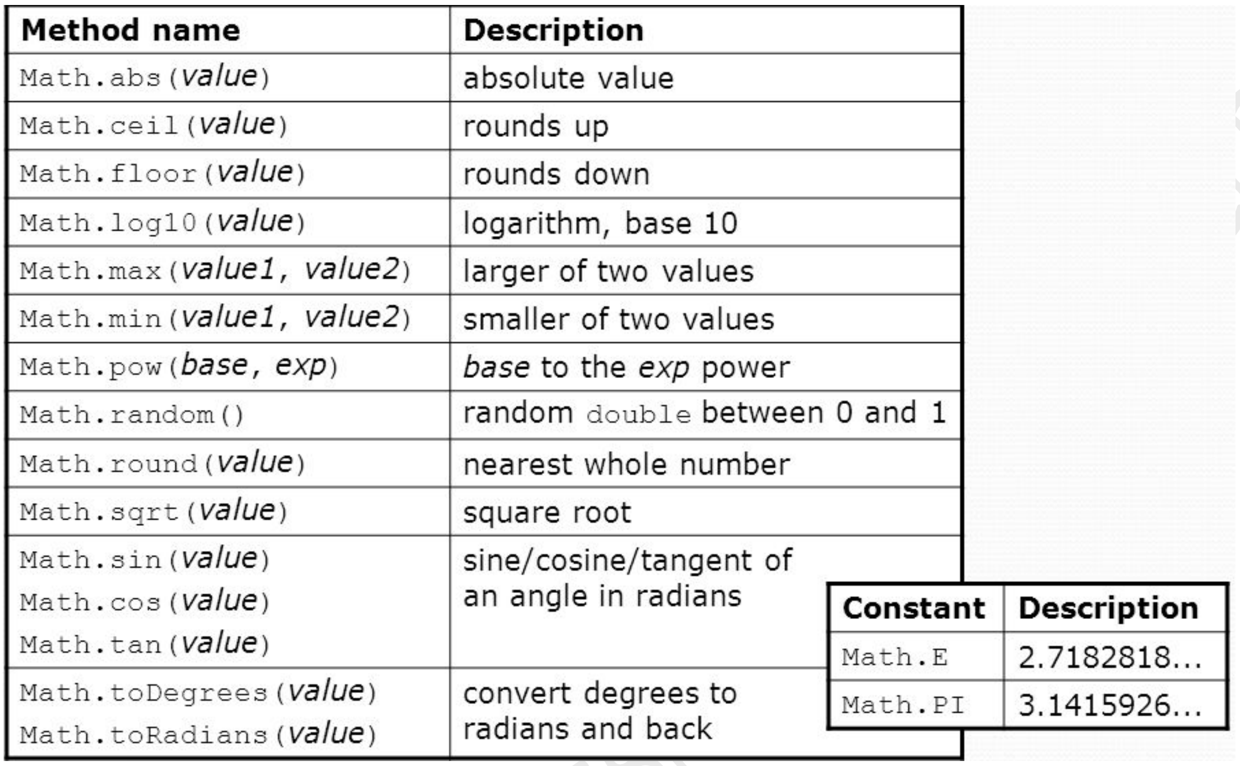
Notation	Character represented
\n	Newline (0x0a)
\r	Carriage return (0x0d)
\f	Formfeed (0x0c)
\b	Backspace (0x08)
\s	Space (0x20)
\t	tab
\"	Double quote
\'	Single quote
\\	backslash
\ddd	Octal character (ddd)
\uxxxx	Hexadecimal UNICODE character (xxxx)

MATHEMATICAL FUNCTION:

- The Math class provides a number of useful methods . All Math methods are called by this syntax : Math . method(parameters)

- The Math class is part of java.lang package.
- It is in the java.lang package, the Math class need not be imported. The lang package is default for all the programs.

- The constants E and PI are defined in Math class.
- Math.E : 2.71828 math.PI : 3.14159
- All methods in Math class are static and hence we can use the methods using class name.
- The Math class cannot be instantiated, which means we cannot create objects of Math class . because math class is declared as final. Any final class cannot be instantiated.
- All methods take t least one argument, except random which takes no arguments.



Method name	Description
`Math.abs(`*value*`)`	absolute value
`Math.ceil(`*value*`)`	rounds up
`Math.floor(`*value*`)`	rounds down
`Math.log10(`*value*`)`	logarithm, base 10
`Math.max(`*value1, value2*`)`	larger of two values
`Math.min(`*value1, value2*`)`	smaller of two values
`Math.pow(`*base, exp*`)`	*base* to the *exp* power
`Math.random()`	random `double` between 0 and 1
`Math.round(`*value*`)`	nearest whole number
`Math.sqrt(`*value*`)`	square root
`Math.sin(`*value*`)` `Math.cos(`*value*`)` `Math.tan(`*value*`)`	sine/cosine/tangent of an angle in radians
`Math.toDegrees(`*value*`)` `Math.toRadians(`*value*`)`	convert degrees to radians and back

Constant	Description
`Math.E`	2.7182818...
`Math.PI`	3.1415926...

Take Xerox of page no:5.35 , 5.36 and 5.37.

<u>CHAPTER 5:</u>

1. What is instance of operator?(2010)
2. What are different operators supported in java
3. Explain arithmetic operator with an example
4. What are unary operator
5. Explain the difference between pre increment and post increment
6. Explain assignment and compound assignment operators
7. Explain relational and logical operators with example
8. What is the difference between normal operator and short circuit operator
9. Explain bitwise operator
10. What is the difference between >> and >>> operators
11. What is bitwise XOR operator
12. Write a program to demonstrate bitwise operator
13. What is Boolean expression
14. What is string conversion
15. What is string expression
16. Explain type conversion in expression
17. How Type promotion happens in Java
18. What is operator precedence and associatively
19. Explain operator precedence
20. What is Math class? Mention important mathematical function of Math class with example

21. What are the constant defined in Math class
22. Why we cannot create a object of Math Class
23. Write a program to demonstrate mathematical functions of Math Class.

5MARKS

1. Explain the operators in java
2. What are the different types of expression in java? Explain with example
3. Difference between equal() method and = = symbol
4. Explain the difference between logical and short-circuit operators.
5. Explain math class in java(2010)
6. What is conditional operator? Explain with an example(2010)
7. Explain bitwise and logical operator with example(2016).

DECISION MAKING STATEMENTS:

WHAT ARE THE DIFFERENT CATEGORIES OF CONTROL STATEMENT:

- Selection statements : if and switch
- Iteration statements : for, while and do-while
- Jump statements: break, continue and return.

SIMPLE IF STATEMENT: The if statement is Java's conditional branch statement. The if statements executes a block of code only if the specified expression is true . If the value is false then if block is skipped And execution continues with the rest of the program.

Syntax:

```
if(expression)
{
     Statements;
}
```

Example:

```
 int age;
 if(age>=18)
{
                  System.out.println("vote ");
}
```

IF ELSE STATEMENT :

The if/else statements is an extension of the if statement. if the statement on the if statement fails, the statements in the else block are executed . You can either have a single statement or a block of code within if – else blocks.

Syntax:

```
if(expression)
{
   // statement1
}
else
{
   // statement2
}
```

This if else work like: if condition is true than statement1 is executed otherwise statement2 is executed.
Example:

```
public class A
{
            public static void main(String args[])
            {
                        int a = 10;
                        int b = 20;
                        if(a>b)
                        {
                                    System.out.println("a is greater than b");
                        }
                        else
                        {
                                    System.out.println("b is greater than a");
                        }
            }
}
```

ELSE-IF-LADDER:
The else-if ladder is a multi way decision maker which contains two or more else –if , from which any one block is executed.
Syntax:

```
if(expression)
{
        // statement1;
}
else if(expression)
{
        // statement2;
}
else if(expression)
{
        // statement3;
}
........
else
{
        // else statement
}
```

The if statement executed from top to down. As soon as one of the condition is true, statement associated with that if statement executed.

If none of the condition is true than else statement will be executed, only one of the statement executed from list of else if statements.

Example:

```
public class IfDemo
{
            public static void main(String args[])
            {
                    int percentage = 65;
                    if(percentage >= 70)
                    {
                              System.out.println("First class with Distinction");
                    }
                    else if(percentage >= 60)
                    {
                              System.out.println("First Class");
                    }
                    else if(percentage >= 48)
                    {
                              System.out.println("Second Class");
                    }
                    else if(percentage >= 36){
                              System.out.println("Pass Class");
                    }
                    else
                    {
                              System.out.println("Fail");
                    }

          }
}
```

NESTED IF STATEMENT:

We can put if statements inside other if statements which will make them nested if statements.

Syntax:

```
if(expression 1)
{
          if(expression 2)
                    statement 1;
          else
                    statement 2;
}

else
{
          if(expression 3)
                    statement 3;
          else
                    statement 4;
}
```

Example:

```
public class largestnumber
{
          public static void main(String args[])
          {
                    int a =10 , b=20,c=30;
                    if(a>b)
                    {
                              if(a>c)
```

```
                        {
                                System.out.println(" a is largest");
                        else
                                System.out.println(" c is largest");
                        }
                else
                {
                        if(b>c)
                        {

                                System.out.println(" b is largest");
                        else
                                System.out.println(" c is largest");
                        }
                }
        |}
}
```

SWITCH STATEMENT:
The switch statement is multi way branch statement in java programming. It is use to replace multilevel if-else-if statement.
Syntax:

```
switch(expression)
{
case value 1:
        // statement 1
        break;
case value 2:
        // statement 2
        break;
case value n:
        // statement n
        break;
default:
        //statements
        break;
}
```

The expression type must be the byte, short, int and char.
Each case value must be a unique literal(constant not a variable). Duplicate case value not allowed.
The each case value compare with expression if match found than corresponding statement will be executed.
If no match is found than default statement will be executed. Default case if optional.
The break statement use to terminate statement sequence, if break statement is not written than all statement execute after match statement.
Example:

```
public class Ifdemo
{
        public static void main(String args[]){
                int day = 1;
                switch(day)
```

```
                    {
                              case 0:
                                        System.out.println("Sunday");
                                        break;
                              case 1:
                                        System.out.println("Monday");
                                        break;
                              case 3:
                                        System.out.println("Tuesday");
                                        break;
                              case 4:
                                        System.out.println("Wednesday");
                                        break;
                              case 5:
                                        System.out.println("Thursday");
                                        break;
                              case 6:
                                        System.out.println("Friday");
                                        break;
                              case 7:
                                        System.out.println("Saturday");
                                        break;
                              default:
                                        System.out.println("Invalid Day");
                                        break;
                    }
          }
}
```

JAVA LOOP CONTROL:

There may be a situation when we need to execute a block of code several number of times, and is often referred to as a loop.
Java has very flexible three looping mechanisms. You can use one of the following three loops:

- while Loop
- do...while Loop
- for Loop

As of java 5 the *enhanced for loop* was introduced. This is mainly used for Arrays.

THE WHILE LOOP:

A while loop is a control structure that allows you to repeat a task a certain number of times.
Syntax:
The syntax of a while loop is:

```
while (expression)
{
//Statements
}
```

When executing, if the *expression* result is true then the actions inside the loop will be executed. This will continue as long as the expression result is true.
Here key point of the *while* loop is that the loop might not ever run. When the expression is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.
Example:

```
public class Test
{
        public static void main(String args[])
        {
                int x = 10;
                while( x < 20 )
                {
                    System.out.println("value of x : " + x );
                    x++;

                }
        }
}
```

THE DO...WHILE LOOP:

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.
Syntax:
The syntax of a do...while loop is:

```
do
{
//Statements
}while(expression);
```

Notice that the Boolean expression appears at the end of the loop, so the statements in the loop execute once before the Boolean is tested.
If the Boolean expression is true, the flow of control jumps back up to do, and the statements in the loop execute again. This process repeats until the Boolean expression is false.

Example:

```
public class Test
{
        public static void main(String args[])
        {
        int x = 10;
        do
        {
                System.out.println("value of x : " + x );
                x++;
        }while( x < 20 );
        }
}
```

THE FOR LOOP:
A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.
A for loop is useful when you know how many times a task is to be repeated.

Syntax:
The syntax of a for loop is:

```
for(initialization; expression; update)
{
//Statements
}
```

Example:

```
public class Test
{
        public static void main(String args[])
        {
                for(int x = 10; x < 20; x = x+1)
                {
                System.out.println("value of x : " + x );
                }
        }
}
```

ENHANCED FOR LOOP IN JAVA:
As of java 5 the enhanced for loop was introduced. This is mainly used for Arrays.
Syntax:
The syntax of enhanced for loop is:

```
for(declaration : expression)
{
//Statements
}
```

Example:

```
public class Foreachdemo
{
        public static void main(String args[])
        {
                int[] number={10,20,38,40,650};
                for(int x:numbers)
                {
                        System.out.println(x);
                }
}
}
```

THE BREAK KEYWORD:
The *break* keyword is used to stop the entire loop. The break keyword must be used inside any loop or a switch statement.
The break keyword will stop the execution of the innermost loop and start executing the next line of code after the block.

```
while(condition)
{
        ---------
        if(condition)
                break;
                ---------
                -----------
}
statement-x;
```

ex:

```
int i=1;
while(true)
{
        if(i==11)
                break;
        system.out.println(i)
```

```
    i++;                                                        }
```

THE CONTINUE KEYWORD:

The *continue* keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop.

- In a for loop, the continue keyword causes flow of control to immediately jump to the update statement.
- In a while loop or do/while loop, flow of control immediately jumps to the Boolean expression.

```
while(condition)
{
    ---------
    if(condition)
            continue;
            ---------
            -----------
}
statement-x;
ex:
int i=1;
while(true)
{
    if(i==11)
            continue;
    system.out.println(i)
    i++;
}
```

LABELLED LOOP:

A label is java identifier. Label is not a variable. It is just a name.

```
Outer: for (int i=1;i<=10;i++)
      {

            Inner: while(true)
            {
                    If(i==5)
                    Continue outer;
            }
      Statements;
       }
```

Chapter 6:

1. Give general form of switch statements.(2016)

2.	What is different type of control statements in java
3.	Explain simple if statements with an example
4.	What are the jump statements
5.	Explain if else statements with an example
6.	Explain nested if else statements
7.	Write a program to find the largest of three numbers using nested if else
8.	Explain else if ladder
9.	Explain switch statements with an example
10.	What is the difference between switch and if else
11.	Write any four rules of using switch statements

12. What are the advantages and disadvantages of switch statements
13. What are the different loop statements in java
14. Explain while loop and do loop while with an example
15. Explain the difference between while and do while
16. Explain for loop with an example
17. Explain break continue statements with an example
18. Why goto statements are not available in a java

5 MARKS:

1. What is for each loop? How it is useful? Explain with an example
2. Explain the iterative statements in java.
3. Write a program to reverse a number using for loop
4. What is labeled loop? explain with an example(2010)
5. Write a java program to check whether the given number is prime or not.(2011)
6. Compare a) while and do while b) while and for c) break and continue(2011)

UNIT 2-JAVA

CHAPTER 7:

Java is an Object Oriented Language. As a language that has the Object Oriented feature Java supports the following fundamental concepts:

- Polymorphism
- Inheritance
- Encapsulation
- Abstraction
- Classes
- Objects
- Instance
- Method
- Message Parsing

CLASSES:

Definition:

➔ A Class is a template that defines the form of an object. It specifies both the data and code that will operate on that data. A Class is s template for creating multiple objects with similar features.

➔ A class contains data members and methods. Class=data+methods

➔ A class is created by using the keyword "class".

Data members: These are variables that store data items that typically differentiate one object of the class from another.

Methods: These define the operations you can perform for the class, they determine what you can do to or with objects of the class.

GENERAL FORM OF A CLASS / DEFINING A CLASS

```
[AccessModifier][ClassModifier] class class_name [extend SuperclassName][ImplementsInterfaceList]
{
        [Variable declaration]
        [Method declaration]
}
```

Example:

```
        Class Test
        {
        }
```

➔ The elements between the pair of square brackets [] are optional.

➔ The access modifier specifies who can access this class.

➔ The class modifiers specify the behavioral restrictions on this class. E.g. the class modifier abstract indicates that we cannot create objects from this class.

RULES FOR DEFINING A CLASS:

➔ The class name starts with capital letter.

- ➔ There can be only one public class per source file.
- ➔ It must be legal java identifier, it must start with a digit,letter, $ or _. EX: $5_a$$8

Can we declare a class a private:

We cannot declare a class as private, because it is not available to java compiler and hence we get compile time error? But inner classes can be declared as private.

ADDING VARIABLE TO CLASS:

- ➔ We can declare the variable inside the class. The variables inside a class are of two types. The class variables and instance variables.
- ➔ The class variables will always have the modifier static in front of them. The class variables are also called as static variables.

```
Ex:     static int age=30;
```

- ➔ The same variable defines as instance variable.

```
Ex:     int age=30;
```

EXAMPLE:

```
public class VariablesDemo
{
        //declaring instance variable
        int age;
        intregno;
        float per;     //declaring class variables.
        static int collegecode;     //declaring char variable
        char sex='m';        //declaring class variable with initial value.
        static double fees=30000.00;
}
```

ADDING METHODS TO CLASS:

- ➔ A method contains one or more statements. Each methods performs only one task.
- ➔ Java keywords should not be used as methods names.

SYNTAX:

```
[modifiers]return_typemethod_name(parameter_list)
{
//body of method
}
```

EX:

```
public class MethodDemo
{
        int num1,num2;
        //method which does not return any value
        void subtract(int a, int b)
        {
                num1=a;
                num2=b;
                int c = num1-num2;
                System.out.println(c);
        }
}
```

CREATING OBJECT:

- An object is created by instantiating a class. The process of creating a class is called as instantiation and created object is called as instance.
- To create a new object java uses the new created.
- Object contains its own copies of non-static variables and addresses of the methods.

GENERAL FORM:

```
class_name reference_variable=new class_name (arguments);
```

- class_name is the name of the class
- reference_variable is can refer to an object
- arguments are optional

WHAT HAPPENS WHEN AN OBJECT IS CREATED?

- The below code creates an instance of Account class.
  Account acc;   //declaring reference to object
  acc = new Account();   // allocates a Account object
  Account acc= new Account (); This declaration performs two functions.
- First: It declares a variable called acc of the class type Account. This variable does not define an object. Instead, it is simply a variable that can refer to an object.
- Second, the new Account() creates a physical copy of the object in heap memory and assigns the address of the object to acc reference variable.
- Whenever JVM encounters new operators, its duty is to create the memory for that object in heap memory.
- The new operator dynamically allocates memory for an object and returns a reference to it. This reference is the address in memory of the object allocated by new. This reference is then stored in a variable.
- In java all class objects must be dynamically allocated.
- The memory model of creating an objects in shown below

WHAT HAPPENS WHEN MULTIPLE OBJECTS ARE CREATED?

Accounts acc1 = new Account ();
Accounts acc2 = new Account ();
Accounts acc3 = acc2;

ACCESSING CLASS MEMBERS:

To access variables: object • variable;
To access methods: object • method ();

Example:

```
class Demo
{
	void funx()
	{
		int i=10 , j=20;
		System.out.println (“i=”+i);
		System.out.println (“j=”+j);
	}
	public static void main (String args[])
	{
		Demo ob= new Demo ();
		ob.funx();
	}
}
```

CONSTRUCTORS:

- ➔ A constructor is a special method whose task is to initialize the object of its class. It is special it means its name is the same as the class name. A constructor is invoked whenever an object of its associated class is created.
- ➔ They do not have return types, so they cannot return values.
- ➔ All classes have constructor, whether we define it are not, because java automatically provides a default constructor that initializes all member variables to zero.
- ➔ Once we define our own constructor, the default constructor is no longer is used.
- ➔ A constructor is called concurrently when the object creation is going on. The JVM first allocates memory for the object and then executes the constructor to initialize the instance variable.

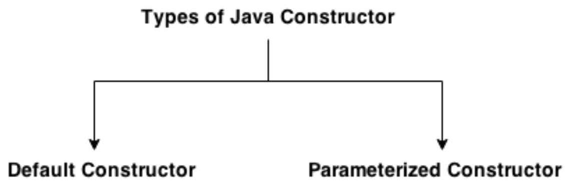
RULES FOR CONSTRUCTOR:

- The constructor name must match the name of the class.
- The constructors must not have a return type.

- The constructor can use any access modifiers like public, protected and public.
- The static keyword cannot be applied to constructors.
- The constructor is not compulsory for any class.
- The constructors cannot be called from methods.
- We can call constructors from other constructor.

THERE ARE TWO TYPES OF CONSTRUCTORS:

1. Default constructor
2. Parameterized constructor



JAVA DEFAULT CONSTRUCTOR

A constructor that has no parameter is known as default constructor. The Java provides the default zero arguments constructor, only when none of the constructors are defined in the class.

SYNTAX OF DEFAULT CONSTRUCTOR:

```
class_name ()
{
}
Ex:
A ( )
{
}
```

EXAMPLE OF DEFAULT CONSTRUCTOR

In this example, we are creating the no-args constructor in the Bike class. It will be invoked at the time of object creation.

```
class Bike1
{
    Bike1()
    {
            System.out.println("Bike is created");
    }
    public static void main(String args[])
    {
            Bike1 b=new Bike1();
    }
}
```

JAVA PARAMETERIZED CONSTRUCTOR

A constructor that have parameters is known as parameterized constructor

```
class Student
{
    int x,y;
    Student(int i ,  int j)
    {
            x = i;
            y= j;
    }
    void display()
    {
            System.out.println( x+ “     ”+y);
    }
    public static void main(String args[])
    {
            Student s1 = new Student (10, 20);
            s1.display();
    }
}
```

WHY USE PARAMETERIZED CONSTRUCTOR?

Parameterized constructor is used to provide different values to the distinct objects.

EXAMPLE OF PARAMETERIZED CONSTRUCTOR

In this example, we have created the constructor of Student class that has two parameters. We can have any number of parameters in the constructor.

```
A(int a)
{
}
```

CONSTRUCTOR OVERLOADING IN JAVA

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

The below constructors are having the same name and same number of arguments but different types of arguments.

```
A(int a )              A(byte b)              A(long l)
{                      {                      {
}                      }                      }
```

EXAMPLE OF CONSTRUCTOR OVERLOADING

```
classConsDemo
{
inti,j;
ConsDemo()
{
        i=10; j=20;
}
ConsDemo( int a )
{
        i=a;
        j=20;
}
ConsDemo(int a int b)
{
        i=a;
        j=b;
}
void display()
{
        System.out.println(" i ="+i);
        System.out.println(" j ="+j);
}

public static void main( String args[])
{
        ConsDemo a1 = new ConsDemo();
        ConsDemo a2 = new ConsDemo(11);
        ConsDemo a3 = new ConsDemo(111,222);
        a1.display ();
        a2.display ();
        a3.display ();
}
}
```

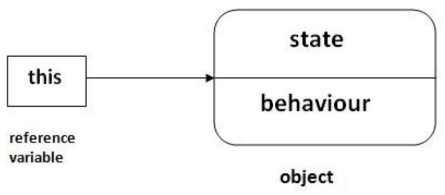
DEFAULT CONSTRUCTOR	PARAMETERIZED CONSTRUCTOR
This default constructor is useful to initialize all objects with same data	Parameterized constructor is useful to initialize each object with different data
It does not have any arguments	It will have one or more arguments
When data is not passed at the time of creating an objects , default constructor will be called	When data is passed at the time of creation of an object, parameterized constructor will be called.

CONSTRUCTOR	METHODS
The constructor is used to initially the instance variables of class	A method is used for any general purpose task like calculation
A constructor name should be always same as class name	The method name and class name can be same and different.
A constructor is called at the time of creating object	A method can be called after creating the object
A constructor is called only one once per object	A method can be called any number of times on the object
A constructor is called and executed automatically	A method is executed only when we want it.

THIS KEYWORD

In java "this" is a reference variable that refers to the current object. Whenever it is required to point an object from a functionality which is under execution then we use the "this keyword" .The job of this keyword is only to point. It always points to an object that is executing the block in which "this" keyword is present.

The first function of "this" is to call constructor from another constructor and second function is to avoid namespace conflicts between a method's or constructor's parameter list and its variables.



USAGE OF JAVA THIS KEYWORD

Here is given the 6 usage of java this keyword.

1. this keyword can be used to refer current class instance variable.
2. this() can be used to invoke current class constructor.
3. this keyword can be used to invoke current class method (implicitly)
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this keyword can also be used to return the current class instance

.

The “this” keyword can be used to refer current class instance variable.
If there is ambiguity between the instance variable and parameter, this keyword resolves the problem of ambiguity.

EAXMPLE

```
class A
{
        int i,j;
A( )
{
}
A ( int i, int j)
{
        i=i;      // it is confusing which "i" is assigning to which 'i'
        j=j;      //it is confusing, which 'j' is assigning to which 'j'
}
}
```

To avoid these conflicts, we can use this keyword as shown below.

```
class A
{
        int i,j;
A( )
{
}
A ( int i, int j)
{
        this.i=i;        // this.i refers to instance variables and i refers to method arguments
        this.j=j;        // this.j refers to instance variables and j refers to method arguments

}
}
```

The “this” keyword can be used to call a constructor from other constructor. The process of calling the constructors from other constructors is called CONSTRUCTOR CHAINING.

EXAMPLE:

```
class A()
{
        int i , j;
        A()
        {
                this(100);
        }
        A(int a )
        {
                this (a,200);
        }
        A(int a , int b)
        {
                i=a;
                j=b;
        }
}
```

The “this” keyword can be used inside non static methods to resolve the names conflict as shown below

```
class A()
{
        int i,j;
        void fun()
        {
        int i=100;
        System.out.println(i);  // prints the value of local variable ‘i’
        System.out.println(this.i);// prints the value of instance variable ‘i’
        }
}
```

<u>EXAMPLE FOR THIS KEYWORD</u>

```
classThisDemo
{
        inti, j;
        ThisDemo()
        {
                this(100);
        }
        ThisDemo(int a)
        {
                this(a,200);
        }
        ThisDemo(int i, int j)
        {
                this.i=i;
                this.j=j;
        }
        void display()
        {
                System.out.println(“i=”+i);
                System.out.println(“j=”+j);
        }
        public static void main(String args[])
        {
                ThisDemo a1=new ThisDemo();
                a1.display();
        }
}
```

METHOD OVERLOADING:

If a class has multiple methods by same name but different parameters, it is known as Method Overloading. Defining more than one functionality with the same name but with different arguments in the same class is known as static polymorphism. Why it is called as compile time because if there are multiple methods are overloaded, than which method has to be executed is decided at the time of compile time itself.

ADVANTAGE OF METHOD OVERLOADING?

Method overloading increases the readability of the program.

PROGRAM TO DEMONSTRATE METHOD OVERLOADING

```
Class DemoOver
{
        void test()
        {
                System.out.println(“no parameters”);
        }
        void test(int a , int b)
        {
                System.out.println(“a=”+a+ “b=” +b);
        }
        void test(int a ,float f)
        {
                System.out.println(“a=”+a+ “f=” +f);
        }
        double test(double d)
        {
                System.out.println(“d= “+d);
                return d;
        }
}
classOverLoad
{
        public static void mian(String args[])
        {
                OverLoadob = new OverLoad();
                ob.test();
                ob.test(10,20);
                ob.test(30,3.14f);
                double d2=ob.test(3.14);
        }
}
```

DIFFERENT WAYS TO OVERLOAD THE METHOD

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type.

In java, Method Overloading is not possible by changing the return type of the method.

1) EXAMPLE OF METHOD OVERLOADING BY CHANGING THE NO. OF ARGUMENTS

In this example, we have created two overloaded methods, first sum method performs addition of two numbers and second sum method performs addition of three numbers.

```
class Calculation
{
        void sum(int a, int b)
        {
                System.out.println(a+b);
        }
        void sum(int a, int b,int c)
        {
                System.out.println(a+b+c);
        }
        public static void main(String args[])
        {
                Calculation obj=new Calculation();
                obj.sum(10,10,10);
                obj.sum(20,20);
        }
}
```

2) EXAMPLE OF METHOD OVERLOADING BY CHANGING DATA TYPE OF ARGUMENT

In this example, we have created two overloaded methods that differs in data type. The first sum method receives two integer arguments and second sum method receives two double arguments.

```
class Calculation
{
	void sum(int a,int b)
	{
		System.out.println(a+b);
	}
	void sum(double a,double b)
	{
		System.out.println(a+b);
	}
	public static void main(String args[])
	{
		Calculation obj=new Calculation();
		obj.sum(10.5,10.5);
		obj.sum(20,20);

	}
}
```

JAVA'S AUTOMATIC TYPE CONVERSION IN OVERLOADING:

When there is no direct match between a set of arguments and a set of parameters, the method with the closet matching set parameters is used if the arguments can be automatically converted to the type of the parameter.

```
	void test(short s){ }
	void test(float f){ }
	void test(double d){ }
```

When we call test (6), first JVM searches for test method with int as an arguments, since it is found, JVM considers the priority concepts. The priority is given to float and hence test (float f) method will be executed.

JAVA STATIC KEYWORD

A class member must be accessed through an object of its class, but it is possible to create a member that can be used without creating an object. To create such members we need to use the keyword "static".
The static keywords can be used in 3 scenarios:

1. Static variable (also known as class variable)
2. Static method (also known as class method)
3. Static block

1) JAVA STATIC VARIABLE

If you declare any variable as static, it is known static variable.

- ➔ The instance variables are non-static and its part of an object. But static variable are special type of variable that are not associated with an object, they are associated with class.
- ➔ The static variables are also called as class variables.
- ➔ The static variables can be accessed without an object.
- ➔ The static variables are common to all objects. If we declare one static variable then only one copy of the variable will be available in memory. If we create 10 objects, then all objects will share the single copy of static variable. This means static variable are common to all the objects.
- ➔ Static variables are initialized only once, at the start of the execution .These variables will be initialized first, before execution of first statement of main () methods.
- ➔ The static variable will not be created in heap area. It is created in context area or methods area.
- ➔ A static variable can be accessed directly by the class name and doesn't need any object.

```
Syntax: class_name. variable name
```

Ex: Staticdemo.z;

DECLARING STATIC VARIABLES:

```
class StaticDemo
{
        intx,y;
        static int z;
}
```

EXAMPLE PROGRAM FOR STATIC VARIABLE:

```
class StaticDemo
{
        intx,y;
        static int z=10;
        public static void main (String args [])
        {
                StaticDemo ob1=new StaticDemo();
                StaticDemo ob2=new StaticDemo();
                System.out.println (ob1.x);
                System.out.println (ob2.y);
                System.out.println (StaticDemo.z);
        }
}
```

ADVANTAGE OF STATIC VARIABLE

It makes your program memory efficient (i.e. it saves memory).

2) JAVA STATIC METHOD.

- ➔ If you apply static keyword with any method, it is known as static method.
- ➔ A static method belongs to the class rather than object of a class.
- ➔ The most common example of static method is main (). The main () is declared as static because it must be called by the operating system when our program begins.
- ➔ A static method can be invoked directly by the class name and doesn't need any objects.

```
Syntax: class_name .method_name(arguments)
        StaticDemo .method ( );
```

- ➔ Static method can access only static data. It cannot access non - static data (instance members).
- ➔ A static method can call only other static methods and cannot call a non - static method from it.

DECLARING A STATIC METHODS.

```
class StaticDemo
{
        int x, y;
        static int z;
        voidstaticMethod( )
        {
                System.out.println( z);
        }
}
```

A static method can access only static data. It cannot access non static data

```
class StaticDemo
{
        int x ,y ;
        static int z;
        voidstaticMethod( )
        {
                System.out.println(x); // x is non static and can't be accessed in static method.
                System.out.println(y); // y is non static and can't be accessed in static method
                System.out.println (z); // z is static. It is allowed here
        }
}
```

A static method can call only other static methods and cannot call a non – static method from it.

```
class StaticDemo
{
        Voidstaticmethod( )
        {
                method2( );  // allowed
                method3( );  // not allowed , method3( ) is not static
        }
        void static method2( )
        {
        }
        void method3( )
        {
        }
}
```

3) JAVA STATIC BLOCK

- Is used to initialize the static data member.
- It is executed before main method at the time of class loading.
- A static block is executed when the class is first loaded. It is executed before the class can be used for any other purpose. We can also have non static blocks in the class.

EXAMPLE OF STATIC BLOCK

```
class A2
{
        static
        {
        System.out.println("static block is invoked");
        }
        public static void main(String args[])
        {
        System.out.println("Hello main");
        }
}
Output: static block is invoked
          Hello main
```

4) JAVA NON STATIC METHOD:

- ➔ If we have many constructors in a class and if every constructor has some common statements. Then instead of repeating those statements in each constructor, we place those statements in a non static block. In this way we can avoid duplication of the code.

EXAMPLE OF NON STATIC BLOCK

```
class A2
{
        A2()
        {
                System.out.println(“Inside the constructor”);
        }

        {
        System.out.println("non static block is invoked");
        }

        public static void main(String args[])
        {
        System.out.println("Hello main");
        }
}
```

NESTING OF METHODS:

Nesting of methods is calling methods inside another method.

```
ClassmethodDemo
{
        void method1()
        {
                method2();
        }
        void method2( )
        {
                method3( ) ;
        }
        void method3( )
        {
                System.out.println( “ Hello Java”);
        }
}
```

WHEN JVM LOADS STATIC ELEMENTS OF THE CLASS:

Whenever we ask JVM to execute a .class file , JVM opens the .class file and searches for the static elements in the .class file and then loads all the static elements of the .class file into one specific memory location inside the RAM . This memory is called as method area or context area. This happens before executing any statements in the main( ) method.

USE OF STATIC BLOCK	USE OF NON STATIC BLOCK
In the real scenario. Static blocks are used to provide	If we have constructors in a class and if every

information regarding the project, before actually the project is executed.	constructor has some common statements. Then instead of repeating those statements in each constructor, we place those statements in a non static block.

STATIC VARIABLE	INSTANCE VARIABLE
The static variables are not part of an object	Instance variables are part of object
The static variable are initialized only once during start up of the execution. All the objects will share the same copy of static variables.	Every object will have its own copies of instance variables.

CHAPTER 7:

1. How to define a class and what are the rules of defining class.
2. Define an object? How to create an object
3. What is difference between method and constructor(2010)
4. What is the difference between overloading and overriding?
5. How to create an object. What happens when we create an object(2016)
6. Explain inner class
7. What is the use of this keyword (2010).
8. Differentiate between class and an interface.(2012)
9. What is static class(2012)
10. Define class and write down its syntax(2015)
11. Write a few points about default constructor (2016).
12. Write the general syntax for defining syntax
13. What class constructor
14. What are instance variable? What is the difference between class variable and instance variable
15. What do you mean by attribute and behavior? Give an example
16. Which operator is used to create an object
17. In which memory the objects and class variables are created
18. What is reference variable? Give an example
19. Explain how to access class members using an object
20. What JVM does when an object is created
21. Which operator is used to access class members
22. What is dynamic loading
23. What is constructor
24. What are the types of constructor? Explain with an example
25. What is constructor overloading
26. What is the difference between zero argument constructor and parameterized constructor
27. What are the rules for defining constructor
28. When will java provides default constructor explain with example
29. What is this operator what are the uses of this operator
30. How to call constructor of the same class
31. What are the rules of using this constructor
32. What is method overloading? explain with example
33. What is polymorphism? explain the classification of polymorphism
34. What is static or compile time polymorphism. how it is achieved
35. What are the rules of method overloading
36. What is the use of static keyword in java
37. Explain the use of static variables

38. Explain the use of static methods
39. How static variable and methods are accessed
40. How JVM loads static variables of the class
41. Where the memory is allocated for the static members
42. What are static and non static blocks? Give example
43. What is the use of static and non static block

5 marks:

1. What is constructor overloading? Write a program to demonstrate it.(2010)
2. What is the difference between constructor and method(2016)
3. Explain methods supported in java object class
4. What is static class
5. Explain vector class with an example? How is it different from array.
6. Explain method overloading and constructor overloading with an example.(2010)

CHAPTER 8:

INHERITANCE IN JAVA

Inheritance in java is a process by which objects of one class acquire the properties of objects of another class. The class that does the inheriting is called a subclass. Inheritance is done by using the keyword extends.
The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.
Inheritance represents the IS-A relationship, also known as parent-child relationship.

WHY USE INHERITANCE IN JAVA

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

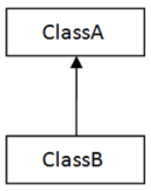
SYNTAX OF JAVA INHERITANCE / DEFINING SUBCLASS:

```
class subclass_name extends superclass_name
{
        body of the class: Methods and Varibales
}
```

The extends keyword indicates that you are making a new class that derives from an existing class.
In the terminology of Java, a class that is inherited is called a super class. The new class is called a subclass.
Understanding the simple example of inheritance
In the above example, Programmer object can access the field of own class as well as of Employee class i.e. code reusability.

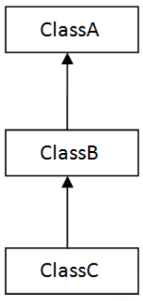
TYPES OF INHERITANCE IN JAVA

1. Single Inheritance: The one class extends another class.

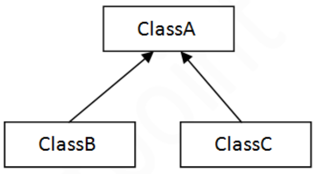


```
class Employee
{
float salary=40000;
}
class Programmer extends Employee
{
int bonus=10000;
public static void main(String args[])
{
Programmer p=new Programmer();
System.out.println("Programmer salary is:"+p.salary);
System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
 Programmer salary is:40000.0
 Bonus of programmer is:10000
```

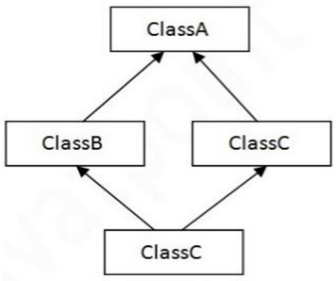
2. Multilevel Inheritance: One class extending another class as in a hierarchical structure is termed as multilevel inheritance.



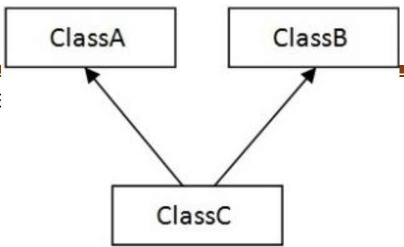
3. Hierarchical Inheritance: Many classes extending from single super class. One super class and many sub classes



4. Hybrid Inheritance:



5. Multiple Inheritances: The one class extends from more than one class.
The java supports only single, multilevel and hierarchical Inheritance. It does not support multiple



inheritance.
The multiple inheritances will be achieved in different way by using interfaces.

Inheritance Type	Example Code
Single Inheritance	Class A ; Class B extends A
Multilevel Inheritance	Class A ; class B extends A ; class C extends B
HierarchicalInheritance	Class A Class B extends A Class c extends A Class D extends A
Multiple Inheritance	Class A extends B,C // Not allowed

METHOD OVERRIDING IN JAVA

The process of overriding the method present in super class in subclass is called method overriding.
In other words, if subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

USAGE OF JAVA METHOD OVERRIDING

- Method overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method overriding is used for runtime polymorphism

RULES FOR JAVA METHOD OVERRIDING

1. The method must have same name as the method in the super class and type and order of the arguments must be same.

```
class A
{
        void shoot(int a)
        {
                // code
        }
}
class B extends A
{
        void shoot(int b)
        {
                // code
        }

}
```

2. The overriding method must have the same return type as the overridden method.

```
class A
{
    void shoot()
    {
        // code
    }
}
class B extends A
{
    void shoot()
    {
            // code
    }

}
```

3. The access modifier in subclass but they must be less restrictive than the original method in super class. private→default→protected→public

```
protected class A
{
    void shoot()
    {
            // code
    }
}
public class B extends A
{
    void shoot()
    {
            // code
    }

}
```

4. The overridden method may not throw any checked exception at all.
5. We cannot override a final method of super class.

EXAMPLE OF METHOD OVERRIDING

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method is same and there is IS-A relationship between the classes, so there is method overriding.

```
class Vehicle
{
void run()
{
        System.out.println("Vehicle is running");
}
}
class Bike2 extends Vehicle
{
void run()
{
        System.out.println("Bike is running safely");
}

public static void main(String args[])
{
    Bike2 obj = new Bike2();
    obj.run();
}
```

```
Output:Bike is running safely
```

CAN WE OVERRIDE STATIC METHOD?
No, static method cannot be overridden.

WHY WE CANNOT OVERRIDE STATIC METHOD?
Because static method is bound with class whereas instance method is bound with object. Static belongs to class area and instance belongs to heap area.

CAN WE OVERRIDE JAVA MAIN METHOD?
No, because main is a static method.

DIFFERENCE BETWEEN METHOD OVERLOADING AND METHOD OVERRIDING IN JAVA

METHOD OVERLOADING	METHOD OVERRIDING

Overloading deals with multiple methods in the same class with the same name but different method signature.
Class Myclass
{
Public void get (int a) {………}
Public void get (int a, int b) {………………….}
}
Both the above methods have same methods names but different methods signatures, which means the methods are overloaded.
	Overriding deals with two methods, one in the parent class and other one in the child class and has the same name as signature.
	Class Mydemo
	{
	Public void get (int a) {……………..}
	}
	Class Myclass extends Mydemo
	{
	Public void get (int a){………………..}
	}
	Both the above methods have the same methods names and signatures but the method in the sub class Myclass overrides the method in the superclass Mydemo

Signature has to be different. Just a difference in return type is not enough	Signature has to be the same including the return type.
Any access modifiers can be used	Overrides methods cannot be more restrictive then the overridden methods.
The method exception list may vary freely	Overridden methods cannot throw more checked exception then the overridden methods
The which to be called decided at the time of compilation	The which to be called will be decided at the time of runtime based on type of the object.
Methods can be static and non-static	The static method does not participate in overriding since they are resolved at compile time based on the type of reference variable.
There's no limit on number of overloaded methods a class can have	Each parent class method may be overridden at most once in any sub class. That is we cannot have two identical methods in the same class
Method overloading is the example of compile time polymorphism.	Method overriding is the example of run time polymorphism.
Method overloading is performed within class.	Method overriding occurs in two classes that have IS-A (inheritance) relationship.

POLYMORPHISM:

Defining more than one functionality with the same name is nothing but a polymorphism.

TYPES OF POLYMORPHISM:

1. Compile time polymorphism / static polymorphism
2. Runtime time polymorphism / dynamic polymorphism

WHAT IS COMPILE TIME POLYMORPHISM?

Defining more than one functionality with the same name but with different arguments in the same class is known as static polymorphism.
Static polymorphism is achieved by method overloading.

WHAT IS DYNAMIC POLYMORPHISM?

The dynamic polymorphism is the biding of method call to the method body will happen at runtime.
Dynamic polymorphism is achieved by using method overriding.

SUPER KEYWORD IN JAVA

The super keyword in java is a reference variable that is used to refer parent class object.
Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.

USAGE OF JAVA SUPER KEYWORD

1. super is used to refer immediate parent class instance variable. If the same variables are defined in both super class and subclass.
2. super() is used to invoke immediate parent class constructor.
3. super is used to invoke immediate parent class method. If the same method is present in super class and subclass

RULES FOR USING SUPER KEYWORD:

1. The super should be the first statement inside the constructor.
2. The super and this cannot be used together.
3. The super cannot be used inside the static method.
4. If we don’t write super then java provides the super ( ) by default.

1) SUPER IS USED TO REFER IMMEDIATE PARENT CLASS INSTANCE VARIABLE.

```
//example of super keyword

class Vehicle
{
    int speed=50;
}
class Bike4 extends Vehicle
{
    int speed=100;

    void display()
    {
    System.out.println (super. speed); //will print speed of Vehicle now
    }
    public static void main(String args[])
    {
    Bike4 b=new Bike4();
        b.display();
    }
}
Output:50
```

2) SUPER IS USED TO INVOKE PARENT CLASS CONSTRUCTOR.

```
class Vehicle
{
Vehicle()
{
System.out.println("Vehicle is created");
}
}
class Bike5 extends Vehicle
{
Bike5( )
{
super();//will invoke parent class constructor
System.out.println ("Bike is created");
}
public static void main(String args[])
{
Bike5 b=new Bike5( );
}
}
Output:   Vehicle is created
           Bike is created
```

The super keyword can also be used to invoke the parent class constructor as given below:

super() is added in each class constructor automatically by compiler.

As we know well that default constructor is provided by compiler automatically but it also adds super() for the first statement. If you are creating your own constructor and you don't have either this() or super() as the first statement, compiler will provide super() as the first statement of the constructor.

Another example of super keyword where super() is provided by the compiler implicitly.

```
class Vehicle
{
Vehicle( )
{
        System.out.println("Vehicle is created");
}
}

class Bike6 extends Vehicle
{
int speed;
Bike6(int speed)
{
this.speed=speed;
System.out.println(speed);
}
public static void main(String args[])
{
        Bike6 b=new Bike6(10);
}
}
Output: Vehicle is created
      10
```

3) SUPER CAN BE USED TO INVOKE PARENT CLASS METHOD

The super keyword can also be used to invoke parent class method. It should be used in case subclass contains the same method as parent class as in the example given below:

```
class Person
{
void message( )
{
System.out.println("welcome");
}
}

class Student16 extends Person
{
void message( )
{
System.out.println("welcome to java");
}

void display( )
{
message();//will invoke current class message() method
super.message();//will invoke parent class message() method
}
public static void main(String args[])
{
Student16 s=new Student16();
s.display();
}
}
Output:welcome to java
welcome
```

In the above example Student and Person both classes have message() method if we call message() method from Student class, it will call the message () method of Student class not of Person class because priority is given to local.

In case there is no method in subclass as parent, there is no need to use super. In the example given below message() method is invoked from Student class but Student class does not have message() method, so you can directly call message() method.

PROGRAM IN CASE SUPER IS NOT REQUIRED

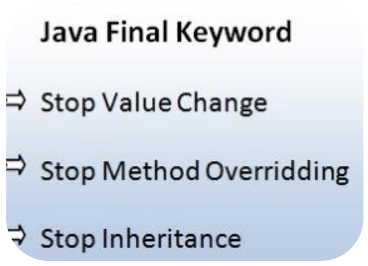
```
class Person
{
void message()
{
System.out.println("welcome");
}
}
class Student17 extends Person
{
void display()
{
message();//will invoke parent class message() method
}

public static void main(String args[])
{
Student17 s=new Student17();
s.display();
}
}
Output:welcome
```

THIS	**SUPER**
It refers to current object	It refers to super class object
It is used to call the constructors of same class	It is used to call the constructor of super class
It is used to differentiate between instance variable and local variable of same class	It is used to differentiate between instance variables of subclass and super class.

FINAL KEYWORD IN JAVA

The final keyword in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class



1) JAVA FINAL VARIABLE

Declaring a variable with the final keyword makes that variable as constant. Once we assign the value to final variables, we cannot change the value of final variable (It will be constant).
If a reference variable marked as final, you cannot reassigned to refer to a different object. The data within an object can be modifies but the reference variable cannot be changed.

EXAMPLE OF FINAL VARIABLE

There is a final variable speed limit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
class Bike9
{
final int speedlimit=90;  //final variable
void run( )
{
speedlimit=400;  // error , speedlimit is final
}
public static void main(String args[])
{
Bike9 obj=new  Bike9( );
obj.run();
final Bike ob2=new Bike( );
Bike ob3=new Bike( );
ob2=ob3; // this is not allowed. ob2 is final reference variable.
}
}//end of class
```

output: Compile time error because speed limit is final and cannot be changed.

2) JAVA FINAL METHOD:

The methods in the class can be declared as final. A method can be declared final to prevent subclasses from overriding it. If you make any method as final, you cannot override it.
Example of final method

```
class Bike
{
final void run( )
{
System.out.println ("running");
}
}
class Honda extends Bike
{
void run( )
{                                                    //cannot override void run() is final in super
class
System.out.println("running safely with 100kmph");
}
public static void main(String args[])
{
Honda honda= new Honda();
honda .run ();  // gives compile time error
}
}
```

output: Compile time error because run() is final and cannot be overridden.

3) JAVA FINAL CLASS

The keyword final can also be used for classes. If we declare a class as final, then that class cannot be inherited. Int final classes cannot be sub classed. If you make any class as final, you cannot extend it. The java has lot of built in final classes. Example: String and Math are final classes.
Example: String and Math are final classes.

Example of final class

```
final class Bike
{
}

class Honda1 extends Bike // error class Bike is final
{
void run()
{
System.out.println("running safely with 100kmph");
}
public static void main(String args[])
{
Honda1 honda= new Honda();
honda.run();
}
}
```

Output: Compile Time Error because class Bike is final and cannot be inherited.

IS FINAL METHOD INHERITED?

Yes, final method is inherited but you cannot override it. For Example:

```
class Bike
{
final void run( )
{
System.out.println ("running...");
}
}
class Honda2 extends Bike
{
public static void main(String args[])
{
new Honda2( ).run();
}
}
```

Output: running...

WHAT IS BLANK OR UNINITIALIZED FINAL VARIABLE?

A final variable that is not initialized at the time of declaration is known as blank final variable.
If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful. For example PAN CARD number of an employee.
It can be initialized only in constructor.

EXAMPLE OF BLANK FINAL VARIABLE

```
class Student{
int id;
String name;
final String PAN_CARD_NUMBER;
...
}
```

CAN WE INITIALIZE BLANK FINAL VARIABLE?

Yes, but only in constructor. For example:

```
class Bike10
{
final int speedlimit;//blank final variable
Bike10( )
{
speedlimit=70;
System.out.println (speedlimit);
}
public static void main(String args[])
{
new Bike10();
}
}
Output:70
```

STATIC BLANK FINAL VARIABLE

A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

Example of static blank final variable

```
class A
{
static final int data;//static blank final variable
static
{
data=50;
}
public static void main(String args[])
{
System.out.println (A.data);
}
}
```

WHAT IS FINAL PARAMETER?

```
class Bike
{
 int cube(final int n)
{
  n=n+2;//can't be changed as n is final
  n*n*n;
 }
 public static void main(String args[]){
  Bike b=new Bike11();
  b.cube(5);
 }
}
Output: Compile Time Error
```

If you declare any parameter as final, you cannot change the value of it.

CAN WE DECLARE A CONSTRUCTOR FINAL?

No, because constructor is never inherited.

ABSTRACT CLASSES AND METHODS

ABSTRACT CLASS IN JAVA

A class that is declared with abstract keyword and a class which contains 0 or more abstract method , is known as abstract class in java. It can have abstract method (method without body) and non-abstract methods (method with body).

ABSTRACTION IN JAVA

Abstraction is a process of hiding the implementation details and showing only functionality to the user. Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.
Abstraction lets you focus on what the object does instead of how it does it. Ways to achieve Abstraction There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

ABSTRACT CLASS IN JAVA

A class that is declared as abstract is known as abstract class. It needs to be extended and its method implemented. It cannot be instantiated.
If you are extending any abstract class that have abstract method, you must either provide the implementation of the method or make this class abstract.
If there is any abstract method in a class, that class must be abstract.
Abstract class can also have concrete methods or normal methods besides the abstract methods.

EXAMPLE ABSTRACT CLASS

```
abstract class A{}
```

ABSTRACT METHOD

A method that is declared as abstract and does not have implementation is known as abstract method.

EXAMPLE ABSTRACT METHOD

```
abstract void printStatus();//no body and abstract
```

EXAMPLE OF ABSTRACT CLASS THAT HAS ABSTRACT METHOD

In this example, Bike the abstract class that contains only one abstract method run. It implementation is provided by the Honda class.

```
abstract class Bike
{
abstract void run();
}
```

```
class Honda extends Bike
{
void run()
{
System.out.println("running safely..");
}

public static void main(String args[])
{
Bike obj = new Honda();
obj.run();
}
}
running safely
```

ABSTRACT CLASS HAVING CONSTRUCTOR, DATA MEMBER, METHODS ETC.

An abstract class can have data member, abstract method, method body, constructor and even main () method.

File: TestAbstraction2.java

```
//example of abstract class that have method body
abstract class Bike
{
Bike()
{
System.out.println("bike is created");
}
abstract void run();
void changeGear()
{
System.out.println("gear changed");
}
}
class Honda extends Bike
{
void run()
{
System.out.println("running safely..");
}
}
class TestAbstraction2
{
public static void main(String args[])
{
Bike obj = new Honda();
obj.run();
obj.changeGear();
}
}
bike is created
running safely..
gear changed
```

JAVA GARBAGE COLLECTION

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.orThe mechanism Java uses to automatically release the allocated memory is called as the automatic garbage collection.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
- It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

HOW CAN AN OBJECT BE UNREFERENCED?

There are many ways:

- By nulling the reference
- By assigning a reference to another
- By annonymous object etc.

1) By nulling a reference:

1. Employee e=new Employee();
2. e=null;

2) By assigning a reference to another:

1. Employee e1=new Employee();
2. Employee e2=new Employee();
3. e1=e2;//now the first object referred by e1 is available for garbage collection

3) By annonymous object:

1. new Employee();

FINALIZE() METHOD

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

1. protected void finalize(){}

Ex:
```
class Dog
{
        protected void finalize( ) throws Throwable
        {
                System.out.println("Garbage collecting the DOG object….");
        }
}
```

Note: The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

GC() METHOD

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

1. public static void gc(){}

Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.

SIMPLE EXAMPLE OF GARBAGE COLLECTION IN JAVA

```
public class TestGarbage1
{
public void finalize()
{
System.out.println("object is garbage collected");
}
public static void main(String args[])
{
TestGarbage1 s1=new TestGarbage1();
TestGarbage1 s2=new TestGarbage1();
s1=null;
s2=null;
System.gc();
}
}
```

```
object is garbage collected
object is garbage collected
```

CHAPTER 8:

1. What is garbage collection?(2010)
2. What is inheritance? List types of inheritance
3. Why super keyword is used in inheritance
4. What is abstract class?(2011)
5. When do we declare a method final?
6. What are the uses of final keyword
7. Why multiple inheritance is not supported in java
8. What is use of extend keyword in java
9. How to declare a final method(2011)
10. What is the use of “this” and “super” keyword(2015)
11. What is the use of inheritance
12. What are the types of inheritance? Give an example
13. Which keyword is used for inheritance
14. What is the general syntax of creating subclass
15. What happens when an object of subclass is created
16. What happens when an object is created using new operator
17. What is method overriding
18. What is dynamic polymorphism? How it is achieved in java
19. What is the difference between static and dynamic polymorphism in java
20. What are the rules of method overriding
21. How to call constructor of super class from subclass
22. How to call methods of super class using super. Explain with an example
23. What are the rules of using super keyword
24. What is the difference between super an d this keyword
25. What is final variable
26. What is final methods
27. What is final class
28. What is abstract method? Explain with an example
29. What is abstract class how to create an object on abstract class

30. What is the use of abstract classes
31. What is garbage collection how it is achieved in java
32. How to request for garbage collection
33. Which object are eligible for garbage collection
34. What is the finalize() method
35. What JVM does for garbage collection

5 marks

1. What is the use of super keyword? With example(2010)
2. Write a note on inheritance(2015)
3. Write a program to demonstrate inheritance
4. Explain method overriding with an example(2015)
5. Explain overriding with a program.(2012)
6. What are the uses of final keyword
7. Explain with java program to demonstrate single level inheritance (2010)
8. What are final variable, method and classes with a suitable example(2010)
9. Explain the use of super keyword in java inheritance concept with syntax(2011)
10. Define abstract class. Discuss its importance.(2012)
11. Demonstrate "this" keyword with simple example(2016)
12.

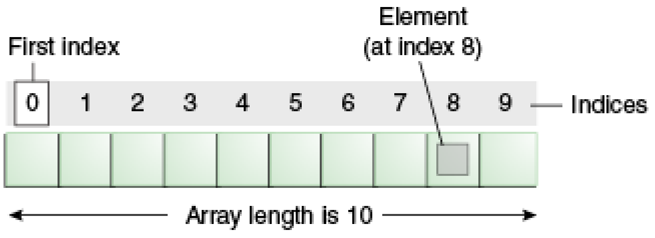
CHAPTER 9:

JAVA ARRAY

An array is an object which represents a fixed sixed ordered grouping of data elements of the same type. Java array is an object that contains elements of similar data type. It is a data structure where we store similar elements. We can store only fixed set of elements in a java array.

Array in java is index based, first element of the array is stored at 0 index.
To create an array in java we use three steps:

1. Declare a variable to hold array.
2. Create a new array object and assign it to the array variable.
3. Initialize or assign values to the array elements.



ADVANTAGE OF JAVA ARRAY

- Code Optimization: It makes the code optimized, we can retrieve or sort the data easily.
- Random access: We can get any data located at any index position.

DISADVANTAGE OF JAVA ARRAY

- Size Limit: We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java.

TYPES OF ARRAY IN JAVA

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

CREATING AN ARRAY:

Like other objects, array objects also need to be declared at the compile time. The actual array construction with a new keyword involves the memory allocation and the array object creation at runtime.

```
int[ ] arr; // array declaration.
arr = new int [5]; // array construction at runtime.
```

At compile time, the reference variable arr of type int [ ] is created. At runtime, java will create an array object of integers with five elements on memory heap and assign its reference to the variable arr.
We can also write the array declaration and creation on the same line. But the array will still be constructed at runtime.

```
int[ ] arr = new int[5];
```

The pair of empty square ([ ]) brackets in the declaration of the reference variable arr indicates that the arr does not need to know about the size of the array to which it may refer.

SINGLE DIMENSIONAL ARRAY IN JAVA

Syntax to Declare an Array in java

```
datatype[ ] arrayname; (or)
datatype arrayName[ ];
ex: int arr [ ] ; or int[ ] arr;
```

INSTANTIATION OF AN ARRAY IN JAVA

```
arrayRefVar=new datatype[size];
    ex: int[ ] arr = new int[5];
        arr[0]= 10;
        arr[1]= 20;
        arr[2]= 30;
        arr[3]= 40;
        arr[4]= 50;
```

INITIALIZING ARRAY AT THE TIME OF DECLARATION:

Array can also be created with an array initialize without using new operator.
Ex: int [ ] numbers={ 10,20,30,40,50 };

When using the initialize, an array must be declared, constructed and explicitly initialized at the same time.

```
Ex: int[ ] numbers;      //array declaration
numbers = {10,20,30,40,50};// error
numbers = new int[ ] {10,20,30,40,50}; // ok. This statement is called as anonymous array.
```

Example of single dimensional java array

```
class Testarray
{
```

```
          public static void main(String args[])
          {
                    int a[]=new int[5];//declaration and instantiation
                    a[0]=10;//initialization
                    a[1]=20;
                    a[2]=70;
                    a[3]=40;
                    a[4]=50;
                    //printing array
                    for(int i=0;i<a.length;i++)//length is the property of array
                    System.out.println(a[i]);
          }
}
class Testarray1
{
          public static void main(String args[])
          {
                    int a[ ]={33,3,4,5};//declaration, instantiation and initialization
                    //printing array
                    for(int i=0;i<a.length;i++)//length is the property of array
                    System.out.println(a[i]);
          }
}
```

SYNTAX OF ANONYMOUS ARRAY:

new<type>[ ] { <list of value>};
Ex: new int [ ] {1, 2, 3}; // creates a nameless array and initializes it. Here neither the name of the array nor the size is specified.

ARRAY OF OBJECTS REFERENCE:

An array of object reference is created by specifying the object type and the size of the array.

Ex: an array of Date objects is declared and constructed as
Date[ ] birth=new Date[5];

An array of five Date object references will be created with each elements initialized to null . the birth array is declared , constructed and initialized in one line, only the declaration of birth will be done at compile time . The actual construction of the array object and initialization will be done only at runtime.

PASSING ARRAY TO METHOD IN JAVA

We can pass the java array to method so that we can reuse the same logic on any array.
Let's see the simple example to get minimum number of an array using method.

```
class Testarray2
{
          static void min(int arr[])
          {
                    int min=arr[0];
                    for(int i=1;i<arr.length;i++)
                    if(min>arr[i])
                    min=arr[i];
                    System.out.println(min);
```

```
        }
public static void main(String args[])
{
        int a[]={33,3,4,5};
        min(a);//passing array to method
}
}
```

MULTIDIMENSIONAL ARRAY IN JAVA

In such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array in java

1. dataType[][] arrayRefVar; (or)
2. dataType [][]arrayRefVar; (or)
3. dataType arrayRefVar[][]; (or)
4. dataType []arrayRefVar[];

Example to instantiate Multidimensional Array in java

1. int[][] arr=new int[3][3];//3 row and 3 column

Example to initialize Multidimensional Array in java

1. arr[0][0]=1;
2. arr[0][1]=2;
3. arr[0][2]=3;
4. arr[1][0]=4;
5. arr[1][1]=5;
6. arr[1][2]=6;
7. arr[2][0]=7;
8. arr[2][1]=8;
9. arr[2][2]=9;

EXAMPLE OF MULTIDIMENSIONAL JAVA ARRAY

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

```
class Testarray3
{
public static void main(String args[])
{
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
//printing 2D array
for(int i=0;i<3;i++)
{
for(int j=0;j<3;j++)
{
System.out.print(arr[i][j]+" ");
}
System.out.println();
}

}
}
```

ADDITION OF 2 MATRICES IN JAVA

Let's see a simple example that adds two matrices.

```
class Testarray5
{
public static void main(String args[])
{
//creating two matrices
int a[][]={{1,3,4},{3,4,5}};
int b[][]={{1,3,4},{3,4,5}};
//creating another matrix to store the sum of two matrices
int c[][]=new int[2][3];
//adding and printing addition of 2 matrices
for(int i=0;i<2;i++)
{
for(int j=0;j<3;j++)
{
c[i][j]=a[i][j]+b[i][j];
System.out.print(c[i][j]+" ");
}
System.out.println();//new line
}
}
}
```

What is anonymous array?

In java it is perfectly legal to create an anonymous array using the following syntax:

```
new < type>[]{<list of values>};
```

THE FOREACH LOOPS:

JDK 1.5 introduced a new for loop, known as foreach loop or enhanced for loop, which enables you to traverse the complete array sequentially without using an index variable.

Example:

The following code displays all the elements in the array

```
public class Simplearray
{
        public static void main(string args[])
        {
                int[] numbers={10,20,30,40,50};
                for(int a : numbers)
                {
                        System.out.println(a);
                }
        }
}
```

COMMAND LINE ARGUMENTS:

The command line arguments represent the values passed to main method. To receive and store the values, main ( ) method provides arguments called as String args[ ] .

The main( ) method must always have the following method signature.

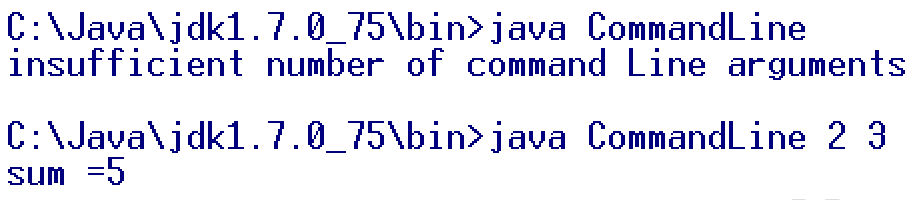
public static void main(String args[ ])

Here, args is the name of the array of strings that contains the list of arguments. The arguments that we pass in the command line are considered as Strings and hence we receive it in array of strings in the main ( ) method. If we pass the value 10, then it is considered as String not as primitive.

Example:

```
public class CommandLine
{
        public static void main(String args[ ] )
        {
                if(args.length>=2)
                {
                        int num1=Integer.parseInt(args[0]);
                        int num2=Integer.parseInt(args[1]);
                        int sum=num1+num2;
                        System.out.println(“sum =” + sum);
                }
                else
                {
                System.out.println (“insufficient number of command Line arguments”);
                }
        }
}
```

OUTPUT:



```
C:\Java\jdk1.7.0_75\bin>java CommandLine
insufficient number of command Line arguments

C:\Java\jdk1.7.0_75\bin>java CommandLine 2 3
sum =5
```

VECTOR:

A vector is a collection class in java.util.package and it holds collection of objects. The array are fixed size but vectors can grow dynamically. The array can contains elements of same type but vector can hold objects of any class.

CREATING A VECTOR:

Vector v=new Vector ( );

THREE WAYS TO CREATE VECTOR CLASS OBJECT:

METHOD 1:

Syntax: `Vector vec = new Vector ();`

It creates an empty Vector with the default initial capacity of 10. It means the Vector will be re-sized when the 11th elements needs to be inserted into the Vector. Note: By default vector doubles its size. i.e. In this case the Vector size would remain 10 till 10 insertions and once we try to insert the 11th element It would become 20 (double of default capacity 10).

METHOD 2:

Syntax: `Vector object= new Vector(int initialCapacity);`

```
Vector vec = new Vector(3);
```

It will create a Vector of initial capacity of 3.

METHOD 3:

Syntax: `Vector object= new vector(int initialcapacity, capacityIncrement)`

```
Vector vec= new Vector(4, 6)
```

Here we have provided two arguments. The initial capacity is 4 and capacityIncrement is 6. It means upon insertion of 5th element the size would be 10 (4+6) and on 11th insertion it would be 16(10+6).

COMPLETE EXAMPLE OF VECTOR IN JAVA:

```
import java.util.*;
public class VectorDemo
{
        public static void main(String args[])
        {
                Vector v=new Vector();
                v.add(“C++”);
                v.add(“Java”);
                v.add(“Java”);
                v.add(“J2EE”);
                System.out.println(“The vector Contents:”+v.toString( ));
                System.out.println(“The Element at the Last:”+v.lastElement( ));
                System.out.println(“The Element at 2nd Position:”+v.elementAt(2 ));

                v.insertElementAt(“VB,1);
                System.out.println(“The vector Contents:”+v.toString( ));

                v.removeElementAt(3);
                System.out.println(“The vector Contents:”+v.toString( ));

                v.setElementAt(“C++”,3);
                System.out.println(“The vector Contents:”+v.toString( ));
        }
}
```

IMPORTANT METHODS OF VECTOR CLASS:

1. voidaddElement(Object element): It inserts the element at the end of the Vector.
2. int capacity(): This method returns the current capacity of the vector.
3. int size(): It returns the current size of the vector.
4. voidsetSize(int size): It changes the existing size with the specified size.
5. boolean contains(Object element): This method checks whether the specified element is present in the Vector. If the element is been found it returns true else false.
6. booleancontainsAll(Collection c): It returns true if all the elements of collection c are present in the Vector.
7. Object elementAt(int index): It returns the element present at the specified location in Vector.
8. Object firstElement(): It is used for getting the first element of the vector.
9. Object lastElement(): Returns the last element of the array.
10. Object get(int index): Returns the element at the specified index.
11. booleanisEmpty(): This method returns true if Vector doesn’t have any element.
12. booleanremoveElement(Object element): Removes the specified element from vector.

13. voidsetElementAt(Object element, int index): It updates the element of specifed index with the given element.

JAVA STRING

WHAT IS STRING IN JAVA?

String is a sequence of characters. But in java, String is an object that represents a sequence of characters. String class is used to create string object.
How to create String object?
There are two ways to create String object:

1. By string literal
2. By new keyword

1) STRING LITERAL

Java String literal is created by using double quotes. For Example:

1. String s="welcome";

Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance is returned. If string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

1. String s1="Welcome";
2. String s2="Welcome";//will not create new instance

In the above example only one object will be created. Firstly JVM will not find any string object with the value "Welcome" in string constant pool, so it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create new object but will return the reference to the same instance.

Note: String objects are stored in a special memory area known as string constant pool.

Why java uses concept of string literal?

To make Java more memory efficient (because no new objects are created if it exists already in string constant pool).

2) BY NEW KEYWORD

Ex:

1. String s=new String("Welcome");//creates two objects and one reference variable
2. String str=new String("Hello");
   String str2=new String(str);

In such case, JVM will create a new string object in normal (non pool) heap memory and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in heap (non pool).

JAVA STRING EXAMPLE

```
public class StringExample
{
public static void main(String args[])
{
    String s2=new String("String);
    String s3=new String("example");
    System.out.println(s2);
    System.out.println(s3);
    }
}
```

OUTPUT:

```
Strings
example
```

JAVA STRING CLASS METHODS

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

WHY STRING ARE CALLED IMMUTABLE:

String objects are created by either using new operator or enclosing a sequence of characters in double quotes. The string object created by either way is immutable.

It simply means once we create a string object by specifying a sequence of characters, that object will always represent that same sequence of character throughout its life.

ex: String str= new String ("Hello");

Now the string object created has a sequence of 5 characters. We can never change this sequence of characters in this object. No method in java.lang.String class allows us to do so. Any method that may appear to be

changing the string actually create new string object.

for

JAVA STRING CHARAT

The java string charAt() method returns a char value at the given index number. The index number starts from 0.
The signature of string charAt() method is given below:

```
public char charAt(int index)
```

Java String charAt() method example

JAVA STRING COMPARETO:

```
public class CharAtExample
{
	public static void main(String args[])
	{
		String name="javatpoint";
		char ch=name.charAt(4);//returns the char value at the 4th index
		System.out.println(ch);
	}
}
output: t
```

The java string compareTo() method compares the given string with current string lexicographically. It returns positive number, negative number or 0.
If first string is greater than second string, it returns positive number (difference of character value). If first string is less than second string, it returns negative number and if first string is equal to second string, it returns 0.

1. s1 > s2 => positive number
2. s1 < s2 => negative number
3. s1 == s2 => 0

Signature

```
public int compareTo(String anotherString)
```

Java String compareTo() method example:

```
public class LastIndexOfExample
{
	public static void main(String args[])
	{
		String s1="hello";
		String s2="hello";
		String s3="meklo";
		String s4="hemlo";
		System.out.println(s1.compareTo(s2));
		System.out.println(s1.compareTo(s3));
		System.out.println(s1.compareTo(s4));
	}
}
Output:
0
-5
-1
```

JAVA STRING CONCAT

The java string concat() method combines specified string at the end of this string. It returns combined string. It is like appending another string.

THE SIGNATURE OF STRING CONCAT() METHOD IS GIVEN BELOW:

1. `public String concat(String anotherString)`

Java String concat() method example

```
public class ConcatExample
{
        public static void main(String args[])
        {
                String s1="java string";
                s1.concat("is immutable");
                System.out.println(s1);
                s1=s1.concat(" is immutable so assign it explicitly");
                System.out.println(s1);
        }
}
java string
java string is immutable so assign it explicitly
```

JAVA STRING EQUALS

The java string equals() method compares the two given strings based on the content of the string. If any character is not matched, it returns false. If all characters are matched, it returns true.
The String equals() method overrides the equals() method of Object class.

SIGNATURE

`public boolean equals(Object anotherObject)`

JAVA STRING EQUALS() METHOD EXAMPLE

```
public class EqualsExample
{
        public static void main(String args[])
        {
                String s1="javatpoint";
                String s2="javatpoint";
                String s3="JAVATPOINT";
                String s4="python";
                System.out.println(s1.equals(s2));//true because content and case is same
                System.out.println(s1.equals(s3));//false because case is not same
                System.out.println(s1.equals(s4));//false because content is not same
        }
}
true
false
false
```

JAVA STRING INDEXOF

The java string indexOf() method returns index of given character value or substring. If it is not found, it returns -1. The index counter starts from zero.

SIGNATURE

There are 4 types of indexOf method in java. The signature of indexOf methods are given below:

No.	Method	Description

1	int indexOf(int ch)	returns index position for the given char value
2	int indexOf(int ch, int fromIndex)	returns index position for the given char value and from index
3	int indexOf(String substring)	returns index position for the given substring
4	int indexOf(String substring, int fromIndex)	returns index position for the given substring and from index

PARAMETERS

ch: char value i.e. a single character e.g. 'a'
fromIndex: index position from where index of the char value or substring is retured
substring: substring to be searched in this string

JAVA STRING INDEXOF() METHOD EXAMPLE

```
public class IndexOfExample
{
        public static void main(String args[])
        {
                String s1="this is index of example";
                //passing substring
                int index1=s1.indexOf("is");//returns the index of is substring
                int index2=s1.indexOf("index");//returns the index of index substring
                System.out.println(index1+"  "+index2);//2 8

                //passing substring with from index
                int index3=s1.indexOf("is",4);//returns the index of is substring after 4th index
                System.out.println(index3);//5 i.e. the index of another is

                //passing char value
                int index4=s1.indexOf('s');//returns the index of s char value
                System.out.println(index4);//3
        }
}
2  8
5
3
```

JAVA STRING LENGTH

The java string length () method length of the string. It returns count of total number of characters. The length of java string is same as the Unicode code units of the string.

SIGNATURE

The signature of the string length () method is given below:

```
1.  public int length()
```

JAVA STRING LENGTH ( ) METHOD EXAMPLE

```
public class LengthExample
{
    public static void main(String args[])
    {
                String s1="javatpoint";
                String s2="python";
                System.out.println("string length is: "+s1.length());//10 is the length of javatpoint string
                System.out.println("string length is: "+s2.length());//6 is the length of python string
    }
}
string length is: 10
string length is: 6
```

JAVA STRING REPLACE

The java string replace() method returns a string replacing all the old char or CharSequence to new char or CharSequence.
Since JDK 1.5, a new replace() method is introduced, allowing you to replace a sequence of char values.

SIGNATURE

There are two type of replace methods in java string.

1. public String replace(char oldChar, char newChar)

The second replace method is added since JDK 1.5.

PARAMETERS

oldChar : old character
newChar : new character
target : target sequence of characters
replacement : replacement sequence of characters

Java String replace(char old, char new) method example

```
public class ReplaceExample1
{
	public static void main(String args[])
	{
		String s1="javatpoint is a very good website";
		String replaceString=s1.replace ('a','e');//replaces all occurrences of 'a' to 'e'
		System.out.println (replaceString);
	}
}
jevetpoint is e very good website
```

JAVA STRING SUBSTRING

The java string substring() method returns a part of the string.
We pass begin index and end index number position in the java substring method where start index is inclusive and end index is exclusive. In other words, start index starts from 0 whereas end index starts from 1.
There are two types of substring methods in java string.

SIGNATURE

1. public String substring(int startIndex)
2. and
3. public String substring(int startIndex, int endIndex)

If you don't specify endIndex, java substring() method will return all the characters from startIndex.

PARAMETERS

startIndex : starting index is inclusive
endIndex : ending index is exclusive

JAVA STRING SUBSTRING() METHOD EXAMPLE

```
public class SubstringExample
{
	public static void main(String args[])
	{
		String s1="javatpoint";
		System.out.println(s1.substring(2,4));//returns va
		System.out.println(s1.substring(2));//returns vatpoint
	}
}
 va
vatpoint
```

JAVA STRING TOLOWERCASE()

The java string toLowerCase() method returns the string in lowercase letter. In other words, it converts all characters of the string into lower case letter.
The toLowerCase() method works same as toLowerCase(Locale.getDefault()) method. It internally uses the default locale.

SIGNATURE

There are two variant of toLowerCase() method. The signature or syntax of string toLowerCase() method is given below:

1. public String toLowerCase()
2. public String toLowerCase(Locale locale)

The second method variant of toLowerCase(), converts all the characters into lowercase using the rules of given Locale.

JAVA STRING TOLOWERCASE() METHOD EXAMPLE

```
public class StringLowerExample
{
	public static void main(String args[])
	{
		String s1="JAVATPOINT HELLO stRIng";
		String s1lower=s1.toLowerCase();
		System.out.println(s1lower);
	}
}
Output:javatpoint hello string
```

TOUPPERCASE

The java string toUpperCase() method returns the string in uppercase letter. In other words, it converts all characters of the string into upper case letter.
The toUpperCase() method works same as toUpperCase(Locale.getDefault()) method. It internally uses the default locale.

SIGNATURE

There are two variant of toUpperCase() method. The signature or syntax of string toUpperCase() method is given below:

1. public String toUpperCase()
2. public String toUpperCase(Locale locale)

The second method variant of toUpperCase(), converts all the characters into uppercase using the rules of given Locale.

```
Java String toUpperCase() method example:

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

```
        {
                String s1="hello string";
                String s1upper=s1.toUpperCase();
                System.out.println(s1upper);
        }
}
Output:HELLO STRING
```

IMMUTABLE STRING IN JAVA

In java, string objects are immutable. Immutable simply means unmodifiable or unchangeable.
Once string object is created its data or state can't be changed but a new string object is created.
Let's try to understand the immutability concept by the example given below:

```
class Test
{
      public static void main(String args[])
     {
                String s="Sachin";
                s.concat(" Tendulkar");//concat() method appends the string at the end
                System.out.println(s);//will print Sachin because strings are immutable objects
        }
}
Output:Sachin
```

if we explicitely assign it to the reference variable, it will refer to "SachinTendulkar" object.For example:

```
class Testimmutablestring1
{
         public static void main(String args[])
        {
                String s="Sachin";
                 s=s.concat(" Tendulkar");
                 System.out.println(s);
         }
}
Output:Sachin Tendulkar
```

In such case, s points to the "Sachin Tendulkar". Please notice that still sachin object is not modified.

WHY STRING OBJECTS ARE IMMUTABLE IN JAVA?

Because java uses the concept of string literal. Suppose there are 5 reference variables, allrefers to one object "sachin".If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.

JAVA STRINGBUFFER CLASS

Java StringBuffer class is used to created mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.
Note: Java StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order.
Important Constructors of StringBuffer class

1. StringBuffer(): creates an empty string buffer with the initial capacity of 16.
2. StringBuffer(String str): creates a string buffer with the specified string.
3. StringBuffer(int capacity): creates an empty string buffer with the specified capacity as length.

WHAT IS MUTABLE STRING

A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

1) STRINGBUFFER APPEND() METHOD

The append() method concatenates the given argument with this string.

```
class A
{
	public static void main(String args[])
	{
		StringBuffer sb=new StringBuffer("Hello ");
		sb.append("Java");//now original string is changed
		System.out.println(sb);//prints Hello Java
	}
}
```

2) STRINGBUFFER INSERT() METHOD

The insert() method inserts the given string with this string at the given position.

```
class A
{
	public static void main(String args[])
	{
		StringBuffer sb=new StringBuffer("Hello ");
		sb.insert(1,"Java");//now original string is changed
		System.out.println(sb);//prints HJavaello
	}
}
```

3) STRINGBUFFER REPLACE() METHOD

The replace() method replaces the given string from the specified beginIndex and endIndex.

```
class A
{
	public static void main(String args[])
	{
		StringBuffer sb=new StringBuffer("Hello");
		sb.replace(1,3,"Java");
		System.out.println(sb);//prints HJavalo
	}
}
```

4) STRINGBUFFER DELETE() METHOD

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

```
class A
{
	public static void main(String args[])
	{
		StringBuffer sb=new StringBuffer("Hello");
		sb.delete(1,3);
		System.out.println(sb);//prints Hlo
	}
}
```

5) STRINGBUFFER REVERSE() METHOD

The reverse() method of StringBuilder class reverses the current string.

```
class A
{
        public static void main(String args[])
        {
                StringBuffer sb=new StringBuffer("Hello");
                sb.reverse();
                System.out.println(sb);//prints olleH
    }
}
```

DIFFERENCE BETWEEN STRING AND STRINGBUFFER

There are many differences between String and StringBuffer. A list of differences between String and StringBuffer are given below:

No.	String	StringBuffer
1)	String class is immutable.	StringBuffer class is mutable.
2)	String is slow and consumes more memory when you concat too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when you cancat strings.
3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.

JAVA TOSTRING() METHOD

If you want to represent any object as a string, toString() method comes into existence.
The toString() method returns the string representation of the object.
If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object etc. depends on your implementation.
Advantage of Java toString() method
By overriding the toString() method of the Object class, we can return values of the object, so we don't need to write much code.

EXAMPLE OF JAVA TOSTRING() METHOD

```
class Student
{
         int rollno;
        String name;
        String city;
        Student(int rollno, String name, String city)
        {
                 this.rollno=rollno;
                this.name=name;
                 this.city=city;
        }
     public String toString()
    {
        //overriding the toString() method
        return rollno+" "+name+" "+city;
    }
    public static void main(String args[])
   {
               Student s1=new Student(101,"Raj","lucknow");
               Student s2=new Student(102,"Vijay","ghaziabad");
               System.out.println(s1);//compiler writes here s1.toString()
               System.out.println(s2);//compiler writes here s2.toString()
      }
}
Output:101 Raj lucknow
102 Vijay ghaziabad
```

CHAPTER 9

1. What is vector?(2011)(2012)
2. How to get primitive values for wrapper objects
3. Why strings are called immutable? Mention any two string operation.
4. What is the difference between string and string buffer?(2015)
5. What is the default capacity of string buffer class?
6. How to declare and initialize array?(2010)
7. What is an array? How it is created in java?
8. How string different from string buffer
9. List out various methods of string class
10. What is string buffer class(2011)(2012)
11. Explain command line arguments(2012)(2015)
12. Explain a) character extraction b)string comparison c)string concatenation.(2012)
13. What are wrapper classes?(2012)
14. What are the advantages of vector over arrays(2015)
15. How to find the size of an array?
16. How to access array elements
17. What is an anonymous array?
18. How to declare array of object references?
19. How to pass array to methods
20. How to create a multidimensional array
21. How to declare initialize and access multidimensional array
22. What are regular and non regular multidimensional array? give an example
23. What are command line arguments? How to use
24. What is string
25. How string are created
26. Explain concat() , substr(), indexOf(), equals() and length() methods of string class
27. What is string buffer class?
28. What is the use of vector class
29. How to add and remove elements in vector class
30. What are the important methods of vector class
31. How to create wrapper object
32. Explain difference ways of creating wrapper objects
33. How to get string objects from wrapper objects.

5 marks

1. Explain array of object reference

2. How to declare string and string buffer? What is the difference between them(2015)
3. Explain the various uses of wrapper classes
4. Write a short notes on wrapper classes and string buffer
5. Explain any 5 methods of vector classes(2010)
6. Differentiate between vectors and array(2010)
7. Explain any 7 methods of string class.(2011)(2015)
8. Illustrate array declaration and accessing and accessing data elements using an example.(2016)
9. Write a program to display all prime numbers between two limits using command line arguments.(2015)
10. Write a program to sort a list of elements in ascending order(2016)

CHAPTER 10:

INTERFACE IN JAVA

- An interface in java is a blueprint of a class. It is a programming construct which contains only public, abstract methods and public static final variable.
- The interface in java is a mechanism to achieve fully abstraction. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritances in Java.
- Java Interface also represents IS-A relationship.
- It cannot be instantiated just like abstract class.
- An interface contains any number of methods.
- An interface is not extended by a class, it is implemented by a class.
- An interface can extend multiple interface.

WHY USE JAVA INTERFACE?

There are mainly three reasons to use interface. They are given below.

- It is used to achieve fully abstraction.
- By interface, we can support the functionality of multiple inheritances.
- It can be used to achieve loose coupling.

The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.
In other words, Interface fields are public, static and final by default, and methods are public and abstract.

DEFINING AN INTERFACE:

```
interface interface_name
{
public static variables
public abstract methods
}
```

Here interface is a keyword and interface_name is any valid Java identifier. All the variable in the interface are public static final variable. Methods are public and abstract.

EXAMPLE:

```
interface Animal
{
public static final int age=10;
public abstract void eat( );
public abstract void travel( );
}
```

OR

```
interface Animal
{
int age=10;
void eat( );
void travel( );
}
```

The abstract and public are default. All methods in interface are by default public and abstract.

EXTENDING INTERFACE:

We can extend an interface from another interface just like we extend a java class from another java class. An interface can only be extended from another interface. We cannot extend an interface from a class. Class can be extended from only one class but an interface can be extended from multiple interfaces.

EXAMPLE 1:

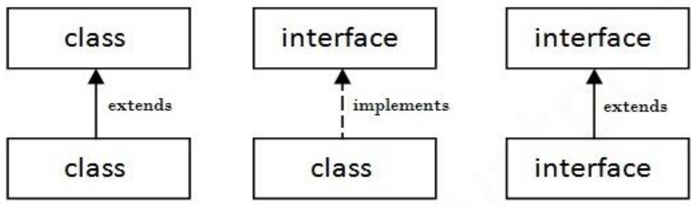
```
interface i1
{
        void method1( );
}
interface i2 extends i1
{
        void method2( );
}
```

EXAMPLE 2:

```
interface i1
{
void method1( );
}
interface i2 extends i1
{
void method2( );
}
interface i3 extends i1, i2
{
void method3( );
}
```

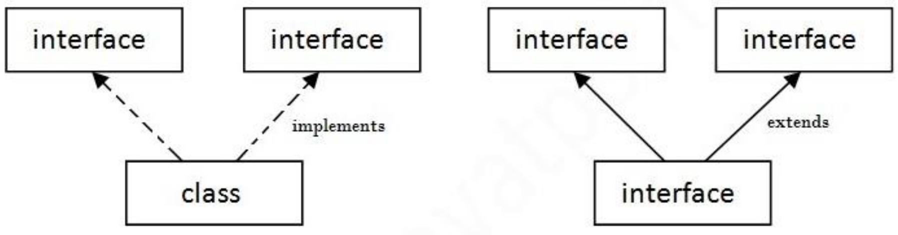
UNDERSTANDING RELATIONSHIP BETWEEN CLASSES AND INTERFACES

As shown in the figure given below, a class extends another class, an interface extends another interface but a class implements an interface.



MULTIPLE INHERITANCE IN JAVA BY INTERFACE

If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.



Multiple Inheritance in Java

IMPLEMENTING INTERFACES:

To implement an interface, a class must provide bodies (implementations) for the methods described by the interface. Each class is free to determine the details of its own implementation.
so two classes might implement the same interface in different ways

```
class class_name implements interface_name
{
body of class.
}
```

A CLASS CAN IMPLEMENT MORE THAN ONE INTERFACE AS SHOWN

```
class class_name implements interfacename1,interfacename2
{
body of class
}
```

EXAMPLE:

```
interface XYZ
{
public void function( );
public void function( );
}
class ABC implements XYZ
{
        public void functionx( )
        {
        //body
        }
        public void function( )
        {
        //body
        }

}
```

THE CLASS IMPLEMENTING AN INTERFACE:

```
interface XYZ
{
        public void funx( );
        public void funy( );
}
class ABC implements XYZ
{
        public void funx( )
        {
                System.out.println("void functionx( )");
        }
        public void funy( )
        {
```

```
                    System.out.println("void function( )");
          }
}

classInterfaceDemo
{
          public static void main(String args[ ])
          {
                    ABC a1=new ABC( );
                    a1.funx( );
                    a1.funy( );
          }
}
```

<u>THE CLASS IMPLEMENTING MORE THAN ONE INTERFACE</u>

```
interface XYZ
{
          public void funx( );
          public void funy( );
}
interface MSD
{
          public void funM( );
          public void funS( );
}

class ABC implements XYZ,MSD
{
          public void funx( )
          {
                    System.out.println("void functionx( )");
          }
          public void funy( )
          {
                    System.out.println("void function( )");
          }
          public void funM( )
          {
                    System.out.println("void function( )");
          }
          public void funS( )
          {
                    System.out.println("void function( )");
          }
}
classInterfaceDemo
{
          public static void main(String args[ ])
          {
                    ABC a1=new ABC( );
                    a1.funx( );
                    a1.funy( );
                    a1.funM( );
```

```
                a1.funS( );
        }
}
```

THE INTERFACE EXTENDING MORE THAN ONE INTERFACE. THIS IS EXAMPLE OF ACHIEVING MULTIPLE INHERITANCE USING INTERFACES

```
interface XYZ
{
        public void funx( );

interface MSD
{
        public void funM( );

}
interface PQR extends XYZ,MSD
{
        public void funP( );
}
class ABC implements PQR
{
        public void funX( )
        {
                System.out.println("void functionX( )");
        }
        public void funM( )
        {
                System.out.println("void functionM( )");
        }
        public void funP( )
        {
                System.out.println("void functionP( )");
        }
}
classInterfaceMulti
{
public static void main(String args[ ])
        {
                ABC a1=new ABC( );
                a1.funX( );
                a1.funM( );
                a1.funP( );
        }
}
```

ACCESSING INTERFACE VARIBALE:

```
interface XYZ
{
        int a=100;
        int b=200;
        public void funx( );
}
```

```
class ABC implements XYZ
{
        int c=300;
        public void funx( )
        {
                System.out.println(“a = ” + a);
                System.out.println(“b = ” + b);
                System.out.println(“c = ” + c);
                a=400; // Error .the final variables cannot be altered
        }
}
classInterfaceVariable
{
        public static void main(String args[ ])
        {
                ABC a1=new ABC ( );
                a1.funx ( );
        }
}
```

DIFFERENCE BETWEEN ABSTRACT CLASS AND INTERFACE

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

ABSTRACT CLASS	INTERFACE
Abstract is partially implemented and partially unimplemented structure	Interface is fully unimplemented structure.
The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.
Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods.
Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
Abstract class can have final, non-final, static and instance variables.	Interface has only static and final variables.
Abstract class can have static methods, main method and constructor.	Interface can't have static methods, main method or constructor.
Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.

Example:
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

CLASS	INTERFACE
Class is fully implemented structure	Interface is fully unimplemented structure
Class can be instantiated	Interface cannot be instantiated
Class contain both static and non static variables	Interface contains only final variables
No abstract methods are allowed	It allows only abstract methods
The class is declared using the keyword class	The interface is declared using the keyword interface
Class can extend only one class	Interface can extends more than one interface

Classes implements interface	An interface extends another interface.

ACCESS MODIFIERS IN JAVA

There are two types of modifiers in java: access modifiers and non-access modifiers.
The access modifiers in java specify accessibility (scope) of a data member, method, constructor or class.
There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc. Here, we will learn access modifiers.

1) PRIVATE ACCESS MODIFIER

The private access modifier is accessible only within class. We can apply it to methods and members variables. The private methods and variable are accessible only within the declaring class.

SIMPLE EXAMPLE OF PRIVATE ACCESS MODIFIER

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is compile time error.

```
class A
{
        private int data=40;
        private void msg()
        {
                System.out.println("Hello java");
        }
}
public class Simple
{
    public static void main(String args[])
    {
                A obj=new A();
                System.out.println(obj.data);//Compile Time Error
                obj.msg();//Compile Time Error
    }
}
```

Note: A class cannot be private or protected except nested class.

2) DEFAULT ACCESS MODIFIER

If you don't use any modifier, it is treated as default by default. The default modifier is accessible only within package.A class or its members with default access are available to all the classes within the same package in which the class is declared. For example if we declare a class with default access, it can only instantiated by the classes within the same package. The member variable or methods with default access can only be accessed by classes within the same package.

EXAMPLE OF DEFAULT ACCESS MODIFIER

In this example, we have created two packages pack and my pack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

```
//save by A.java
package pack;
class A
{
             void msg()
            {
                        System.out.println("Hello");
            }
}
//save by B.java
package mypack;
import pack.*;
class B
{
            public static void main(String args[])
            {
                        A obj = new A();//Compile Time Error
                        obj.msg();//Compile Time Error
            }
}
//save by A.java
package pack;
class A
{
             void msg()
            {
                        System.out.println("Hello");
            }
}
//save by B.java
package pack;
class B
{
              public static void main(String args[])
            {
                        A obj = new A();//it is accessible because in same package
                        obj.msg();//it is accessible because in same package
             }
}
```

3) PROTECTED ACCESS MODIFIER

The protected access modifier is accessible within package and outside the package but through inheritance only.
The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

EXAMPLE OF PROTECTED ACCESS MODIFIER

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

```
//save by A.java
```

```
package pack;
public class A
{
      protected void msg()
      {
      System.out.println("Hello");
      }
}
//save by B.java
package mypack;
import pack.*;
class B extends A
{
        public static void main(String args[])
       {
                  B obj = new B();
                  obj.msg();
        }
      }  Output:Hello
```

4) PUBLIC ACCESS MODIFIER

The public access modifier is accessible everywhere. It has the widest scope among all other modifiers. Example of public access modifier.

```
//save by A.java
package pack;
public class A
{
      public void msg()
      {
            System.out.println("Hello");
      }
}
//save by B.java
package mypack;
import pack.*;
class B
{
      public static void main(String args[])
      {
                   A obj = new A();
                   obj.msg();
       }
}
Output: Hello
```

UNDERSTANDING ALL JAVA ACCESS MODIFIERS

Let's understand the access modifiers by a simple table.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Public :

- Least restrictive of all access modifiers.
- Applicable to top level classes, nested classes , methods and member variables.

Protected:

- Applicable to methods, member variables and nested classes. The top level class cannot be declared as protected.
- Allow access to all subclasses as per inherence
- Also allows access to all the classes in the same package.

Default:

- Absence of any modifier indicates a default or package access.
- Applicable to top level classes, inner classes, methods and member variable.
- Allows access to all the classes in the same package.

Private:

- Most restrictive of all access modifiers.
- Applicable to inner classes , methods and member variables.
- Allows access only to the declaring class.

JAVA ACCESS MODIFIERS WITH METHOD OVERRIDING

If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive. The default modifier is more restrictive than protected. That is why there is compile time error.

```
class A
{
protected void msg()
{
System.out.println("Hello java");
}  }
public class Simple extends A
{
void msg()
{
System.out.println("Hello java");}//C.T.Error
public static void main(String args[])
{
Simple obj=new Simple();
obj.msg();
}  }
```

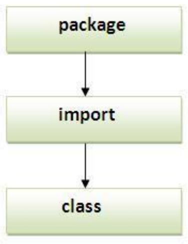
JAVA PACKAGE

The package is a logical organization of related classes. We can logically organize java classes into packages. That means we can put all related classes into one package
For example: in banking application, we decide to put all classes related to accounting in one package..
A java package is a group of similar types of classes, interfaces and sub-packages. The package statements
Package in java can be categorized in two form, built-in package and user-defined package.
There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.
A package serves two purposes:

- First: it provides a mechanism by which related pieces of a program can be organized as a unit. Classes defined within a package must be accessed through their packages name. Thus a package provides a way to name a collection of classes.
- Second: a package participates in java’s access control mechanism. Classes defined within a package can be made private to that package and not accessible by code outside the package. Thus the package provides a means by which classes can be encapsulated.

Default package: java.lang package

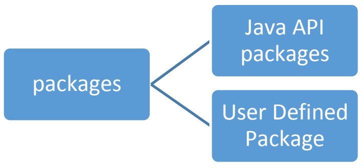
Sequence of the program must be package then import then class.



ADVANTAGE OF JAVA PACKAGE

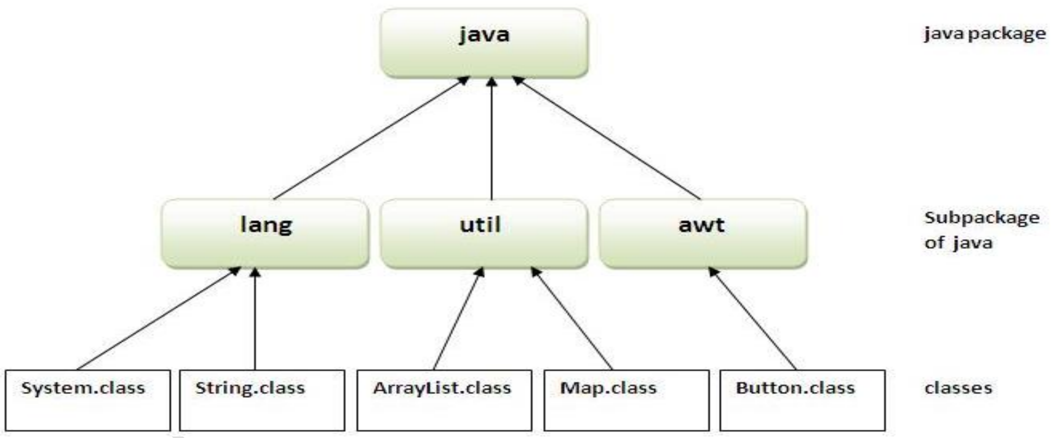
1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2) Java package provides access protection.
3) Java package removes naming collision.

The java packages are classified into two categories:



The java API packages contain built in classes provided by the Java. The User Defined packages are the packages which the programmer develops.

Java API Packages:



Purpose	Package Name	Description
Language support package	java.lang	These classes support the basic language features and the handling of arrays and strings. Classes in this package are always available directly in our programs by default
Input/output package	java.io	Classes for a data input and output operation
Utilities package	java.util	A collection of classes to provide utility functions such as date and time functions.
Networking package	java.net	A collection of classes for communicating with other computer via internet.
AWT	java.awt	The Abstract Window Tool kit package contains classes that implements platform independent graphical user interface.
Applet	java.applet	This includes a set of classes that allows us to create java applets.
Swings	javax.swing	These classes provide easy to use and flexible components for building graphical user interfaces (GUIs) . the components in this package are referred as Swing components.
Java Database connectivity	javax.sql	Classes related to database access

HOW TO ACCESS JAVA API PACKAGES:

There are three ways to access the package from outside the package.

1. import package.*;
2. import package.classname;
3. Fully qualified name.

1) USING PACKAGENAME.*

If you use package.* then all the classes and interfaces of this package will be accessible but not sub packages.
The import keyword is used to make the classes and interface of another package accessible to the current package.

```
import java.util.*;
Vector v= new Vector();
ArrayList a = new ArrayList();
HashSet s= new hashSet();
```

```
Example of package that import the packagename.*
import java.util.*;
class A
{
        public static void main(String args[])
        {
                Vecotor v=new Vector();
                // code
        }
}
```

2) USING PACKAGENAME.CLASSNAME

If you import package.classname then only declared class of this package will be accessible.

```
Example of package using class name
```

```
import java.util.Vector;
class A
{
	public static void main(String args[])
	{
		Vecotor v=new Vector();
		// code
	}
}
```

3) USING FULLY QUALIFIED NAME

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.
It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

```
Example of package by import fully qualified name
class A
{
	public static void main(String args[])
	{
		java.util.Vecotor v=new java.util.Vector();
		// code
	}
}
```

USER DEFINED PACKAGE:

To create a package, put a package command at the top of java source file. The classes declared within that file will then belong to the specified package. Since a package defined a namespace , the names of the classes that we put into the file become part of that package's namespace.

General form of the package statements:

package pkg;// here pkg is the name of the package.

General form of a multileveleved package statement is shown

package pack1,pack2,pack3……packn;

HOW TO COMPILE JAVA USER DEFINED PACKAGE

If you are not using any IDE, you need to follow the syntax given below:

1. javac -d directory javafilename

For example

1. javac -d . Simple.java

The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

HOW TO RUN JAVA PACKAGE PROGRAM

You need to use fully qualified name e.g. mypack.Simpleetc to run the class.
To Compile:javac -d . Simple.java
To Run: java mypack.Simple
Output:Welcome to package
The -d is a switch that tells the compiler where to put the class file i.e. it represents destination. The represents the current folder.

CREATING A USER DEFINED PACKAGE:

Step 1: let us create a program which is part of packge called pack1:

```
        package pack1;
        public class PackageExample
        {
                public void packmethod
                {
                        System.out.println("I'm packmethod() in pack1.packagename"):
                }
        }
```

Step 2: place this file PackageExample.java in C:\ and compile the program as shown below. We have to use –d option to create the pack1 directory automatically.

The command is javac –d.PackageExample.java

Step 3: create another file called packageDemo.java in pack2 package as shown below:

```
        package pack2;
        import pack1.*;
        public class PackageDemo
        {
                public static void main( String args[])
                {
                        PackageExample obj = new PackageExmaple();
                        obj.packmethod();
                }
        }
```

In this file, we are trying to import the pack1 classes and creating an object of PackageExample class and call to packmethod().

Step4: Place this file also C:\ and the program using –d option as shown below
The command is javac –d.PackageDemo.java

Step5: The compilations of both the classes are compiled. Now lets us execute the PackageDemo class. Go to the C:\ prompt and execute the file using
java pack2.PackageDemo.java

Hidden Classes:

One of the advantages of packages is to hide the cases from other packages. If we declare the class as public, then the class is visible in all packages. If we do not want to expose the class to other packages then we can use default access.

```
package pack1;
public class ABC
{
}
```

If want to hide the class, than we can use just default access to the class. In the below example ABC is visible to all and XYZ is visible to only pack1 package. The classes outside of pack1 cannot use this class.

```
package pack1;
public class ABC
{
}
class xyz
{
}
```

How to add class to a package:

The class can be added to the package by mentioning the below statement in the beginning of the java source file

```
package pack1;
class ABC
{
}
```

SUBPACKAGE IN JAVA

Package inside the package is called the subpackage. It should be created to categorize the package further.
Let's take an example, Sun Microsystem has definded a package named java that contains many classes like System, String, Reader, Writer, Socket etc. These classes represent a particular group e.g. Reader and Writer classes are for Input/Output operation, Socket and ServerSocket classes are for networking etc and so on. So, Sun has subcategorized the java package into subpackages such as lang, net, io etc. and put the Input/Output related classes in io package, Server and ServerSocket classes in net packages and so on.
The standard of defining package is domain.company.package e.g. com.javatpoint.bean or org.sssit.dao.

EXAMPLE OF SUBPACKAGE

```
package com.javatpoint.core;
class Simple{
  public static void main(String args[]){
   System.out.println("Hello subpackage");
  }
}
```

To Compile:javac -d . Simple.java
To Run: java com.javatpoint.core.Simple
Output:Hellosubpackage

CHAPTER 10:

1. What is interface?
2. What is import keyword(2012)
3. What is package? Explain with an example? What its purpose
4. List all the access modifiers in java? Explain default access modifier.
5. What is the default package imported in java source code.
6. What is difference between class and interface? (2010).
7. How multiple inheritance is achieved in java(2015)

6. What is the use of interface?
7. How to define an interface
8. What are the rules of defining an interface?
9. How interface are implemented in java?
10. Explain the difference forms of implementing interface
11. Explain extending interface
12. What is the difference between abstract class and interface?
13. How multiple inheritances are achieved using interface? Explain with a program
14. How to access control modifiers
15. Explain private protected default and public access modifiers with an example
16. Explain the role of each access modifier with an example
17. How package are classified
18. How to use java API package

19. How to import the java classes
20. How to define a user defined package
21. Explain the process of creating user defined package with an example
22. How to hide the classes

5 marks

1. What is package how to create user defined package./how do you create an access package(2012)(2015)
2. How to define an interface? Explain various forms of implementing interface
3. Explain visibility control in java
4. How to import java package to the source code
5. Explain interface concept with an example? Can interface be extended.(2015)
6. Explain with an programming example how to create and use an interface.(2010)
7. Why do we need import statements? explain with an example(2011)
8. What is interface? How to implement interface? Give an example (2011)
9. Give the general form of interface with an example(2016)
10. Give steps to create and use a java package with an example(2016)