Que. Difference between C++ and Java.
Ans.

C++	Java
C++ support object oriented programming but it is not pure object oriented language.	Java is a pure object oriented programming language.
C++ generates object code and the same code may not run on different platforms. Means C++ is platform-dependent.	Java is interpreted for the most part and hence platform independent.
C++ uses compiler only.	Java uses compiler and interpreter both.
C++ support operator overloading.	Java does not support operator overloading.
C++ have template classes.	Java does not have template classes.
C++ support multiple inheritance of classes. The keyword virtual is used to resolve ambiguities during multiple inheritance if there is any.	Java does not support multiple inheritance of classes. This is accomplished using a new feature called "interface".
C++ support global variables.	Java does not support global variables. Every variable and method is declared within a class and forms part of that class.
C++ support pointers. (Asst. Prof. Viral S. Patel)	Java does not use pointers.
C++ support structures and unions.	Java does not support unions, structures.
C++ does not have Boolean data type.	Java has Boolean data type. And this type of variable can store true or false value.
C++ has destructor function.	Java has replaced the destructor function with a finalize() function.
C++ support typedef.	Java does not support typedef.
In C++ we can declare unsigned integers.	It is not possible to declare unsigned integers in java.
C++ have many header files.	There are no header files in Java.
C++ have delete operator. C++ does not have garbage collection.	Java does not have delete operator. Java has garbage collection that automatically frees blocks of memory.
In C++ objects are pass by value or pass by reference.	In java objects are pass by reference only.
C++ have simple break and continue statements.	Java has enhanced break and continue statements called label break and label continue concept.
In C++ '<<' and '>>' operators are overloaded for I/O operations.	In java '<<' and '>>' operators are not overloaded for I/O operations.
In C++ '>>>' operator is not available like '>>>' operator used in java.	In java '>>>' operator is used as unsigned right shift.
C++ has only single-line and multi-line comments.	Java has single-line, multi-line comments and also has documentation comment which is begin with /** and end with */.
C++ has no built in support for threads.	Java has built in support for threads.
C++ has goto statement.	There is no goto statement in java.
In C++ there is no concept of package.	In java there is a concept of package.
C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.

Que. Why pointers don't exist in java.
Ans. Pointers don't exist in java for **two reasons** :
➔Pointers are **inherently insecure**. By using pointer it is possible to access memory addresses and data from outside of program's code. A malicious program could make use of this fact to damage the system, perform unauthorized accesses (such as obtaining passwords), or otherwise violate security restrictions.
➔Even if pointers could be restricted to the confines of the Java run-time system (which is theoretically possible), the designers of Java believed that they were **inherently troublesome**.

Que. Explain the Features of Java (or Java Buzzwords)
Ans.

- Simple
- Secure
- Portable
- Object-oriented
- Robust
- Multithreaded
- Architecture-neutral
- Interpreted
- High performance
- Distributed
- Dynamic

Simple
➔Java was designed to be easy for the professional programmer to learn and use effectively. If we already understand the basic concepts of object-oriented programming, learning Java will be even easier.
➔Because Java **inherits the C/C++ syntax** and many of the object-oriented features of C++, most programmers have little trouble learning Java.

Secure
➔Most users worried about the possibility of infecting their systems with a virus. This type of program can gather private information, such as credit card numbers, bank account balances, and passwords, by searching the contents of your computer's local file system.
➔Java answers of these concerns by providing a "**firewall**" between a networked application and your computer. (Asst. Prof. Viral S. Patel)
➔When we use a Java-compatible Web browser, we can safely download **Java applets** without fear of virus infection or malicious intent.
➔Java achieves this protection by confining a Java program to the **Java execution environment** and **not allowing it access to other parts of the computer.**

Portable
➔Many types of computers and operating systems are in use throughout the world—and many are connected to the Internet. For programs to be dynamically downloaded to all the various types of platforms connected to the Internet, some means of generating portable executable code is needed. Java interpretation of **bytecode** is the easiest way to create truly portable programs.

Object Oriented
➔In java "everything is an object" paradigm. The object model in java is simple and easy to extend.

Robust
➔Two of the main reasons for program failure: **memory management mistakes and mishandled exceptional conditions** (run-time errors). The program must execute reliably in a variety of systems.
➔Java is robust as deallocation is completely automatic in Java, because Java provides **garbage collection** for unused objects and providing object-oriented **exception handling**.

Multithreaded
➔Java supports multithreaded programming, which allows to write **programs that do many things simultaneously**. The Java run-time system comes with an elegant yet sophisticated solution for **multiprocess synchronization** that enables you to construct smoothly running interactive systems.

Architecture-Neutral
➔A central issue for the Java designers was that of code longevity and portability. One of the main problems facing programmers is that no guarantee exists that if you write a program today, it will run tomorrow—even on the same machine. Operating system upgrades, processor upgrades, and changes in core system resources can all combine to make a program malfunction.
➔The Java designers made several hard decisions in the Java language and the Java Virtual Machine in an attempt to alter this situation. Their goal was "**write once; run anywhere, any time, forever.**" (Asst. Prof. Viral S. Patel)

Interpreted
➔Java enables the creation of cross-platform programs by compiling into an intermediate representation called **Java bytecode.** This code can be **interpreted on any system that provides a Java Virtual Machine.**

High Performance
➔ Java, however, was designed to perform well on very **low-power CPUs**. Java was engineered for interpretation, the Java bytecode was carefully designed so that it would be easy to translate directly into native machine code for very high performance by using a **just-in-time compiler.**

Distributed
Java is designed for the distributed environment of the **Internet**, because it handles TCP/IP protocols. Java allowed objects on two different computers to execute procedures remotely. Java revived these interfaces in a package called **Remote Method Invocation (RMI).** This feature brings client/server programming.

Dynamic
Java programs carry with them substantial amounts of run-time type information that is used to **verify and resolve accesses to objects at run time**. This is crucial to the robustness of the

applet environment, in which small fragments of bytecode may be dynamically updated on a running system. (Asst. Prof. Viral Patel)

Que. Explain Java Program Structure.
Ans.

Document Section	**/* comment */**
Package Statement	**package p1;**
Import Statements	**import java.io.*;**
Interface Statements	**interface C{ …}** **interface D{…}**
Class Definitions **{** **Data members** **User Defined methods** **}**	**class A extends B implements C,D** **{** **int a,b;** **void disp( ) { …}** **}**
Main method class **{** **Main Method** **}**	**class Demo** **{** **public static void main(String args[]) { …}** **}**

- ⇨ A **package** is a collection of classes, interfaces and sub-packages. A sub package contains collection of classes, interfaces and sub-sub packages etc. **java.lang.*;** This package is imported by default and this package is known as default package.
- ⇨ **Import** statement is used to import classes of other package in current package.
- ⇨ **Interface** is like a class but includes a group of method declarations. It is used only when we wish to implement multiple inheritance feature in program.
- ⇨ **Class** is used for developing user defined data type and every java program must start with a concept of class. Class can extends other class and can implements interfaces.
- ⇨ **Data member** represents either instance or static they will be selected based on the name of the class.
- ⇨ **User-defined methods** represents either instance or static they are meant for performing the operations either once or each and every time.
- ⇨ Each and every java program starts execution from the **main() method**. And hence main() method is known as program driver.

  main( ) must be declared as public, since it must be called by code outside of its class when the program is started.

  The keyword static allows main( ) to be called without having to instantiate a particular instance of the class. This is necessary since main( ) is called by the Java interpreter before any objects are made.

  The keyword void simply tells the compiler that main( ) does not return a value.

  Each and every main() method of java must take array of objects of String.

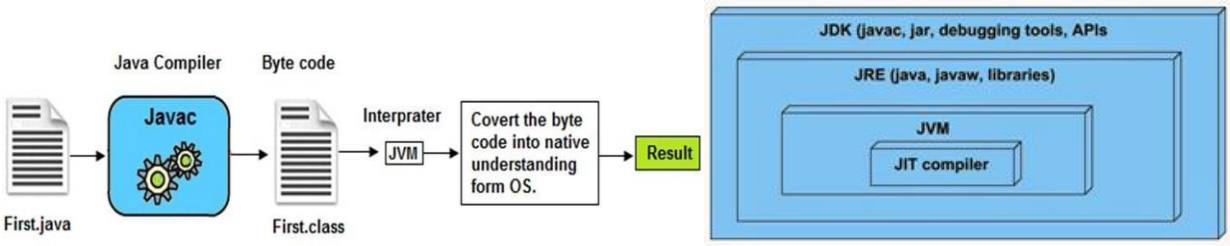
Que. What is compiler ?
Ans. Compiler is a program that convert High level language program/code in low level language program/code. (Asst. Prof. Viral S. Patel)

Que. What is Java Compiler ?
Ans. Compile java code (.java source file) and convert in bytecode (.class file).

Que. What is Interpreter ?
Ans. An interpreter is a computer language processor (computer program) that **reads a program line-by-line (statement-by-statement) and directly executes** instructions written in a programming or scripting language, without requiring them previously to have been compiled into a machine language program.

Que. What is Java Interpreter ?
Ans. A java interpreter is usually referred to as the Java Virtual Machine (or JVM). It **reads line by line bytecodes and executes** the bytecodes (.class file).

***Note : Following figure just for understand**



Que. What is bytecode ?
Ans. The output of a Java compiler is bytecode. Bytecode is a **highly optimized set of instructions** designed to be executed by the Java Virtual Machine (JVM). This bytecode interpreted by all JVM so java become platform independent.

Que. What is JVM ? or What is Java's Magic ?
Ans. The output of a Java compiler is not executable code. Rather, it is bytecode.
Bytecode is a highly optimized set of instructions designed to be executed by the **Java run-time system**, which is called the **Java Virtual Machine (JVM).** That is, in its standard form, the JVM is an interpreter for bytecode.
JVM will **differ from platform to platform**, All interpret the same Java bytecode. So only the JVM needs to be implemented for each platform.
JVM make Java program secure as the execution of every Java program is under the control of the JVM. (Asst. Prof. Viral S. Patel)

Que. What is JIT ?
Ans. JIT is just-in-time compiler for bytecode, which is included in the Java 2 release. In order to improve performance, JIT compiler interact with the Java Virtual Machine (JVM) at run time. When the JIT compiler is part of the JVM, it **compiles bytecode into executable code in real time, on a piece-by-piece, demand basis**. Java **performs various run-time checks**, so JIT compiles codes as it is needed during execution. (Asst. Prof. Viral S. Patel)

The JIT compiler is able to perform certain optimization while compiling a series of bytecode to native machine language. Some of these optimizations performed by JIT compilers are data analysis, reduction of memory accesses by register allocation, translation from stack operations to register operations, elimination of common sub-expressions etc.

Once the bytecode is compiled into that particular **machine code**, it is **cached by the JIT compiler** and will be reused for the future needs. Hence the main **performance improvement**

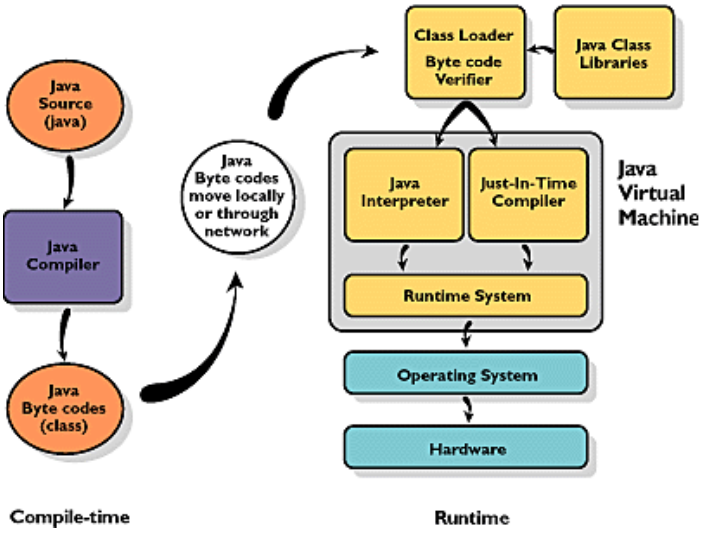
by using JIT compiler can be seen **when the same code is executed again and again** because JIT make use of the machine code which is cached and stored.

Que. What is JRE ?
Ans. The **Java Runtime Environment (JRE)** facilitates the execution of programs developed in Java. It contains **JVM, set of libraries and other files** that JVM uses at runtime.

Que. Write down the significance of CLASSPATH.
Ans. CLASSPATH environment variable used to specify/set **path of the class files** which we want to load using ClassLoader. If this path is not set and we need to load a class file that is not present in the current directory then operating system give ClassNotFoundException or NoClassDefFoundError.

Que. Explain Java Architecture.

Ans.



1. Compilation and interpretation in Java

Java combines both the approaches of compilation and interpretation.
➔First, java compiler compiles the source code into bytecode.
➔At the run time, Java Virtual Machine (JVM) interprets this bytecode and generates machine code which will be directly executed by the machine in which java program runs.
So java is both compiled and interpreted language.

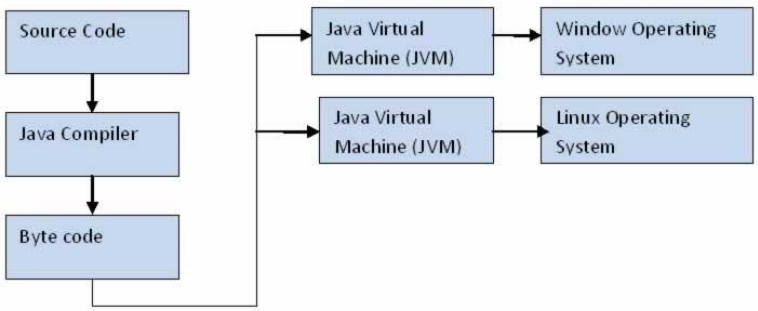
2. Java Virtual Machine (JVM)
JVM is a component which provides an environment for running Java programs. JVM interprets the bytecode into machine code which will be executed the machine in which the Java program runs. (Asst. Prof. Viral S. Patel)

Why Java is Platform Independent?

Platform independence is one of the main advantages of Java. In another words, java is portable because the same java program can be executed in multiple platforms without making any changes in the source code. You just need to write the java code for one platform and the **same program will run in any platforms**. But **how** does Java make this possible?

First the Java code is compiled by the Java compiler and generates the bytecode. This bytecode will be stored in class files. **Java Virtual Machine (JVM) is unique for each platform.** Though JVM is unique for each platform, all interpret the same bytecode and convert it into machine code required for its own platform and this machine code will be directly executed by the machine in which java program runs. This makes Java platform independent and portable.

In following diagram we can see that the same compiled Java bytecode is interpreted by two different JVMs to make it run in Windows and Linux platforms.



3. Java Runtime Environment (JRE)

Java Runtime Environment contains JVM, class libraries and other supporting components.

As you know the Java source code is compiled into bytecode by Java compiler. This bytecode will be stored in class files. During runtime, this bytecode will be loaded, verified and JVM interprets the bytecode into machine code which will be executed in the machine in which the Java program runs. (Asst. Prof. Viral S. Patel)

A Java Runtime Environment performs the following main tasks respectively.

1. **Loads the class**
This is done by the class loader
2. **Verifies the bytecode**
This is done by bytecode verifier.
3. **Interprets the bytecode**
This is done by the JVM

Class loader

Class loader loads all the class files required to execute the program. Class loader makes the program secure by separating the namespace for the classes obtained through the network from the classes available locally. Once the bytecode is loaded successfully, then next step is bytecode verification by bytecode verifier.

Byte code verifier

The bytecode verifier verifies the byte code to see if any security problems are there in the code. It checks the byte code and ensures the followings.

1. The code follows JVM specifications.
2. There is no unauthorized access to memory.
3. The code does not cause any stack overflows.
4. There are no illegal data conversions in the code such as float to object references.

Once this code is verified and proven that there is no security issues with the code, JVM will convert the byte code into machine code which will be directly executed by the machine in which the Java program runs.

Why Java is Secure?

The byte code is inspected carefully before execution by Java Runtime Environment (JRE). This is mainly done by the "Class loader" and "Byte code verifier". Hence a high level of security is achieved.

4. Just in Time Compiler

This is a component which helps the program execution to happen faster.

When the Java program is executed, the byte code is interpreted by JVM. But this interpretation is a slower process. To overcome this difficulty, JRE include the component JIT compiler. JIT makes the execution faster. (Asst. Prof. Viral S. Patel)

If the JIT Compiler library exists, when a particular bytecode is executed first time, JIT complier compiles it into native machine code which can be directly executed by the machine in which the Java program runs. Once the byte code is recompiled by JIT compiler, the execution time needed will be much lesser. This compilation happens when the byte code is about to be executed and hence the name "Just in Time".

Once the bytecode is compiled into that particular **machine code**, it is **cached by the JIT compiler** and will be reused for the future needs. Hence the main **performance improvement** by using JIT compiler can be seen **when the same code is executed again and again** because JIT make use of the machine code which is cached and stored.

5. Garbage Collection

Garbage collection is a process by which Java achieves better memory management. As you know, in object oriented programming, objects communicate to each other by passing messages.

Whenever an object is created, there will be some memory allocated for this object. This memory will remain as allocated until there are some references to this object. **When there is no reference to this object, Java will assume that this object is not used anymore.** When

garbage collection process happens, **these objects will be destroyed and memory will be reclaimed.**

Garbage collection happens automatically. There is no way that you can force garbage collection to happen. There are two methods "System.gc()" and "Runtime.gc()" through which you can make request for garbage collation. But calling these methods also will not force garbage collection to happen and you cannot make sure when this garbage collection will happen.

Que. Explain the meaning of ' public static void main(String args[]) '.
Ans. This is the line at which the program will begin executing. All Java applications begin execution by calling **main( ).** Java is case-sensitive. Thus, Main is different from main. It is important to understand that the Java compiler will compile classes that do not contain a main( ) method. However,the Java interpreter would report an error because it would be unable to find the main( ) method.

The **public** keyword is an access specifier, main( ) must be declared as public, since it must be **called** by code **outside of its class** when the program is started.

The keyword **static** allows main( ) to be **called without** having to instantiate a particular **instance (object)** of the class. This is necessary since main( ) is called by the Java interpreter before any objects are made.

The keyword **void** simply tells the compiler that main( ) does not return a value.

In main( ), there is only one parameter. **String args[ ]** declares a parameter named args, which is an array of instances of the class String. Objects of type String store character strings. In this case, args receives any command-line arguments present when the program is executed.

Que. Explain the meaning of ' System.out.println("Hello !!! Prof. Viral Patel") '.
Ans. System is a predefined class that provides access to the system, and **out** is the output stream that is connected to the console. **println( )** displays the string like "Hello !!! Prof. Viral Patel" in new line which is passed to it. println( ) can be used to display other types of information, too.

Que. Explain different types of data in java.
Ans.
➔Java is more strictly typed than C or C++ language. For example, in C/C++ you can assign a floating-point value to an integer. In Java, we cannot.
➔C and C++ allow the size of an integer to vary based upon the dictates of the execution environment. However, Java is different. Because of Java's portability requirement, all data types have a strictly defined range.

For example, an int is always 32 bits, regardless of the particular platform. This allows programs to be written that are guaranteed to run without porting on any machine architecture
➔Java defines eight simple (or elemental) types of data: **byte, short, int, long, char, float, double, and boolean.** These can be put in four groups:

■ **Integers** This group includes **byte, short, int, and long**, which are for wholevalued

signed numbers. Java does not support unsigned(positive-only) integers.

byte => 8 bits (-128 to +127)
short => 16 bits (-32,768 to +32,767)
int => 32 bits (-2,147,483,648 to +2,147,483,647)
long => 64 bits (–9,223,372,036,854,775,808 to +9,223,372,036,854,775,807)

■ **Floating-point numbers** This group includes **float and double**, which represent numbers with fractional precision.

Float => 32 bits (1.4e–045 to 3.4e+038) [1.4 * 10^(-45) to 3.4 * 10^(38)]
Double => 64 bits (4.9e–324 to 1.8e+308) [4.9 * 10^(-324) to 1.8 * 10^(308)]

■ **Characters** This group includes **char**, which represents symbols in a character set, like letters and numbers.

Char => 16 bits (0 to 65,536)

Java uses **Unicode** to represent characters. Unicode defines a fully international character set that can represent all of the characters found in all human languages. It is a unification of dozens of character sets, such as Latin, Greek, Arabic, Cyrillic, Hebrew, Katakana, Hangul, and many more. For this purpose, it requires 16 bits. Thus, in Java char is a 16-bit type. (ASCII still ranges from 0 to 127.)

■ **Boolean** This group includes **boolean**, which is a special type for representing true/false values.

Data type	**Default Value**	**Size**
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Que. Explain Type Conversion (Prof. Viral S. Patel)

Ans. Java can perform type conversion automatically or perform explicit conversion.

Automatic Conversion :

➔Also known as **Widening conversion**.

➔It will take place if the following **two conditions** are met:

1. Two types are compatible
2. Destination type is larger than the source type

➔**Example** : int to float, int to long

Explicit Conversion :
➔Also known as **Narrowing Conversion**
➔Numeric types are not compatible with char or Boolean.
➔char and Boolean are not compatible with each other
➔It will take place if the one of the following two conditions are met:

1. Two types are not compatible
2. Destination type is smaller than the source type

Example : int to byte
Example of int to byte explicit conversion:

```
int a=257;
byte b;
b = (byte) a;  // explicitly casting int to byte as byte is smaller than int data type.
```

Range of a byte reduced modulo means The remainder of an integer division by the byte's range 256. If a=257 then b will be 1 as 257%256=1.

Que. What is automatic type promotion in java. Explain Automatic Type Promotion Rules. Which type of problem causes during automatic type promotion and how can we solve it? Explain with example.
Ans. Java automatically promotes each byte or short operand to int when evaluating an expression. Means automatically promotes lower data type to higher data type.

For example,

```
byte a = 40;
byte b = 50;
byte c = 100;
int d = a * b / c;
```

The result of the intermediate term a * b easily exceeds the range of either of its byte operands. To handle this kind of problem, This means that the sub-expression a * b is performed using integers—not bytes.

Example Type Promotion Rules :
double result = (f * b) + (i / c) - (d * s);
f is float variable, b is byte variable, i is int variable, c is char variable, d is double and s is sort variable.

first result of the sub expression f * b is of type float.
Second result of the sub expression i / c is of type int.
Third result of the sub expression d * s is of type double.
The outcome of float plus an int is a float.
Then the resultant float minus the last double is to double, which is the type for the final result of the expression.

Problem : Some time give compile time error.
For example :

```
byte b = 50;
b = b * 2; // Error! Cannot assign an int to a byte!
```

Solution : Explicit Cast.
For example,

```
byte b = 50;
b = (byte)(b * 2);
```

Que. Explain Wrapper class in java. (Asst. Prof. Viral S. Patel)
Ans. Java provides classes that correspond to each of the simple types. In essence, these classes encapsulate or **wrap the simple types within a class.** Thus, they are commonly **referred to as wrapper classes.**
Wrapper class in java provides the mechanism to convert primitive into object and object into primitive. **Autoboxing and unboxing feature** facilitates the process of generates a code implicitly to **convert primitive type to the corresponding wrapper class type and vice-versa**.
One of the **eight classes of java.lang** package are **known as wrapper class** in java.

The list of eight wrapper classes are given below:

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Wrapper class Example: Primitive to Wrapper

```
public class WrapperExample1
{
public static void main(String args[])
{
//Converting int into Integer
int a=20;
Integer i=Integer.valueOf(a);       //converting int into Integer
Integer j=a;           //autoboxing, now compiler will write Integer.valueOf(a) internally
System.out.println(a+" "+i+" "+j);
}
}
```

Output:
20 20 20

Wrapper class Example: Wrapper to Primitive

```
public class WrapperExample2
{
public static void main(String args[])
{
//Converting Integer to int
Integer a=new Integer(3);
int i=a.intValue();          //converting Integer to int
int j=a;              //unboxing, now compiler will write a.intValue() internally
System.out.println(a+" "+i+" "+j);
}
}
```

Output:
3 3 3

Que. What is boxing conversion and Unboxing conversion ?
Ans. Basic data type to corresponding wrapper class type conversion is called Boxing conversion.

- From type boolean to type Boolean
- From type byte to type Byte
- From type short to type Short
- From type char to type Character
- From type int to type Integer
- From type long to type Long
- From type float to type Float
- From type double to type Double

Example :

```
int a=20;
Integer i=Integer.valueOf(a);   //converting int into Integer
Integer j=a;                    //autoboxing, now compiler will write Integer.valueOf(a) internally
```

➔ Wrapper class type to corresponding basic data type conversion is called Unboxing conversion. (Asst. Prof. Viral S. Patel)

- From type Boolean to type boolean
- From type Byte to type byte
- From type Short to type short
- From type Character to type char
- From type Integer to type int
- From type Long to type long

- From type Float to type float
- From type Double to type double

Example :
Integer a=new Integer(3);
int i=a.intValue(); //converting Integer to int
int j=a; //unboxing, now compiler will write a.intValue() internally

Que. Explain 'break, label break, continue, label continue and return' in java as jump statements.
Ans.
Using break to Exit a Loop : When a break statement is encountered inside a loop, the loop is terminated and program control resumes at the next statement following the loop. The break statement can be used with any of Java's loops (while, do..while, for loops)

Example :
```
class BreakLoop
{
   public static void main(String args[])
   {
     for(int i=0; i<10; i++)
     {
        if(i == 5)
           break;

        System.out.print(i + " ");
     }
     System.out.println("Loop complete.");
   }
}
output: 0 1 2 3 4 Loop complete.
```

Using break in nested loops : When used inside a set of **nested loops**, the break statement will only break out of the innermost loop.

Example:
```
class BreakOnlyInnerLoop
{
   public static void main(String args[])
   {
      for(int i=0; i<3; i++)
      {
           for(int j=0; j<5; j++)
           {
             if(j == 2)
                 break;

             System.out.print(j+ " ");
           }

           System.out.print(i+ " ");
      }

      System.out.println("Loops complete.");
    }
}
```

```
Output:  0 1 0 0 1 1 0 1 2 Loops complete.
```

Using break to Exit a outer Loop in nested loops concept as a Form of Goto : Here, label is the name of a label that identifies a block of code. When this form of break executes, control is transferred out of the named block of code. One of the most common uses for a labeled break statement is to exit from nested loops. (Asst. Prof. Viral S. Patel)

Example:

```
class BreakOuterLoop
{
   public static void main(String args[])
   {
      outer : for(int i=0; i<3; i++)
              {
                  for(int j=0; j<5; j++)
                  {
                     if(j == 2)
                       break outer;

                     System.out.print(j+ " ");
                  }

                  System.out.print(i+ " ");
              }

              System.out.println("Loops complete.");
    }
}
Output:  0 1 Loops complete.
```

Using continue in single loop:
The Java *continue statement* is used to continue current flow of the program and skips the remaining code at specified condition.

Example :

```
class Continue
{
   public static void main(String args[])
   {
     for(int i=0; i<5; i++)
     {
        if(i == 2)
           continue;

        System.out.print(i + " ");
     }

     System.out.println("Loop complete.");
   }
}
output: 0 1 3 4 Loop complete.
```

Using continue in nested Loop to continue outer loop : continue may specify a label to describe which enclosing loop to continue. (Asst. Prof. Viral S. Patel)

Example :

```
class Continue
{
   public static void main(String args[])
   {
      outer : for(int i=0; i<3; i++)
              {
                  for(int j=0; j<5; j++)
                  {
                     if(j == 2)
                       continue outer;

                     System.out.print(j+ " ");
                  }

                  System.out.print("This statement does not exit.");
              }

              System.out.println("Loops complete.");
    }
}
Output:  0 1 0 1 0 1 Loops complete.
```

Return :
The return statement is used to explicitly return from a method. That is, it causes program control to transfer back to the caller of the method.

Example :

```
class Return
{
   public static void main(String args[])
   {
              for(int i=0; i<3; i++)
              {
                  for(int j=0; j<5; j++)
                  {
                     if(j == 2)
                       return;

                     System.out.print(j+ " ");
                  }
                  System.out.print("This statement does not exit.");
              }
              System.out.println("This statement also does not exit.");
    }
}
Output:  0 1
```

Que. Which keyword equivalent to go to statement in java.
Ans. We can use **label continue or label break** statements for equivalent to go to statement in java. (Asst. Prof. Viral S. Patel)

Que. Difference between >> and >>>.
Ans. The >> is signed right shift operator. The right shift operator >> shifts all of the bits in a value to the right a specified number of times. Rightmost bit is lost when this shift occur.

leftmost position after ">>" depends on sign extension. If the number is negative, then 1 is used as a filler and if the number is positive, then 0 is used as a filler.

For example:

int a = 35; **a = a >> 2; // a now contains 8**	00000000 00000000 00000000 00100011 35 >> 2 00000000 00000000 00000000 00001000 8
int a = -8; **a = a >> 1; // a now contains -4**	11111111 11111111 11111111 11111000 –8 >>1 11111111 11111111 11111111 11111100 –4

➔ **The >>> is right shift unsigned operator.** It shifts bits towards right. Zeros are fill in the left bits regardless of sign.

For example:

int a = -1; **a = a >>> 24;**	11111111 11111111 11111111 11111111 –1 in binary as an int >>>24 00000000 00000000 00000000 11111111 255 in binary as an int

Que. What will be the output of following statements ?

```
System.out.println("ans:"+2+2);
System.out.println("ans:"+(2+2));
System.out.println(2+2);
```

Ans.

ans: 22
ans: 4
4

Que. What is the difference between & and &&.

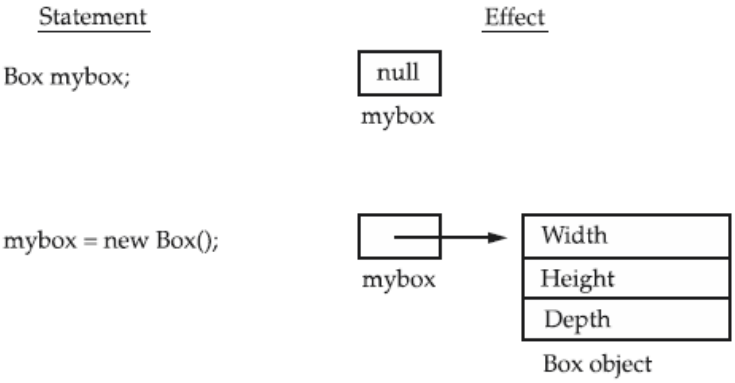
Ans.

	&	**&&**
1	It is Binary AND operator.	It is Boolean AND operator. Also called as short circuit logical operator.
2	In the case of & operator we must have to evaluate the both side expression first.	If we use the && operator, java will not bother to evaluate the right-hand expression.
3	For example, if (c==1 & e++ < 100) d = 100; Here, using a single & ensures that the increment operation will be applied to e whether c is equal to 1 or not.	For example, if (denom != 0 && num / denom > 10) Here there is no risk of causing a run-time exception where **denom** is zero. If this line of code were written using single & , both sides would have to be evaluated, causing a run-time exception when **denom** is zero.

Que. What is class and object ? How to create object and reference of class ?

Ans. Class is a user defined data type. A class is a **template** for an object and Object is a **instance** of a class. (Asst. Prof. Viral S. Patel)

➔A **class** creates a **logical framework** that defines the relationship between its members. An **object** of a class is creating an **instance** of that class.
➔Thus, **a class is a logical construct.** An **object has physical reality**. An object **occupies space in memory**. Keyword **'new' allocates memory** for an object during **run time.**
➔A **reference** is an **address** that indicates where an object's variables and methods are stored. We aren't actually using objects when we assign an object to a variable or pass an object to a method as an argument. We aren't even using copies of the objects. we're only using copy of reference.



Example :

```
class Box
{
    int width; int height; int depth;
    Box() { }
    Box(int w, int h,  int d)
    {
        width = w; height = h; depth = d;
    }
}
class DemoBox
{
   public static void main(String args[])
   {
       Box mybox; // declare reference for object name
       mybox = new Box(); // allocate memory of object

       Box mybox1 = new Box(); // declare reference and allocate memory simultaneously

       Box mybox2 = new Box(2,3,4); // declare, allocate and initialize simultaneously.

       mybox2.display( ); // calling method of class using object
   }
}
```

Que. Explain this keyword. (3 to 4) (Asst. Prof. Viral S. Patel)
Ans. Sometimes a method will need to refer to the object that invoked it. To allow this, Java defines the this keyword. this can be used inside any method to **refer to the current object**.

// A redundant use of this.

Box(double w, double h, double d) //constructor
{
this.width = w;
this.height = h;
this.depth = d;
}
	Box ob1 = new Box(2,3,4);
	// this refer to object ob1 here
	
	Box ob2 = new Box(5,6,7);
	//this refer to object ob2 here

void display( ) // method {	ob1.display(); // this refer to object ob1 here.

System.out.println(this.width+" "+this.height+" "+this.depth); }	

➔It is illegal in Java to declare two local variables with the same name inside the same or enclosing scopes. When a **local variable has the same name as an instance variable**, the local variable hides the instance variable.

Because 'this' refer directly to the object, we can use it to resolve any name space collisions that might occur between instance variables and local variables**.**

```
// Use this to resolve name-space collisions.
Box(double width, double height, double depth)
{
    this.width = width;
    this.height = height;
    this.depth = depth;
}
```

Que. What is Garbage Collection ? (2 or 3)

Ans. Since objects are dynamically allocated by using the new operator, such objects must be destroyed and their memory released for later reallocation.

➔In some languages, such as **C++,** dynamically allocated objects must be manually released by use of a **delete operator.**

➔**Java** takes a different approach; it handles **deallocation automatically**. The technique that accomplishes this is called **garbage collection.**

It works like this: **when no references to an object exist**, that object is assumed to be no longer needed, and the memory occupied by the **object can be reclaimed**. There is no explicit need to destroy objects as in C++. Garbage collection only occurs sporadically (if at all) during the execution of your program. It will not occur simply because one or more objects exist that are no longer used.

Que. Explain static keyword. Also explain calling sequence of static variables, methods and blocks in java with example. (7)

Ans. When a member (variable or method) of class declared static, it can be accessed before any objects of its class are created, and without reference to any object.

➔**Variables declared as static:** When variables declared as static, they work **like global variables**. When objects of its class are declared, no copy of a static variable is made (means static variable is **not part of object memory**). All instances **(objects)** of the class **share** the same static variable. (Asst. Prof. Viral S. Patel)

➔**Methods declared as static:** The most common example of static method is **main method**. Main is declared as static because it must be called before any objects exist.

Methods declared as static have several restrictions :

- They can only call other **static** methods.

- They must only access **static** data.
- They cannot refer to **this or super** in any way.

➔Static methods and variables can be used independently of any object. We can access them only by specify the name of their class followed by the dot operator.

classname.method( )

classname.variableOfClass

Example :

```
class StaticDemo
{
  static int a = 42;
  static void callme()
  {
     System.out.println("a = " + a);
  }
}
class StaticByName
{
  public static void main(String args[])
  {
    StaticDemo.callme();
    System.out.println("a = " + StaticDemo.a);
  }
}
```

output:

```
a = 42
```

➔In **calling sequence** of static variables, static methods and blocks : (Asst. Prof. Viral S. Patel)

1. Fist **static variables** statements execute
2. Then **static block** is execute
3. Then **static main method** is execute
4. Then **other static methods** are execute.

Example:

```
class UseStatic
{
   static int a = 3;
   static int b;
   static void meth(int x)
   {
     System.out.println("x = " + x);
     System.out.println("a = " + a);
     System.out.println("b = " + b);
   }
   static
   {
     System.out.println("Static block initialized.");
     b = a * 4;
   }
   public static void main(String args[])
   {
     meth(42);
   }
}
```

Output :

```
Static block initialized.
```

```
x = 42
a = 3
b = 12
```

As soon as the UseStatic class is loaded, all of the **static statements** are run. First, **a** is set to **3**, then the **static block** executes (printing a message), and finally, **b** is initialized to **a * 4 or 12**. Then **main( )** is called, which calls **meth( )**, passing **42 to x**.

Que. Explain Overloading and Overriding methods with example. (Prof. Viral S. Patel)
Ans.
Overloading :
In Java it is possible to define two or more methods within the same class that share the same name, as long as their parameter declarations are different. This methods are said to be overloaded and the process is referred to as method **overloading.**
➔Overloaded methods must **differ in the type and/or number of parameters**.
➔The return type alone is insufficient to distinguish two versions of a method.

Example :

```
Class A
{
    void disp()
    {
       System.out.println("No parameters");
    }
    void disp(int a)
    {
       System.out.println("a: "+a);
    }
    void disp(int a, int b)
    {
       System.out.println("a and b: "+a+" "+b);
    }
    Void disp(double a)
    {
       System.out.println(a);
    }

}
```

```
class Overload
{

     public static void main(String args[])
     {
        A ob = new A();
        ob.disp();
        ob.disp(10);
        ob.disp(10, 20);
        ob.disp(123.25);
     }

}

Output :
No parameters
a: 10
a and b: 10 20
double a: 123.25
```

Overriding :
➔In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass, then the method in the subclass is said to override the method in the superclass.
➔When an overridden method is called from within a subclass, it will always refer to the version of that method defined by the subclass. The version of the method defined by the superclass will be hidden. (Asst. Prof. Viral S. Patel)
➔ If we need to access the superclass version of an overridden function, we can do so by using **super keyword which is resolved at compile time**. **Dynamic method dispatch** is the mechanism by which a call to an overridden method is **resolved at run time**, rather than compile time.

Example :

```
class A
{
    void show()
    {
     System.out.println("SUPER A”);
    }
}

class B extends A
{
    void show()
    {
     System.out.println("SUB B”);
    }
}

class C extends A
{
    void show()
    {
    super.show(); // this calls A's show()
    System.out.println("SUB C”);
    }
}
```

```
class Override
{
  public static void main(String args[])
  {
    A a = new A(); // object of type A
    B b = new B();
    C c = new C();
    b.show(); // this calls show() in B
    c.show(); // this calls show() in C
    A r;        // obtain a reference of type A
    r = a;      // r refers to an A object
    r.show(); // calls A's version of show()
    r = b;      // r refers to a B object
    r.show(); // calls B's version of show()
  }
}
Output :
SUB B   // by overriding method
SUPER A // solved by super key word
SUB C
SUPER A // by using run time polymorphism
SUB B   // by using run time polymorphism
```

Que. Explain dynamic method dispatch with example.
Or Explain run-time polymorphism.
Ans. Overridden methods in Java are similar to virtual functions in C++ language.
➔Dynamic method dispatch is the mechanism by which a call to an **overridden method is resolved at run time**, rather than compile time.
➔Dynamic method dispatch is important because this is how Java implements **run-time polymorphism**. (Asst. Prof. Viral S. Patel)
➔ A **superclass reference** variable can **refer to a subclass object**. Java uses this fact to resolve calls to overridden methods at run time.
➔When an overridden method is called through a superclass reference, Java determines which version of that **method to execute based upon the type of the object being referred** to at the time the call occurs.

Example :

```
class A
{
  void callme()
  {
  System.out.println("Inside A's");
  }
}

class B extends A
{
   void callme()
  {
   System.out.println("Inside B's");
  }
}

class C extends A
{
  void callme()
  {
```

```
class Dispatch
{
  public static void main(String args[])
  {
    A a = new A(); // object of type A
    B b = new B(); // object of type B
    C c = new C(); // object of type C
    A r;        // obtain a reference of type A
    r = a;      // r refers to an A object
    r.callme(); // calls A's version of callme
    r = b;      // r refers to a B object
    r.callme(); // calls B's version of callme
    r = c;      // r refers to a C object
    r.callme(); // calls C's version of callme
  }
}
```

Output :
```
Inside A’s
```

```
  System.out.println("Inside C's");
  }
}
```

```
Inside B's
Inside C's
```

➔This program creates one superclass called A and two subclasses of it, called B and C. Subclasses B and C override callme( ) declared in A. Inside the main( ) method, objects of type A, B, and C are declared. Also, a reference of type A, called r, is declared. The program then assigns a reference to each type of object to r and uses that reference to invoke callme( ). As the output shows, the version of callme( ) executed is determined by the type of object being referred to at the time of the call.

Que. What is Native code ? What is Native Method in Java?
Ans.
➔Native code is known as machine code to run with a particular processor and its set of instructions.
➔A ***native method*** is a Java method (either an instance method or a class method) whose implementation is written in another programming language such as C. We can integrate native methods into Java code. These methods can be called from inside your Java program just as you call any other Java method. (Asst. Prof. Viral S. Patel)
➔**For example:**
public native int meth() ;

After we declare a native method, we must write the native method and follow some complex series of steps to link it with Java code. Most native methods are written in C. The mechanism used to integrate C code. (Asst. Prof. Viral S. Patel)

Native methods seem to offer great promise, because they enable you to gain access to your existing base of library routines, and they offer the possibility of faster run-time execution. But native methods also introduce two significant problems:
■ **Potential security risk** Because a native method executes actual machine code, it can gain access to any part of the host system. That is, native code is not confined to the Java execution environment. This could allow a virus infection, for example. For this reason, applets cannot use native methods. Also, the loading of DLLs can be restricted, and their loading is subject to the approval of the security manager.
■ **Loss of portability** Because the native code is contained in a DLL, it must be present on the machine that is executing the Java program. Further, because each native method is CPU- and operating-system-dependent, each DLL is inherently nonportable. Thus, a Java application that uses native methods will be able to run only on a machine for which a compatible DLL has been installed.

The use of native methods should be restricted, because they render your Java programs nonportable and pose significant security risks.

Que. What is JDK ?
Ans. The Java Development Kit (JDK) is a software development environment used for developing Java applications and applets. (Asst. Prof. Viral S. Patel)
It includes JRE (Java Runtime Environment), APIs (Application programming Interface) and tools needed in Java development some of them like :
➔appletviewer (for viewing java applets)
➔javac (Java compiler)

➔java (Java interpreter),
➔javap (Java disassembler),
➔javah (for c header files),
➔jar (Java archiver),
➔javadoc (for creating HTML documents)
➔jdb (Java debugger)
etc.

Que. What is API ?
Ans. Java application programming interface (API) is also known as Java Standard Library (JSL). API is a list of all classes that are part of the Java development kit (JDK). It includes all Java packages, classes, and interfaces, along with their methods, fields, and constructors. These pre-written classes provide a tremendous amount of functionality to a programmer.
Most commonly used packages are :
➔**Language Support Package:** A collection of classes and methods required for implementing basic features of java.
➔**Utilities Package:** A collection of classes to provide utility functions such as date and time functions.
➔**Input/Output Package:** A collection of classes required for input/output manipulation.
➔**Networking Package:** A collection of classes for communicating with other computers via Internet. (Asst. Prof. Viral S. Patel)
➔**AWT Package:** The Abstract Window Tool Kit package contains classes that implements platform-independent graphical user interface.
➔**Applet Package:** This includes a set of classes that allows us to create Java applets.

Que. Explain how to force the garbage collection in Java.
Ans. First of all Garbage collection is an automatic process and **can't be forced**. Although, you can request it by calling System.gc(). JVM does not guarantee that GC will be started immediately. (Every class inherits finalize() method from java.lang.Object. The finalize() method is called by garbage collector when it determines no more references to the object exists.) You can sent request to recycle the unused objects by calling **System.gc()** and **Runtime.gc()** , but there is no guarantee when all the objects will garbage collected.

Que. How to set CLASSPATH ?
Ans. There are two ways to set CLASSPATH.

1) ➔Right click on computer/mycomputer/system icon.
   ➔Click on Advanced System Settings and use Advanced window.
   ➔Click on Environment variables
   ➔Now we can use system / user variables. If the CLASSPATH already exists in System Variables, click on the edit button then put a semicolon (;) at the end. Paste the Path of class files which we want. If the CLASSPATH doesn't exist in System/User Variables, then click on the new button and type variable name as CLASSPATH and give variable value as path of class files which we want and click on OK. (Asst. Prof. Viral S. Patel)

2) Open command Prompt and write following.
   set CLASSPATH =%CLASSPATH%;D:\sybca\java\;

Here we assume that our class files are in 'D:\sybca\java\' directory which we want. %CLASSPATH% means existing environment variable which will remain as it is and we can add our new path using semicolon(;).

Que. Explain Lexical Issues in JAVA.

OR

Explain Java Tokens.

Ans. Java program is a collection of **tokens, comments and white spaces.**

WHITE SPACES:

Java is a free-form language. This means that you do not need to follow any special indentation rules. For example, the Example program could have been written all on one line or in any other strange way you felt like typing it, as long as there was at least one whitespace character between each token that was not already delineated by an operator or separator. In Java, whitespace is a space, tab, or newline. (Asst. Prof. Viral S. Patel)

COMMENTS :

There are three types of comments defined by Java. We have already seen two: single-line and multiline. The third type is called a documentation comment. This type of comment is used to produce an HTML file that documents your program. The documentation comment begins with a /** and ends with a */.

TOKENS:

➔Java language includes **five types of tokens.**

- Keywords
- Identifiers
- Literals
- Operators
- Separators

➔**Keywords :** Java language has reserved 50 words as keywords. These keywords can combined with the operators and separators syntax and form the definition of the java language. These keywords **cannot be used as names for a variable, class or method.**

The keywords **const** and **goto** are reserved but not used. The **assert** keyword was added by Java 2, version 1.4. In addition to the keywords, java reserves the **true, false** and **null**. These words cannot be used for the names of variables, classes and so on.

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

➔**Identifiers :** Identifiers are used for class names, method names, and variable names. An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters. They must not begin with a number, lest they be confused with a numeric literal. Again, Java is case-sensitive, so VALUE is a different identifier than Value.
Some examples of **valid** identifiers are:
AvgTemp count a4 $test this_is_ok
Invalid variable names include:
2count high-temp Not/ok

➔**Literals :** A constant value in Java is created by using a literal representation of it.
For example, here are some literals: **100 98.6 'X' "This is a test"**
Left to right, the first literal specifies an integer, the next is a floating-point value, the third is a character constant, and the last is a string. A literal can be used anywhere a value of its type is allowed (Asst. Prof. Viral S. Patel)

➔**Seperators :** In Java, there are a few characters that are used as separators. The most commonly used separator in Java is the semicolon. It is used to terminate statements. The separators are shown in the following :

( )	Parentheses
{ }	Braces
[ ]	Brackets
;	Semicolon
,	Comma
.	Period

➔**Operators :** Java provides operators can be divided into the following four groups: **arithmetic, bitwise, relational, and logical.**

Arithmetic Operators	**Bitwise Operators**
+ Addition	~ Bitwise unary NOT
– Subtraction	& Bitwise AND
* Multiplication	| Bitwise OR
/ Division	^ Bitwise exclusive OR
% Modulus	>> Shift right
Arithmetic Assignment Operators	>>> Shift right zero fill
+= Addition assignment	<< Shift left
–= Subtraction assignment	**Bitwise Assignment Operators**
*= Multiplication assignment	&= Bitwise AND assignment
/= Division assignment	|= Bitwise OR assignment
%= Modulus assignment	^= Bitwise exclusive OR assignment
Increment and decrement Operators	>>= Shift right assignment
++ Increment	>>>= Shift right zero fill assignment
– – Decrement	<<= Shift left assignment

Relational Operators	Logical Operators
== Equal to	|| Short-circuit OR
!= Not equal to	&& Short-circuit AND
> Greater than	! Logical unary NOT
< Less than	**Conditional Operators**
>= Greater than or equal to	?: Ternary if-then-else
<= Less than or equal to	

➔**Special Operators : Instanceof Operator and member selection operator (.)**

- **Instanceof Operator**
  The instanceof operator has this general form:
  object instanceof type
  Here, object is an instance of a class, and type is a class type. If object is of the specified THE JAVA LANGUAGE type or can be cast into the specified type, then the instanceof operator evaluates to true. Otherwise, its result is false.
  Thus, instanceof is the means by which your program can obtain run-time type information about an object. (Asst. Prof. Viral S. Patel)
- **member selection operator (.)**
  The dot operator (.) is used to access the instance variables and methods of class objects. Examples :
  Obj.a // obj access to the variable a
  Obj.disp() // obj call to the method disp()
  Dot operator also used to access classes and sub-packages from a package.

Que. What is difference between reference and pointer ?
Ans. An object reference is similar to a memory pointer. The main difference—and the key to Java's safety—is that you cannot manipulate references as you can actual pointers. Thus, you cannot cause an object reference to point to an arbitrary memory location or manipulate it like an integer. (Asst. Prof. Viral S. Patel)