1. INTRODUCTION TO JAVA

1. Features of Java:

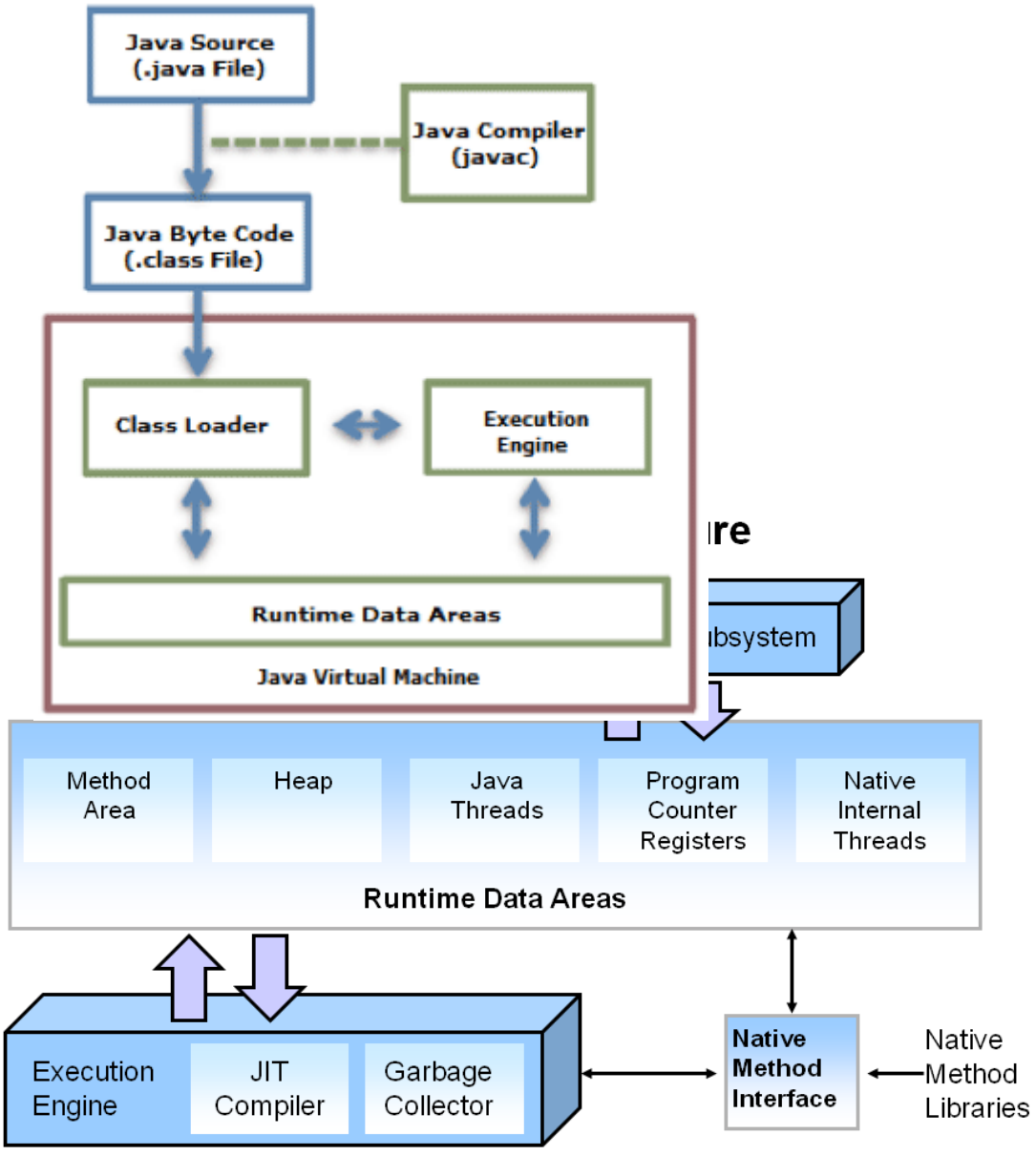
- **Object Oriented:** In Java, everything is an Object. Java can be easily extended since it is based on the Object model. Java inherits the Oops features like abstraction, encapsulation, inheritance, polymorphism.
- **Platform independent:** Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by virtual Machine (JVM) on whichever platform it is being run.
- **Simple:** Java is designed to be easy to learn. If you understand the basic concept of OOP we can easily become master in java.
- **Secure:** With Java's secure feature it enables to develop virus-free systems.
    - It allows all the Java programs to run inside the sandbox only and prevents many activities from untrusted resources including reading or writing to the local disk, creating any new process or even loading any new dynamic library while calling a native method.
    - Java never uses any kind of pointers. Java has its internal mechanism for memory management.
    - Byte code is another thing that makes Java more secure
- **Architectural-neutral:** Java compiler generates an architecture-neutral object file format which makes the compiled code to be executable on many processors, with the presence of Java runtime system.
- **Portable:** Java is portable because of being no implementation dependent aspects of the specification. Compiler in Java is written in ANSI C which makes it portable.
- **Robust:** Java is robust because of strong exception handling techniques and strong memory management. Java makes an effort to eliminate error prone situations by focussing on compile time error checking as well as runtime checking. Also Java has automatic garbage collection process which frees the memory by deleting no longer used objects.
- **Multithreaded:** With Java's multithreaded feature it is possible to write programs that can do many tasks simultaneously. This allows to minimize the workload of single thread and to minimize the execution time.
- **Interpreted:** Java byte code is translated on the fly to native machine instructions and is not stored anywhere.
- **High Performance:** With the use of Just-In-Time compilers, Java enables high performance. Because JIT executes the bytecode only on requirement basis.
- **Distributed:** Java is designed for the distributed environment of the internet where we can implement client server applications, also invoke remote methods.
- **Dynamic:** Java is considered to be more dynamic than C or C++ since it is designed to adapt

to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

2. Architecture of Java Virtual Machine:

Java is known as platform independant language because, when java compiler compiles the java code which resides in .java file, it will get converted to byte code which resides in .class file. This byte code can be executed on every platform(operating system) by JVM on that platform. Though JVM is implemented differently for different platforms, all JVMs can read same byte code.

HotSpot provides a Just-In-Time (JIT) compiler for bytecode. When a JIT compiler is part of the JVM, selected portions of bytecode are compiled into executable code in realtime, on a piece-by-piece, demand basis.



1. INTRODUCTION TO JAVA

3. Java Concepts:

- **Object** - Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors -wagging, barking, eating. An object is an instance of a class.
- **Class** - A class can be defined as a template/ blue print that describes the behaviors/states that object of its type support. It is a logical costruct.
- **Methods** - A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Instance Variables** - Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

4. Object Oriented Programming Principles:

Java is known as truely object oriented language. It is fully object oriented , but still primitive types are used as primitive for simplicity though in java we have Wrapper classes to use primitive types as objects. Java has following OOP features.

1. Abstraction:

Abstraction is the act of representing the data without knowing the background / internal details. It concerns with hiding the unnecessary details and showing only important part. In java it is achieved

via abstract classes and interfaces. It hides the complexity.
For example, When we drive a car we need not to worry about how internal parts are moving, how petrol/diesel flows through engine. We only know to move steers, to use gears and to use breaks. Thus abstraction hides functionality.

2. Encapsulation:
It is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse. In java, class is the basis of encapsulation. By making the members of class private, we can encapsulate the data in class. The private data can be accessed only through public methods of that class from outside. As abstraction hides functionality, encapsulation hides data.
For example, while drive a car we are not allowed to access the private parts of Car, like engine, but we have a public environment to set the car as we want, like we can use breaks as public enviroment to slow or stop the car.

3. Inheritance:
It is a mechanism where one class can acquire the properties of another class. Using inheritance, a class can inherit the general properties from its parent class and it has to define only its unique properties. This mechanism achieves code reusability.
For example, The company manufactures different versions of car but the car is extended from its previous version, so that new features will get added with original features remain same. In technical, java also has several classes which can be extended to add the new functionality and accesssing existing functionality.

4. Polymorphism:
Polymorphism means many forms where different actions are performed though one interface. In java polymorphism is achieved through overloading and overriding.
For example, If a dog smells cat it will bark and if a dog smells food, it salivates. Here 'smelling' is behaviour but whether to bark or salivate depends on type of parameter passed like cat or food. It is known as compile time polymorphism.
Another example, if a new car is replacing some functionality from exising car then, it will be dynamic polymorphism where existing functionality will get replaced with new functionality and bydefault, new functionality will get accessed.

5. Java Identifiers:
All Java components require names. Names used for classes, variables and methods are called identifiers.

In Java, there are several points to remember about identifiers. They are as follows:

- 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 key word 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

6. Java Keywords:

The following list shows the reserved words in Java.

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

Among the above, only java keywwords are : abstract, finally, extends, implements, import, new, package, strictfp, static, synchronized, throw, throws, transient, volatile

7. Java Data types:

There are two data types available in Java:

- Primitive Data Types
- Non-Primitive Data Types

1. Primitive Data Types:

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

byte:

- byte data type is an **8-bit** signed two's complement integer.
- 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 **2 bytes** (16-bit**)** signed two's complement integer.
- 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 **4 byets** (32-bit) signed two's complement integer.
- 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 **8 bytes** (64-bit) signed two's complement integer.
- This type is used when a wider range than int is needed.
- Default value is 0L.
- Example: long a = 100000L, int b = -200000L

float:

- float data type is a single-precision **4 bytes** (32-bit) IEEE 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 **8 bytes** (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 **2 bytes** (16-bit) Unicode character. Its size is more because java handles global character set.
- char data type is used to store any character.

- Example: char letterA ='A'

2. Non_Primitive Data Types:
There are again two types in non primitve data types

2.1 Derived data types:
Arrays:
Arrays are used to strore elements of simlilar data types. These are of fixed size. The elements will get stored in contiguous memory location.
Syntax:

datatype arrayname[]=new datatype[size];

Suppose we want to store the 10 integeres in array then,
int ar[]=new int[10];

2.2 User defined data types:
Enums:
Enums were introduced in Java5. Enums restrict a variable to have one of only a few predefined values. The values in this enumerated list are called enums. An enumeration is given a type name so that the elements of the enumeration can be referenced.
Syntax:

enum type-name { element-name, element-name, ... };

For example, if we consider an application for a fresh juice shop, it would be possible to restrict the glass size to small, medium and large. This would make sure that it would not allow anyone to order any size other than the small, medium or large.
Another example, we can access the days in a week through enums and we can restrct the day having only 7 values.

```
 enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
}
public class EnumDemo {
          Day day;

            public EnumDemo(Day day) {
               this.day = day;
            }

            public void worker() {
               switch (day) {
```

```
                case MONDAY:
                    System.out.println("Mondays are bad.");
                    break;

                case FRIDAY:
                    System.out.println("Fridays are better.");
                    break;

                case SATURDAY: case SUNDAY:
                    System.out.println("Weekends are best.");
                    break;

                default:
                    System.out.println("Midweek days are so-so.");
                    break;
            }
        }

    public static void main(String[] args) {
            EnumDemo firstDay = new EnumDemo(Day.MONDAY);
        firstDay.worker();
        EnumDemo thirdDay = new EnumDemo(Day.WEDNESDAY);
        thirdDay.worker();
        EnumDemo fifthDay = new EnumDemo(Day.FRIDAY);
        fifthDay.worker();
        EnumDemo sixthDay = new EnumDemo(Day.SATURDAY);
        sixthDay.worker();
        EnumDemo seventhDay = new EnumDemo(Day.SUNDAY);
        seventhDay.worker();
    }
}
```

Output:

Mondays are bad.
Midweek days are so-so.
Fridays are better.
Weekends are best.
Weekends are best.

Classes :

A class is a logical blue print which binds data and code together. User may have any class name.

Syntax:

```
class Address
{
        String street, sector, landmark, apartment;
        int flat_no, plot_no;
}
```

We can use the class name Address as data type.
Like, Address add;

8. Java Operators:

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups:

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Misc Operators

1. The Arithmetic Operators:

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators:

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

Operator	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
++	Increment - Increases the value of operand by 1	B++ gives 21
--	Decrement - Decreases the value of operand by 1	B-- gives 19

2. The Relational Operators:

There are following relational operators supported by Java language
Assume variable A holds 10 and variable B holds 20, then:

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values 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.

3. The Bitwise Operators:

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.

Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60; and b = 13; now in binary format they will be as follows:

a = 0011 1100

b = 0000 1101

---

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

The following table lists the bitwise operators:

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

Operator	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 either operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in	(A ^ B) will give 49 which is

	one operand but not both.	0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operan	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

4. The Logical Operators:

The following table lists the logical operators:

Assume Boolean variables A holds true and variable B holds false, then:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands are non-zero, then the 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.

5. The Assignment Operators:

There are following assignment operators supported by Java language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign 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 left operand and assign the	C *= A is equivalent to C = C * A

	result to left operand	
/=	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

6. Misc Operators:

There are few other operators supported by Java Language.

6.1 Conditional Operator (? :):

Conditional operator is also known as the ternary operator. 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. The operator is written as:

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 );
    }
}
```

Output:

```
Value of b is : 30
Value of b is : 20
```

9. Java Modifiers:

Like other languages, it is possible to modify classes, methods, etc., by using modifiers. There are two categories of modifiers:

1. Access Modifiers: define scope

Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The four access levels are:

- **private :** Visible to the class only.
- **default (no access modifier) :** Visible to the package, the default.
- **public :** Visible to the world.
- **protected :** Visible to the package and all subclasses from outside

Access By	private	package	protected	public
the class itself	yes	yes	yes	yes
a subclass in same package	no	yes	yes	yes
non-subclass in same package	no	yes	yes	yes
a subclass in other package	no	no	yes	yes
non-subclass in other package	no	no	no	yes

2. Non-access Modifiers: define behaviour

- **final :** classes, methods, variables

  final class – A class when set to final cannot be extended.
  final method – A method when set to final cannot be overridden.
  final variable – A final variable value cannot be changed. Final variable are like constants.

- **abstract :** classes and methods.

  abstract class – An abstract class may have abstract and concrete methods in it
  abstract method - It has only method signature. Its doesn't have any method body.If a clas has an abstract method, the class becomes an abstract class.

- **static :** methods, variables, nested classes, block.

  static variables – Single shared copy.
  static methods – Single shared copy.
  static nested classes – static members of other class
  local variables can not be made static

- **strictfp :** applicable to class and to a method .

  strictfp class- strictfp non access modifier forces floating point or floating point operation to adhere to IEEE 754 standard.

- **synchronized :** methods and blocks: used in thread synchronization

- **volatile :** variables: value of volatile variable is shared among threads and any thread can update its value

- **transient :** variables
  the value of transient instance variable of class can not be persisted to file.

- **native :** method
  native – native method indicates that a method is implemented on platform dependent code like in C or C++.

10. Java Variables:

We would see following type of variables in Java:

- Local Variables
- Instance Variables (Non-static variables)
- Class Variables (Static Variables)

1. Local variables: Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.

```
public class Circle {
        public void area() {
            int r=67;                           // local variable
            double areac=3.14*r*r;                      // areac – local variable
            System.out.println("Area of circle is:"+areac);
          }
}
```

2. Instance variables: Instance variables are variables within a class but outside any method. These variables are instantiated whenever new object is created. Each object has its own copy of instance variables. Instance variables can be accessed from inside any method, constructor or blocks of class using object of that class where it is declared.

```
public class Circle {
        int r=10;                               // instance variable
        public void area() {
            double areac=3.14*r*r;                      // areac – local variable
            System.out.println("Area of circle is:"+areac);
          }
}
```

In the above program, two objects of Employee are being created. Each employee has its own name

and salary. If any one's salary get changed it will not affect on other objects.

3. Class variables: Class variables are variables declared with in a class, outside any method, with the static keyword. These are loaded once in the memory area known as static context at the time of class loading. It has only one copy shared among all the objects.

Example:

```
public class Employee {
  public static  String DEPARTMENT = "Development ";     // static variable

 public void display() {
                System.out.println("Department: "+DEPARTMENT);
        }
}
```

11. Garbage Collection:

The garbage collection is the process where the objects that are no longer used in memory and the objects which are unrefered are deleted from the memory and memory is freed automatically. This process is executed periodically and automatically. Thus the automatic memory management is done. The garbage collector, call the finalize method to release the resources held by the object before the object being garbage collected.
The syntax of finalize method is

```
protected void finalize()
{
        // resource releasing code...
}
```

But we should not rely on finalize method to free the resources as we can not predict when this method will get called by the gc.

12. First Java Program:

```
public class  MyFirstJavaProgram {

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

1. Why main is declared as public static void main?
Explanation:
1. public : It is accessed from outside by JAVA VIRTUAL MACHINE(JVM).
2. static : main method should get loaded frist in memory as it is the entry point of execution. Also it should be accessed by JVM without using object of the class i.e. by using classname.
3. void : It does not return anytthing
4. main : name of the method

5. String args[] : It is a string array used to store command line arguments passed to main method while executing code. String is a class in Java.

2. Explain System.out.println
Explanation:
1. System : It is a final class in java.lang package.
2. out : It is a final static member of System class. It is of type PrintStream. PrintStream is a class.
3. println : It is a method of PrintStream class.

ASSIGNMENTS

1. Implement the program of addition, subtraction, division and multiplication of two numbers. Take the input from user.
2. Calculate the areas of different shapes using Switch Case.
3. What will be the output of following program?

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

                int num1 = 1;
                int num2 = 1;

                num1++;
                num2++;

                System.out.println("num1 = " + num1);
                System.out.println("num2 = " + num2);
          }
}
```

4. What will be the output of following program

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

           int num1 = 1;
           int num2 = 1;

           --num1;
           --num2;

           System.out.println("num1 = " + num1);
           System.out.println("num2 = " + num2);
        }
}
```

5. Implement a program on bitwise operators.

6. Implement a program on logical operators.
7. Find out given number is divisible by 5 as well as 7 or not.
8. Display the elements from an array which are divisible by 5 or 7.
9. Display the elements from array which are not divisible by 3.
10. Display the smallest element from an array.
11. Display the greatest element from an array.
12. Display the smallest even number from an array.
13. Display the greatest odd number from an array.
14. Write any program which shows the use of ternary operator.