Chapter 1: Introduction to Java

- The History and Evolution of Java
- Java's Lineage
    - The Birth of Modern Programming: C
    - C++: The Next Step
    - The Stage Is Set for Java
- The Creation of Java
- The Java Buzzwords
- The Evolution of Java

The History and Evolution of Java

To fully understand Java, one must understand the reasons behind its creation, the forces that shaped it, and the legacy that it inherits. Like the successful computer languages that came before, Java is a blend of the best elements of its rich heritage combined with the innovative concepts required by its unique mission.

This lecture explains how and why Java came about, what makes it so important, and how it has evolved over the years.

Although Java has become inseparably linked with the online environment of the Internet, it is important to remember that Java is first and foremost a programming language. Computer language innovation and development occurs for two fundamental reasons:

• To adapt to changing environments and uses
• To implement refinements and improvements in the art of programming

Java's Lineage

Java is related to C++, which is a direct descendant of C. Much of the character of Java is inherited from these two languages. From C, Java derives its syntax. Many of Java's object oriented features were influenced by C++.

The Birth of Modern Programming: C

Invented and first implemented by Dennis Ritchie on a DEC PDP-11 running the UNIX operating system, C was the result of a development process that started with an older language called BCPL, developed by Martin Richards. BCPL influenced a language called B, invented by Ken Thompson, which led to the development of C in the 1970s.

C was formally standardized in December 1989, when the American National Standards Institute (ANSI) standard for C was adopted.

C was designed, implemented, and developed by real, working programmers, reflecting the way that they approached the job of programming. Its features were honed, tested, thought about, and rethought by the people who actually used the language.

In short, C is a language designed by and for programmers.

C++: The Next Step

By the early 1980s, many projects were pushing the structured approach past its limits. To solve this problem, a new way to program was invented, called object-oriented programming (OOP). Object-oriented programming helps organize complex programs through the use of inheritance, encapsulation, and polymorphism.

C++ was invented by Bjarne Stroustrup in 1979, while he was working at Bell Laboratories in Murray Hill, New Jersey. Stroustrup initially called the new language "C with Classes." However, in 1983, the name was changed to C++. C++ extends C by adding object-oriented features. Because C++ is built on the foundation of C, it includes all of C's features, attributes, and benefits.

The Stage Is Set for Java

By the end of the 1980s and the early 1990s, object-oriented programming using C++ took hold. Indeed, for a brief moment it seemed as if programmers had finally found the perfect language. Because C++ blended the high efficiency and stylistic elements of C with the object-oriented paradigm, it was a language that could be used to create a wide range of programs. However, just as in the past, forces were brewing that would, once again, drive computer language evolution forward. Within a few years, the World Wide Web and the Internet would reach critical mass. This event would precipitate another revolution in programming.

The Creation of Java

Java was conceived by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems, Inc. in 1991. It took 18 months to develop the first working version. This language was initially called "Oak," but was renamed "Java" in 1995. Between the initial implementation of Oak in the fall of 1992 and the public announcement of Java in the spring of 1995, many more people contributed to the design and evolution of the language. Bill Joy, Arthur van Hoff, Jonathan Payne, Frank Yellin, and Tim Lindholm were key contributors to the maturing of the original prototype.

Somewhat surprisingly, the original impetus for Java was not the Internet! Instead, the primary motivation was the need for a platform-independent (that is, architecture-neutral) language that could be used to create software to be embedded in various consumer electronic devices, such as microwave ovens and remote controls.

About the time that the details of Java were being worked out, a second, and ultimately more important, factor was emerging that would play a crucial role in the future of Java. This second force was, of course, the World Wide Web. Had the Web not taken shape at about the same time that Java was being implemented, Java might have remained a useful but obscure language for programming consumer electronics. However, with the emergence of the World Wide Web, Java was propelled to the forefront of computer language design, because the Web, too, demanded portable programs.

By 1993, it became obvious to members of the Java design team that the problems of portability frequently encountered when creating code for embedded controllers are also found when attempting to create code for the Internet.

So, while the desire for an architecture-neutral programming language provided the initial spark, the Internet ultimately led to Java's large-scale success.

The Java Buzzwords

Following are the features or buzzwords of Java language which made it popular:

1. Simple
2. Portable

3. Powerful
4. Secure and Robust
5. Dynamic
6. Distributed
7. Multithreaded
8. True OOPS
9. Compiled and Interpreted, yet High Performance
10. Platform Independent

Simple:

- Java easy to learn
- It is easy to write programs using Java
- Expressiveness is more in Java.
- Most of the complex or confusing features in C++ are removed in Java like pointers etc.

Portable:

- Applications written using Java are portable in the sense that they can be executed on any kind of computer containing any CPU or any operating system.
- When an application written in Java is compiled, it generates an intermediate code file called as “bytecode”.
- Bytecode helps Java to achieve portability.
- This bytecode can be taken to any computer and executed directly.

Secure:

- Java does not provide pointers, programs written in Java does not interact with hardware directly.
- Programs written in Java are executed within the JVM which protects from unauthorized or illegal access to system resources.

Robust:

- An application is said to be robust(reliable) when it is able to give some response in any kind of context.
- Java’s features help to make the programs robust. Some of those features are:
    1. Type checking
    2. Exception handling

Dynamic:

- The Java Virtual Machine (JVM) maintains a lot of runtime information about the program and the objects in the program.
- Libraries are dynamically linked during runtime.
- Even if you make dynamic changes to pieces of code, the program is not affected.

Distributed:

- Java supports distributed computation using Remote Method Invocation (RMI) concept.
- The server and client(s) can communicate with another and the computations can be divided among several computers which makes the programs to execute rapidly.
- In distributed systems, resources are shared.

Multithreaded:

- Java supports multithreading which is not supported by C and C++.
- A thread is a light weight process.
- Multithreading increases CPU efficiency.
- A program can be divided into several threads and each thread can be executed concurrently or in parallel with the other threads.
- Real world example for multithreading is computer. While we are listening to music, at the same time we can write in a word document or play a game.

Object - Oriented:

- Java follows object oriented model.
- It supports all the features of object oriented model like:
    1. Encapsulation
    2. Inheritance
    3. Polymorphism
    4. Abstraction

Interpreted and High Performance:

- In Java 1.0 version there is an interpreter for executing the bytecode. As interpreter is quite slow when compared to a compiler, java programs used to execute slowly.
- After Java 1.0 version the interpreter was replaced with JIT(Just-In-Time) compiler.
- JIT compiler uses Sun Microsystem's Hot Spot technology.
- JIT compiler converts the byte code into machine code piece by piece and caches them for future use.
- This enhances the program performance means it executes rapidly.

Platform Independent/Architecture - Neutral:

- Bytecode can be executed on computers having any kind of operating system.
- Since Java applications can run on any kind of platform, Java is architecture – neutral.

<u>The Evolution of Java</u>

Java Version History

There are many java versions that have been released. Current stable release of Java is Java SE 9.

JDK Alpha and Beta (1995)
JDK 1.0 (23rd Jan, 1996)
JDK 1.1 (19th Feb, 1997)
J2SE 1.2 (8th Dec, 1998)
J2SE 1.3 (8th May, 2000)
J2SE 1.4 (6th Feb, 2002)
J2SE 5.0 (30th Sep, 2004)
Java SE 6 (11th Dec, 2006)
Java SE 7 (28th July, 2011)
Java SE 8 (18th March, 2014)
Java SE 9 (21st September, 2017)
Java SE 10 (20th March, 2018)
Java SE 11 (25th September, 2018)
Java SE 12 (19th March, 2019)
Java SE 13 (17th September, 2019)

Java SE 14 (17th March, 2020)
Java SE 15 (15th September, 2020)
Java SE 16 (16th March, 2021)
Java SE 17 (14th September, 2021) - LTS
Java SE 18 (22nd March, 2022)
Java SE 19 (20th September, 2022)
Java SE 20 (21st March, 2023)
Java SE 21 (19th September, 2023)
Java SE 22 (19th March, 2024)
Java SE 23 (17th September, 2024)
Java SE 24 (18th March, 2025)

Theory Questions

1. When was Java introduced? Give a detailed description about the history of Java.
2. List down the various technologies and editions of Java.
3. What are the buzzwords/characteristics of Java?

Chapter 2: Sample Program- Steps to Create and Execute

- Software Requirements
- Sample Program
- Steps to create and execute
- Troubleshooting

Software Requirements

JDK: Java Development Kit (1.21)

Tools	Command
Compiler	javac
Interpreter	java
Debugger	jdb
Documentation tool	javadoc
Jar utility	jar
Java API	

Editor: Notepad, gEdit, Subline, Notepad++, Visual Code
OR
Integrated Development Environment (IDE) : Eclispe IDE, NetBeans IDE, BlueJ, JCreator

Sample Program

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

// My First Java Program

Explanation:

```
class <ClassName> {
                //Body
}
```

public: It can called from anywhere
static : No object is required
void : Does not return any value
main : Name of the function
String args[] : Command line arguments, data type is string array []
System.out.println("Message and New Line");
System.out.print("Message");
\n \t \v \a \\ \" \'

Types of Comments in Java

// Single line comments

```
/*
*Multiline comments
*/

/**
* Documentation comments
*/
```

<u>Steps to create and execute</u>

1. Open Notepad. (Start | Run | Notepad)
2. Type the Program.
3. Save it:
    a. Class name and file name should be same
    b. Extension should be .java
    c. It is case-sensitive.
    Example: C:\Data\First.java
4. Compile it:
    a. Go to Command/DOS prompt
        - Start | Run | Cmd
```
C:\windows\desktop>cd\
C:\> cd Data
C:\Data>javac First.java
```
    Syntax: javac FileName.java
5. Execute it/interpret it
```
C:\Data>java First
```
    Syntax: java ClassName

<u>Troubleshooting</u>

```
C:\Data>javac First.java
Not an internal or external DOS command
```

Problem: My OS does not recognizes javac
Solution: You need to set the path. Path specifies OS the location of the compiler and interpreter.
1. Search for “javac.exe” within My Computer (in case not found, install jdk)
2. Select javac.exe | properties, select and copy the Location
<u>Example</u>: C:\Program Files\Java\jdk-21\bin
3. Go to My Computer | Properties | Advanced | Environment Variables | System Variables
4. Select path | Edit and at the end paste the path after ;
    Variable Name: path
    Variable Value: ....; C:\Program Files\Java\jdk-21\bin
5. Select OK and Restart the computer or open another command prompt.

Chapter 3: Data Types, Variables, Literals, Type Casting, Assignments, Operator

- Data Types
- Variables
- Literals
- Type Casting
- Assignments
- Operators

Data Types

- Data Types in a programming language specifies the type of data to be stored and the size of memory to be allocated.
- Data Types can be classified as:
  1. Fundamental Data Types
  2. Reference Types

Fundamental Data Types

- Fundamental data types are the data types that the compiler understands.
- These are also referred as primitive, pre-defined or built in data types.

Data Types	Size in Bytes	Size in Bits	Range
char	2	16	0 to 65,535
byte	1	8	–128 to 127
short	2	16	–32,768 to 32,767
int	4	32	–2,147,483,648 to 2,147,483,647
long	8	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4	32	4.9e–324 to 1.8e+308
double	8	64	1.4e–045 to 3.4e+038
boolean	virtual machine dependent		true, false

Variables

- Variables are temporary memory locations used to store a value.
- The value stored in a variable can change/vary.
- Variable naming rules:
  - It should start with an alphabet, underscore (_) or dollar ($)
  - It can contain alphabets, digits, underscore (_) or dollar ($)
  - No spaces, no special characters except underscore (_) and dollar ($)
  - It should not be a keyword
  - It should be unique within its scope
- Variable naming conventions:
  - Variables in Java make use of camelCase.
    Example: my**V**ar, one**T**wo, one**T**wo**T**hree

Variable Declaration

char c;
int num;

Variable Initialization

c = ‘a’;
num = 10;

Variable Declaration and Initialization

char c = ‘a’;

int num = 10;

Literals

Literals are used to specify data type of a value.

Literals are not used with variables.

There are no literals for byte and short.

Literals are case insensitive.

Literals values for all primitive data types

```
24                          //int
124.543                     // double
true                        // boolean
'v'                         //char
```

Integer Literals

An integer literal can be represented in the following forms:

- Decimal (base 10)
- Octal (base 8)
- Hexadecimal (base 16)

Decimal Literals

int num = 21, 99_123;

There is no prefix or suffix

Octal Literals

- Numbers can be from 0 to 7
- Prefix used is 0

int num = 05;

Hexadecimal Literals

- Numbers can be from 0 to 9, a to f
- Prefix used is 0x

NOTE: a to f is not case sensitive, a is equal t o A

```
int a = 0x5;
int b = 0xF;
long a = 05l;               // long
long b = 0xFl;              // long
```

Floating Point Literals

```
float pi = 3.14;                    // Incorrect
float pi = 3.14f;
double pi = 3.14;
double pi = 3.14d;
```

Boolean Literals

```
boolean married = true;                 // Correct
boolean married = 1;                          // Incorrect

if(1) {                                       // Incorrect
        // statements
}
```

Character Literals

```
char c = 'A';
char value = 'y';
char newline = '\n';
```

NOTE: Character literals can also be written in unicode

```
char c = '\u004A';
```

String Literals

```
String str = "Hello World";
String str = "Hell\u0001 W\u0001rld";
```

<u>TypeCasting</u>

TypeCasting is converting from one data type to another data type
It can be inform of:

1. **Narrowing/Explicit**: Assigning larger data types to smaller data types
2. **Widening/Implicit**: Assigning smaller data types to larger data types

```
int i = 10;
long j = 20;
i = j;                              // Incorrect
i = (int) j;                        // Correct
j = i;                              // Correct
j = (long) i;                       // Correct
```

<u>Assignments</u>

Rule 1: Precedence (double, float, long, int)

```
class Test {
        public static void main(String args[]) {
                byte b1 = 10;
                byte b2 = 20;
                byte b3 = b1 + b2;
                System.out.println(b3);
        }
}
```

Rule 2: Range Rule (Applicable on byte, short and char)

```
class Test {
        public static void main(String args[]) {
                char b = 65536;
                System.out.println(b);
        }
}
```

Rule 3: Shortcut Assignment Operators (+= , -=)

- Range rules does not applies
- if outside range, it becomes cyclic

```
class Test {
        public static void main(String args[]) {
                byte b = 122;
                b += 6;
                System.out.println(b);    //-1 28
        }
}
```

Rule 4: Variables are evaluated at runtime, whereas values are evaluated at compiletime

Error:

```
class Test {
        public static void main(String args[]) {
                byte b, b1 = 10;
                b = b1 + b1;
                System.out.println(b);
        }
}
```

Will Compile:

```
class Test {
        public static void main(String args[]) {
                byte b;
                b = 10 + 10;
                System.out.println(b);
        }
}
```

<u>Operators</u>

An operator is a symbol that allows a programmer to perform certain arithmetic or logical operations on data and variables.

1. Arithmetic (*,/,%,+,-,++,--)
2. Assignment (= , +=, -=, *=, /= , %=)
3. Relations (==, !=, <, > , <= , >=)
4. Instance of (instanceof)
5. Logical (&, |, ^, !, &&, ||)
6. Conditional (? :)

Example:

```
class Op1 {
        public static void main(String args[]) {
                int a = 5;
                int b = 3;
                int c = 0;
                c = a + b;
                System.out.println(c);
                c = a - b;
                System.out.println(c);
                c = a * b;
                System.out.println(c);
                c = a / b;
                System.out.println(c);
                c = a % b;
                System.out.println(c);
        }
}
```

Example:

```
class Op2 {
        public static void main(String args[]) {
                int a = 6, b = 5;
                a = b++;
                System.out.println(a);
                System.out.println(b);
                a = ++b;
                System.out.println(a);
                System.out.println(b);
                b++;
                System.out.println(b);
                System.out.println(b++);
        }
}
```

Example:

```
class Op3 {
        public static void main(String args[]) {
                int a = 5;
                int b = 6;
                if ((b == a) && (a < ++b))
                        System.out.println("Here");
                System.out.println(" a : " + a + " b : " + b);
        }
}
```

Example:

```
class Op4 {
        public static void main(String args[]) {
                String str = " ";
                int a = 333;
                int b = 666;
                System.out.println(str + a + b);
                System.out.println(str + (a + b));
```

```
		System.out.println(a + b + str);
		str += 10;
		System.out.println(str);
	}
}
```

Example:

```
class Op5 {
	public static void main(String args[]) {
		int a = 10;
		a = a + 5 * 15;
		System.out.println(a);
	}
}
```

Example:

```
class Op6 {
	public static void main(String args[]) {
		String str = "Welcome";
		str += " to ";
		str += "Java";
		System.out.println(str);
	}
}
```

Example:

```
class Op7 {
	public static void main(String args[]) {
		int a = 10;
		int b = 20;
		int c = 30;
		int d = a > b & a > c ? a : b > c ? b : c;
		System.out.println(d);
	}
}
```

Example:

```
class Op8 {
	public static void main(String args[]) {
		int a = 9;
		float b = 9.0f;
		double c = 9.0;
		if (a == b && a == c)
			System.out.println("Equal");
		else
			System.out.println("Not Equal");
	}
}
```

Example:

```
class Op9 {
        public static void main(String args[]) {
                int a = 65;
                char b = 'A';
                System.out.println(b);
                if (a == b)
                        System.out.println("Equal");
                else
                        System.out.println("Not Equal");
        }
}
```

<u>Assignment</u>

Theory Questions

1. How are data types classified? Explain the fundamental data types that Java provides.
2. What are variables? Explain variable naming rules and conventions.
3. What is type casting? Explain its types.
4. What are literals? Explain the types of literals that Java provides.
5. List the various operators that Java supports.

Chapter 4: Input and Output, Conditional and Looping Structures

- Scanner Class
- Conditional Structures – if and switch
- Looping Structures - while, do while, for, enhanced for
- Jumping Statements

Scanner Class

- To take input from user “Scanner” class is used
- This “Scanner” class comes from java.util package
- To use “Scanner” class in Java program, statement is
  - import java.util.Scanner;
- To associate Scanner to keyboard (create an object)
  Scanner a = new Scanner(System.in);

To accept a word
String word = a.next();

To accept a line
String line = a.nextLine();

To accept a character
char c = a.nextLine().charAt(0); or char c = a.next().charAt(0);

To accept an integer
int num = a.nextInt();

To accept a float
float fp = a.nextFloat();

To accept a double
double d = a.nextDouble();

Write a program to accept your name and address and display it on Screen

```
import java.util.Scanner;
class Test1 {
        public static void main(String args[]) {
                String name, add;
                Scanner a = new Scanner(System.in);
                System.out.println("Enter Name :");
                name  = a.nextLine();
                System.out.println("Enter Address :");
                add  = a.nextLine();
                System.out.println("Name is : "  + name);
                System.out.println("Address is : " + add);
        }
}
```

Write a program to radius of a circle and print its area and circumference

```
import java.util.Scanner;
class Test2 {
	public static void main(String args[]) {
		float radius, area = 0,cir = 0;
		Scanner a = new Scanner(System.in);
		System.out.println("Enter Radius :");
		radius  = a.nextFloat();
		area = 3.14f * radius * radius;
		// area = (float) (Math.PI * Math.pow(radius, 2));
		cir = 2 * 3.14f * radius;
		System.out.println("Area is : " + area);
		System.out.println("Circumference is : " + cir);
	}
}
```

Conditional / Decision / Selection Structures

- To select single out of multiple options
- Like C/C++ even Java provides:
  1. if
  2. switch
- if cane be used in 3 ways

```
if(<cond>) {
//T-Part
}
```

```
if(<cond1>) {
//T-Part
}
else {
//F-Part
}
```

```
if(<cond1>) {
//T-Part of cond1
}
else if (<cond2>) {
//T-Part of con2
}
:
else {
//F-Part
}
```

```
switch(<variable>) {
	case <value1> : // T-Part of Value 1
			break;
	case <value2> : // T-Part of Value 2
			break;
	case <value3> : // T-Part of Value 3
			break;
	:
	default  :		// F-Part
}
```

Write a program to accept a character and print whether it is upper case, lower case or a special character

```
import java.util.Scanner;
class Test3 {
	public static void main(String args[]) {
		char c;
		Scanner a = new Scanner(System.in);
		System.out.println("Enter Character :");
		c = a.nextLine().charAt(0);
		if(c >= 65 && c <= 90)
			System.out.println("Upper Case");
		else if(c >= 'a' && c <= 'z')
			System.out.println("Lower Case");
		else
			System.out.println("Special");
	}
}
```

Write a program to accept a character and print whether it is a vowel or a consonant

```
import java.util.Scanner;
class Test4 {
	public static void main(String args[]) {
		char c;
		Scanner a = new Scanner(System.in);
		System.out.println("Enter Character :");
		c = a.nextLine().charAt(0);
		switch(c) {
			case 'a' : System.out.println("Vowel a.");
				break;
			case 'e' : System.out.println("Vowel e.");
				break;
			case 'i' : System.out.println("Vowel i.");
				break;
			case 'o' : System.out.println("Vowel o.");
				break;
			case 'u' : System.out.println("Vowel u.");
				break;
			default : System.out.println("It is a consonant.");
		}
	}
}
```

__**Looping/Repetitive Structures**__

It is used to repeat set of statements or block of code

Like C/C++, even Java provides:

1. while
2. do ... while
3. for

```
//initialization
while(<cond>) {
//statements
//re-initialization
}
```

```
//initialization
do {
//statements
//re-initialization
} while(<cond>);
```

```
for(<initialization>;<condition>;<re-init>) {
//statements
}
```

Parts of looping structure:

1. Initialization : Defines the starting value
2. Condition: Specifies the ending value
3. Re-Initialization : Defines the step value

```
class Test4 {
	public static void main(String args[]) {
		int i;
		i = 1;
		while(i <= 100) {
			System.out.print("\t" + i) ;
			i++;
		}
		do {
			System.out.print("\t" + i);
			i++;
		} while( i<= 200);
		for(;i = 300;i++)
			System.out.print("\t" + i);
	}
}
```

Enhanced For

- it was introduced with jdk5
- it is used to enumerate arrays and collections

```
class Test {
	public static void main(String args[]) {
		int num[] = {10, 20, 30, 40, 50};
		for(int x : num)
			System.out.println(x);
		String weekDays[] = { “Monday” ,  “Tuesday”, “Wednesday”, “Thursday”, “Friday”};
		for(String w : weekDays)
			System.out.println(w);
	}
}
```

Jumping Statements

- break
- continue
- return

Example:

```
class Jump1 {
	public static void main(String args[]) {
	int j = 0;
	while(true) {
		j = j + 2;
		if(j == 6) {
			break;
		}
```

```
                    System.out.println(j);
                    }
        }
}
```

Example:

```
class Jump2 {
        public static void main(String args[]) {
                int j, k;
                abc:
                for(j = 0 ; j < 5 ; j++) {
                        System.out.println("j= " + j);
                        for( k = 2 ; k < 5 ; k++) {
                                System.out.println("k= " + k);
                                if( j == k) {
                                        break  abc;
                                }
                        }
                }
        }
}       // abc: (label) should be immediate before the loop
```

Example:

```
class Jump3 {
        public static void main(String args[]) {
                for(int i = 2; i <= 10 ; i+=2) {
                        if( i == 6) {
                                continue;
                        }
                        System.out.println(i);
                }
        }
}
```

Example:

```
class Jump4 {
        public static void main(String args[]) {
                int x , y;
                skip:
                for( x = 2 ; x  < 4 ; x++) {
                        System.out.println("x = " + x);
                        for( y = 2 ; y < 6 ; y++) {
                                System.out.println("y = " + y);
                                if(x == y) {
                                        continue skip;
                                }
                        }
                }
        }
}
```

Example:

```
class Jump5 {
	public static void main(String args[]) {
		int input = 10;
		int result = input * input;
		System.out.println(result);
		return;
	}
}
```

<u>Assignment</u>

Theory Questions

1. What is the software requirement for Java? Name some of the IDEs used for Java development.
2. Why to be set the PATH variable? How do we set PATH for Java?
3. What is the different between byte code and machine code?

Practical Questions

1. Write a program that displays **Hello World** on screen.
2. Write a program that displays your name on screen within double quotes.
3. Write a program that accepts your name, address and mobile number and display it on screen.
4. Write a program that accepts radius of a circle and display its area and circumference.
5. Write a program that accepts principle, rate and time and display simple and compound interest.
6. Write a program that accepts width and height of a rectangle and display its area and perimeter.
7. Write a program that accepts temperature in Centigrade and display it in Fahrenheit.
   (F=C*9/5+32)
8. Write a program that accepts 2 numbers and display their sum, difference, product, and quotient.
9. Write a program that accepts a character and print whether it is an uppercase, lowercase, digit or a special character.
10. Write a program that accepts a character and print whether it is an uppercase, lowercase or a special character without using logical operators.
11. Write a program that accepts an integer and displays the number of digits in that integer.
12. Write a program that accepts a character and print whether it is a vowel or a consonant.
13. Write a program that displays even numbers from 100 to 200.
14. Write a program that displays multiples of 3 from 300 to 100.
15. Write a program that displays sum and average of multiples of 5 from 500 to 600.
16. Write a program that accepts a number and print its table.
17. Write a program that accepts a number and print its reverse multiplication table.
18. Write a program that accepts a number and print whether it is a prime number or a composite number.
19. Write a program that display prime numbers from 100 to 200.
20. Write a program that display Fibonacci series of 20 elements.