Instructional Material

For

COMP 20023
Computer Programming 1

Compiled by:

Michael Anjelo O. Miguel, MIT

Overview

This instructional material introduces the students to the fundamentals of logic formulation and allows the student to learn and apply the art and style of procedural programming to solve computational problems adhering to the standards and guidelines of documentation. It includes discussion on I/O statements, loop and branching instructions, and creating functions and procedures.

The course covers the use of general purpose programming language to solve problems. The emphasis is to train students to design, implement, test, and debug programs intended to solve computing problems using fundamentals constructs.

Course Outcomes

Design, implement, test and debug a program, based on a given specification that uses such of the following fundamental programming components: (1) primitive data types (2) basic computation (3) simple I/O (4) conditional and iterative structures (5) definition of functions and parameter

Assess and recommend revisions to another programmer's code (1) regarding documentation and program style standards that contribute to readability and maintainability of software, (2) regarding appropriateness of chosen conditional and iterative constructs given a programming task, and (3) regarding thoroughness in applying procedural abstraction.

Assessment

Assessment is the systematic basis for making inferences about the learning and development of students. It is the process of defining, selecting, designing, collecting, analyzing, interpreting, and using information to increase students' learning and development.

Herewith, assessment is required at the end of every chapter in this instructional material.

Grading System

Midterm Grade = 70% Class Standing (Seatwork, Quizzes; Research, Exercises & Projects) + 30% Midterm Examination

Finals Grade = 70% Class Standing (Seatwork, Quizzes; Research, Exercises & Projects) + 30% Finals Examination

Final Grade = (Midterm Grade + Finals Grade) / 2

Table of Contents

Programming Concepts

Learning Outcomes:

After successful completion of this lesson, you should be able to:

- Apply steps in program development.
- Design algorithms.
- Determine the Importance of Java
- Identify Applications in Java
- Familiarize with the process of Java

Introduction

Computers massively affect modern human life and computer is run by a computer program. Computer programs are arranged by programming languages, which is the reason why they are the foundation of all the conveniences that we feel. Programming languages are responsible for all programs that have been made, such as networking systems, virtual reality, scheduling software, online games, anti-virus and so on. Mastering the programming languages thus takes us closer how all modern technology originates.

Computer Programming Languages introduces students to the fundamental concepts of computer Programming Languages and provides them with the tools needed to evaluate contemporary and future languages. Often preparing students to research compiler design is an in-depth review of Programming Language constructs, such as syntax, lexical and syntactic analysis.

Programming languages refer to different types of expressions and logical structuring rules that are used to generate recurring and systematic tasks. They are of great importance because they allow the generation of different systems that serve tasks which also meet the needs of users. Nearly anything we can do in computer science is due in large part to programming languages.

Developing a system requires a variety of steps. The programmer identifies a problem, preparing a solution, coding the software, testing the software and finally recording the program. The programmer typically determines what he knows and the target, chooses a program to use, debugs the software in stages after completion to ensure no errors are introduced, and then records the program's design, development, and testing.

Awareness of some programming language may be very useful in developing capacity for problem-solving and task automation. Certainly, this kind of awareness, combined with some simple notions of algorithms, will open up a whole new panorama of job opportunities.

Computer Hardware and Software

Hardware

Refers to the physical elements of your device. This is also often referred to as the system or computer equipment. In the case of a typical desktop computer, this includes the main machine panel, the display screen, the keyboard, the mouse, and occasionally the printer. Speakers, webcams and external hard drives for backup storage are also often included. Computer hardware is made up of several different components, but the motherboard may be the most important of them. The motherboard is made up of many more parts to power and monitor the machine.

Computer hardware is any physical component used in or with your system, whereas software is a program code set mounted on your computer's hard drive. In other words, hardware is something that you can hold in your hand, while software cannot be held in your hand. Hardware and software are intertwined, without the use of software, the computer hardware would have no purpose. Nonetheless, without the development of hardware for the execution of tasks guided by software through the central processing unit, software will be useless. Hardware is restricted to particular tasks that are, taken separately, very basic. Computer implements algorithms (problem solutions) that allow the machine to perform far more complex tasks.

Software

The interface between the machine and the user. It's a series of instructions, programs that are used to send hardware commands. It is responsible for monitoring, integrating and handling the hardware components of the computer system and for performing different tasks. Software is capable of performing several tasks, as opposed to hardware that can perform only the mechanical tasks for which it is built. Code offers the means to perform several different tasks using the same basic hardware. Practical computer systems split information systems into two primary groups. The system software and the application software.

System software helps run your computer hardware and your own operating program. Computer software contains operating systems, application drivers, diagnostic tools, etc. System software is almost always pre-installed on your computer.

Application software allows users to perform one or more functions. It involves word processing, web surfing, and almost every other function for which you might be downloading apps. (Some software is pre-installed on most computer systems)

Program (noun)

An executable software that runs on a computer. A program is a detailed plan or procedure for solving a problem with a computer; more specifically, an unambiguous, ordered sequence of computational instructions necessary to achieve such a solution. The distinction between computer programs and equipment is often made by referring to the former as software and the latter as hardware.

Programmer

A person who designs the algorithms and writes the necessary code to create an application or computer program. A computer programmer can be a specialist in one field of computer programming, or a generalist who writes code for several kinds of applications.

Programming language is used by programmers to create software programs, scripts, or other collection of instructions for computers to run. While several languages share similarities, each has its own syntax. When a programmer knows the rules, syntax, and structure of the languages, the source code is written in a text editor or an IDE.

Instruction

The main element in the computer, since it informs the processor which action should be taken. The instructions to be followed are displayed in the source tab, and the machine moves from one instruction to the next following the instructions from top to bottom. Instructions are usually made up of two components.

- **The Operator:** the action that the processor is to carry out.

- **The Operands:** one or more pieces of data on which the operation is performed.

Types of Operator

- **Unary operator –** those which operate one operand only
- **Binary operator** - these operators are those which, contrary to what you would think, do not work on binary operands, but rather work on two operands (binary here means the number of operands that are affected, addition, often denoted as +, is therefore a binary operator)
- **Ternary operator** - are operators which allow three operands (for example, conditional operators are ternary operators

Operators can also be classified into many groups based on the type of action they conduct when they are executed.

- Arithmetic Operators
- Comparison Operators
- Logical Operators
- Bit Operators
- Assignment Operators
- Conditional Operators
- Sequential Operators

Every language usually has an operator's choice, so that the machine knows in which direction to measure the operator where many operators exist in the same expression.

Code

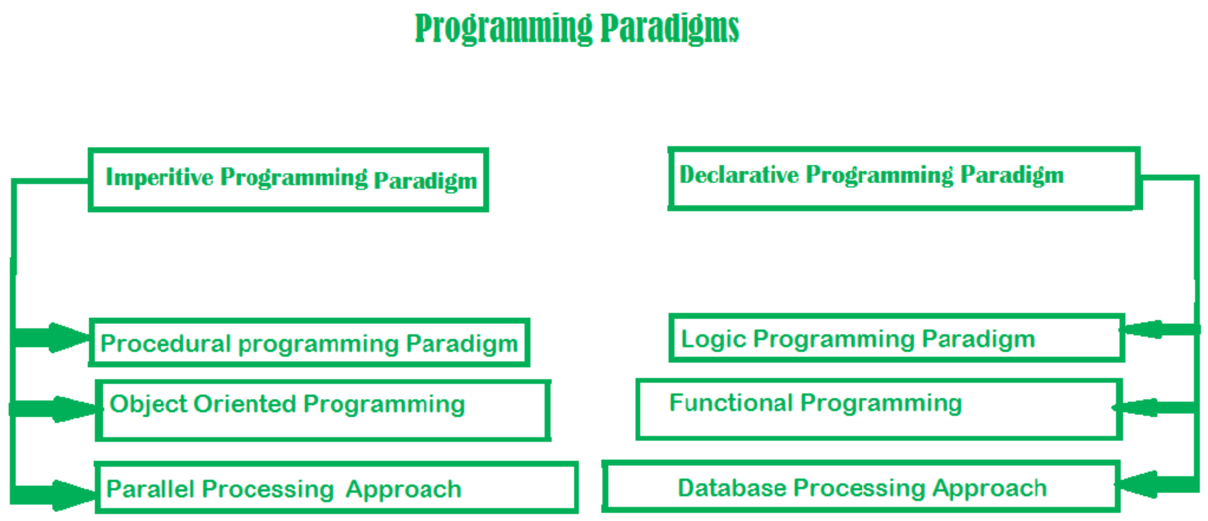
Short for source code, is a term used to describe text that is written by a computer programmer using a common language protocol. Types of programming languages include C, C #, C++, Java, Perl, and PHP. Code may also be used less formally to refer to text written for markup or modeled languages, such as HTML and CSS (Cascading Style Sheets).

Example of scripting code that prints "Hello World" to the screen is displayed below.

```
/* HelloWorld.java
*/

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

Paradigm

Referred to as a strategy to solve a problem or to perform a task. Programming methodology is a problem-solving approach using any programming language, or we may also suggest that it is a problem-solving process utilizing tools and methods that are accessible to us in any way. Paradigm can also be referred to as a technique to solve a problem or perform a mission. Programming methodology is a problem-solving technique using any programming language, or we can also say that it is a problem-solving mechanism utilizing tools and techniques that are available to us in every way.


Programming Paradigms
Imperitive Programming Paradigm
Declarative Programming Paradigm
Procedural programming Paradigm
Object Oriented Programming
Parallel Processing Approach
Logic Programming Paradigm
Functional Programming
Database Processing Approach

Imperitive Programming Paradigm

It's one of the oldest programming paradigms in the world. This has a similar relationship to computer architecture. This is based on the architecture of Von Neumann. It works by changing the status of the program by assignment statements. It performs a step-by - step function by changing the state. The main emphasis is on how to accomplish this aim. The model consists of multiple statements and all the results are processed after the execution.

Advantage of Imperitive Programming Paradigm

- Quite simple to execute
- Contains loops, variables, etc.

Disadvantage of Imperitive Programming Paradigm

- Complex problem cannot be resolved
- Less active and more productive
- No Parallel programming is necessary

Imperative programming is divided into three broad categories: procedural, OOP and parallel processing. These paradigms are the following:

Procedural programming paradigm

This paradigm emphasizes the process in terms of a lying machine model. There is no difference between a procedural and an imperative approach. It has the ability to reuse the code, and it was good at the time it was in use due to its reusability.

Examples of **Procedural** programming paradigm:

Programming Language	**Developer**
C	Dennis Ritchie and Ken Thompson
C++	Bjarne Stroustrup
Java	James Gosling at Sun Microsystems
ColdFusion	J J Allaire
Pascal	Niklaus Wirth

```
#include <iostream>
using namespace std;
int main()
{
        int i, fact = 1, num;
        cout << "Enter any Number: ";
        cin >> number;
        for (i = 1; i <= num; i++) {
                fact = fact * i;
        }
        cout << "Factorial of " << num << " is: " << fact << endl;
        return 0;
}
```

Object Oriented Programming

The software is written as a set of classes and artifacts intended for communication. The smallest and most fundamental entity is an atom, and all kinds of computing are done on objects only. Further focus is placed on the data rather than the process. This can tackle almost any kind of real life question that is currently in the scenario.

Advantages:

- Data security
- Inheritance
- Code reusability
- Flexible and abstraction is also present

```
import java.io.*;
 class GFG {
public static void main(String[] args)
{
System.out.println("GfG!");
            Signup s1 = new Signup();
            s1.create(22, "riya", "riya2@gmail.com", 'F', 89002);
    }
}
          class Signup {
    int userid;
    String name;
    String emailid;
    char sex;
    long mob;

    public void create(int userid, String name,
                                        String emailid, char sex, long mob)
    {
            System.out.println("Welcome to
                            GeeksforGeeks\nLets create your account\n");
            this.userid = 132;
            this.name = "Radha";
            this.emailid = "radha.89@gmail.com";
            this.sex = 'F';
            this.mob = 900558981;
            System.out.println("your account has been created");
    }
}
```

Examples of **Object Oriented** programming paradigm:

Programming Language	Developer/Description
Simula	First OOP Language
C++	Bjarne Stroustrup
Java	James Gosling at Sun Microsystems
Objective-C	Brad Cox
Visual Basic .NET	Microsoft
Python	Guido van Rossum
Ruby	Yukihiro Matsumoto
Smalltalk	Alan Kay, Dan Ingalls, Adele Goldberg

Parallel Processing

Approach is the processing of program instructions by splitting them between several processors. A parallel processing system has multiple processor numbers with the goal of running a program in less time by splitting it. This strategy seems to be like separating and winning. Examples are NESL (one of the oldest) and C / C++ is also supported because of some library feature.

Declarative Programming Paradigm

This is divided into Logic, Functional, and Database. Throughout computer science, declarative programming is a method of constructing programs that communicates the logic of computation without thinking about its control flow. This also considers programs to be hypotheses of certain nature. It can simplify the writing of parallel programs. The emphasis is on what needs to be done rather as to how it should be done, simply emphasizing what the code actually does. It only declares the result that we would just like to see how it was made. It is the only distinction between imperative (how to do) and declarative (what to do) style programming. Going deeper, we'd see reasoning, accessibility, and database.

Logic Programming Paradigms

Referred to as an abstract programming model. This will solve logical problems such as puzzles, sequence, etc. Through logic programming, we have a knowledge base that we know before and that, along with the problem and knowledge base that is presented to the computer, produces results**.** In normal programming languages, such a definition of knowledge base is not available, but when using the concept of artificial intelligence, machine learning we have some models like the Perception model that use the same mechanism.

Functional Programming Paradigms

The central principle of these paradigms is the implementation of a set of mathematical functions. The core model for abstraction is the feature that is intended for some particular computation, not the structure of the data. The data is loosely related to the functions. The role is covering their implementation. The function can be replaced with its values without altering the context of the program. Some of the languages like perl, javascript, also use this framework.

Examples of **Functional** programming paradigm:

Programming Language	Developer/Description
JavaScript	Brendan Eich
Haskwell	Lennart Augustsson, Dave Barton
Scala	Martin Odersky
Erlang	Armstrong, Robert Virding
Lisp	John Mccarthy
ML	Robin Milner
Clojure	Rich Hickey

Database/Data driven programming approach

The technique for programming is focused on data and its movement. Program statements are described by data rather than by hard-coding a sequence of steps. The database software is at the core of the business information system which offers file creation, data entry, update, query which reporting functions. There are many programming languages that are mainly designed for database applications. SQL, for example. It is used for streaming structured data, filtering, transforming, aggregating (such as computing statistics) or calling other programs. And it has a broad application of its own.

```
CREATE DATABASE databaseAddress;
CREATE TABLE Addr (
   PersonID int,
   LastName varchar(200),
   FirstName varchar(200),
   Address varchar(200),
   City varchar(200),
   State varchar(200)
);
```

Steps in Program Development

1. Analyze the problem. The computer user must figure out the problem, then decide how to resolve the problem - choose a program.

2. Design the program. A flow chart is important to use during this step of the PDLC. This is a visual diagram of the flow containing the program. This step will help you break down the problem.

3. Code the program. This is using the language of programming to write the lines of code. The code is called the listing or the source code. The computer user will run an object code for this step.

4. Debug the program. The computer user must debug. This is the process of finding the "bugs" on the computer. The bugs are important to find because this is known as errors in a program.

5. Formalize the solution. One must run the program to make sure there are no syntax and logic errors. Syntax are grammatical errors and logic errors are incorrect results.

6. Document and maintain the program. This step is the final step of gathering everything together. Internal documentation is involved in this step because it explains the reason one might have made a change in the program or how to write a program.

Different Programming Languages

Java

Java is top pick as one of the most popular programming languages, used for building server-side applications to video games and mobile apps. It's also the core foundation for developing Android apps, making it a favorite of many programmers. With its WORA mantra (write once, run anywhere), it's designed to be portable and run happily across multiple software platforms. I first got started with Java server programming back in 1999--it was so exciting, I actually wrote a few books about it. Java is everybody's pal!

Python

Python is a one-stop shop. There's a Python framework for pretty much anything, from web apps to data analysis. In fact, WordStream is written in Python! You're the best bud. Python is often heralded as the easiest programming language to learn, with its simple and straightforward syntax. Python has risen in popularity due to Google's investment in it over the past decade (in fact, one recent study has shown Python to be the most commonly taught programming language in U.S. schools). Other applications built with Python include Pinterest and Instagram.

Ruby

Ruby (also known as Ruby on Rails) is a major supplier of web apps. Ruby is popular due to its ease of learning (it's very straightforward) and power. Ruby knowledge is in high demand these days!

JavaScript

JavaScript (which, confusingly, is not at all related to Java) is another favorite programming language because it's so ubiquitous on the web--it's basically everywhere. JavaScript allows developers to add interactive elements to their website, and its presence is felt across the internet. At WordStream, we use a JavaScript library called JQuery to make our JavaScript work even easier.

C#

C# (pronounced C-sharp, not C-hashtag for you Twitter fans) is the language used in order to develop Microsoft apps. C# is syntactically nearly identical to Java. I've spent much time training with C#, but if you're good at Java, you'll likely have an easy time jumping onto C#. If you're looking to work on Microsoft apps, C# is the way to go. C# opens a lot of Windows (har-har).

Objective-C

Objective-C is the programming language behind iOS apps. Apple's new language Swift is rising in the ranks, but Objective-C is still the recommended starting point for those looking to craft Apple apps for iPhones and iPads. Next stop--the iOS App Store!

C

C is the predecessor to more complex programming languages like Java and C#. C is best when you want to work small and when dealing with low-level applications. It's widely used for embedded systems like the firmware of your television or the operating system of an airplane, as well as computer operating systems like Windows. For me personally, C was more of an academic language. It was nice to learn how to write a kernel back in college, and you gain a more solid understanding of how newer languages work under the covers, but it's rare for most application developers to ever have to use this today.

Importance of Java

Java is a programming language that produces software for multiple platforms. When a programmer writes a Java application, the compiled code (known as bytecode) runs on most operating systems (OS), including Windows, Linux and Mac OS. Java derives much of its syntax from the C and C++ programming languages.

Java is defined by a specification and consists of a programming language, a compiler, core libraries and a runtime (Java virtual machine) The Java runtime allows software developers to write program code in other languages than the Java programming language which still runs on the Java virtual machine. The *Java platform* is usually associated with the *Java virtual machine* and the *Java core libraries*.

Over time new enhanced versions of Java have been released. The current version of Java is Java 1.8 which is also known as Java 8.

The Java language was designed with the following properties:

- Platform independent: Java programs use the Java virtual machine as abstraction and do not access the operating system directly. This makes Java programs highly portable. A Java program (which is standard-compliant and follows certain rules) can run unmodified on all supported platforms, e.g., Windows or Linux.
- Object-orientated programming language: Except the primitive data types, all elements in Java are objects.
- Strongly-typed programming language: Java is strongly-typed, e.g., the types of the used variables must be pre-defined and conversion to other objects is relatively strict, e.g., must be done in most cases by the programmer.
- Interpreted and compiled language: Java source code is transferred into the bytecode format which does not depend on the target platform. These bytecode instructions will be interpreted by the Java Virtual machine (JVM). The JVM contains a so called Hotspot-

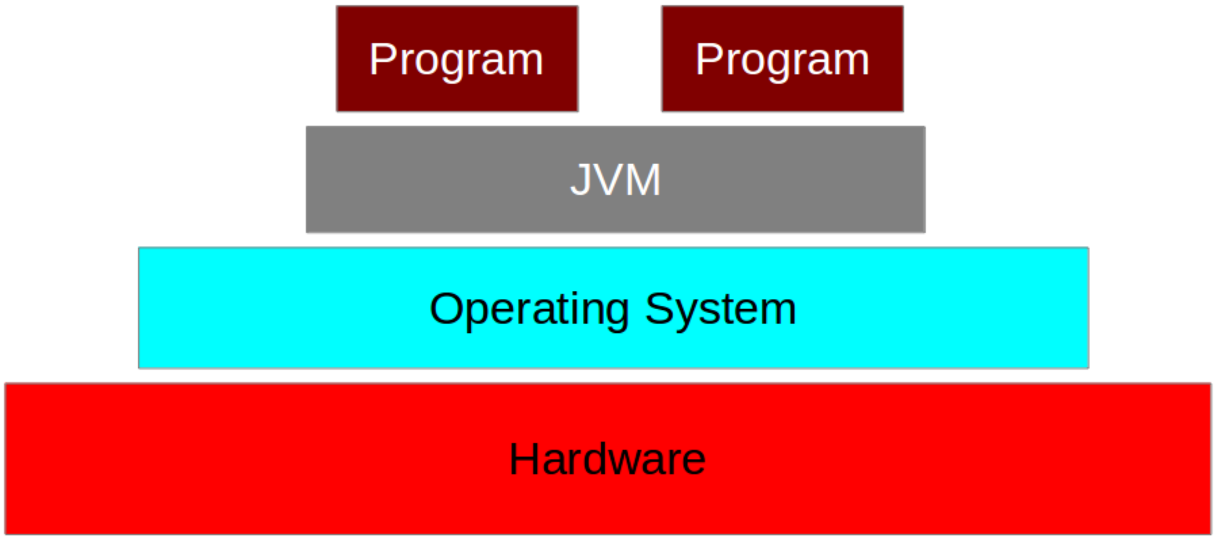
Compiler which translates performance critical bytecode instructions into native code instructions.

- Automatic memory management: Java manages the memory allocation and de-allocation for creating new objects. The program does not have direct access to the memory. The so-called garbage collector automatically deletes objects to which no active pointer exists.

Java Virtual Machine

The Java virtual machine (JVM) is a software implementation of a computer that executes programs like a real machine.

The Java virtual machine is written specifically for a specific operating system, e.g., for Linux a special implementation is required as well as for Windows.



Java programs are compiled by the Java compiler into *bytecode*. The Java virtual machine interprets this *bytecode* and executes the Java program.

Java Runtime Environment vs. Java Development Kit

A Java distribution typically comes in two flavors, the *Java Runtime Environment* (JRE) and the *Java Development Kit* (JDK).

The JRE consists of the JVM and the Java class libraries. Those contain the necessary functionality to start Java programs.

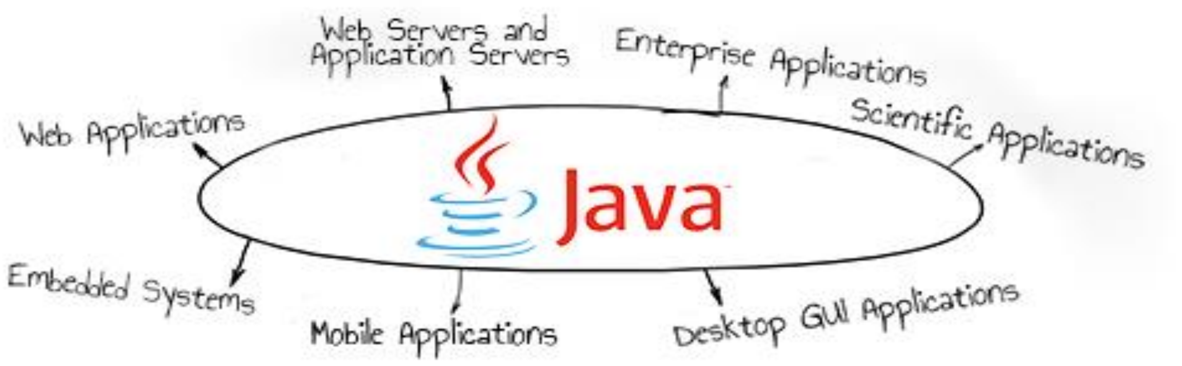
The JDK additionally contains the development tools necessary to create Java programs. The JDK therefore consists of a Java compiler, the Java virtual machine and the Java class libraries.

Development Process with Java

Java source files are written as plain text documents. The programmer typically writes Java source code in an *Integrated Development Environment* (IDE) for programming. An IDE supports the programmer in the task of writing code, e.g., it provides auto-formating of the source code, highlighting of the important keywords, etc.

At some point the programmer (or the IDE) calls the Java compiler (javac). The Java compiler creates the *bytecode* instructions. These instructions are stored in .class files and can be executed by the Java Virtual Machine.

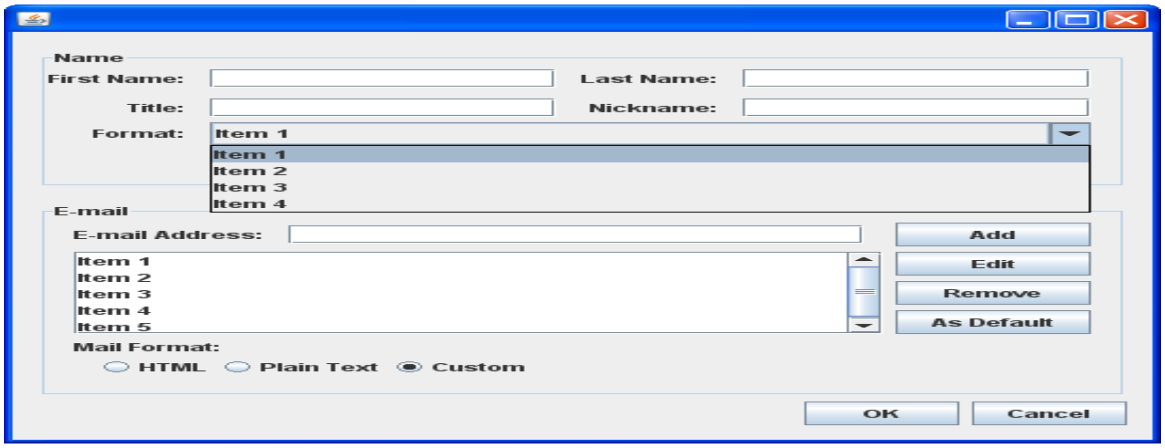
Java is a general purpose programming language, much like Python or JavaScript. The language itself is specifically an object oriented programming language, so bears similarities to C++, C#. **Java** is also a platform, which means that **Java** code can run on any machine that has a **Java** Virtual Machine (JVM) on it.



Types of Applications that runs in Java

Desktop GUI Applications

Java provides GUI development through various means like Abstract Windowing Toolkit (AWT), Swing and JavaFX. While AWT contains a number of pre-constructed components such as menu, button, list, and numerous third-party components, Swing, a GUI widget toolkit, additionally provides certain advanced components like trees, tables, scroll panes, tabbed panel and lists. JavaFX, a set of graphics and media packages, provides Swing interoperability, 3D graphic features and self-contained deployment model which facilitates quick scripting of Java applets and applications.



Format: Item 1

Mail Format: [✓ Custom]

Mobile Applications

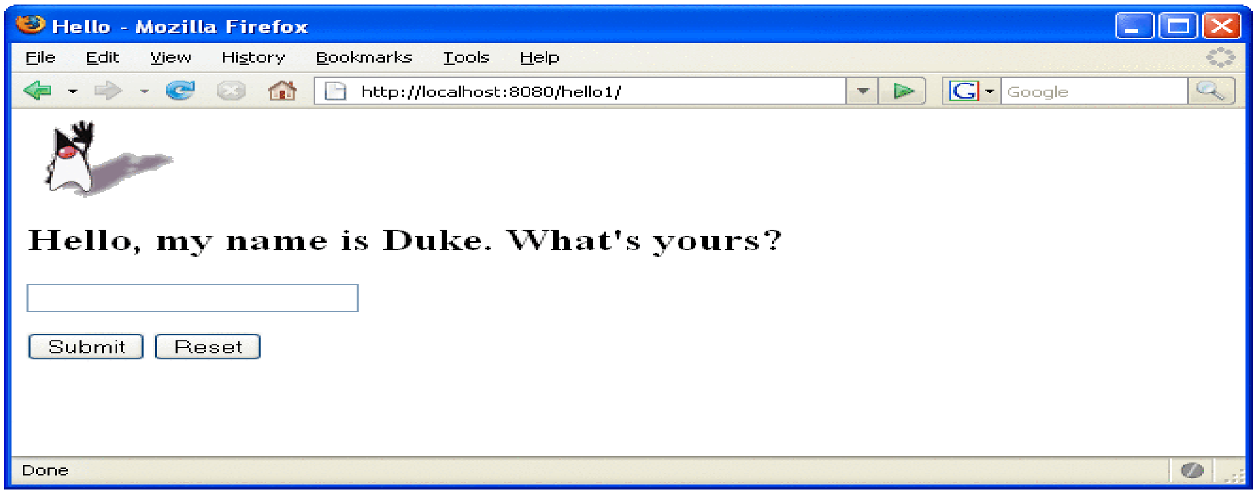
Java Platform, Micro Edition (Java ME or J2ME) is a cross-platform framework to build applications that run across all Java supported devices, including feature phones and smart phones. Further, applications for Android, one of the most popular mobile operating systems, are usually scripted in Java using the Android Software Development Kit (SDK) or other environments.

Embedded Systems

Embedded systems, ranging from tiny chips to specialized computers, are components of larger electromechanical systems performing dedicated tasks. Several devices, such as SIM cards, blue-ray disk players, utility meters and televisions, use embedded Java technologies. According to Oracle, 100% of Blu-ray Disc Players and 125 million TV devices employ Java.

Web Applications

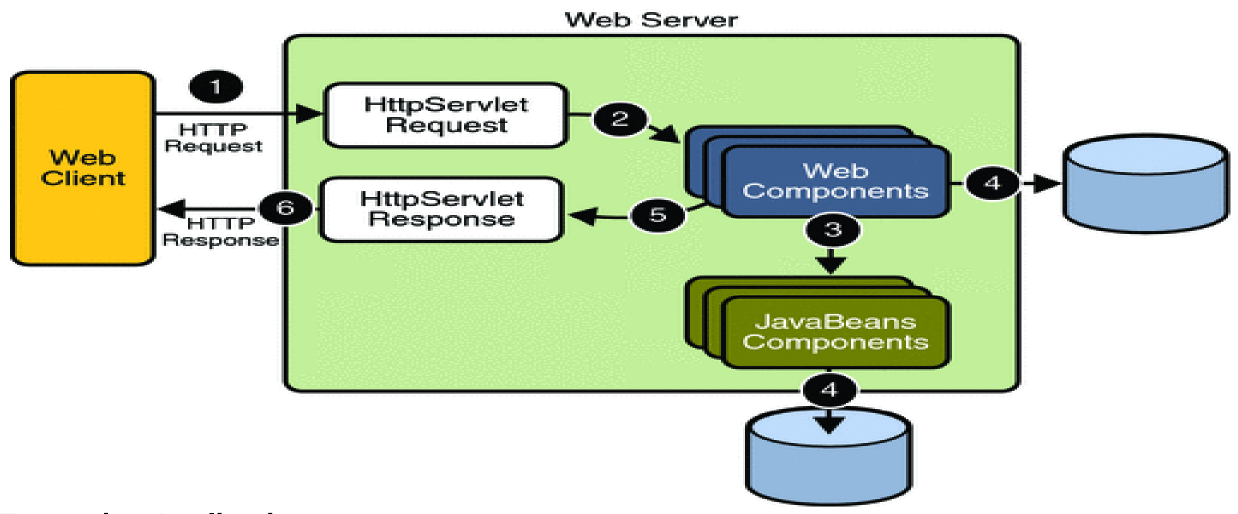
Java provides support for web applications through Servlets, Struts or JSPs. The easy programming and higher security offered by the programming language has allowed a large number of government applications for health, social security, education and insurance to be based on Java. Java also finds application in development of eCommerce web applications using open-source eCommerce platforms, such as Broadleaf.


Hello - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/hello1/
Google
Submit
Reset
Done

Hello, my name is Duke. What's yours?

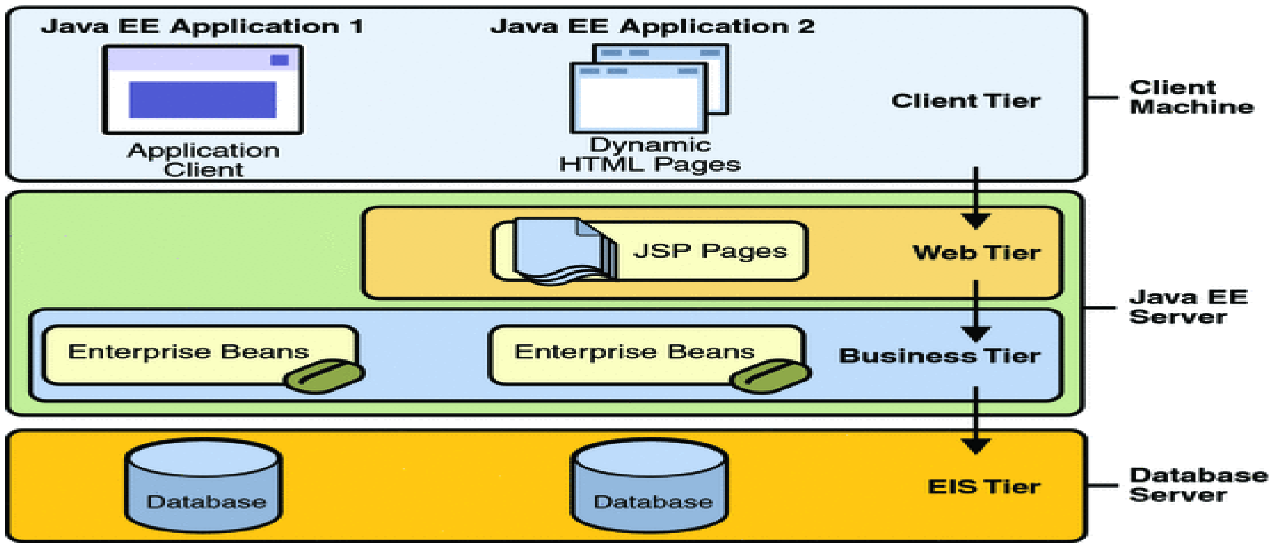
Web Servers and Application Servers:

The Java ecosystem today contains multiple Java web servers and application servers. While Apache Tomcat, Simple, Jo!, Rimfaxe Web Server (RWS) and Project Jigsaw dominate the web server space, WebLogic, WebSphere, and Jboss EAP dominate commercial application server space.



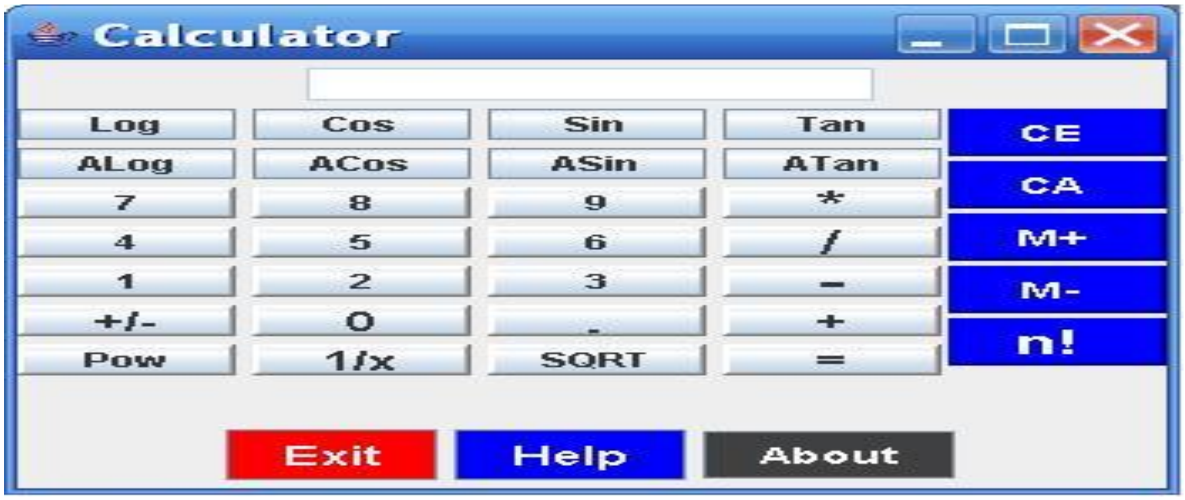
Enterprise Applications

Java Enterprise Edition (Java EE) is a popular platform that provides API and runtime environment for scripting and running enterprise software, including network applications and web-services. Oracle claims Java is running in 97% of enterprise computers. The higher performance guarantee and faster computing in Java has resulted in high frequency trading systems like Murex to be scripted in the language. It is also the backbone for a variety of banking applications which have Java running from front user end to back server end.



Scientific Applications

Java is the choice of many software developers for writing applications involving scientific calculations and mathematical operations. These programs are generally considered to be fast and secure, have a higher degree of portability and low maintenance. Applications like MATLAB use Java both for interacting user interface and as part of the core system.



In conclusion, Java is widely applicable across different types of applications.

Java has major advantages over other languages and environments that make it ideal for any programming tasks.

- Java is easy to learn. Java was designed to be easy to use and is therefore easy to write, compile, debug, and learn than other programming languages.

- Java is object-oriented. This allows you to create modular programs and reusable code.

- Java is platform-independent. One of the most significant advantages of Java is its ability to move easily from one computer system to another. The ability to run the same program on many different systems is crucial to World Wide Web software, and Java succeeds at this by being platform-independent at both the source and binary levels.

Due to Java's robustness, ease of use, cross-platform functionality and security features, it has become the language of choice for delivering global Internet solutions.

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

Object-Oriented: Object oriented programming is associated with concepts like class, object, inheritance, encapsulation, abstraction, polymorphism, etc. which allows you to create modular programs and reusable code. You can declare classes, create objects inside classes, and interact between two objects.

Platform-Independent: Java offers the comfort of write program once and run on any hardware and software platform and any Java compatible browser. This gives the ability to move easily from one computer system to another.

Distributed: Java has great networking capability, it is designed to make distributed computing easy with the networking capability that is inherently integrated into it.

Secure: Java is the first programming language to include security an integral part of the design. Java's compiler, interpreter, and runtime environment were each developed with security in mind. Java Virtual Machine has a unique identifier that identifies the bytecode and verifies it before running it.

Allocation: Java has the feature of Stack allocation system. It follows LIFO (Last in First Out) which helps the data to be stored and retrieved easily.

Multithreaded: Java is one of the programming languages to support Multithreading. Multithreading is the capability for a program to perform several tasks simultaneously within a program.

Rich APIs: Java offers various APIs for application development. Java APIs (Application Programming Interface) is the set of commands or methods of communication among various activities like Database connection, networking, I/O, XML parsing, utilities, and much more.

Powerful Open source Rapid Development Tools: Over the year's several open source development tools i.e., IDEs such as Eclipse and Netbeans, have been created with Java as a base which makes Java more powerful for application development. IDEs makes application development simpler with powerful coding and debugging features.

Robust: Java is one of the most robust programming languages that is Java is more reliable. Java compilers can detect any errors in the coding. There are also other features like exception handling and garbage collection which makes Java more robust.

Resource Availability: There are tons of online java training courses available to learn java. You don't have to spend a fortune to learn java programming.

Disadvantages of Java

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

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

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

Activities/Assessment

I. **Multiple Choice.** Read each question carefully, and then **CIRCLE THE ANSWER** that best fits the question.

1. It is any physical component used in or with your system.
   A. Software
   B. Hardware
   C. Code
   D. Machine
2. This is the interface between the machine and the user.
   A. Javascript
   B. Code
   C. Program
   D. Software
3. It is executable software that runs on a computer.
   A. Javascript
   B. Code
   C. Program
   D. Software
4. The one who designs the algorithms and writes the necessary code to create an application or computer program.
   A. Programmer
   B. Instructor
   C. Programmerist
   D. Operator
5. It informs the processor which action should be taken.
   A. Code
   B. Instruction
   C. Programming Language
   D. Javascript

II. **True or False.** Read each statement below carefully. Place a T on the line if you think a statement it TRUE. Place an F on the line if you think the statement is FALSE.

1. _____ Ternary Operator are operators which allow three operands
2. _____ Operator cannot be classified into many groups based on their type of action
3. _____ A Programming Language is used by Programmers to create software programs
4. _____ The Computer is run by computer program
5. _____ Programming Languages refers to the different types of expression and logical structuring rules.

III. Application. Read the questions carefully and confine your responses to an analysis of the questions as written.

1. What is programming language in simple words?

2. What is the difference between computer hardware and computer software?

3. What is a programmer?

4. Explain the importance of programming?

5. Explain how the programming language works?

Java Structure

Learning Outcomes:

After successful completion of this lesson, you should be able to:

- Design algorithms to solve problems.
- Identify different java Compilers
- Differentiate Data Types
- Predict the results when an expression is evaluated.
- Distinguishes assignment, input, and output functions.
- Identify different input and output statements in Java
- Describe the six expression categories in Java.
- Identify different flowcharting symbols
- Explain the different uses of flowchart

A typical structure of a Java program contains the following elements

- Package declaration
- Import statements
- Comments
- Class definition
- Class variables, Local variables
- Methods/Behaviors

Package declaration

A class in Java can be placed in different **directories/packages** based on the module they are used. For all the classes that belong to a **single parent source directory**, a path from source directory is considered as **package declaration**.

Import statements

There can be classes written in other **folders/packages** of our working java project and also there are many classes written by individuals, companies, etc which can be useful in our program. To use them in a class, we need to **import** the class that we intend to use. Many classes can be imported in a single program and hence multiple import statements can be written.

Comments

The comments in Java can be used to **provide information about the variable, method, class or any other statement**. It can also be used to hide the program code for a specific time.

Class Definition

A name should be given to a **class** in a java file. This name is used while creating an **object of a class**, in other classes/programs.

Variables

The Variables are **storing the values of parameters** that are required during the execution of the program. Variables declared with modifiers have **different scopes,** which define the life of a variable.

Main Method

Execution of a Java application starts from the main method. In other words, it’s an **entry point for the class** or **program** that starts in **Java Run-time**.

Methods/Behaviors

A set of instructions which form a **purposeful functionality** that can be required to run multiple times during the execution of a program. To not repeat the same set of instructions when the same functionality is required, the instructions are enclosed in a method. A method’s behavior can be exploited by **passing variable values** to a method.

Example

```
package abc; // A package declaration
import java.util.*; // declaration of an import statement
// This is a sample program to understnd basic structure of Java (Comment Section)
public class JavaProgramStructureTest { // class name
int repeat = 4; // global variable
public static void main(String args[]) { // main method
JavaProgramStructureTest test = new JavaProgramStructureTest();
test.printMessage("Hello World!");
}
public void printMessage(String msg) { // method
Date date = new Date(); // variable local to method
for(int index = 0; index < repeat; index++) { // Here index - variable local to for loop
System.out.println(msg + "From" + date.toGMTString());
}
}
}
```

Output:

```
Hello World!;from 2 Jul 2019 08:35:15 GMT
Hello World! from 2 Jul 2019 08:35:15 GMT
Hello World! from 2 Jul 2019 08:35:15 GMT
Hello World! from 2 Jul 2019 08:35:15 GMT
```

Java Compilers

Java Compilers are the compilers for the programming language. Every programming language has its own set of program which executes the code return in that particular language. There is no magic behind the code running in a particular language. There is something written by someone to convert the code written in human-understandable to translate in the language which understood by a machine. The same thing gets applied with java also. Java is easy to understand by humans. If we write programs in java it should be get converted into machine language.

Machine language is nothing but 0’s and 1’s, to convert this code into byte code java has its own set of compilers. Generally, we knew only a few compilers. And if you are new to java then the possibility is you must be knowing only one compiler called javac. Compilers give us the ability to interact with other platforms. We can run our program written in Java on any platform like Windows, Linux, MAC, etc. There is no restriction on which compiler should be used. But we should know the availability of different compilers for the time being. Now let’s look at what exactly it means.

What is Java Compilers?

Compilers are an interface between human language and machine understandable language. The Java compiler operates on the .java file or on the source code file. It then converts every class in the .java file into its corresponding .class file. This .class file can operate on any Operating System.

Hence java is a platform-independent language. Note that when we compile our java file with command javac it converts the code into machine language. That code called bytecode. Now it’s time to check out different environments that are currently there available for us.

Types of Java Compilers:

1. Javac
2. Edison Design Group
3. GCJ
4. ECJ
5. Jikes
6. Power J
7. JIT
8. Client-Side Compiler
9. Server-Side Compiler

Lets us study in details about the different types of Compilers which are as follows:

Javac

It is implemented by Martin Odersky at Sun Microsystems which was further owned by Oracle. This javac compiler needs to be installed with any IDE to run a java program. Javac itself is written in Java language. This compiler is available for Windows, UNIX, and other OS.

Edison Design Group

It is a company that makes the EDG compiler. It is implemented by J. Stephen “Steve” Adamczyk in 1988. They mainly write compilers for preprocessing and parsing. This compiler is also available for Windows, UNIX, and other Oss but this compiler is not available for any IDE.

GCJ

GCJ stands for GNU Compiler for Java. This is a free compiler available for Java Programming Language. This compiler compiles the java source code to a machine-understandable format. It can also compile JARs which contain bytecode. This compiler is only available for UNIX and not for other Operating Systems like Windows and this is neither available for any IDE. This compiler can also compile C, C++, Fortran, Pascal, and other programming languages.

ECJ

This is an Eclipse compiler for Java and comes with Eclipse IDE and available for operating systems like Windows, UNIX, etc. With this compiler, if some part of the code is having a compile-time error, however, the other part of the code can be tested whether working fine or not. This is not the case with javac as you need to fix all the errors before compilation.

With Eclipse compiler, if your java source code is having any compile-time error then it will through it as a runtime exception. Also, this compiler can run in the background of IDE and it also speeds up the compilation as compared to javac.

Jikes

This compiler is developed by Dave Shields and Philippe Charles at IBM and it's an OSI certified open-source Java Compiler and written in C++. It is a high-performance compiler used for large projects and much faster in compiling small projects than Sun's compiler. Jikes was released in 1998 for Linux. The issue with Jikes compiler is that it does not support Java 5 and above versions since there is no updation from IBM on the same. This compiler works best with JDK 1.3 and below versions.

Power J

This compiler was written at Sybase which was further owned by SAP. This is available for Windows and also for IDEs.

JIT

It stands for Just In Time compiler and used to improve the performance of Java application. This compiler is enabled by default. It gets active when any method in java is called. JIT compiles the byte code of that method into machine code. It's a component of Java Runtime Environment which improves the performance at run time.

Netbeans

NetBeans IDE is a free and open source integrated development environment for application development on Windows, Mac, Linux, and Solaris operating systems. The IDE simplifies the development of web, enterprise, desktop, and mobile applications that use the Java and HTML5 platforms.

Java Data Types

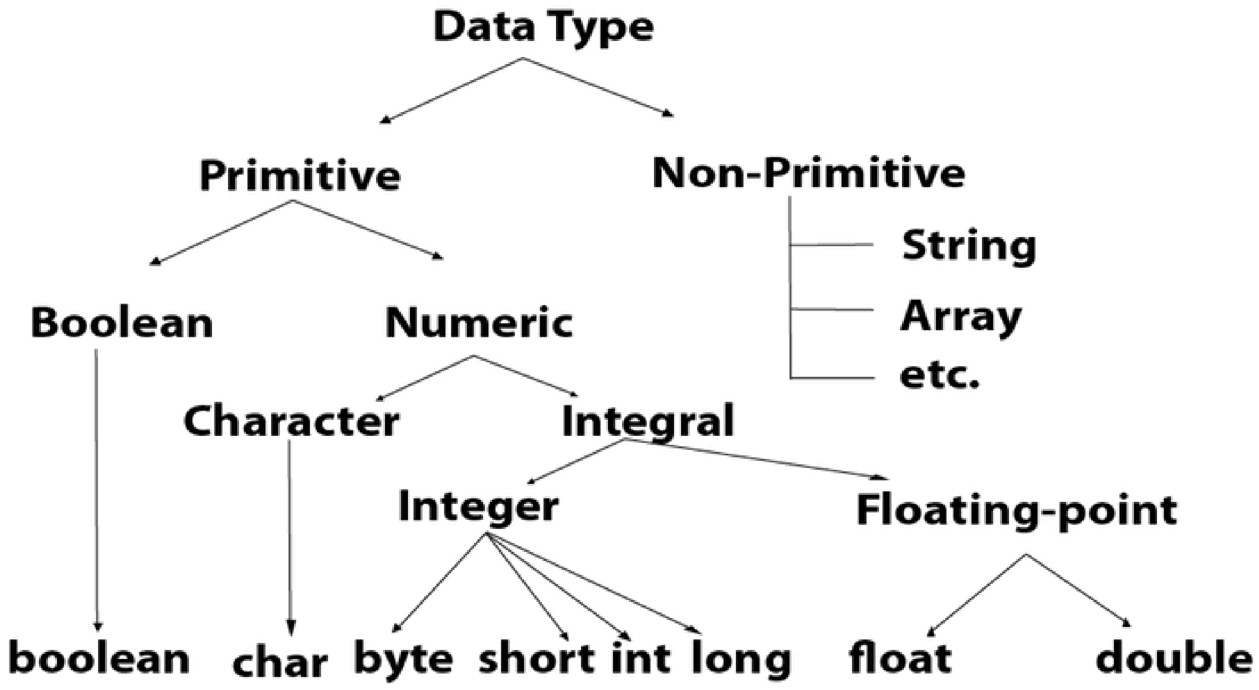
Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.

2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

There are 8 types of primitive data types:

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type
- double data type



Boolean Data Type

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

Example: Boolean one = false

Byte Data Type

The byte data type is an example of primitive data type. It isan 8-bit signed two's complement integer. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

Example: byte a = 10, byte b = -20

Short Data Type

The short data type is a 16-bit signed two's complement integer. Its default value is 0. The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

Example: short s = 10000, short r = -5000

Int Data Type

The int data type is a 32-bit signed two's complement integer. Its default value is 0. The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example: int a = 100000, int b = -200000

Long Data Type

The long data type is a 64-bit two's complement integer. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

Example: long a = 100000L, long b = -200000L

Float Data Type

The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example: float f1 = 234.5f

Double Data Type

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.
Example: double d1 = 12.3

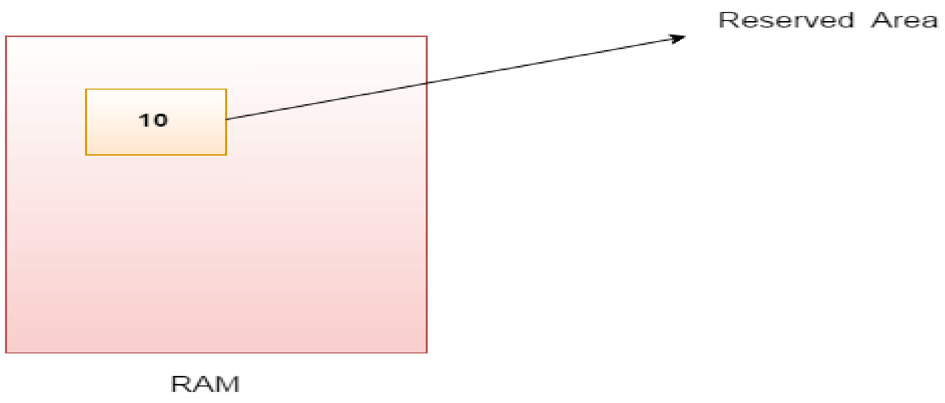
Char Data Type

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.
Example: char letterA = 'A'

Java Variables

A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type. Variable is a name of memory location. There are three types of variables in java: local, instance and static.

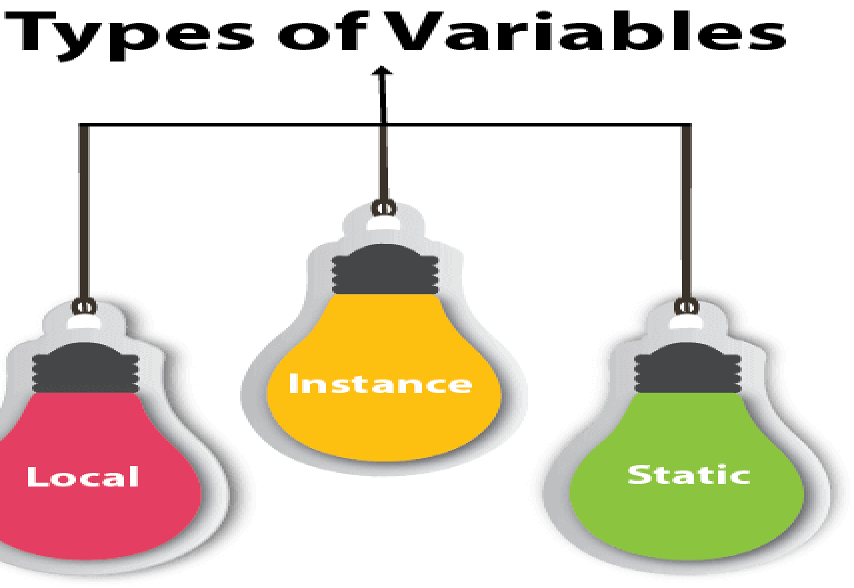
Variable is name of *reserved area allocated in memory*. In other words, it is a *name of memory location*. It is a combination of "vary + able" that means its value can be changed.



int data=50;//Here data is variable

There are three types of variables in Java:

- local variable
- instance variable
- static variable



Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists. A local variable cannot be defined with "static" keyword.

Instance Variable

A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static. It is called instance variable because its value is instance specific and is not shared among instances.

Static variable

A variable which is declared as static is called static variable. It cannot be local. You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

Example to understand the types of variables in java

```
class A{
int data=50;//instance variable
static int m=100;//static variable
void method(){
int n=90;//local variable
}
}//end of class
```

Java Variable Example: Add Two Numbers

```
class Simple{
public static void main(String[] args){
int a=10;
int b=10;
int c=a+b;
System.out.println(c);
}}
```

Output:

```
20
```

Java Constant

Constants in Java are used when a **'static'**value or a permanent value for a variable has to be implemented. Java doesn't directly support constants. To make any variable a constant, we must use 'static' and 'final' modifiers in the following manner:

Syntax to assign a constant value in java:

static final datatype identifier_name = constant;

- The **static modifier** causes the variable to be available without an instance of it's defining class being loaded
- The **final modifier** makes the variable unchangeable

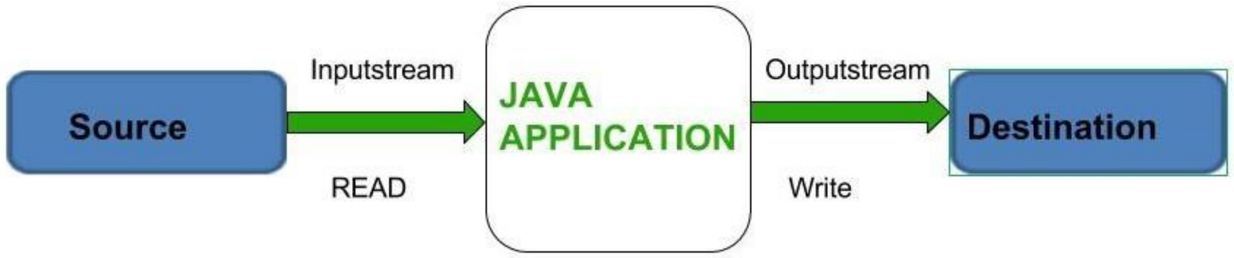
Static and Final Modifiers

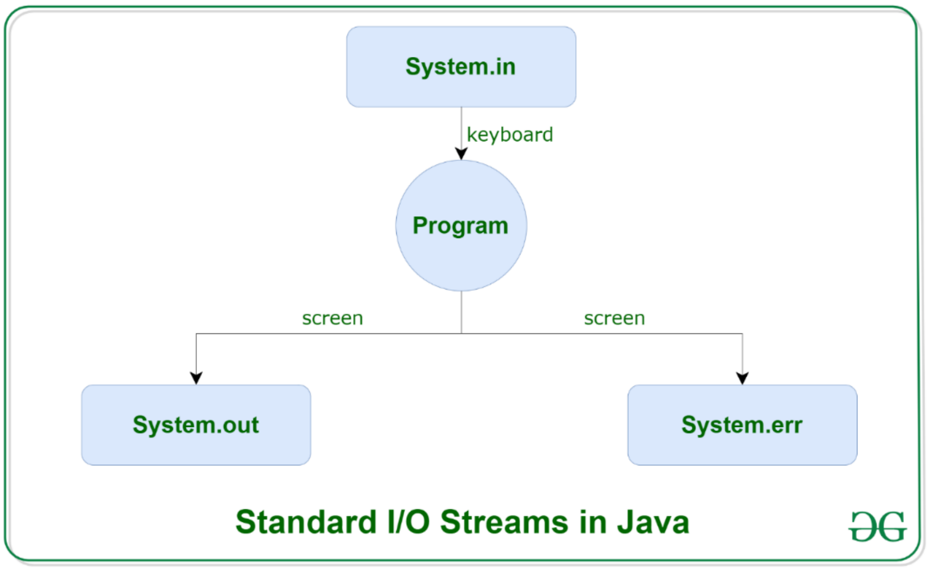
- The static modifier is mainly used for memory management.
- It also allows the variable to be available without loading any instance of the class in which it is defined.
- The final modifier means that the value of a variable cannot change. Once the value is assigned to a variable, a different value cannot be reassigned to the same variable.

By using the final modifier, Primitive data types like int, float, char, byte, long, short, double, Boolean all can be made immutable/unchangeable.

Java IO: Input-output in Java

Java brings various Streams with its I/O package that helps the user to perform all the input-output operations. These streams support all the types of objects, data-types, characters, files etc to fully execute the I/O operations.





System.in:

This is the **standard input stream** that is used to read characters from the keyboard or any other standard input device.

System.out

This is the **standard output stream** that is used to produce the result of a program on an output device like the computer screen.

print()

This method in Java is used to display a text on the console. This text is passed as the parameter to this method in the form of String. This method prints the text on the console and the cursor remains at the end of the text at the console. The next printing takes place from just here.

Syntax:

System.out.print(*parameter*);

```
import java.io.*;

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

      // using print()
      // all are printed in the
      // same line
      System.out.print("GfG! ");
      System.out.print("GfG! ");
      System.out.print("GfG! ");    } }
```

Output:

```
GfG! GfG! GfG!
```

println()

This method in Java is also used to display a text on the console. It prints the text on the console and the cursor moves to the start of the next line at the console. The next printing takes place from the next line.

Syntax:

System.out.println(*parameter*);

Example:

```
// Java code to illustrate println()

import java.io.*;

class Demo_print {
public static void main(String[] args)
{
// using println()
// all are printed in the
// different line
System.out.println("GfG! ");
System.out.println("GfG! ");
System.out.println("GfG! ");
}
}
```

Output:

```
GfG!
GfG!
GfG!
```

printf()

This is the easiest of all methods as this is similar to printf in C. Note that System.out.print() and System.out.println() take a single argument, but printf() may take multiple arguments. This is used to format the output in Java.

System.err

This is the **standard error stream** that is used to output all the error data that a program might throw, on a computer screen or any standard output device.

This stream also uses all the 3 above-mentioned functions to output the error data:

- print()
- println()
- printf()

Input Stream

These streams are used to read data that must be taken as an input from a source array or file or any peripheral device. For eg., FileInputStream, BufferedInputStream, ByteArrayInputStream etc.

Output Stream

These streams are used to write data as outputs into an array or file or any output peripheral device. For eg., FileOutputStream, BufferedOutputStream, ByteArrayOutputStream etc.

ByteStream

This is used to process data byte by byte (8 bits). Though it has many classes, the FileInputStream and the FileOutputStream are the most popular ones. The FileInputStream is used to read from the source and FileOutputStream is used to write to the destination. Here is the list of various ByteStream Classes:

Stream class	**Description**
BufferedInputStream	It is used for Buffered Input Stream.
DataInputStream	It contains method for reading java standard datatypes.
FileInputStream	This is used to reads from a file
InputStream	This is an abstract class that describes stream input.
PrintStream	This contains the most used print() and println() method
BufferedOutputStream	This is used for Buffered Output Stream.
DataOutputStream	This contains method for writing java standard data types.
FileOutputStream	This is used to write to a file.
OutputStream	This is an abstract class that describe stream output.

```
// Java Program illustrating the
// Byte Stream to copy
// contents of one file to another file.
import java.io.*;
public class BStream {
    public static void main(
        String[] args) throws IOException {
        FileInputStream sourceStream = null;
        FileOutputStream targetStream = null;
        try {
            sourceStream
                = new FileInputStream("sorcefile.txt");
            targetStream
                = new FileOutputStream("targetfile.txt");
            // Reading source file and writing
            // content to target file byte by byte
            int temp;
            while ((
                       temp = sourceStream.read())
                   != -1)
                targetStream.write((byte)temp);
        }
        finally {
            if (sourceStream != null)
                sourceStream.close();
            if (targetStream != null)
                targetStream.close();
        }
    }
}
```

Output:

Shows contents of file test.txt

CharacterStream

In Java, characters are stored using Unicode conventions (Refer this for details). Character stream automatically allows us to read/write data character by character. Though it has many classes, the FileReader and the FileWriter are the most popular ones. FileReader and FileWriter are character streams used to read from the source and write to the destination respectively. Here is the list of various CharacterStream Classes:

Stream class	Description
BufferedReader	It is used to handle buffered input stream.
FileReader	This is an input stream that reads from file.
InputStreamReader	This input stream is used to translate byte to character.
OutputStreamReader	This output stream is used to translate character to byte.
Reader	This is an abstract class that define character stream input.
PrintWriter	This contains the most used print() and println() method
Writer	This is an abstract class that define character stream output.
BufferedWriter	This is used to handle buffered output stream.
FileWriter	This is used to output stream that writes to file.

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

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-	B % A will give 0

	hand operand and returns remainder.	
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

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
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (greater than)	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.
< (less than)	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.
>= (greater than or equal to)	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.
<= (less than or equal to)	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.

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
```

Operator	Description	Example
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (bitwise compliment)	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.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (right shift)	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
>>> (zero fill right shift)	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

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
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true
! (logical not)	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

The Assignment Operators

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

Miscellaneous Operators

There are few other operators supported by Java Language.

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

Following is an 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
```

Instance of Operator

This operator is used only for object reference variables. The operator checks whether the object is of a particular type (class type or interface type). instance of operator is written as − (Object reference variable) instance of (class/interface type).

If the object referred by the variable on the left side of the operator passes the IS-A check for the class/interface type on the right side, then the result will be true. Following is an example:

```
public class Test {

   public static void main(String args[]) {

      String name = "James";

      // following will return true since name is type of String
      boolean result = name instanceof String;
      System.out.println( result );
   }
}
```

Output:

```
true
```

Precedence of Java Operators

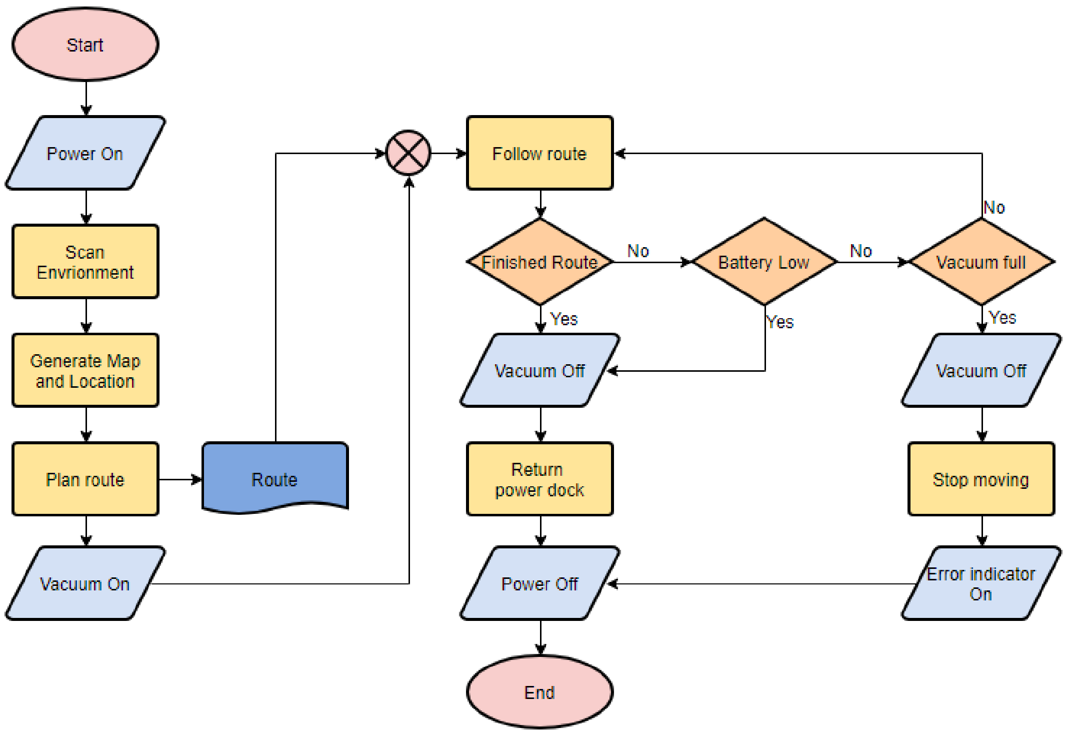
Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator

For example, x = 7 + 3 * 2; here x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3 * 2 and then adds into 7. Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	expression++ expression--	Left to right
Unary	++expression –-expression +expression –expression ~ !	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >> >>>	Left to right
Relational	< > <= >= instanceof	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= ^= |= <<= >>= >>>=	Right to left

Flowchart

A flowchart is simply a graphical representation of steps. It shows steps in sequential order and is widely used in presenting the flow of algorithms, workflow or processes. Typically, a flowchart shows the steps as boxes of various kinds, and their order by connecting them with arrows.



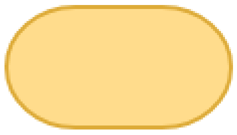
A flowchart is a graphical representations of steps. It was originated from computer science as a tool for representing algorithms and programming logic but had extended to use in all other kinds of processes. Nowadays, flowcharts play an extremely important role in displaying information and assisting reasoning. They help us visualize complex processes, or make explicit the structure of problems and tasks. A flowchart can also be used to define a process or project to be implemented.

Flowchart Symbols

Different flowchart shapes have different conventional meanings. The meanings of some of the more common shapes are as follows:

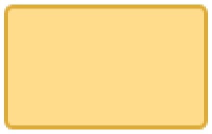
Terminator

The terminator symbol represents the starting or ending point of the system.



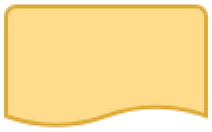
Process

A box indicates some particular operation.



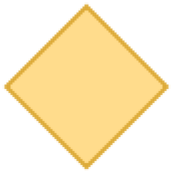
Document

This represents a printout, such as a document or a report.



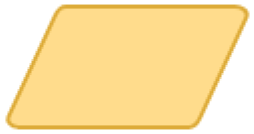
Decision

A diamond represents a decision or branching point. Lines coming out from the diamond indicates different possible situations, leading to different sub-processes.



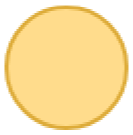
Data

It represents information entering or leaving the system. An input might be an order from a customer. Output can be a product to be delivered.



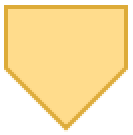
On-Page Reference

This symbol would contain a letter inside. It indicates that the flow continues on a matching symbol containing the same letter somewhere else on the same page.



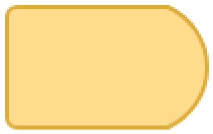
Off-Page Reference

This symbol would contain a letter inside. It indicates that the flow continues on a matching symbol containing the same letter somewhere else on a different page.



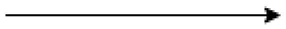
Delay or Bottleneck

Identifies a delay or a bottleneck.



Flow

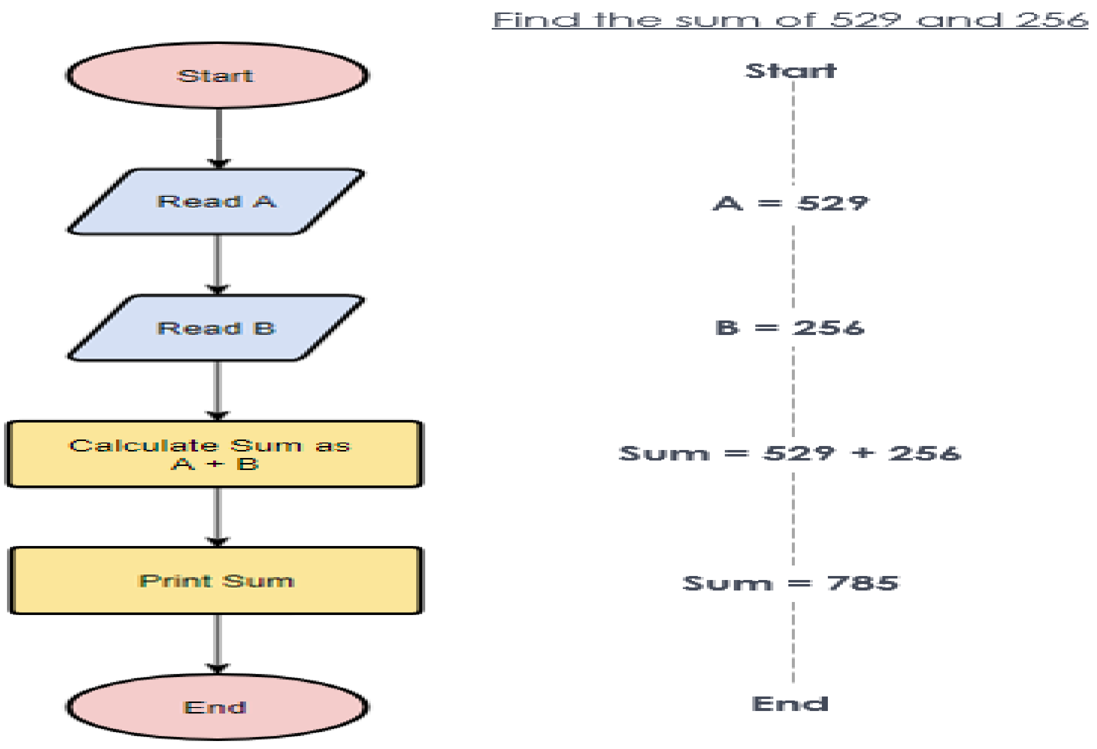
Lines represent the flow of the sequence and direction of a process.



When to Draw Flowchart?

Using a flowchart has a variety of benefits:

- It helps to clarify complex processes.
- It identifies steps that do not add value to the internal or external customer, including delays; needless storage and transportation; unnecessary work, duplication, and added expense; breakdowns in communication.
- It helps team members gain a shared understanding of the process and use this knowledge to collect data, identify problems, focus discussions, and identify resources.
- It serves as a basis for designing new processes.



Uses of Flowchart

Visual Clarity

One of the biggest benefits of a flowchart is the tool's ability to visualize multiple progresses and their sequence into a single document. Stakeholders throughout an organization can easily understand the workflow while finding out which step is unnecessary and which progress should be improved.

Instant Communication

Teams can use flowcharts to replace meetings. Simply clarifying progresses offers an easy, visual method to help team members instantly understand what they should do step by step.

Effective Coordination

For project managers and resource schedulers, the benefits of a flowchart include the ability to sequence events and reduce the potential for overburdening team members. Eliminating the unnecessary steps help to save time and resources.

Efficiency Increase

Efficiency increases are a significant benefit of flowcharts. The flowchart lists each step necessary to perform a process. The flowchart helps a designer remove unnecessary steps in a process, as well as errors. The flowchart should only include the steps that are requirements to reach the endpoint of the process.

Effective Analysis

With the help of flowchart, problem can be analyzed in more effective way. It specifically shows what type of action each step in a process requires. Generally, a rectangle with rounded edges defines the beginning or end of the process, a diamond shape shows the point at which a decision is required, and a square block shows an action taken during the process. A flowchart may also include symbols that show the type of media in which data is stored, such as a rectangle with a curved bottom to show a paper document or a cylinder to symbolize a computer hard drive.

Problem Solving

Flowcharts break a problem up into easily definable parts. The defined process displayed by the flowchart demonstrates the method of solving a complex problem. A flowchart reduces the chance that a necessary step for solving a problem will be left out because it appears obvious. In this way, it reduces cost and wastage of time.

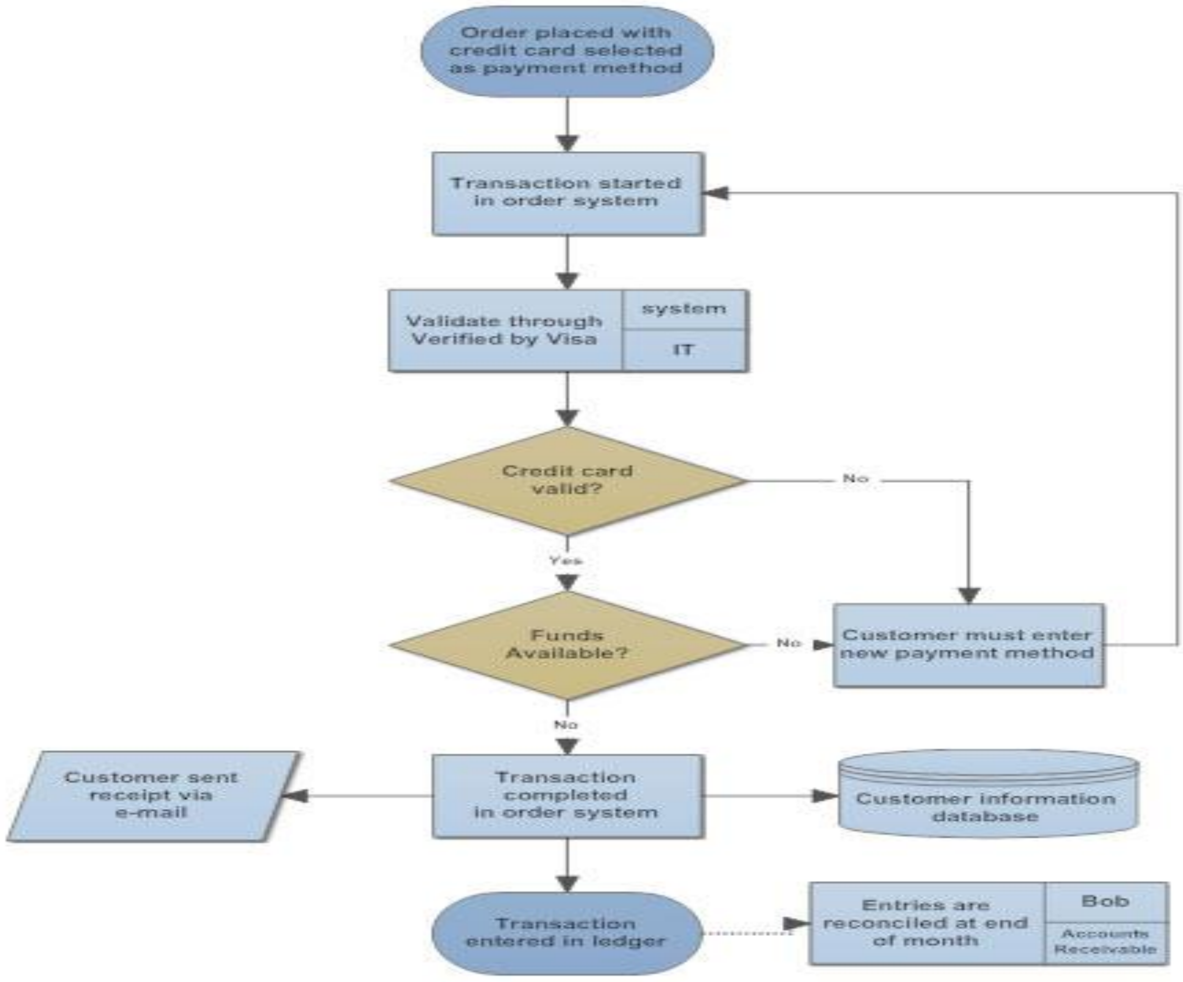
Proper Documentation

Digital flowcharts serve as a good paperless documentation, which is needed for various purposes, making things more efficient.

Sample Flowchart

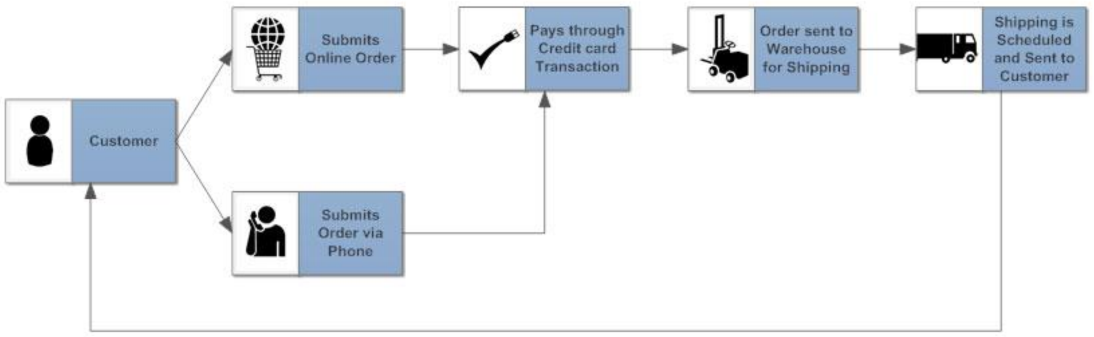
Basic Process Flowchart

One of the most frequent uses of flowcharts is to map out a new project. Engineers and software designers often use flowcharts for this purpose, but others may find them useful, as well. They are particularly helpful when the project will involve a sequence of steps that involve decisions. Here's a basic flowchart that shows this:


Order placed with credit card selected as payment method
Transaction started in order system
Validate through Verified by Visa
system
IT
Credit card valid?
No
Yes
Funds Available?
No
Customer must enter new payment method
No
Customer sent receipt via e-mail
Transaction completed in order system
Customer information database
Transaction entered in ledger
Entries are reconciled at end of month
Bob
Accounts Receivable

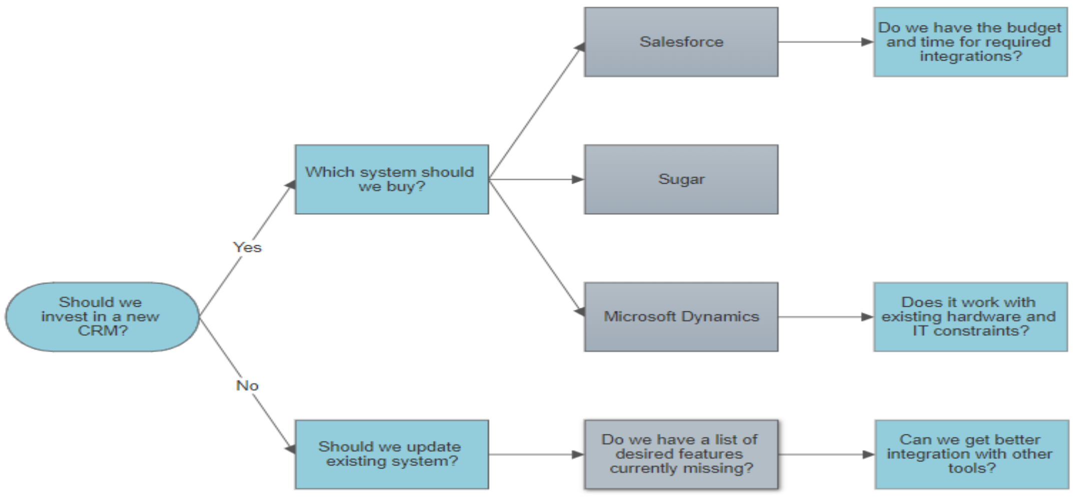
Workflow Process

Systems for managing workflow are best illustrated using a workflow diagram. These systems can focus on process integration, human task orientation, or both. The goal is to create a consistent, quality output based on a standardized set of procedures. Here is an example of a simple workflow diagram:


Customer
Submits Online Order
Submits Order via Phone
Pays through Credit card Transaction
Order sent to Warehouse for Shipping
Shipping is Scheduled and Sent to Customer

Decision Flowchart

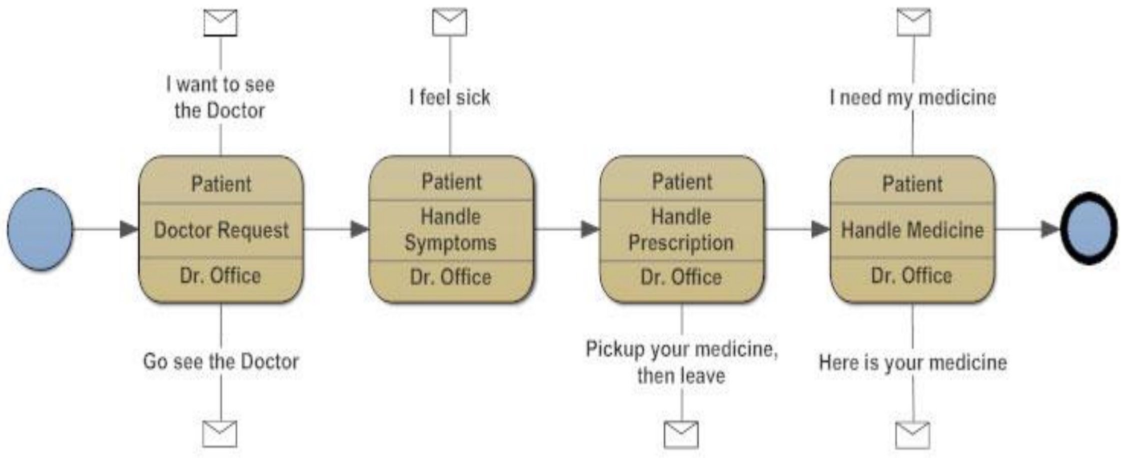
A decision flowchart lets you visualize the options in any important business decision. Walk through potential outcomes and make sure you consider all the questions before making a decision.


Should we invest in a new CRM?
Yes
No
Which system should we buy?
Salesforce
Sugar
Microsoft Dynamics
Do we have the budget and time for required integrations?
Does it work with existing hardware and IT constraints?
Should we update existing system?
Do we have a list of desired features currently missing?
Can we get better integration with other tools?

EPC Diagram

Business processes can cover a wide range of activities. They might be a simple set of tasks or a complex array of them that cover a number of possible situations. Modeling these processes is done to ensure a consistent, predictable outcome. Documenting or modeling a business process is using an event-driven process chain (EPC) diagram. A basic example is shown here:

The EPC diagram is a specialized type of flowchart designed specifically for this purpose. It has a unique library of symbols not found in traditional flowcharts.

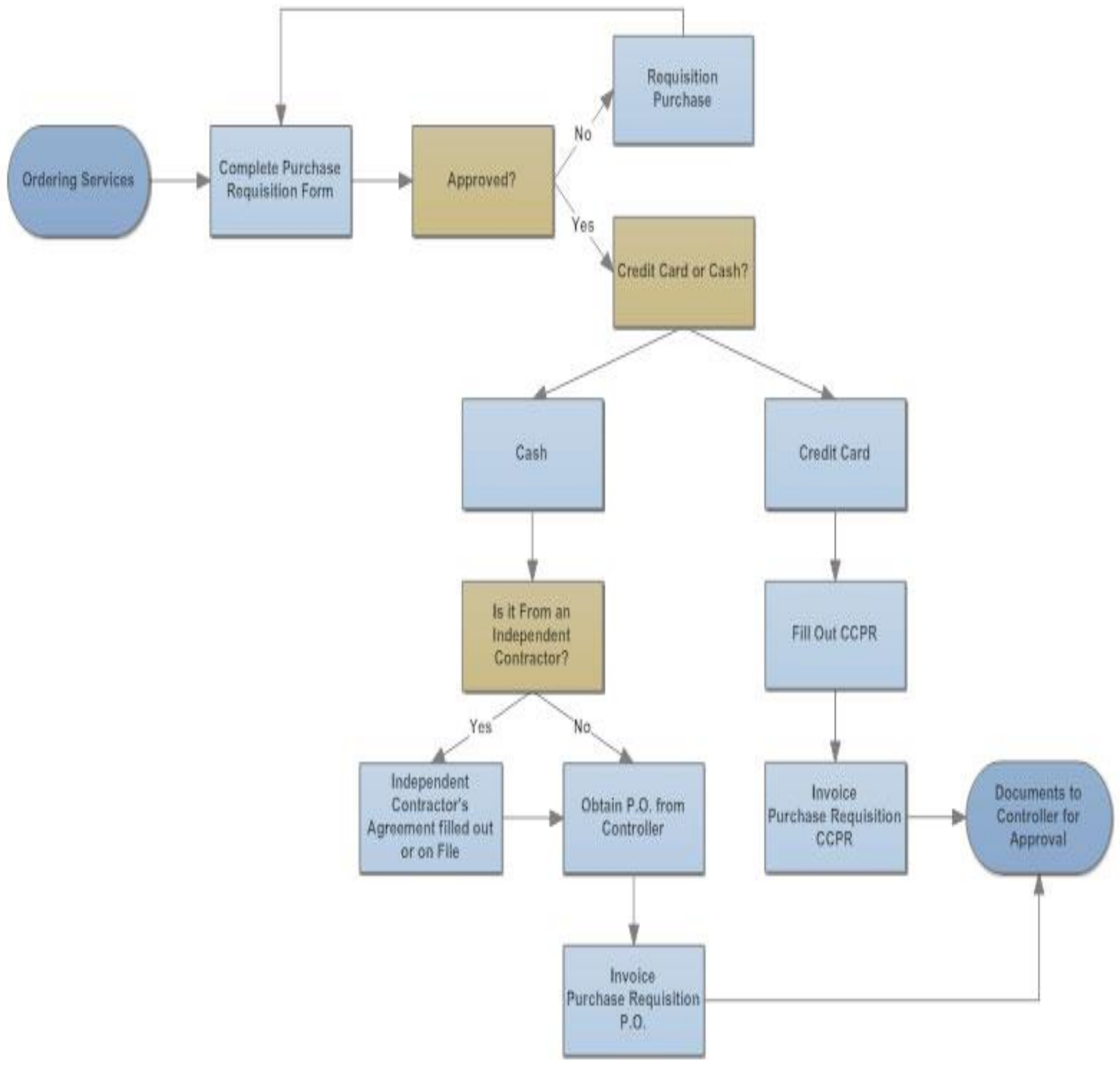


Process Map

Flowcharts may be used for diagnosing a malfunction or to troubleshoot problems. These uses are common in the fields of software and electronics. But they aren't confined to just these disciplines.

A process map is a detailed flowchart that is a useful tool for auditing a process. There are four steps used in creating a process map:

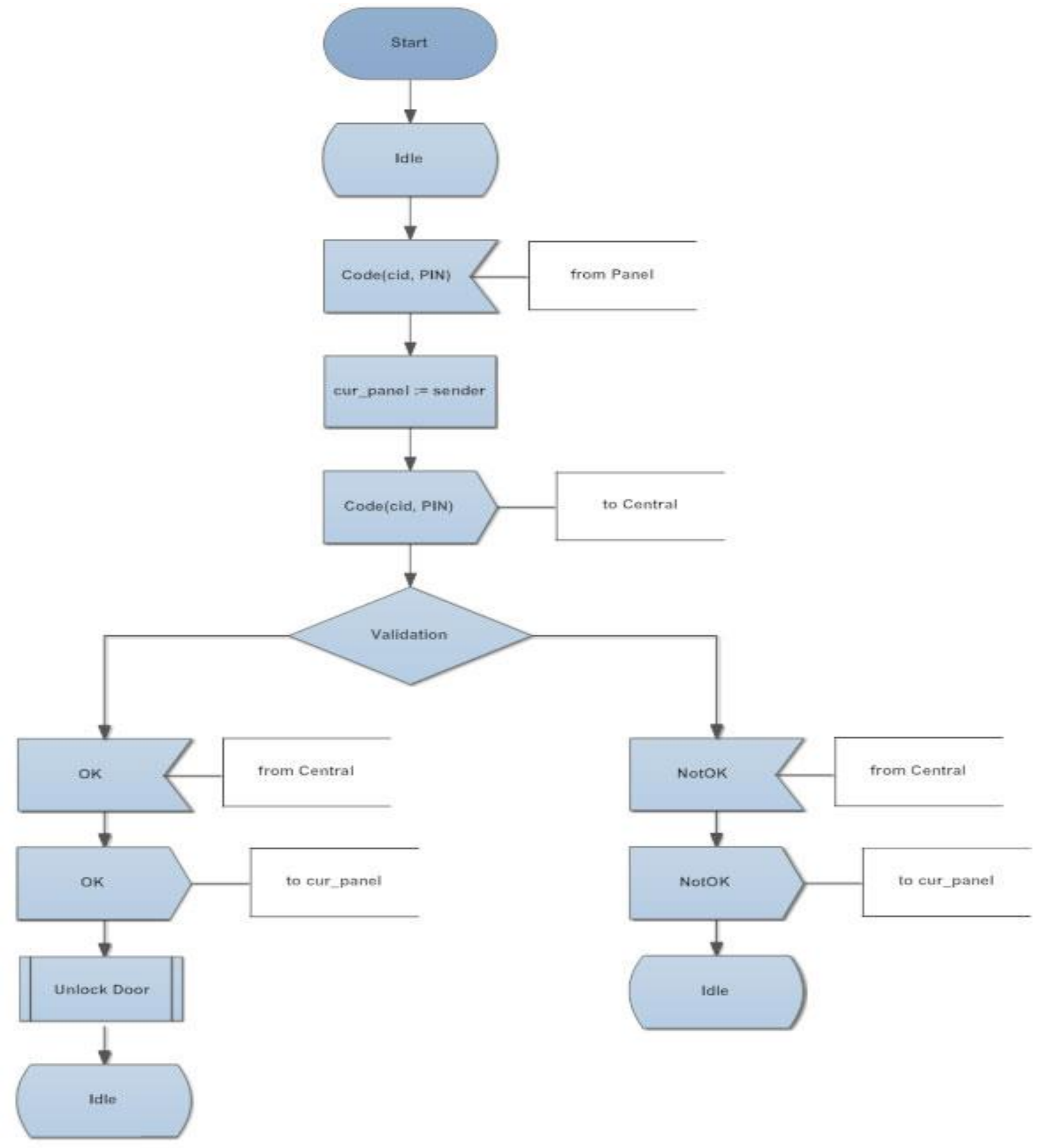
1. Identifying and understanding the steps in a process.
2. Gathering information to identify the objectives, risks and controls in a process.
3. Interviewing the individuals involved and creating the process map.
4. Analyzing and effecting changes to improve the process.


Ordering Services
Complete Purchase Requisition Form
Approved?
No
Requisition Purchase
Yes
Credit Card or Cash?
Cash
Credit Card
Is it From an Independent Contractor?
Fill Out CCPR
Yes
No
Independent Contractor's Agreement filled out or on File
Obtain P.O. from Controller
Invoice Purchase Requisition CCPR
Documents to Controller for Approval
Invoice Purchase Requisition P.O.

SDL Diagram

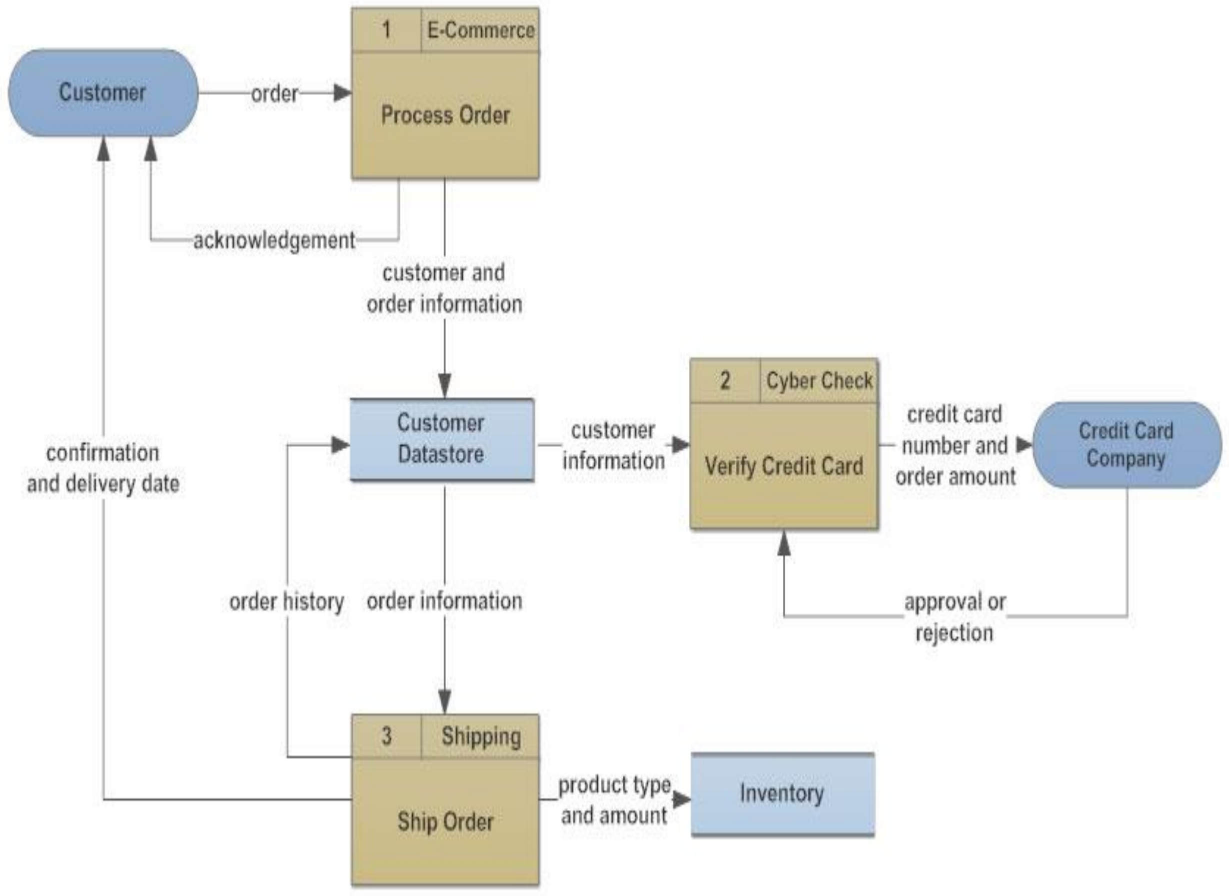
Brainstorming computer algorithms is often accomplished using an SDL diagram. SDL stands for Specification and Description Language. This is a flowchart that offers a unique set of symbols that are used to map out real-time systems. The three basic components of an SDL diagram are the system definition, the block, and the process.

One of the reasons that flowcharts are used frequently for program and network design is that they also offer a good resource for internal problem solving. They are also a great way to show customers how to troubleshoot common problems, because they are visual and are easy to follow, when presented properly.



Data Flow Diagram

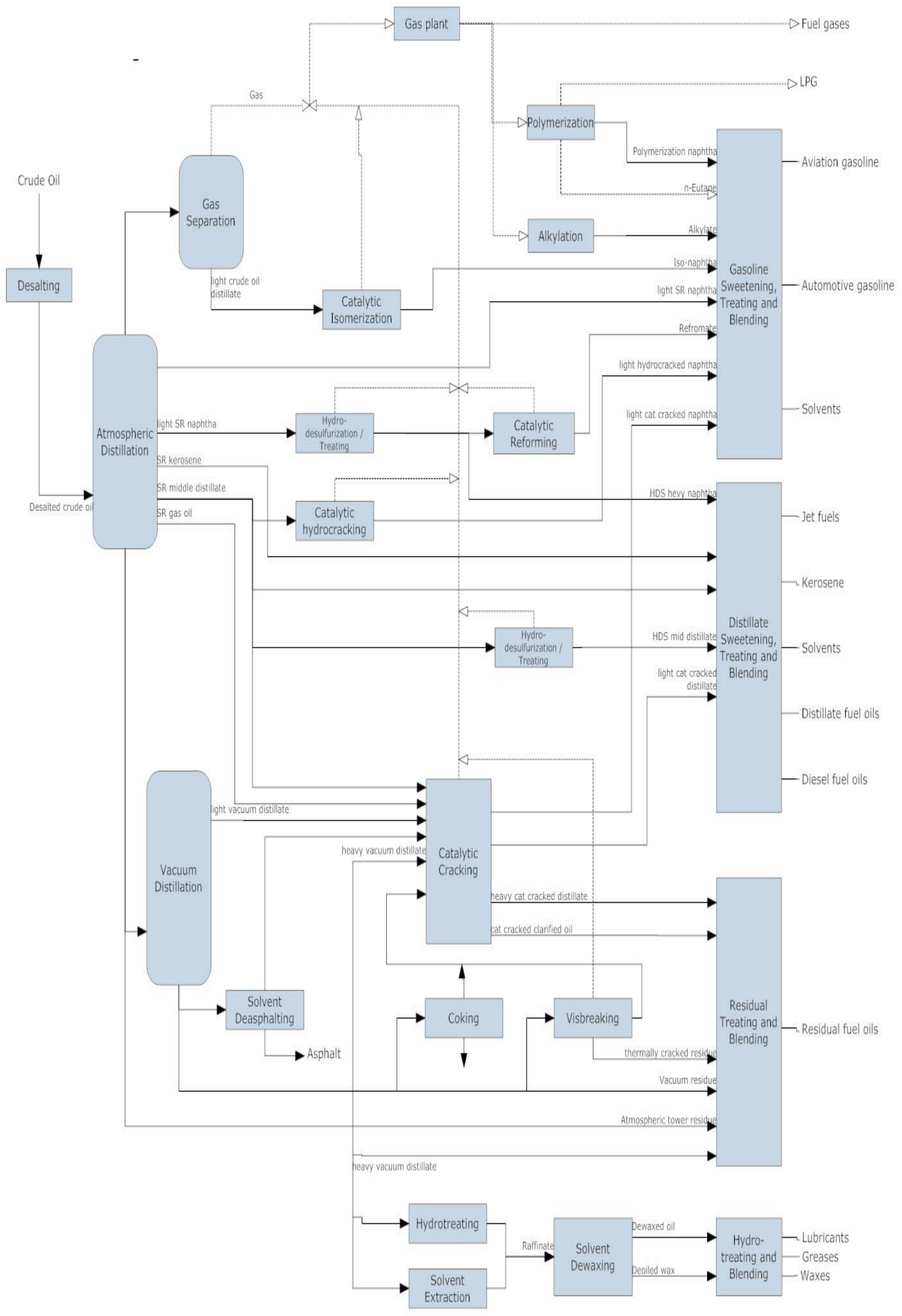
Data flow diagrams (DFD) are an efficient way of bridging the communication gap between system developers and users. They are specialized flowcharts that distill a substantial amount of information into a relatively few symbols and connectors.



Process Flow Diagram

A process flow diagram (PFD) is a technical illustration also known as a flowsheet. It is used to exhibit high-level processes in chemical and process engineering. The PFD will focus on major plant processes but not show minor details. This type of diagram is used for a wide range of engineering applications.

- Oil and petroleum refining
- Natural gas systems
- Green energy, such as wind and solar power
- Water treatment and processes
- Electrical power plants
- Piping and irrigation systems


Gas plant
Fuel gases
LPG
Gas
Polymerization
Polymerization naphtha
n-Eutane
Aviation gasoline
Crude Oil
Gas Separation
Alkylation
Alkylate
Desalting
Iso-naphtha
light crude oil distillate
Catalytic Isomerization
light SR naphtha
Gasoline Sweetening, Treating and Blending
Automotive gasoline
Refromate
light hydrocracked naphtha
Solvents
light cat cracked naphtha
Atmospheric Distillation
light SR naphtha
Hydro-desulfurization / Treating
Catalytic Reforming
SR kerosene
SR middle distillate
Desalted crude oil
SR gas oil
Catalytic hydrocracking
HDS hevy naphtha
Jet fuels
Kerosene
Distillate Sweetening, Treating and Blending
Hydro-desulfurization / Treating
HDS mid distillate
Solvents
light cat cracked distillate
Distillate fuel oils
Diesel fuel oils
light vacuum distillate
heavy vacuum distillate
Catalytic Cracking
Vacuum Distillation
heavy cat cracked distillate
cat cracked clarified oil
Residual Treating and Blending
Residual fuel oils
Solvent Deasphalting
Coking
Visbreaking
Asphalt
thermally cracked residue
Vacuum residue
Atmospheric tower residue
heavy vacuum distillate
Hydrotreating
Raffinate
Solvent Dewaxing
Dewaxed oil
Deoiled wax
Hydro-treating and Blending
Lubricants
Greases
Waxes
Solvent Extraction

Activities/Assessment

I. **Identification.** Read each statement or question below carefully and fill in the blank(s) with the correct answer.

1. _______________ is a symbol would contain a letter inside. It indicates that the flow continues on a matching symbol containing the same letter somewhere else on the same page.
2. _______________is a technical illustration also known as a flowsheet. It is used to exhibit high-level processes in chemical and process engineering.
3. ______________ an operator consists of three operands and is used to evaluate Boolean expressions.
4. ______________ is the standard output stream that is used to produce the result of a program on an output device like the computer screen.
5. ______________ a variable declared inside the body of the method.

II. **Application.** Read the questions carefully and confine your responses to an analysis of the questions as written.

1. Write the structure of a java program.

2. Explain the different java compilers.

3. Compare local and global variable using Venn diagram.

4. Expound the importance of using flowchart.

5. Create a flowchart of your enrollment system

Conditional Constructs

Learning Outcomes:

After successful completion of this lesson, you should be able to:

- Use single, two-way and multi-way selection in an algorithm.

Conditional Statement

In computer science, conditional statements, conditional expressions and conditional constructs are features of a programming language, which perform different computations or actions depending on whether a programmer-specified Boolean condition evaluates to true or false. It helps you to make a decision based on certain conditions. These conditions are specified by a set of conditional statements having Boolean expressions which are evaluated to a Boolean value true or false.

Conditional statements in a computer program support decisions based on a certain condition. If the condition is met, or "true," a certain piece of code is executed.

For example, you want to convert user-entered text to lowercase. Execute the code only if the user entered capitalized text. If not, you don't want to execute the code because it will lead to a runtime error.

There are two main conditional statements used in Java: the if-then and if-then-else statements, and the switch statement.

Conditional Operators

In the example above, we used a single operator. These are the standard operators you can use:

- equal to: =
- less than: <
- more than: >
- greater than or equal to: >=
- less than or equal to: >=

In addition to these, there are four more operators used with conditional statements:

- and: &&
- not:!
- or: ||
- is equal to: ==

For example, the driving age is considered to be from age 16 to age 85, in which case the AND operator can be used.

else if (age > 16 && age < 85)

This will return true only if both conditions are met. The operators NOT, OR, and IS EQUAL TO can be used in a similar way.

The If-Then and If-Then-Else Statements (Single Selection)

The most basic flow control statement in Java is if-then: if [something] is true, do [something]. This statement is a good choice for simple decisions. The basic structure of an if statement starts with the word "if," followed by the statement to test, followed by curly braces that wrap the action to take if the statement is true. It looks like this:

```
if ( statement ) {// do something here....}
```

This statement can also be extended to do something else if the condition is false:

if (statement) { // do something here...}
else {// do something else...}

For example, if you are determining whether someone is old enough to drive, you might have a statement that says "if your age is 16 or older, you can drive; else, you cannot drive."

Int age = 17;

if age >= 16 {System.out.println("You can drive.");}

else { System.out.println("You are not old enough to drive.")

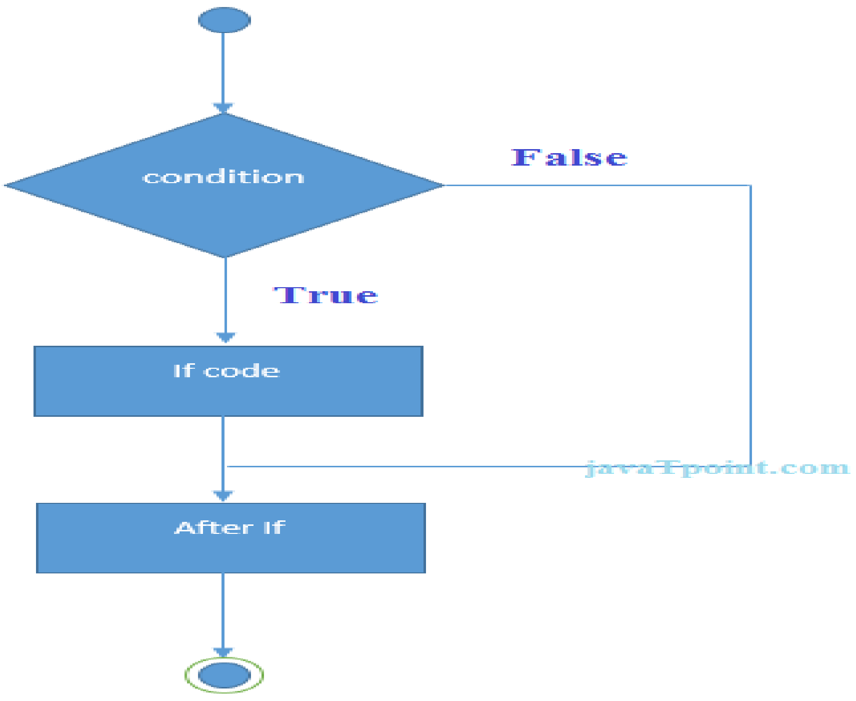
There is no limit to the number of else statements you can add.

If Statement

Use the if statement to specify a block of Java code to be executed if a condition is true.

```
Syntax:

if (condition) {
// block of code to be executed if the condition is true
}
```

Note: that if is in lowercase letters. Uppercase letters (If or IF) will generate an error.



Examples:

```
if (20 > 18) {
System.out.println("20 is greater than 18");
}
```

We can also test variables:

```
int x = 20;
int y = 18;
if (x > y) {
  System.out.println("x is greater than y");
}
```

Example explained

In the example above we use two variables, x and y, to test whether x is greater than y (using the > operator). As x is 20, and y is 18, and we know that 20 is greater than 18, we print to the screen that "x is greater than y".

If Else Statement

Use the else statement to specify a block of code to be executed if the condition is false.

Syntax:

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

Example:

```
int time = 20;
if (time < 18) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
// Outputs "Good evening."
```

Example explained

In the example above, time (20) is greater than 18, so the condition is false. Because of this, we move on to the else condition and print to the screen "Good evening". If the time was less than 18, the program would print "Good day".

A year is leap, if it is divisible by 4 and 400. But, not by 100.

```
public class LeapYearExample {
public static void main(String[] args) {
    int year=2020;
    if(((year % 4 ==0) && (year % 100 !=0)) || (year % 400==0)){
        System.out.println("LEAP YEAR");
    }
    else{
        System.out.println("COMMON YEAR");
    }
}
}
```

Output:

```
LEAP YEAR
```

Example of if-else statement

```
public class IfElseExample {

   public static void main(String args[]){
     int num=120;
     if( num < 50 ){
              System.out.println("num is less than 50");
     }
     else {
              System.out.println("num is greater than or equal 50");
     }
   }
}
```

Output:

```
num is greater than or equal 5
```

Multiple Selection Construct (If, Else If, Else Statement)

Use the else if statement to specify a new condition if the first condition is false.

Syntax:

```
if (condition1) {
// block of code to be executed if condition1 is true
} else if (condition2) {
// block of code to be executed if the condition1 is false and condition2 is true
} else {
// block of code to be executed if the condition1 is false and condition2 is false
}
```

Example:

```
int time = 22;
if (time < 10) {
  System.out.println("Good morning.");
} else if (time < 20) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
// Outputs "Good evening."
```

Example explained

In the example above, time (22) is greater than 10, so the first condition is false. The next condition, in the else if statement, is also false, so we move on to the else condition since condition1 and condition2 is both false - and print to the screen "Good evening".

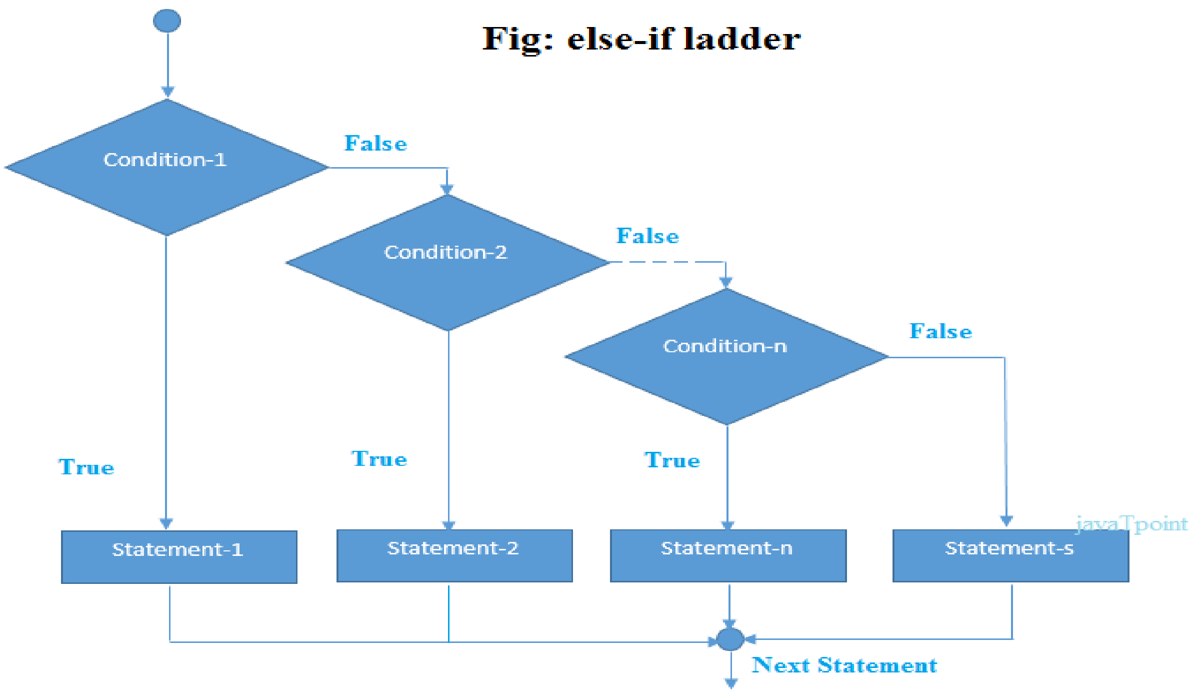
Multi-way Statement (if-else-if ladder Statement)

The if-else-if ladder statement executes one condition from multiple statements.

```
if(condition_1) {
/*if condition_1 is true execute this*/
statement(s);
}
else if(condition_2) {
/* execute this if condition_1 is not met and
* condition_2 is met
*/
statement(s);
}
else if(condition_3) {
/* execute this if condition_1 & condition_2 are
* not met and condition_3 is met
*/
statement(s);
}
.
.
.
else {
/* if none of the condition is true
* then these statements gets executed
*/
statement(s);
}
```

Note:

The most important point to note here is that in if-else-if statement, as soon as the condition is met, the corresponding set of statements get executed, rest gets ignored. If none of the condition is met then the statements inside “else” gets executed.



Example of if-else-if

```
public class IfElseIfExample {

  public static void main(String args[]){
          int num=1234;
          if(num <100 && num>=1) {
            System.out.println("Its a two digit number");
          }
          else if(num <1000 && num>=100) {
            System.out.println("Its a three digit number");
          }
          else if(num <10000 && num>=1000) {
            System.out.println("Its a four digit number");
          }
          else if(num <100000 && num>=10000) {
            System.out.println("Its a five digit number");
          }
          else {
            System.out.println("number is not between 1 & 99999");
          }
  } }
```

Output:

It’s a four digit number

Example:

```
//Java Program to demonstrate the use of If else-if ladder.
//It is a program of grading system for fail, D grade, C grade, B grade, A grade and A+.

public class IfElseIfExample {
public static void main(String[] args) {
    int marks=65;

    if(marks<50){
        System.out.println("fail");
    }
    else if(marks>=50 && marks<60){
        System.out.println("D grade");
    }
    else if(marks>=60 && marks<70){
        System.out.println("C grade");
    }
    else if(marks>=70 && marks<80){
        System.out.println("B grade");
    }
    else if(marks>=80 && marks<90){
        System.out.println("A grade");
    }else if(marks>=90 && marks<100){
        System.out.println("A+ grade");
    }else{
        System.out.println("Invalid!");
   }
       }
  }
```

Output: C grade

Activities/Assessment

I. Application. Read the questions carefully and confine your responses to an analysis of the questions as written.

Use ***If-Then and If-Then-Else Statements only***

1. Write a program that will accept an integer and execute one of the following based on the input statement.
   - If 0, Display only “Hello World”.
   - If 2, Display only “I am Groot”.
   - If 3, Display only “To The Top”.
   - If 4, Display only “Where is the Horizon?”
   - If 5, Display only “I don’t know”.
   - If 6, Display only “Yeah, I will”.

2. Write a java program that keeps a number from the user and generates an integer between 1 and 7 and display the weekdays.

   Sample output:

   ```
   Input number: 3
   Wednesday
   ```

3. Write a Java Program that takes the user to provide a single character from the alphabet. Print Vowel or Consonant depending on the user input. If the user input is not a letter print an error message.

4. Write a program that reads a whole number and prints “zero” if the number is zero. Otherwise, print “positive” or “negative”.

5. Write a program that takes of the age from the user and display “child” if the age below 13, display “teen” if the age around 13-19, and “adult” from 20 and above.

Switched Statement

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

Syntax:

```
switch(expression) {
  case value :
    // Statements
    break; // optional

  case value :
    // Statements
    break; // optional

  // You can have any number of case statements.
  default : // Optional
    // Statements
}
```

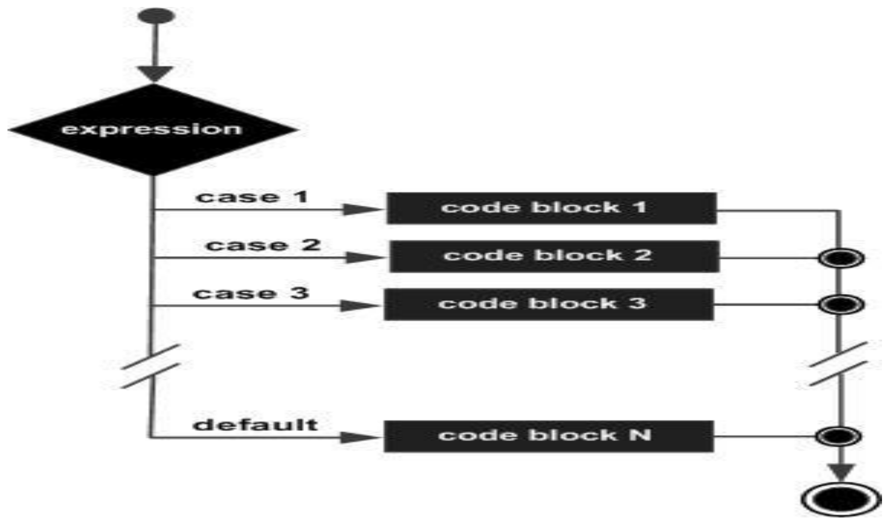
Use the switch statement to select one of many code blocks to be executed.

The break Keyword

- When Java reaches a break keyword, it breaks out of the switch block.
- This will stop the execution of more code and case testing inside the block.
- When a match is found, and the job is done, it's time for a break. There is no need for more testing.
- A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.

The default Keyword

The default keyword specifies some code to run if there is no case match:



Note that if the default statement is used as the last statement in a switch block, it does not need a break.

Points to Remember

- There can be *one or N number of case values* for a switch expression.
- The case value must be of switch expression type only. The case value must be *literal or constant*. It doesn't allow variables.
- The case values must be *unique*. In case of duplicate value, it renders compile-time error.
- The Java switch expression must be of *byte, short, int, long (with its Wrapper type), enums and string*.
- Each case statement can have a *break statement* which is optional. When control reaches to the break statement, it jumps the control after the switch expression. If a break statement is not found, it executes the next case.
- The case value can have a *default label* which is optional.

Example:

```
public class Test {

  public static void main(String args[]) {
    // char grade = args[0].charAt(0);
    char grade = 'C';

    switch(grade) {
      case 'A' :
        System.out.println("Excellent!");
        break;
      case 'B' :
      case 'C' :
        System.out.println("Well done");
        break;
      case 'D' :
        System.out.println("You passed");
      case 'F' :
        System.out.println("Better try again");
        break;
      default :
        System.out.println("Invalid grade");
    }
    System.out.println("Your grade is " + grade);
  }
}
```

Compile and run the above program using various command line arguments. This will produce the following result:

Output:

```
Well done
Your grade is C
```

Example:

```
int day = 4;
switch (day) {
  case 6:
    System.out.println("Today is Saturday");
    break;
  case 7:
    System.out.println("Today is Sunday");
    break;
  default:
    System.out.println("Looking forward to the Weekend");
}
// Outputs "Looking forward to the Weekend"
```

Example:

```
public class SwitchExample {
public static void main(String[] args) {
    //Declaring a variable for switch expression
    int number=20;
    //Switch expression
    switch(number){
    //Case statements
    case 10: System.out.println("10");
    break;
    case 20: System.out.println("20");
    break;
    case 30: System.out.println("30");
    break;
    //Default case statement
    default:System.out.println("Not in 10, 20 or 30");
    }    }    }
```

Output:

20

Finding Month Example:

```
//Java Program to demonstrate the example of Switch statement
//where we are printing month name for the given number
public class SwitchMonthExample {
public static void main(String[] args) {
    //Specifying month number
    int month=7;
    String monthString="";
    //Switch statement
    switch(month){
    //case statements within the switch block
    case 1: monthString="1 - January";
    break;
    case 2: monthString="2 - February";
    break;
    case 3: monthString="3 - March";
    break;
    case 4: monthString="4 - April";
    break;
    case 5: monthString="5 - May";
    break;
    case 6: monthString="6 - June";
    break;

   case 7: monthString="7 - July";
    break;
    case 8: monthString="8 - August";
    break;

case 9: monthString="9 - September";
break;
case 10: monthString="10 - October";

break;
case 11: monthString="11 - November";
break;
case 12: monthString="12 - December";
break;
default:System.out.println("Invalid Month!");
}
//Printing month of the given number
System.out.println(monthString);
}
}
```

Finding Month Example:

Output: 7 – July

Program to check Vowel or Consonant:

```
public class SwitchVowelExample {
public static void main(String[] args) {
    char ch='O';
    switch(ch)
    {
        case 'a':
            System.out.println("Vowel");
            break;
        case 'e':
            System.out.println("Vowel");
            break;
        case 'i':
            System.out.println("Vowel");
            break;
        case 'o':
            System.out.println("Vowel");
            break;
        case 'u':
            System.out.println("Vowel");
            break;
        case 'A':
            System.out.println("Vowel");
            break;
        case 'E':
            System.out.println("Vowel");
            break;
        case 'I':
            System.out.println("Vowel");
            break;
        case 'O':
            System.out.println("Vowel");
            break;
        case 'U':
            System.out.println("Vowel");
            break;
        default:
            System.out.println("Consonant");
    }
}
}
```

Output:

```
Vowel
```

Activities/Assessment

I. Application. Read the questions carefully and confine your responses to an analysis of the questions as written.

Use ***Switched Statement only***

1. Write a program that will accept an integer and execute one of the following based on the input statement.
   - If 0, Display only "Hello World".
   - If 2, Display only "I am Groot".
   - If 3, Display only "To The Top".
   - If 4, Display only "Where is the Horizon?"
   - If 5, Display only "I don't know".
   - If 6, Display only "Yeah, I will".

2. Write a java program that keeps a number from the user and generates an integer between 1 and 7 and display the weekdays.

   Sample output:

   ```
   Input number: 3
   Wednesday
   ```

3. Write a Java Program that takes the user to provide a single character from the alphabet. Print Vowel or Consonant depending on the user input. If the user input is not a letter print an error message.

4. Write a program that reads a whole number and prints "zero" if the number is zero. Otherwise, print "positive" or "negative".

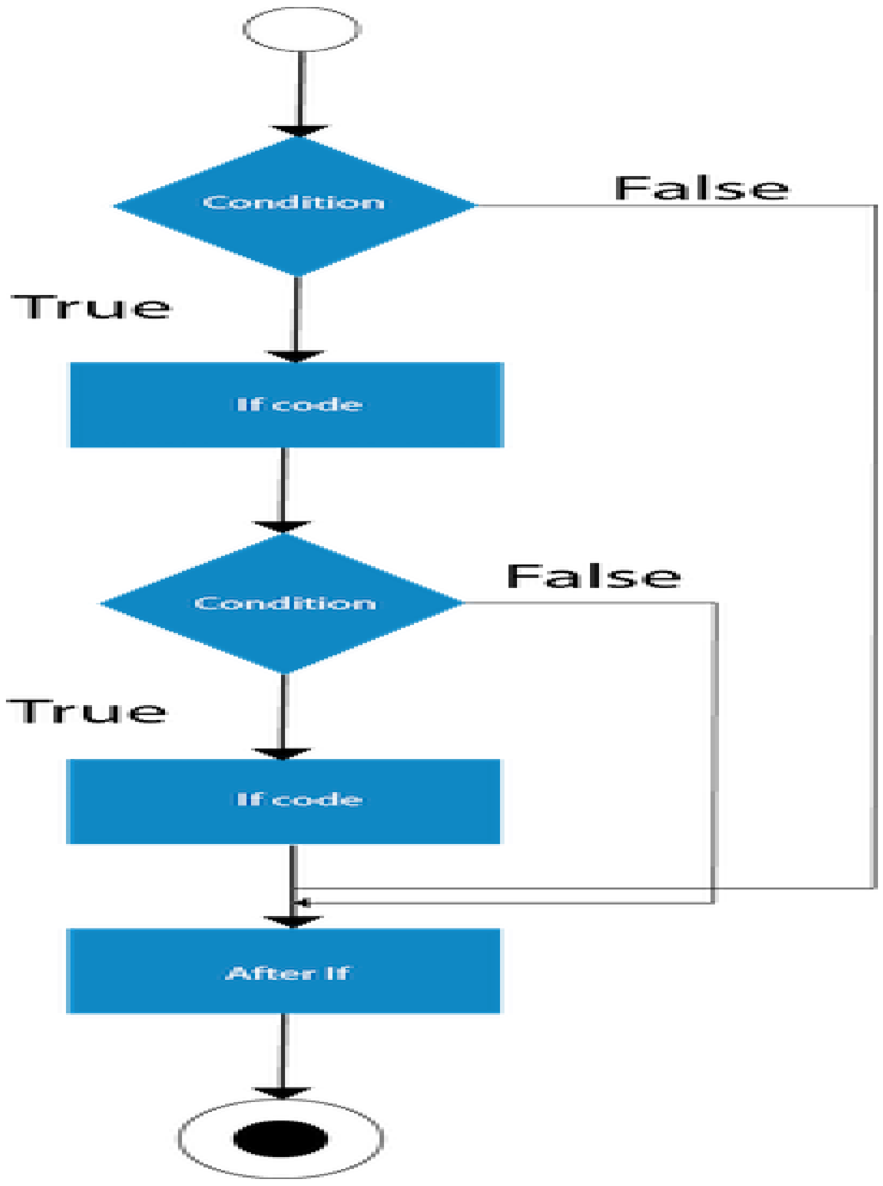
5. Write a program that takes of the age from the user and display "child" if the age below 13, display "teen" if the age around 13-19, and "adult" from 20 and above.

Nested If- Statement

The nested if statement represents the if block within another if block. Here, the inner if block condition executes only when outer if block condition is true.

Syntax:

```
1. if(condition){
2. //code to be executed
3. if(condition){
4. //code to be executed
5. }
6. }
```



Example 1:

```
//Java Program to demonstrate the use of Nested If Statement.
public class JavaNestedIfExample {
public static void main(String[] args) {
    //Creating two variables for age and weight
    int age=20;
    int weight=80;
    //applying condition on age and weight
    if(age>=18){
        if(weight>50){
            System.out.println("You are eligible to donate blood");
        }
    }
}}
```

Output:

```
You are eligible to donate blood
```

Example 2:

```
//Java Program to demonstrate the use of Nested If Statement.
public class JavaNestedIfExample2 {
public static void main(String[] args) {
//Creating two variables for age and weight
int age=25;
int weight=48;
//applying condition on age and weight
if(age>=18){
if(weight>50){
System.out.println("You are eligible to donate blood");
} else{
System.out.println("You are not eligible to donate blood");
}
} else{
System.out.println("Age must be greater than 18");
}
}  }
```

Output:

```
You are not eligible to donate blood
```

```
// Example for Nested If in Java Programming

package ConditionalStatements;

import java.util.Scanner;

public class NestedIf {
	private static Scanner sc;

	public static void main(String[] args) {
		int age;
		sc = new Scanner(System.in);
		System.out.println(" Please Enter you Age: ");
		age = sc.nextInt();

		if (age < 18) {
			System.out.println("You are Minor.");
			System.out.println("You are Not Eligible to Work");
		}
		else  {
			if (age >= 18 && age <= 60 )  {
				System.out.println("You are Eligible to Work");
				System.out.println("Please fill in your details and
apply");
			}
			else  {
				System.out.println("You are too old to work as per
the Government rules");
				System.out.println("Please Collect your pension!");
			}
		}
		System.out.println("\nThis Message is coming from Outside the IF
ELSE STATEMENT");
	}
}
```

Analysis

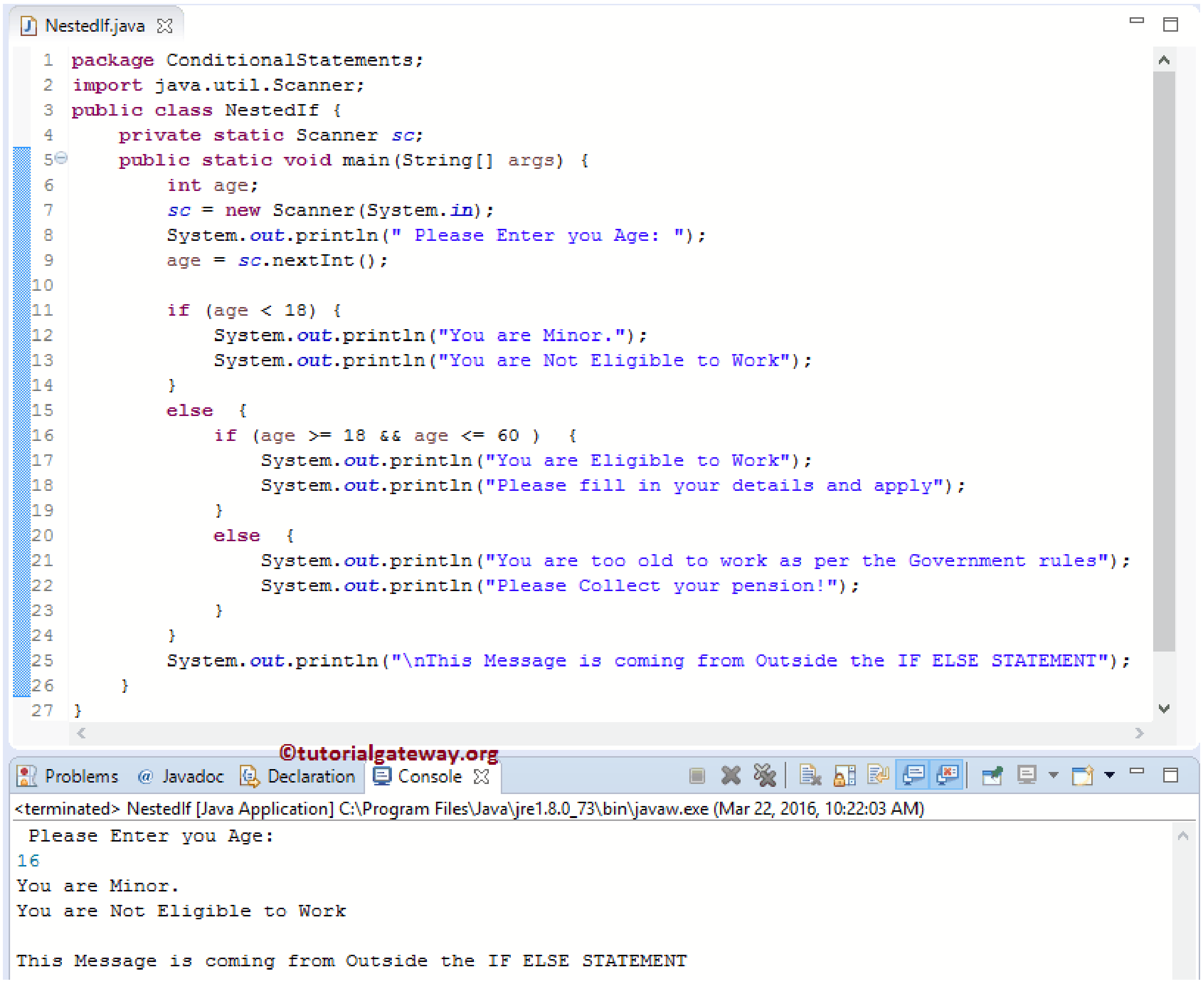
Within this Java nested if else program, If the person's age is less than 18, then he is not eligible to work. If the person's age is greater than or equal to 18, then the first condition fails. It will check the else statement.

Within the Else statement, there is another if condition (called as Nested If).

- Java Nested IF Statement will check whether the person's age is greater than or equal to 18 and less than or equal to 60. If the condition is TRUE, then he can apply for the job.

- If the Nested If condition is FALSE, then he is too old to work as per the government.
- We also place one System.out.println statement outside the If Else block, and it will execute irrespective of condition result.

Output 1: From the screenshot below, you can observe that We entered the age of 16. Here, the first If condition is TRUE. So, the statements inside the first if block will execute.

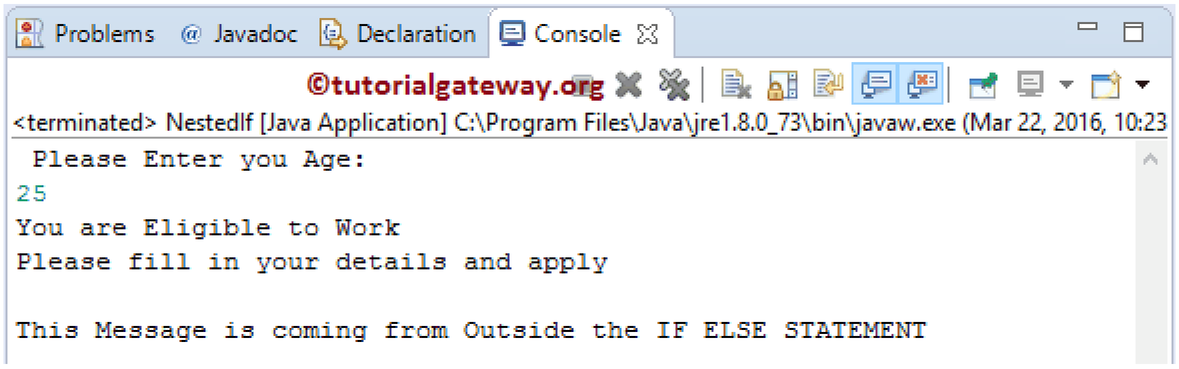


```java
package ConditionalStatements;
import java.util.Scanner;
public class NestedIf {
    private static Scanner sc;
    public static void main(String[] args) {
        int age;
        sc = new Scanner(System.in);
        System.out.println(" Please Enter you Age: ");
        age = sc.nextInt();

        if (age < 18) {
            System.out.println("You are Minor.");
            System.out.println("You are Not Eligible to Work");
        }
        else  {
            if (age >= 18 && age <= 60 )  {
                System.out.println("You are Eligible to Work");
                System.out.println("Please fill in your details and apply");
            }
            else  {
                System.out.println("You are too old to work as per the Government rules");
                System.out.println("Please Collect your pension!");
            }
        }
        System.out.println("\nThis Message is coming from Outside the IF ELSE STATEMENT");
    }
}
```

©tutorialgateway.org

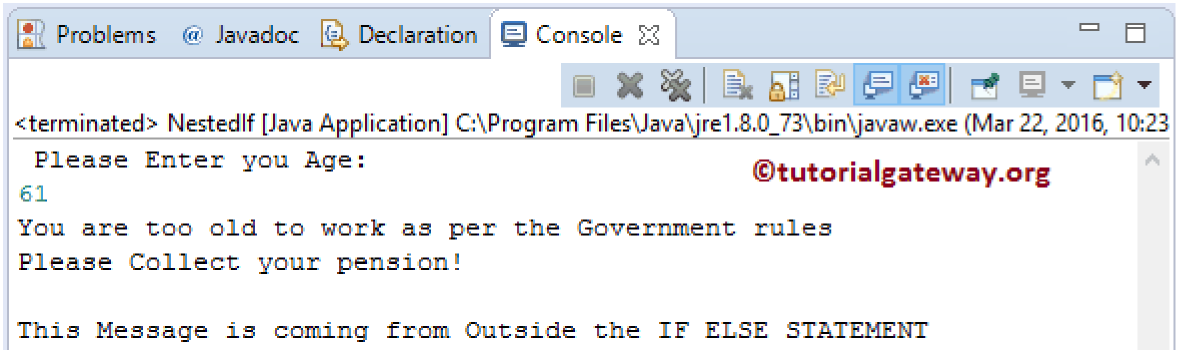
```
<terminated> NestedIf [Java Application] C:\Program Files\Java\jre1.8.0_73\bin\javaw.exe (Mar 22, 2016, 10:22:03 AM)
 Please Enter you Age:
16
You are Minor.
You are Not Eligible to Work

This Message is coming from Outside the IF ELSE STATEMENT
```

Output 2: We are going to enter age as 25 means the first IF condition is FALSE. It will go to else block. Within the else block, Javac will check the if (age >= 18 && age <=60), which is TRUE. So it will print the statements inside this block.



```
<terminated> NestedIf [Java Application] C:\Program Files\Java\jre1.8.0_73\bin\javaw.exe (Mar 22, 2016, 10:23
 Please Enter you Age:
25
You are Eligible to Work
Please fill in your details and apply

This Message is coming from Outside the IF ELSE STATEMENT
```

Output 3: This time, we are going to enter age as 61 to test the Nested If. It means the first IF condition is FALSE. It will go to else block. Within the else block, the Javac compiler will check the ***if (age >= 18 && age <=60)***, which is FALSE. That is why this program will print the statements inside Nested else block.



```
<terminated> NestedIf [Java Application] C:\Program Files\Java\jre1.8.0_73\bin\javaw.exe (Mar 22, 2016, 10:23
 Please Enter you Age:
61
You are too old to work as per the Government rules
Please Collect your pension!

This Message is coming from Outside the IF ELSE STATEMENT
```

Activities/Assessment

I. **Application.** Read the questions carefully and confine your responses to an analysis of the questions as written.

Use ***Nested if only***

1. Write a program that will accept an integer and execute one of the following based on the input statement.
    - If 0, Display only “Hello World”.
    - If 2, Display only “I am Groot”.
    - If 3, Display only “To The Top”.
    - If 4, Display only “Where is the Horizon?”
    - If 5, Display only “I don’t know”.
    - If 6, Display only “Yeah, I will”.

2. Write a java program that keeps a number from the user and generates an integer between 1 and 7 and display the weekdays.

    Sample output:

    Input number: 3

    Wednesday

3. Write a Java Program that takes the user to provide a single character from the alphabet. Print Vowel or Consonant depending on the user input. If the user input is not a letter print an error message.

4. Write a program that reads a whole number and prints “zero” if the number is zero. Otherwise, print “positive” or “negative”.

5. Write a program that takes of the age from the user and display “child” if the age below 13, display “teen” if the age around 13-19, and “adult” from 20 and above.

Using Looping Constructs

Learning Outcomes:

After successful completion of this lesson, you should be able to:

- Apply loop constructs to indicate repetitive tasks.

Event Controlled Loop Construct

In programming languages, loops are used to execute a set of instructions/functions repeatedly when some conditions become true. There are three types of loops in Java.

- for loop
- while loop
- do-while loop

Advantage with looping statement

- Reduce length of Code
- Take less memory space.
- Burden on the developer is reducing.
- Time consuming process to execute the program is reduced.

Difference between conditional and looping statement

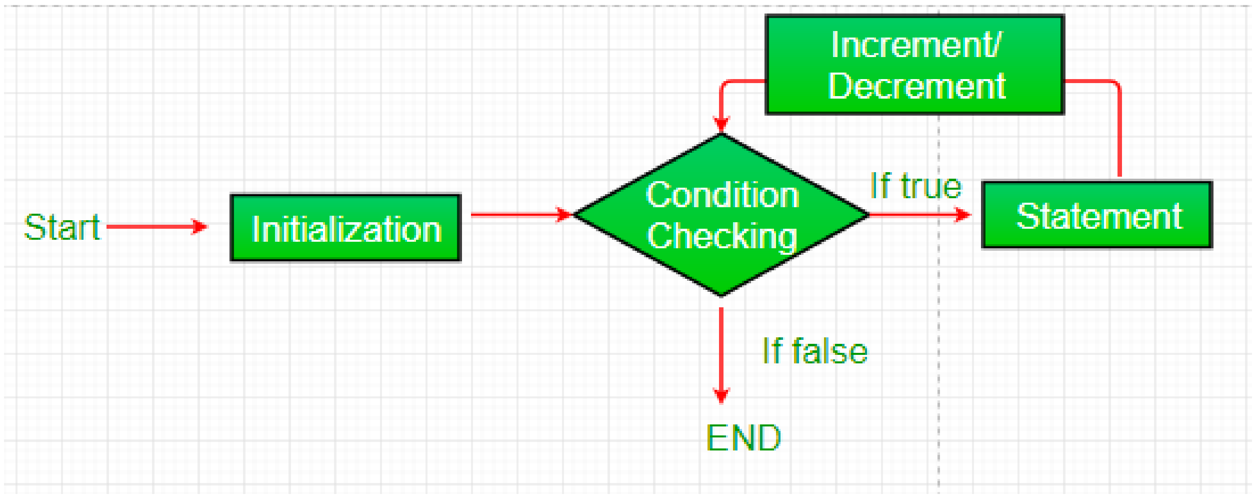
Conditional statement executes only once in the program where as looping statements executes repeatedly several number of time.

for loop: for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

Syntax:

```
for (initialization condition; testing condition;
                              increment/decrement)
{
   statement(s)
}
```

Flowchart:



1. Initialization condition: Here, we initialize the variable in use. It marks the start of a for loop. An already declared variable can be used or a variable can be declared, local to loop only.

2. Testing Condition: It is used for testing the exit condition for a loop. It must return a Boolean value. It is also an Entry Control Loop as the condition is checked prior to the execution of the loop statements.

3. Statement execution: Once the condition is evaluated to true, the statements in the loop body are executed.

4. Increment/ Decrement: It is used for updating the variable for next iteration.

5. Loop termination: When the condition becomes false, the loop terminates marking the end of its life cycle.

Example:

```
// Java program to illustrate for loop.
class forLoopDemo
{
   public static void main(String args[])
   {
      // for loop begins when x=2
      // and runs till x <=4
      for (int x = 2; x <= 4; x++)
         System.out.println("Value of x:" + x);
   }
}
```

Output:

```
Value of x: 2
Value of x: 3
Value of x: 4
```

Example:

```
for (int i = 0; i < 5; i++) {

  System.out.println(i);

}
```

Example explained

- Statement 1 sets a variable before the loop starts (int i = 0).
- Statement 2 defines the condition for the loop to run (i must be less than 5). If the condition is true, the loop will start over again, if it is false, the loop will end.
- Statement 3 increases a value (i++) each time the code block in the loop has been executed.

Another Example:

This example will only print even values between 0 and 10:

```
for (int i = 0; i <= 10; i = i + 2) {
  System.out.println(i);
}
```

Example:

```
class ForLoopExample {
    public static void main(String args[]){
         for(int i=10; i>1; i--){
              System.out.println("The value of i is: "+i);
         }
    }
}
```

The output of this program is:

```
The value of i is: 10
The value of i is: 9
The value of i is: 8
The value of i is: 7
The value of i is: 6
The value of i is: 5
The value of i is: 4
The value of i is: 3
The value of i is: 2
```

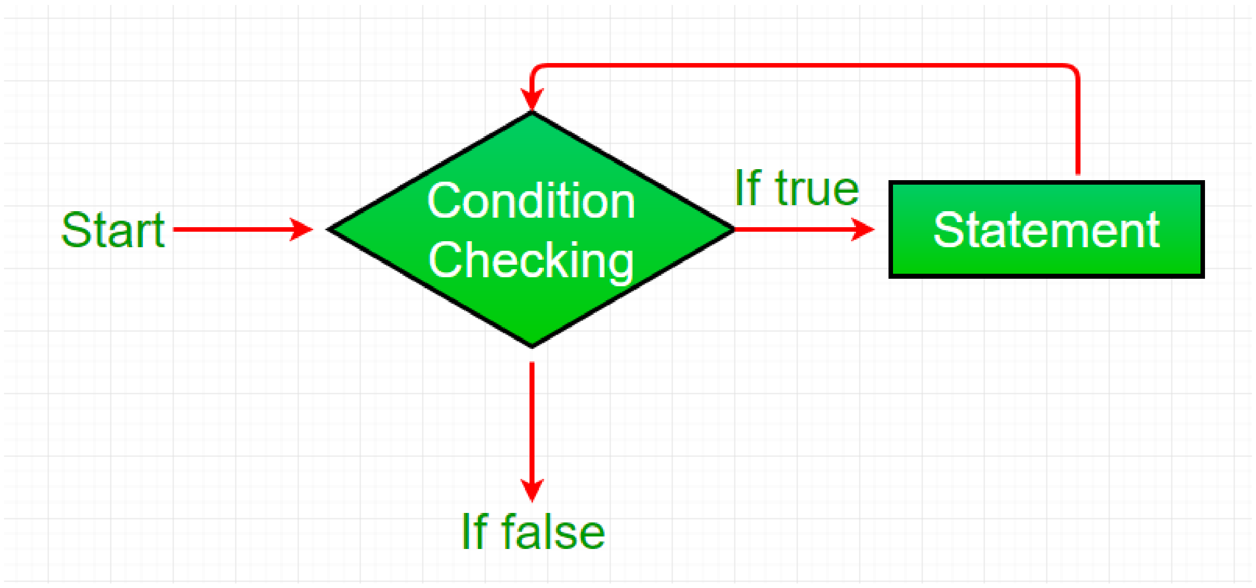
While Loop

A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

Syntax:

```
while (boolean condition)
{
  loop statements...
}
```

Flowchart:



- While loop starts with the checking of condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason it is also called **Entry control loop**

- Once the condition is evaluated to true, the statements in the loop body are executed. Normally the statements contain an update value for the variable being processed for the next iteration.

- When the condition becomes false, the loop terminates which marks the end of its life cycle.

```
// Java program to illustrate while loop
class whileLoopDemo
{
public static void main(String args[])
{
int x = 1;

// Exit when x becomes greater than 4
while (x <= 4)
{
System.out.println("Value of x:" + x);

// Increment the value of x for
// next iteration
x++;
}
}
}
```

Output:

```
Value of x: 1
Value of x: 2
Value of x: 3
Value of x: 4
```

Example:

```
int i = 0;
while (i < 5) {
  System.out.println(i);
  i++;
}
```

Note: Do not forget to increase the variable used in the condition, otherwise the loop will never end!

Example:

```
class WhileLoopExample {
   public static void main(String args[]){
       int i=10;
       while(i>1){
           System.out.println(i);
           i--;
       }
   }
}
```

Output:

```
10
9
8
7
6
5
4
3
2
```

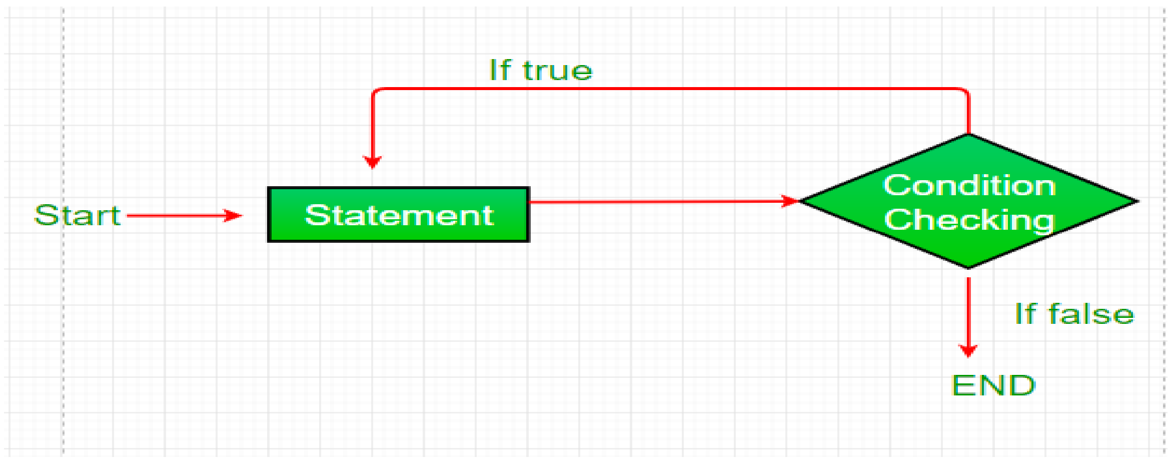
do while

do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of Exit Control Loop.

Syntax:

```
do
{
   statements..
}
while (condition);
```

Flowchart:



1. do while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
2. After the execution of the statements, and update of the variable value, the condition is checked for true or false value. If it is evaluated to true, next iteration of loop starts.
3. When the condition becomes false, the loop terminates which marks the end of its life cycle.
4. It is important to note that the do-while loop will execute its statements atleast once before any condition is checked, and therefore is an example of exit control loop.

Example:

```
// Java program to illustrate do-while loop
class dowhileloopDemo
{
   public static void main(String args[])
   {
      int x = 21;
      do
      {
         // The line will be printed even
         // if the condition is false
         System.out.println("Value of x:" + x);
         x++;
      }
      while (x < 20);
   }
}
```

Output:

```
Value of x: 21
```

Example:

```
int i = 0;
do {
  System.out.println(i);
  i++;
}
while (i < 5);
```

Do not forget to increase the variable used in the condition, otherwise the loop will never end!

```
class DoWhileLoopExample {
    public static void main(String args[]){
        int i=10;
        do{
            System.out.println(i);
            i--;
        }while(i>1);
    }
}
```

Output:

```
10
9
8
7
6
5
4
3
2
```

Count Controlled Loop Construct

Count-controlled repetition requires

- control variable (or loop counter)
- initial value of the control variable
- increment (or decrement) by which the control variable is modified each iteration through the loop
- condition that tests for the final value of the control variable

A count-controlled repetition will exit after running a certain number of times. The count is kept in a variable called an index or counter. When the index reaches a certain value (the loop bound) the loop will end.

Count-controlled repetition is often called definite repetition because the number of repetitions is known before the loop begins executing. When we do not know in advance the number of times we want to execute a statement, we cannot use count-controlled repetition. In such an instance, we would use sentinel-controlled repetition.

Display the character and its' ASCII value for all lower case characters.

```
for (char x = 'a'; x <= 'z'; x++)
   System.out.println(x + " = " + (int) x);
```

Activities/Assessment

I. Application. Read the questions carefully and confine your responses to an analysis of the questions as written.

1. Write a program in java to display the multiplication table of an given integer using for loop.

    Sample output:

    ```
    Enter number of terms: 2
    2 x 1 = 2
    2 x 2 = 4
    ```

2. Using While loop, create a program to display the pattern like right angle triangle with a number.

    Sample ouput:

    ```
    Enter number of rows: 3
    1
    1 2
    1 2 3
    ```

3. Using Do while, create a program that display the alphabet from a – Capital Z.

Array

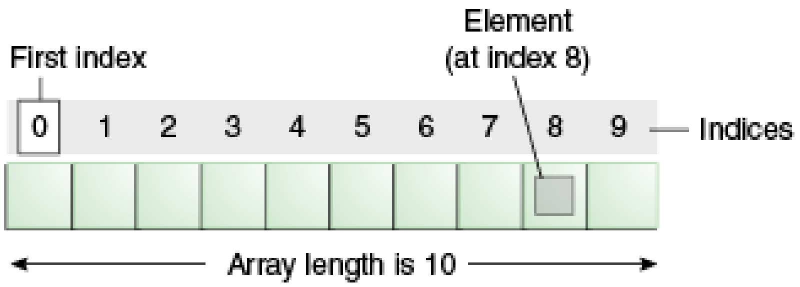
Learning Outcomes:

After successful completion of this lesson, you should be able to:

- Use arrays to implement sort and search algorithms.

Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

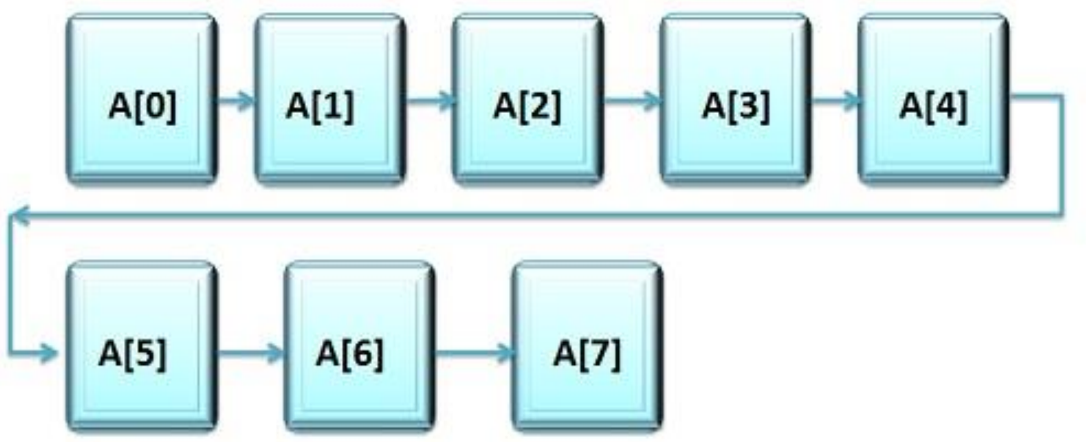
Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.



Types of Array in Java

- **Single Dimensional Array / One dimensional Array**

Considered as the "list of variables of similar data types", and each variable can be distinctly accessed by specifying its index in square brackets preceded by the name of that array.



One-Dimensional Array A[8]

Examples:

One dimensional array declaration of variable:

```
import java.io.*;

class GFG {
          public static void main(String[] args)
          {
                    int[] a; // valid declaration
                    int b[]; // valid declaration
                    int[] c; // valid declaration
          }   }
```

We can write it down in any way.

Now if you want to declare your array as below:

```
import java.io.*;

class GFG {
          public static void main(String[] args)
          {
                    // invalid declaration -- If we want to assign
                    // size of array at the declaration time, it
                    // gives compile time error.
                    int a[5];

                    // valid declaration
                    int b[];
          }   }
```

Now, suppose we want to write multiple variable array declaration, then we can use it like this.

```
import java.io.*;

class GFG {
          public static void main(String[] args)
          {
                    // valid declaration, both arrays are
                    // one dimensional array.
                    int a[], b[];

                    // invalid declaration
                    int c[], [] d;

                    // invalid declaration
                    int[] e, [] f;
          } }
```

When we declare multiple variables at the same time, we must first write variables and then declare the variable except the first variable declaration. There is no restriction on the first variable.

Now, when we're creating an array, it's mandatory to pass the size of the array; otherwise, we'll get a compile time error.

You can use the new operator to create an array.

```
import java.io.*;

class GFG {
        public static void main(String[] args)
        {
                // invalid, here size of array is not given
                int[] a = new int[];

                // valid, here creating 'b' array of size 5
                int[] b = new int[5];

                // valid
                int[] c = new int[0];

                // gives runtime error
                int[] d = new int[-1];
        }   }
```

Printing Array:

```
/* A complete Java program to demonstrate working
of one dimensional arrays */
class oneDimensionalArray {

        public static void main(String args[])
        {
                int[] a; // one dimensional array declaration
                a = new int[3]; // creating array of size 3
                for (int i = 0; i < 3; i++) {
                        a[i] = 100;
                        System.out.println(a[i]);
                }
        }
}
```

Output:

```
100
100
100
```

- **Two Dimensional Array**

The two-dimensional array in the Java programming language is nothing but an array of arrays. In the Java Two Dimensional Array, data stored in rows and columns can be accessed using both the row index and column index (like an Excel file). If the data is linear, the One Dimensional Array can be used. However, in order to work with multi-level data, we need to use the Multi-Dimensional Array.

```
// Java program to demonstrate different ways
// to create two dimensional array.
import java.io.*;

class GFG {
        public static void main(String[] args)
        {
                int a[][]; // valid
                int[][] b; // valid
                int[][] c; // valid
                int[] d[]; // valid
                int[][] e; // valid
                int[] f[]; // valid
                [][] int g; // invalid
                [] int[] h; // invalid
        }  }
```

Now, suppose we want to write multiple declarations of array variable then you can use it like this:

```
// Java program to demonstrate multiple declarations
// of array variable
import java.io.*;

class GFG {
public static void main(String[] args)
        {
                // Here, 'a' is two dimensional array, 'b'
                // is two dimensional array
                int[] a[], b[];
                // Here, 'c' is two dimensional array, 'd'
                // is two dimensional array
                int[] c[], d[];

                // Here, 'e' is two dimensional array, 'f'
                // is three dimensional array
                int[][] e, f[];

                // Here, 'g' is two dimensional array,
                // 'h' is one dimensional array
                int[] g[], h;
        }}
```

Creating a single dimensional array and two dimensional array without a new operator:

```
/* Java program for one and two dimensional arrays.
without new operator*/
class oneTwoDimensionalArray {

        public static void main(String args[])
        {
                int[] a[] = { { 1, 1, 1 }, { 2, 2, 2 },
                                        { 3, 3, 3 } }, b = { 20 };
                // print 1D array
                System.out.println(b[0]);

                // print 2D array
                for (int i = 0; i < 3; i++) {
                        for (int j = 0; j < 3; j++) {
                                a[i][j] = 100;
                                System.out.println(a[i][j]);
                        }
                }
        }
}
```

Output:

```
20
100
100
100
100
100
100
100
100
100
```

Creating one dimensional array and two dimensional array using new operator:

```
/* Java program for one and two dimensional arrays.
using new operator*/
class oneTwoDimensionalArray {

              public static void main(String args[])
              {
                 int[] a[], b = { 20 };
                 a = new int[3][3];
                 b = new int[3];

                 // print 1D array
                 for (int i = 0; i < 3; i++)
```

```
                    System.out.println(b[i]);

                // print 2D array
                for (int i = 0; i < 3; i++) {
                    for (int j = 0; j < 3; j++) {
                        a[i][j] = 100;
                        System.out.println(a[i][j]);
                    }
                }
            }
}
```

Output:

```
0
0
0
100
100
100
100
100
100
100
100
100
```

How to print array in Java

The Java array is a data structure where the elements of the same data type can be stored. The array elements are stored in a contiguous memory location. So, in an array, we can store a fixed set of elements.

There are following ways to print an array in Java:

- Java for loop
- Java for-each loop
- Java Arrays.toString() method
- Java Arrays.deepToString() method
- Java Arrays.asList() method
- Java Iterator Interface
- Java Stream API

Java for loop is used to execute a set of statements repeatedly until a particular condition is satisfied.

Syntax:

```
for(initialization; condition; increment/ decrement)
{
//statements
}
```

Example for loop:

In the following example, we have created an array of lengths of four and initialized elements. We used the loop to get the values from the array. It's the most popular way to print arrays in Java.

```
public class PrintArrayExample1
{
public static void main(String args[])
{
//declaration and instantiation of an array
int arr[]=new int[4];
//initializing elements
arr[0]=10;
arr[1]=20;
arr[2]=70;
arr[3]=40;
//traversing over array using for loop
for(int i=0;i<arr.length;i++)    //length is the property of the array
System.out.println(arr[i]);
}
}
```

Output:

```
10
20
70
40
```

Java for-each loop is also used to pass through an array or collection. It works on the basis of its elements. Returns the elements one by one in a defined variable.

Syntax:

```
for(Type var:array)
```

In the following example, we have generated a string type array with a length of four and initialized elements. We used-each loop to run through the array.

```
public class PrintArrayExample2
{
public static void main(String args[])
{
// declaration and instantiation of an array
String[] city = new String[4];
//initializing elements
city[0] = "Delhi";
city[1] = "Jaipur";
city[2] = "Gujarat";
city[3] = "Mumbai";
//traversing over array using for-each loop
for (String str : city)
{
System.out.println(str);
}
}
}
```

Output:

```
Delhi
Jaipur
Gujarat
Mumbai
```

Java Searching

The simplest type of search is the sequential search. In the sequential search, each element of the array is compared to the key, in the order it appears in the array, until the desired element is found. If you are looking for an element that is near the front of the array, the sequential search will find it quickly. The more data that must be searched, the longer it will take to find the data that matches the key.

public static int search(int [ ] numbers, int key)
{
    for (int index = 0; index < numbers.length; index++)
    {
        if (numbers[index] = = key)
            return index; //We found it!!!
    }
    // If we get to the end of the loop, a value has not yet
    // been returned. We did not find the key in this array.
    return -1;
}
	If the **key** value is found, the index (subscript) of the location is returned. This tells us that the return value x, will be the first integer found such that
	numbers [x] = key.
	
	There may be
	additional **key** locations in this array beyond this location.

Example Program:

```
// Java code to demonstrate search() method
import java.util.*;

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

       // Creating an empty Stack
       Stack<String> STACK = new Stack<String>();
```

```
        // Stacking strings
        STACK.push("Geeks");
        STACK.push("4");
        STACK.push("Geeks");
        STACK.push("Welcomes");
        STACK.push("You");

        // Displaying the Stack
        System.out.println("The stack is: " + STACK);

        // Checking for the element "4"
        System.out.println("Does the stack contains '4'? "
                           + STACK.search("4"));
        // Checking for the element "Hello"
        System.out.println("Does the stack contains 'Hello'? "
                           + STACK.search("Hello"));

        // Checking for the element "Geeks"
        System.out.println("Does the stack contains 'Geeks'? "
                           + STACK.search("Geeks"));
    }
}
```

Output:

```
The stack is: [8, 5, 9, 2, 4]
Does the stack contains '9'? 3
Does the stack contains '10'? -1
Does the stack contains '11'? -1
```

Java Sorting

Sorting is ordering a list of objects. We can distinguish two types of sorting. If the number of objects is small enough to fits into the main memory, sorting is called internal sorting. If the number of objects is so large that some of them reside on external storage during the sort, it is called external sorting.

Bucket sort

Suppose we need to sort an array of positive integers {3,11,2,9,1,5}. A bucket sort works as follows: create an array of size 11. Then, go through the input array and place integer 3 into a second array at index 3, integer 11 at index 11 and so on. We will end up with a sorted list in the second array.

Suppose we are sorting a large number of local phone numbers, for example, all residential phone numbers in the 412 area code region (about 1 million) We sort the numbers without use of comparisons in the following way. Create an a bit array of size 107. It takes about 1Mb. Set all bits to 0. For each phone number turn-on the bit indexed by that phone number. Finally, walk through the array and for each bit 1 record its index, which is a phone number.

We immediately see two drawbacks to this sorting algorithm. Firstly, we must know how to handle duplicates. Secondly, we must know the maximum value in the unsorted array..

Thirdly, we must have enough memory - it may be impossible to declare an array large enough on some systems.

The first problem is solved by using linked lists, attached to each array index. All duplicates for that bucket will be stored in the list. Another possible solution is to have a counter. As an example let us sort 3, 2, 4, 2, 3, 5. We start with an array of 5 counters set to zero.

0	1	2	3	4	5
0	0	0	0	0	0

Moving through the array we increment counters:

0	1	2	3	4	5
0	0	2	2	1	1

Next, we simply read off the number of each occurrence: 2 2 3 3 4 5.

Bubble sort

The algorithm works by comparing each item in the list with the item next to it, and swapping them if required. In other words, the largest element has bubbled to the top of the array. The algorithm repeats this process until it makes a pass all the way through the list without swapping any items.

```
void bubbleSort(int ar[])
{
  for (int i = (ar.length - 1); i >= 0; i--)
  {
    for (int j = 1; j ≤ i; j++)
    {
      if (ar[j-1] > ar[j])
       {
           int temp = ar[j-1];
           ar[j-1] = ar[j];
           ar[j] = temp;
  }  }
 } }
```

Example.

Here is one step of the algorithm. The largest element - 7 - is bubbled to the top:

7, **5**, 2, 4, 3, 9

5, **7,** **2**, 4, 3, 9

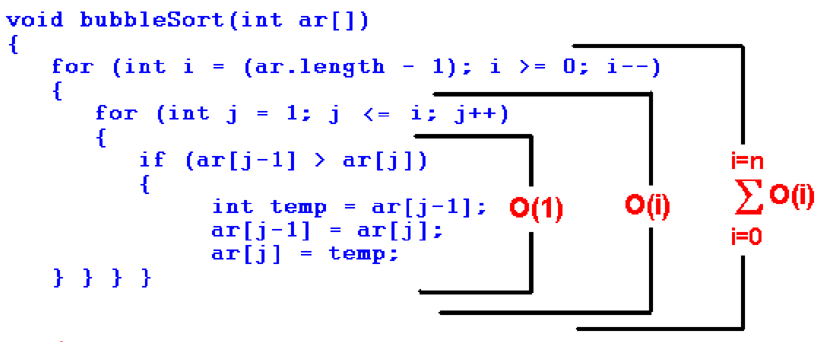
5, 2, **7,** **4**, 3, 9

5, 2, 4, **7,** **3**, 9

5, 2, 4, 3, **7,** **9**

5, 2, 4, 3, 7, 9

The worst-case runtime complexity is O(n2). See explanation below:



```
void bubbleSort(int ar[])
{
   for (int i = (ar.length - 1); i >= 0; i--)
   {
      for (int j = 1; j <= i; j++)
      {
         if (ar[j-1] > ar[j])
         {
              int temp = ar[j-1];
              ar[j-1] = ar[j];
              ar[j] = temp;
   } } } }
```

$$\sum_{i=0}^{i=n} O(i) = 1 + 2 + 3 + \ldots + (n-1) = O(n^2)$$

Selection Sort

The algorithm works by selecting the smallest unsorted item and then swapping it with the item in the next position to be filled.

The selection sort works as follows: you look through the entire array for the smallest element, once you find it you swap it (the smallest element) with the first element of the array. Then you look for the smallest element in the remaining array (an array without the first element) and swap it with the second element. Then you look for the smallest element in the remaining array (an array without first and second elements) and swap it with the third element, and so on.

Here is an example,

```
void selectionSort(int[] ar){
  for (int i = 0; i < ar.length-1; i++)
  {
    int min = i;
    for (int j = i+1; j < ar.length; j++)
        if (ar[j] < ar[min]) min = j;
    int temp = ar[i];
```

```
      ar[i] = ar[min];
      ar[min] = temp;
} }
```

Example.

29,	64,	73,	34, **20**,
20, **64**,	73,		34, **29**,
20,	29, **73**, **34**,		64
20,	29,		34, **73**, **64**
20, 29, 34, 64, 73			

Insertion sort

To sort unordered list of elements, we remove its entries one at a time and then insert each of them into a sorted part (initially empty):

```
void insertionSort(int[] ar)
{
  for (int i=1; i ‹ ar.length; i++)
  {
     int index = ar[i]; int j = i;
     while (j > 0 && ar[j-1] > index)
     {
          ar[j] = ar[j-1];
          j--;
     }
     ar[j] = index;
} }
```

Example. We color a sorted part in green, and an unsorted part in black. Here is an insertion sort step by step. We take an element from unsorted part and compare it with elements in sorted part, moving from right to left.

29,	20,	73,	34,	64
29,	20,	73,	34,	64
20,	**29**,	73,	34,	64
20,	**29,**	**73**,	34,	64
20,	**29,**	**34,**	**73**,	64
20, 29, 34, 64, 73				

Mergesort

Merge-sort is based on the divide-and-conquer paradigm. It involves the following three steps:

1. Divide the array into two (or more) subarrays
2. Sort each subarray (Conquer)
3. Merge them into one (in a smart way!)

Example. Consider the following array of numbers

27 10 12 25 34 16 15 31

divide it into two parts

27 10 12 25 **34 16 15 31**

divide each part into two parts

27 10 **12 25** **34 16** **15 31**

divide each part into two parts

27 **10** **12** **25** **34** **16** **15** **31**

merge (cleverly-!) parts

10 27 **12 25** **16 34** **15 31**

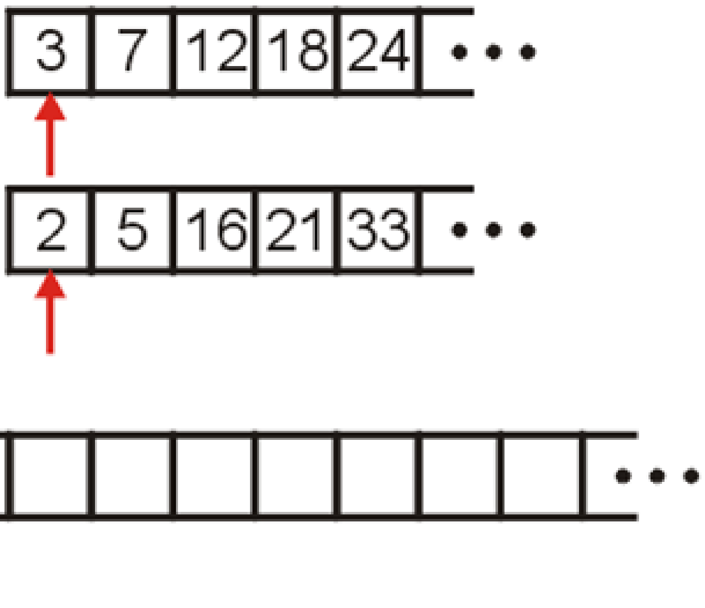
merge parts

10 12 25 27 **15 16 31 34**

merge parts into one

10 12 15 16 25 27 31 34

How do we merge two sorted subarrays? We define three references at the front of each array



Activities/Assessment

I. Application. Read the questions carefully and confine your responses to an analysis of the questions as written.

1. Write a Java program to sort a numeric array and a string array.
2. Write a Java program to print the following grid.

   Expected Output:

   ```
   - - - - - - - - - -

   - - - - - - - - - -

   - - - - - - - - - -

   - - - - - - - - - -

   - - - - - - - - - -
   ```

3. Take 20 integer inputs from user and print the following:

   number of positive numbers

   number of negative numbers

   number of odd numbers

   number of even numbers

   number of 0s.

Functions

Learning Outcomes:

After successful completion of this lesson, you should be able to:

- Identify the different string and character functions used in Java;
- Discuss the different mathematical function;
- Explain purpose of using global and local variables

In Java, all function definitions must be inside classes. We also call functions methods. Let's look at an example method

```
public class Main {
   public static void foo() {
      // Do something here
   }
}
```

foo is a method we defined in class Main. Notice a few things about foo.

- static means this method belongs to the class Main and not to a specific instance of Main. Which means we can call the method from a different class like that Main.foo().
- void means this method doesn't return a value. Methods can return a single value in Java and it has to be defined in the method declaration. However, you can use return by itself to exit the method.
- This method doesn't get any arguments, but of course Java methods can get arguments as we'll see further on.

Arguments

Arguments to Java methods are passed by value, although some might disagree with my choice of words, I find it the best way to explain and understand how it works exactly.

By value means that arguments are copied when the method runs. Let's look at an example.

```
public void bar(int num1, int num2) {
            ...
}
```

Here is a another place in the code, where bar is called

```
int a = 3;
int b = 5;
bar(a, b);
```

You can picture in your head that when bar(a, b) is run, it's like in the beginning of bar the following two lines are written:

```
int num1 = a;
int num2 = b;
```

And only then the rest of the method is run.

This means that a value get copied to num1 and b value get copied to num2. Changing the values of num1 and num2 will not affect a and b.

If the arguments were objects, the rules remain the same, but it acts a bit differently. Here is an example:

```
public void bar2(Student s1, Student s2) {
    ...
}
```

And here is how we use it

```
Student joe = new Student("joe");
Student jack = new Student("jack");
bar2(joe, jack);
```

Again we can picture the same two lines in the beginning of bar2:

```
Student s1 = joe;
Student s2 = jack;
```

But when we assign objects, it's a bit different than assigning primitives. s1 and joe are two different references to the same object. s1 == joe is true. This means that running methods on s1 will change the object joe. But if we'll change the value of s1 as a reference, it will not affect the reference joe.

```
s1.setName("Chuck"); // joe name is now
Chuck as well
s1 = new Student("Norris"); // s1 is a new
student, different than joe with the name of
Norris
// s1 == joe   is not true anymore
```

Non static methods

Non static methods in Java are used more than static methods. Those methods can only be run on objects and not on the whole class.

Non static methods can access and alter the field of the object.

```
public class Student {
    private String name;
    public String getName() {
        return name;   }
    public void setName(String name) {
        this.name = name;
    } }
```

Calling the methods will require an object of type Student.

```
Student s = new Student();
s.setName("Danielle");
String name = s.getName();

Student.setName("Bob"); // Will not work!
Student.getName(); // Will not work!
```

Summary

- Every Java method has to be within a class
- Static methods belong to a class while non-static methods belong to objects
- All parameters to functions are passed by value, primitives content is copied, while objects are not copied and some would say 'passed by reference'

Java Predefined Functional Interface

- The java.util.function package defines several predefined functional interfaces that you can use when creating lambda expressions or method references.
- They are widely used throughout the Java API

Functional Interface	Abstract Method	Function descriptor	Description
Consumer<T>	accept(T t)	T -> void	Represents an operation that accepts a single input argument and returns no result.
Function<T, R>	apply(T t)	T -> R	Represents a function that accepts one argument and produces a result.
Predicate<T>	test(T t)	T -> boolean	Represents a predicate (boolean-valued function) of one argument.
Supplier<T>	get()	() -> T	Represents a supplier of results.

Example using Consumer<T>

```
public class ConsumerApp {
  public static void main(String[] args) {
    String[] players = {"Rafael Nadal", "Novak Djokovic",
      "Stanislas Wawrinka", "David Ferrer",
      "Roger Federer", "Andy Murray", "Tomas Berdych",
      "Juan Martin Del Potro", "Richard Gasquet", "John Isner"};
    // Show the list of players
    System.out.print("Show the list of players:\n");
    // void forEach(Consumer<? super T> action)
    Arrays.asList(players).forEach((player) -> System.out.println(player)); } }
```

Output:

```
Show the list of players:
Rafael Nadal
Novak Djokovic
Stanislas Wawrinka
David Ferrer
Roger Federer
Andy Murray
Tomas Berdych
Juan Martin Del Potro
Richard Gasquet
John Isner
```

Example using Function<T , R>

```
public class FunctionApp {
  public static void main(String[] args) {
    String[] players = {"Rafael Nadal", "Novak Djokovic",
      "Stanislas Wawrinka", "David Ferrer",
      "Roger Federer", "Andy Murray",
      "Tomas Berdych", "Juan Martin Del Potro",
      "Richard Gasquet", "John Isner"};
    Function<String[],String> converter = (all)-> {
      String names= "";
      for (String n : all){
        String forname=n.substring(0, n.indexOf(" "));
        forname=n.substring(n.indexOf(" "))+" "+forname;
        names+=forname+"\n";
      }
      return names;
    };
    System.out.println(converter.apply(players));
  }
}
```

Output:

```
Nadal Rafael
 Djokovic Novak
 Wawrinka Stanislas
 Ferrer David
 Federer Roger
 Murray Andy
 Berdych Tomas
 Martin Del Potro Juan
 Gasquet Richard
 Isner John
```

Example using Predicate<T>

```
public class PredicateApp {
  private static List getBeginWith(List<String> list, Predicate<String> valid) {
    List<String> selected = new ArrayList<>();
    list.forEach(player -> {
      if (valid.test(player)) {
        selected.add(player);
      }
    });
    return selected;
  }

  public static void main(String[] args) {
    String[] players = {"Rafael Nadal", "Novak Djokovic",
      "Stanislas Wawrinka", "David Ferrer",
      "Roger Federer", "Andy Murray", "Tomas Berdych",
      "Juan Martin Del Potro", "Richard Gasquet", "John Isner"};
    List playerList = Arrays.asList(players);
    System.out.println(getBeginWith(playerList,(s)->s.startsWith("R")));
    System.out.println(getBeginWith(playerList,(s)->s.contains("D")));
    System.out.println(getBeginWith(playerList,(s)->s.endsWith("er")));
  }
}
```

Output:

```
[Rafael Nadal, Roger Federer, Richard Gasquet]
[Novak Djokovic, David Ferrer, Juan Martin Del Potro]
[David Ferrer, Roger Federer, John Isner]
```

Example using Supplier<T>

```
public class SupplierApp {
  private static void printNames(Supplier<String> arg) {
    System.out.println(arg.get());
  }
  private static void listBeginWith(List<String> list, Predicate<String> valid) {
    printNames(()->"\nList of players:");
    list.forEach(player -> {
      if (valid.test(player)) {
        printNames(()->player);
      }
    });
  }
  public static void main(String[] args) {
    String[] players = {"Rafael Nadal", "Novak Djokovic",
      "Stanislas Wawrinka", "David Ferrer",
      "Roger Federer", "Andy Murray", "Tomas Berdych",
      "Juan Martin Del Potro", "Richard Gasquet", "John Isner"};
    List playerList = Arrays.asList(players);
    // print which starts with 'R'
    listBeginWith(playerList, (s) -> s.startsWith("R"));
    listBeginWith(playerList, (s) -> s.contains("D"));
    listBeginWith(playerList, (s) -> s.endsWith("er"));
  }
}
```

Output:

```
List of players:
Rafael Nadal
Roger Federer
Richard Gasquet

List of players:
Novak Djokovic
David Ferrer
Juan Martin Del Potro

List of players:
David Ferrer
Roger Federer
John Isner
```

String Functions

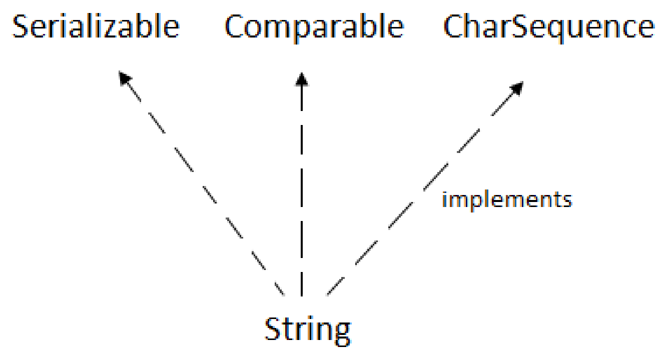
In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. For example:

```
char[] ch={'j','a','v','a','t','p','o','i','n','t'};
String s=new String(ch);
```

Is same as:

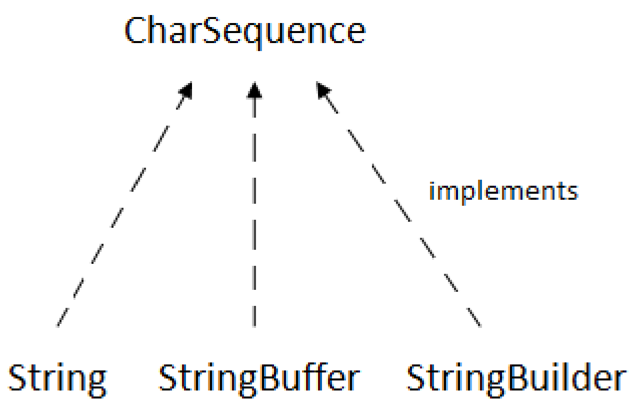
```
String s="javatpoint";
```

Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() et



CharSequence Interface

The CharSequence interface is used to represent the sequence of characters. String, StringBuffer and StringBuilder classes implement it. It means, we can create strings in java by using these three classes.



The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use StringBuffer and StringBuilder classes.

What is string in Java?

Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

There are two ways to create String object:

- By string literal
- By new keyword

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

```
String s="welcome";
```

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

```
String s1="Welcome";
String s2="Welcome";//It doesn't create a new instance
```

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

```
String s=new String("Welcome");//creates two objects and one reference variable
```

Java string example:

```
public class StringExample{
public static void main(String args[]){
String s1="java";//creating string by java string literal
char ch[]={'s','t','r','i','n','g','s'};
String s2=new String(ch);//converting char array to string
String s3=new String("example");//creating java string by new keyword
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```

Activities/Assessment

I. **Application.** Read the questions carefully and confine your responses to an analysis of the questions as written.

1. Write a Java method to find the smallest number among three numbers.

Test Data:

Input First Number:37

Input Second Number: 25

Input Third Number:29

Expected Output:

The smallest value is 25

2. Write a Java method to compute the average of three numbers.

3. Write a Java method to count all vowels in a string.

4. Write a Java method to display the middle character of a string.

   Note:

   a) If the length of the string is even there will be two middle characters.

   b) If the length of the string is odd there will be one middle character.

5. Write a Java method to count all words in a string.

Sample Programs/Activities

Do the following programs and write down the expected out:

Sample 1:

```
public class First {

public static void main(String[] args) {
System.out.print("Hello World\n");
System.out.print("Hello World2\n") ;
System.err.print("warning");
}
}
```

Sample 2:

```
public class Second {

  public static void main(String[] args) {
    System.out.println("Name \t\t\t"  + "Address\t\t\t" +
"Number");
    System.out.println();
    System.out.print("Migs\t\t\t" + "Pampanga\t\t\t" +
"09203817\n");
     System.out.print("Angel\t\t\t" + "Laguna\t\t\t" +
"09167638\n");
      System.out.print("Mike\t\t\t" + "Tarlac\t\t\t" +
"09264301\n");
  }
}
```

Sample 3:

```
import java.util.*;
public class activity {

  public static void main(String[] args) {
     int age, height;
     double weight;
     long contact;
     Scanner sc = new Scanner(System.in);
     System.out.print("Enter your age : ");

     age = sc.nextInt();
     System.out.print("Enter your height (cm) : ");

     height = sc.nextInt();
     System.out.print("Enter your weight (lb) : ");

     weight = sc.nextDouble();
     System.out.print("Enter your contact number : ");
     contact = sc.nextLong();

     System.out.println();

     System.out.println("My age is " + age + ".");
     System.out.println("My height is " + height + "cm.");
     System.out.println("My weight is " + weight + "lb.");
    System.out.println("My contact number is " + “0” + contact);
```

Sample 4:

```
package experiment;
 import java.util.*;
public class NewClass {
  public static void main(String[] args) {

      Scanner sc = new Scanner(System.in);

    int sq, cb, num, age;
    double  ave, php, fh,num1, num2, num3;
    char ch;
    String name, address;

    System.out.println("----------- Welcome to Converstion app -----------\n");

    System.out.println("S - Square");
    System.out.println("C - Cube");
    System.out.println("A - Average");
    System.out.println("D - Money Convertion Dollar - Php");
    System.out.println("T - Temperature");
    System.out.println("R - Area of Rectangle");
    System.out.println("H - Convertion in time (Hour/Minute/Seconds)");
    System.out.println("I - Personal Information");
    System.out.println("L - Convertion of meter to kilometer");
    System.out.println("M - Getting remainder\n");

  System.out.print("Enter your choice : ");
  ch = sc.next().charAt(0);

                                    //continue to next page
```

```
switch (ch) {

    case 'S':case 's':

    System.out.println("You choose Square of a number\n");
    System.out.print("Enter a whole number : ");
    num = sc.nextInt();

    sq = num * num ;
    System.out.println("The square of number " + num + " is " + sq);
    break;

    case 'C' : case 'c':
    System.out.println("You choose Cube of a number\n");
    System.out.print("Enter a whole number : ");
    num = sc.nextInt();

    cb = num * num * num ;
    System.out.println("The Cube of number " + num + " is " + cb);
    break;

      case 'A': case 'a':
        System.out.println("You choose Computing Average Grade\n");
    System.out.print("Enter your Preliminarie Grade : ");
    num1 = sc.nextDouble();

    System.out.print("Enter your Midterm Grade : ");
    num2 = sc.nextDouble();

    System.out.print("Enter your Finals Grade : ");
    num3 = sc.nextDouble();

    ave = (num1 + num2 + num3) /3;

    System.out.println("Your Average grade is : " + ave);
    break;
                              //continue to the next page
```

```
case 'D': case 'd':
System.out.println("You choose Money Convertion (Dollars to Php)");
System.out.print("Enter the money value : ");
num1 = sc.nextDouble();
php = num1 * 51.48;

System.out.println("The amount value of " + num1 + " is " + php + " pesos");
break;

case 'T': case 't':

    System.out.println("You choose Convertion to Temprerature (Celsius to
Farenheit)\n" );

    System.out.print("Enter temperature (Celcius): ");
    num1 = sc.nextDouble();

    fh = num1 * (9/5) + 32;
    System.out.println("The Convertion of Celsius to Fareinheit is " + fh);
    break;

    case 'R':  case 'r':
    System.out.println("You choose computing Area of Rectangle\n");
    System.out.print("Enter length : ");
    num1 = sc.nextDouble();

    System.out.print("Enter width : ");
    num2 = sc.nextDouble();

    num3 = num1 * num2;
    System.out.println("The area of rectangle is " + num3);
    break;
                              //continue to the next page
```

```
case 'H':case 'h':

    System.out.println("You choose convertion of Hours and Minutes to seconds\n");
    System.out.print("Enter hour : ");
    num1 = sc.nextDouble();

    num2= num1/60;
    num3 = num2*60;

    System.out.println("The Convertion of hours to minutes is equal to " + num2 + "
mins.");

    System.out.println("The Convertion of hours to seconds is equal to " + num3 + "
seconds.");
    break;
    case 'I' : case 'i':
    System.out.println("You choose Personal Information\n");

    System.out.print("Enter your name : ");
    name = sc.next();
    System.out.print("Enter your age : ");
    age = sc.nextInt();
    System.out.print("Enter your name : ");
    address = sc.next();
    System.out.println("My name is : " + name);
    System.out.println("My age is : " + age);
    System.out.println("My address is : " + address);

    break;
                                    //continue to the next page
```

```
case 'L': case 'l':
    System.out.println("You choose convertion of meters to Kilometers\n");

    System.out.print("Enter value of meters : ");
    num1 = sc.nextDouble();
    num2 =num1 / 1000;

    System.out.println("The converstion of " + num1 + "meters is " + num2 + " kilometers.");
    break;

    case 'M': case 'm' :
    System.out.println("You choose Finding the Remainder\n");

    System.out.print("Enter First number : ");
    num1 = sc.nextDouble();

    System.out.print("Enter Second number : ");
    num2 = sc.nextDouble();

    num3 = num1 % num2;
    System.out.println("The remainder is " + num3);
    break;

    default :
    System.out.println("Invalid code");
    break;

   }
  }
}
```

Sample 5:

```
 import java.util.*;

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

      Scanner in = new Scanner(System.in);

int start, end;

System.out.print("Enter a whole number start : ");
start = in.nextInt();
System.out.print("Enter a whole number end : ");
end = in.nextInt();

for (int i = start; i<=end;i++){
  System.out.println(i);
}
  //displaying letter using  (for loop)
  char let;
  System.out.print("Enter a letter : ");
  let = in.next().charAt(0);
  for (char letter = let; letter <= 'z';letter++)
  System.out.println(letter);
  }
    }
```

Sample 6:

```
import java.util.*;
public class Act3 {
  public static void main(String[] args) {
   Scanner scan = new Scanner (System.in);
   int  i, j;
   double input [] = new double [6];

   try {

   for (i= 0;i<6;i++) {
     System.out.print("Enter an integer : ");
     input [i] = scan.nextDouble();
   }

   System.out.print("You've entered displaying as ascending are ") ;

   for (j = 0;j < 6; j++) {
    Arrays.sort(input);
     System.out.print(input [j] + " ");
   }

   } catch (Exception e) {
     System.out.println("Invalid Input");
   }
  }
}
```

Sample 7:

```
 import java.util.*;
public class NewClass1 {
  public static void main(String[] args) {

      Scanner inn = new Scanner(System.in);
   int in;
   double array [] = new double [5];
   Scanner scan = new Scanner (System.in);
   try {
   for (in = 0;in < 5;in++) {
   System.out.print("Enter a integer : ");
   array [in] = scan.nextInt();
  }

   for (in = 0;in < 5;in++) {
     System.out.println("Integers that you've enter are : "  + array [in]);
  }
 } catch (Exception e) {
   System.out.println("Invalid input");
   }
 }
}
```

Sample 8:

```
 import java.util.*;
public class NewClass2{
  public static void main(String[] args) {
int in , row, col;
  char letter [] =  {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'};
  Scanner input = new Scanner (System.in);
    try {
     System.out.print("Choose a number from (1-10) : ");
    in = input.nextInt();
       if (in < 0) {
     System.out.println("You've enter a number less than a 1");
    }

    else if  (in > 10){
     System.out.println("You've enter a number is greater than a 10");
    }
else {
            for (row = 1; row <= in;row++) {

        for (col = 1; col <in; col++) {
         System.out.print(" ");
        }
for (col = in - row; col > 0; col--) {
         System.out.print(" ");
        }
        for (col = 1;col<2*row;col++) {
         System.out.print(letter [in - row]);
        }

        System.out.println();
      }
    }

  } catch (Exception e) {
    System.out.println("Invalid Input");
  }
  }
}
```

References

- Two Dimensional array in Java. (2020). Retrieved from https://www.tutorialgateway.org/two-dimensional-array-in-java/
- Array Declarations in Java. (2020). Retrieved from https://www.geeksforgeeks.org/array-declarations-java-single-multidimensional/
- Two Dimensional array in Java. (2020). Retrieved from https://www.journaldev.com/747/two-dimensional-array-java
- How to print array in java?. (2020). Retrieved from https://www.javatpoint.com/how-to-print-array-in-java
- Introductions to array. (2020). Retrieved from https://www.geeksforgeeks.org/introduction-to-arrays/
- Introductions to array. (2020). Retrieved from http://www.w3processing.com/index.php?subMenuLoad=java/oop/PredefinedFunctional.php
- Java Methods. (2020). Retrieved from https://www.w3schools.com/java/java_methods.asp
- Java String. (2020). Retrieved from https://www.javatpoint.com/java-string/
- User Defined Functions in Java. (2020). Retrieved from http://mrbool.com/user-defined-functions-in-java/
- The World's Learning Company. (2020). Retrieved from https://www.pearson.com/us/higher-education/program/Sebesta-Concepts-of-Programming-Languages-11th-
- The importance of Programming Language. (2020). Retrieved from https://epodcastnetwork.com/the-importance-of-programming-languages/
- Importance of Computer Programming. (2020). Retrieved from https://www.techwalla.com/articles/importance-of-computer-programming
- INTRODUCTION. (2020). Retrieved from https://www.explainingcomputers.com/hardware.html
- Introduction to Computers. (2020). Retrieved from http://cs.sru.edu/~mullins/cpsc100book/module02_introduction/module02-03_introduction.html
- Computer Program. (2016). Retrieved from https://www.britannica.com/technology/computer-program
- What does a computer programmer do?. (2019). Retrieved from https://www.careerexplorer.com/careers/computer-programmer/
- What is a Programming Language?. (2020). Retrieved from https://www.computerhope.com/jargon/p/programming-language.htm
- Programming Languages Instructions. (2020). Retrieved from https://ccm.net/contents/313-programming-languages-instructions
- What is Code?. (2020). Retrieved from https://www.computerhope.com/jargon/c/code.htm

- Introduction of Programming Paradigms. (2018). Retrieved from https://www.geeksforgeeks.org/introduction-of-programming-paradigms/
- Introduction to Computer Systems/Program Development. (2020). Retrieved from https://en.wikibooks.org/wiki/Introduction_to_Computer_Information_Systems/Program_Development
- 10 Most Popular Programming Languages Today. (2020). Retrieve from https://www.inc.com/larry-kim/10-most-popular-programming-languages-today.html
- The Different Programming Languages. (2020). Retrieved from https://www.learneroo.com/modules/12/nodes/94
- What is JAVA? – Definition form Techopedia. (2020). Retrieved from https://www.techopedia.com/definition/3927/java
- What is JAVA and Why is it important?. (2020). Retrieved from https://codeinstitute.net/blog/what-is-java/#:~:text='Java%20can%20be%20used%20to,part%20of%20a%20Web%20page.
- IBM Knowledge Center. (2020). Retrieved from https://www.ibm.com/support/knowledgecenter/en/ssw_aix_71/performance/advantages_java.html
- https://www.vogella.com/tutorials/JavaIntroduction/article.html)
- https://java.com/en/download/faq/whatis_java.xml
- https://www.coursereport.com/blog/what-is-java-programming-used-for
- https://www.mindsmapped.com/java-advantages-and-disadvantages/
- https://www.tutorialspoint.com/explain-the-basic-structure-of-a-program-in-java
- https://www.educba.com/java-compilers/
- https://www.javatpoint.com/java-data-types
- https://www.javatpoint.com/java-variabless
- https://www.edureka.co/blog/what-is-java-constant/)
- https://www.geeksforgeeks.org/java-io-input-output-in-java-with-examples/
- https://www.tutorialspoint.com/java/java_basic_operators.htm
- https://www.visual-paradigm.com/tutorials/flowchart-tutorial/
- https://www.edrawsoft.com/flowchart-benefits.html
- https://www.smartdraw.com/flowchart/flowchart-types.htm
- https://www.geeksforgeeks.org/loops-in-java/
- https://www.w3schools.com/java/java_for_loop.asp
- https://beginnersbook.com/2015/03/for-loop-in-java-with-example/
- https://beginnersbook.com/2015/03/while-loop-in-java-with-examples/
- https://www.w3schools.com/java/java_while_loop.asp
- https://www.javatpoint.com/java-do-while-loop
- https://beginnersbook.com/2015/03/do-while-loop-in-java-with-example/
- www.cs.iit.edu/~cs561/cs115/looping/count.html