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

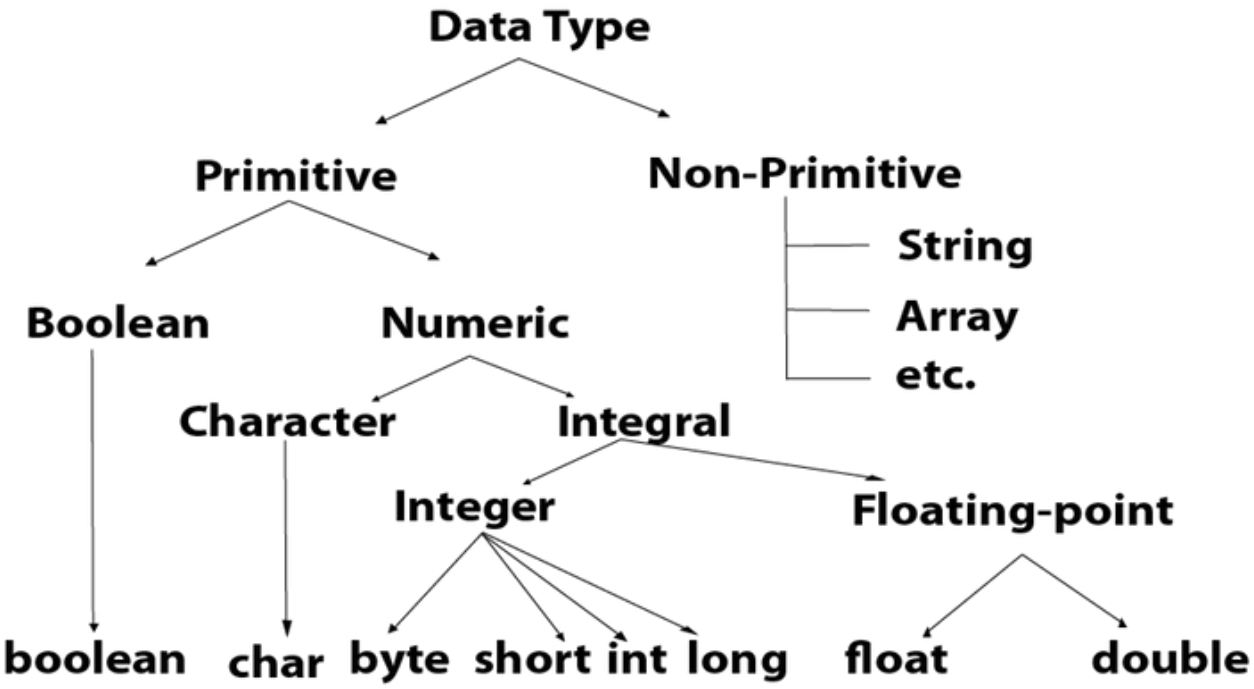
Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

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

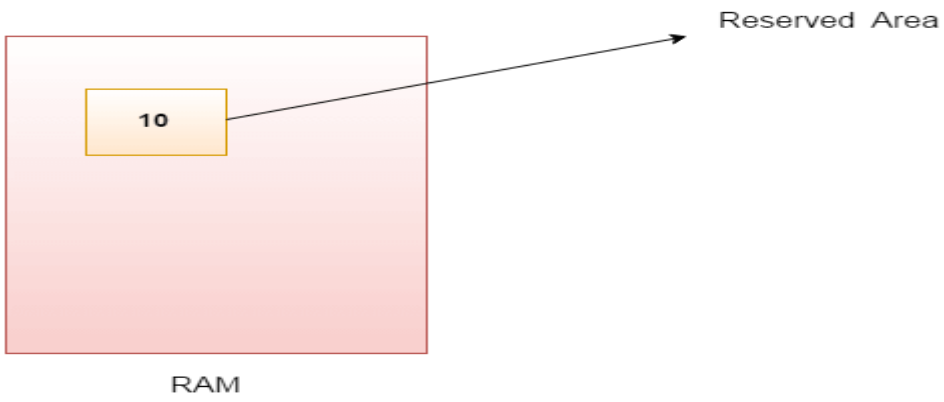
The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

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.

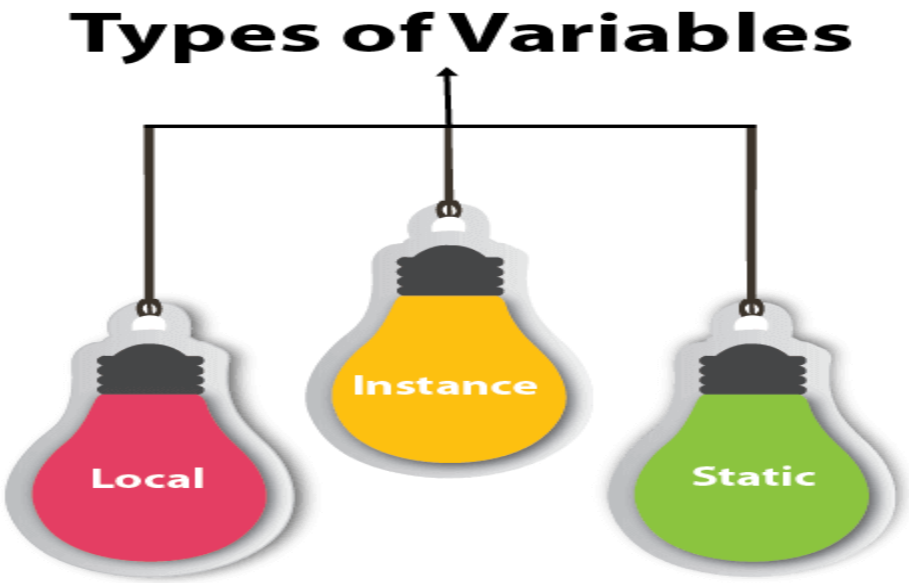
Variable is name of *reserved area allocated in memory*. In other words, it is a *name of memory location*. It is a combination of "vary + able" that means its value can be changed.



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

```
1. class A{
2. int data=50;//instance variable
3. static int m=100;//static variable
4. void method(){
5. int n=90;//local variable
6. }
7. }//end of class
```

Java Variable Example: Add Two Numbers

```
1. class Simple{
2. public static void main(String[] args){
3. int a=10;
4. int b=10;
5. int c=a+b;
6. System.out.println(c);
7. }}
```

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

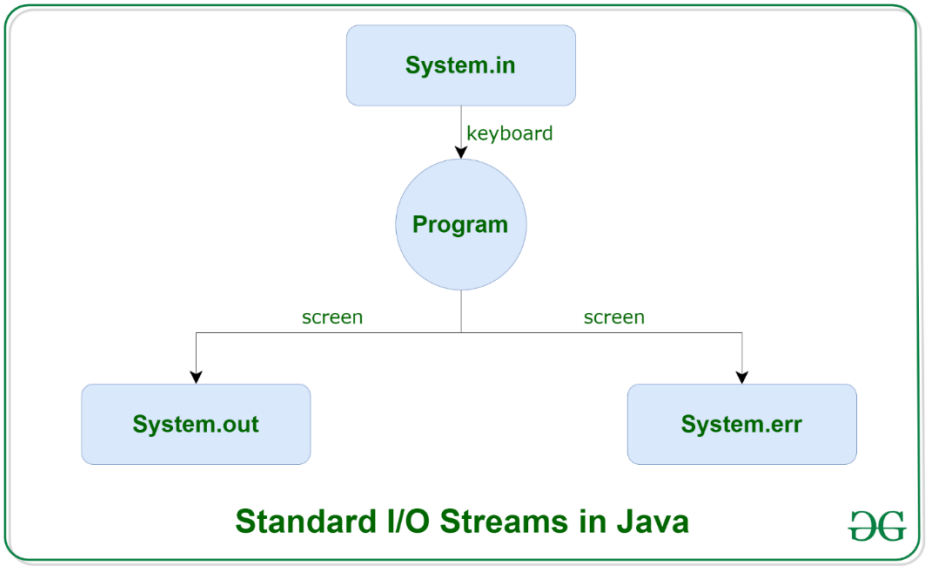
Static and Final Modifiers

- 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

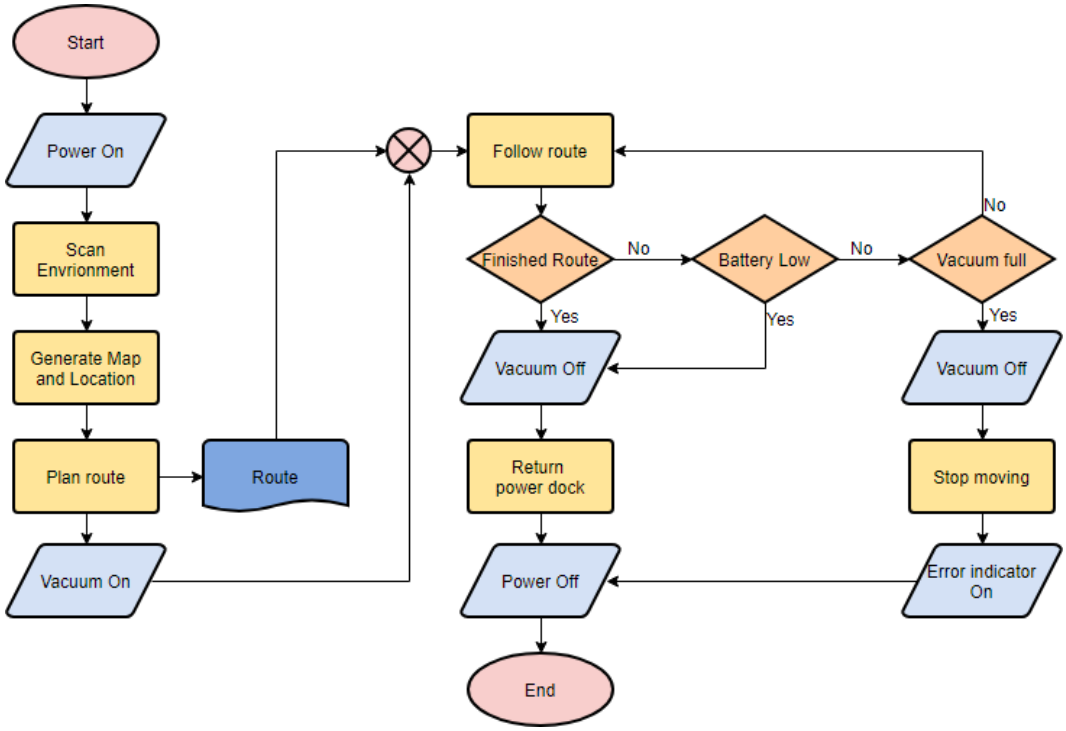
Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator

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.

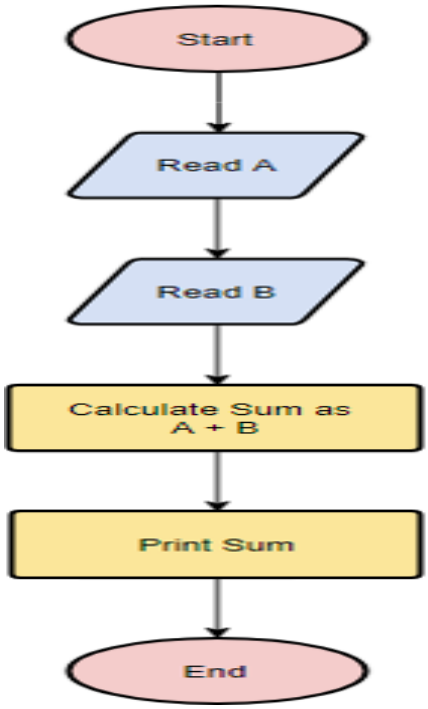
Flow

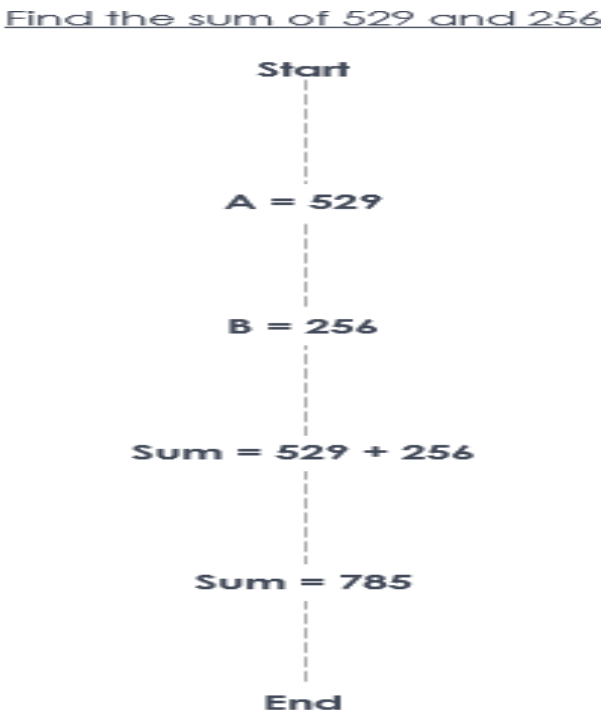
Lines represent the flow of the sequence and direction of a process.

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.

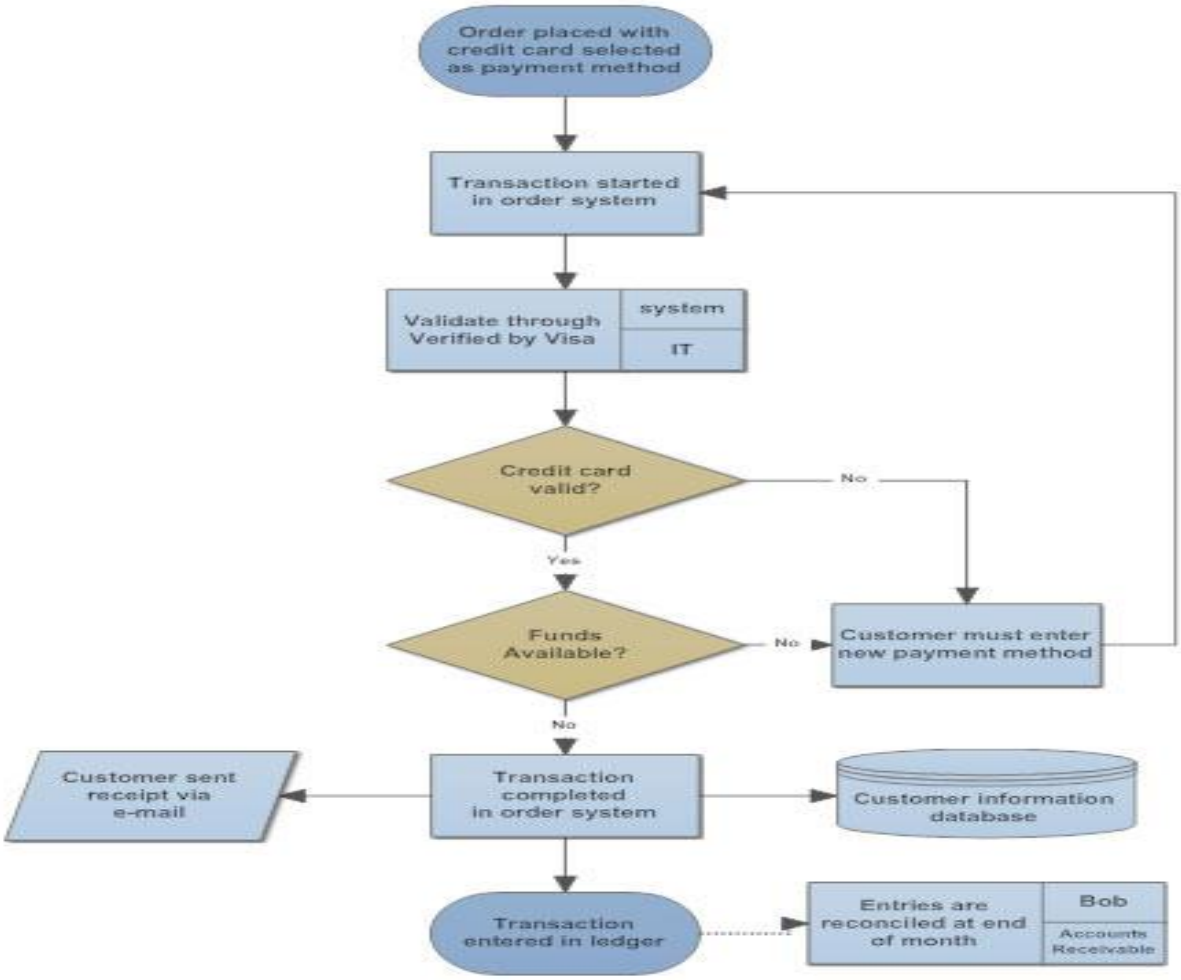
Proper Documentation

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

Sample Flowchart

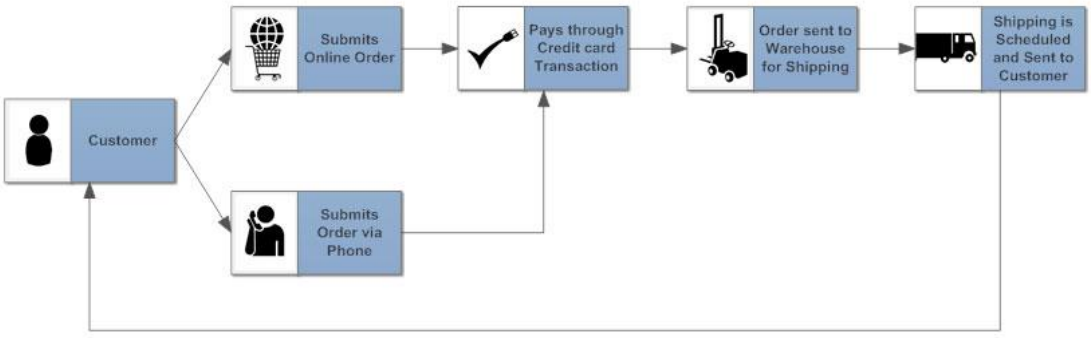
Basic Process Flowchart

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:



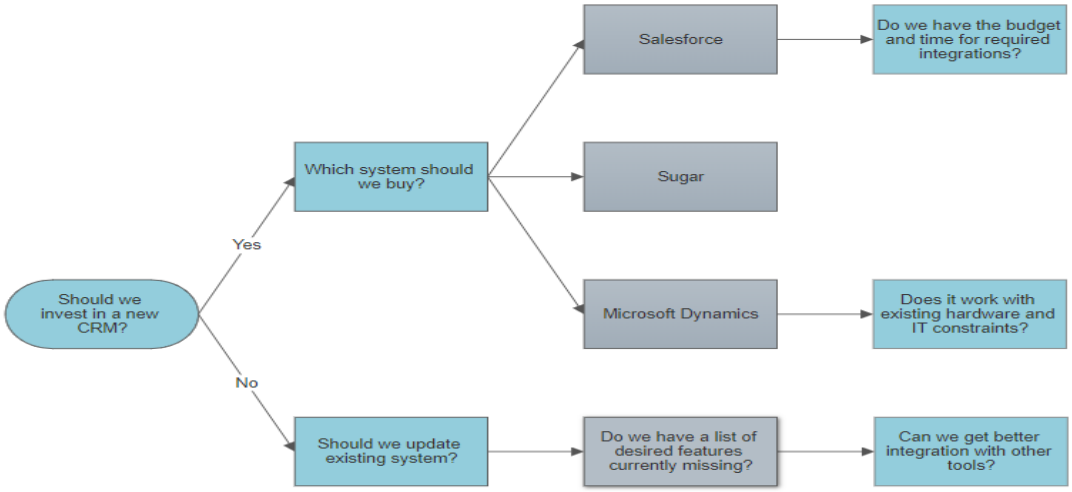
Workflow Process

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:



Decision Flowchart

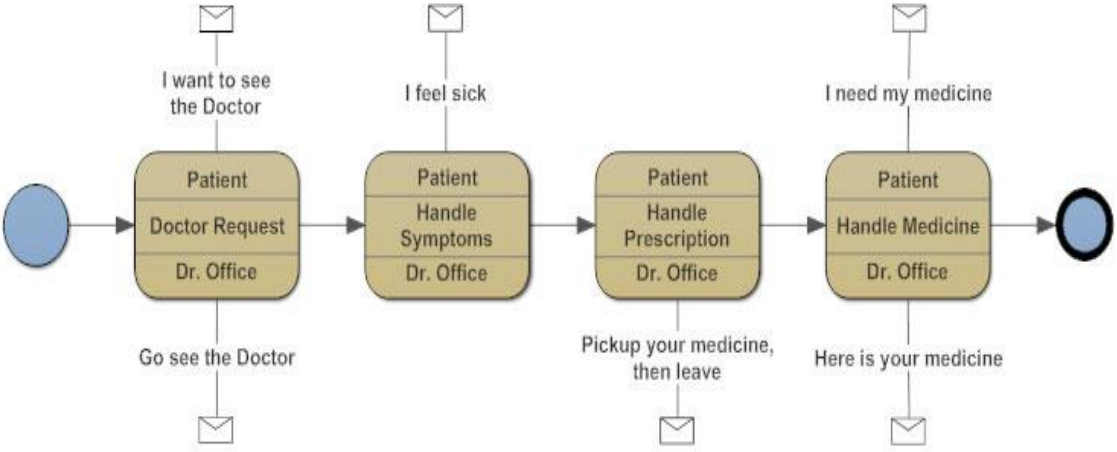
A decision flowchart lets you visualize the options in any important business decision. Walk through potential outcomes and make sure you consider all the questions before making a decision.



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.

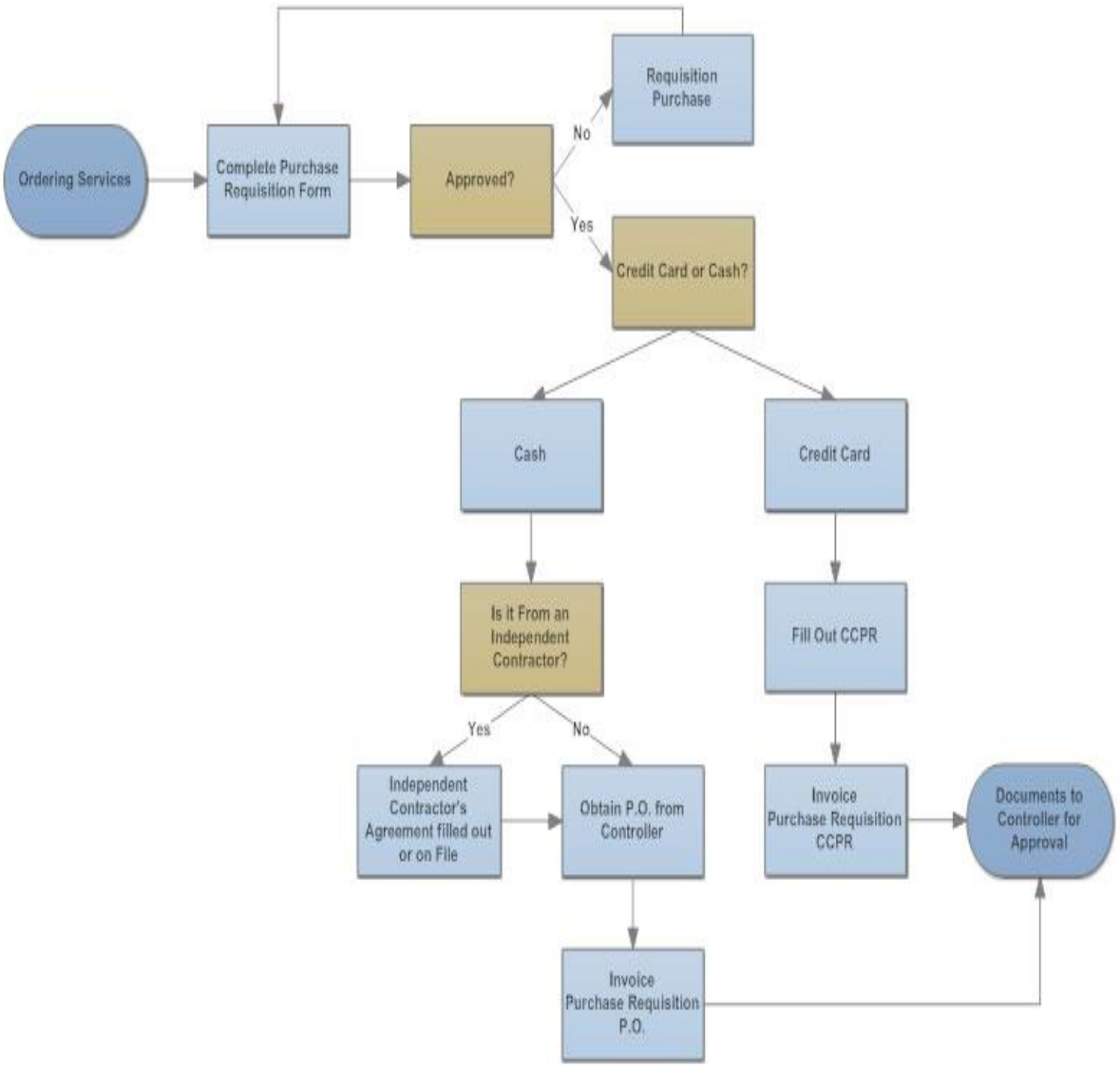


Process Map

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:

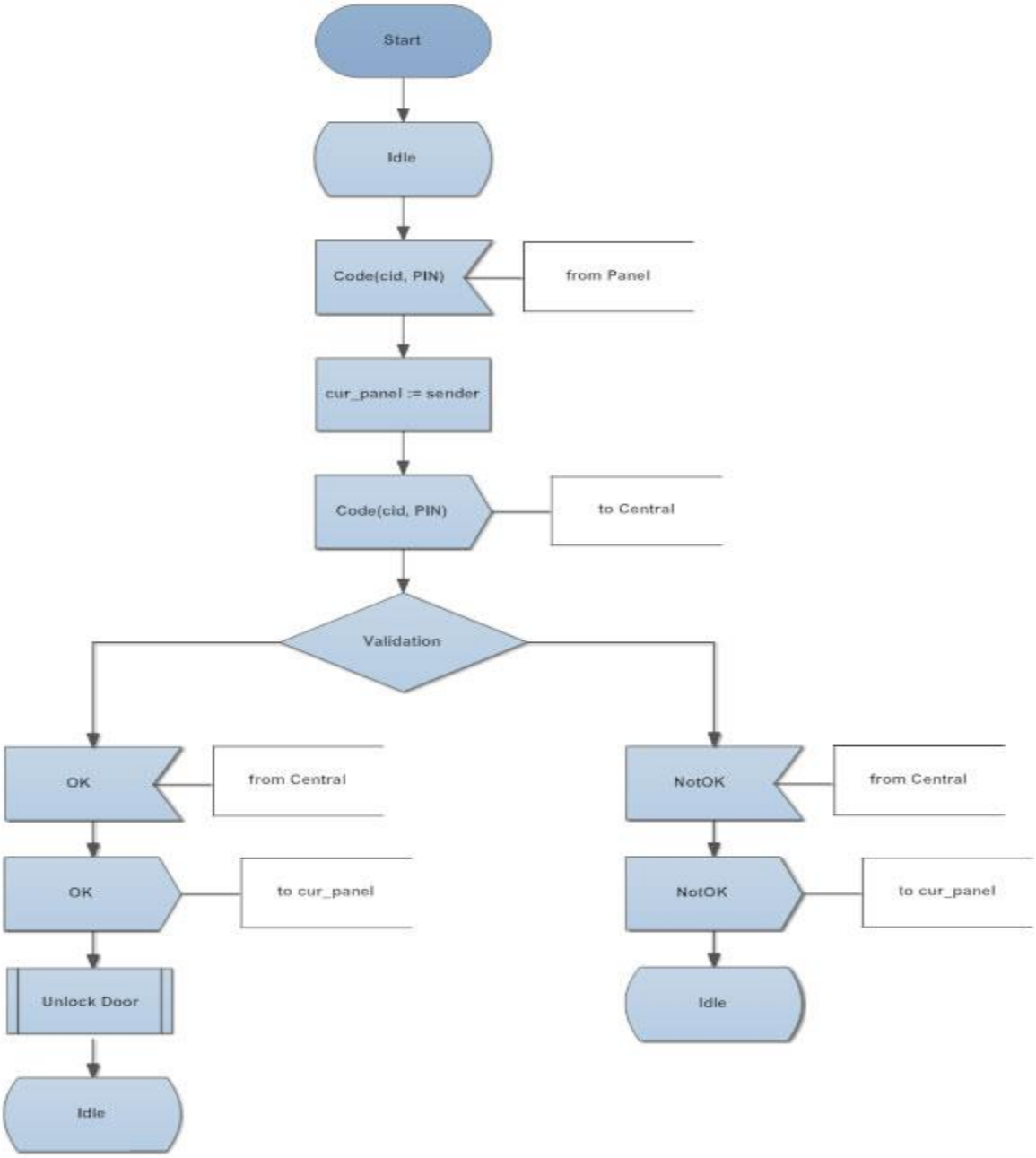
1. Identifying and understanding the steps in a process.
2. Gathering information to identify the objectives, risks and controls in a process.
3. Interviewing the individuals involved and creating the process map.
4. Analyzing and effecting changes to improve the process.



SDL Diagram

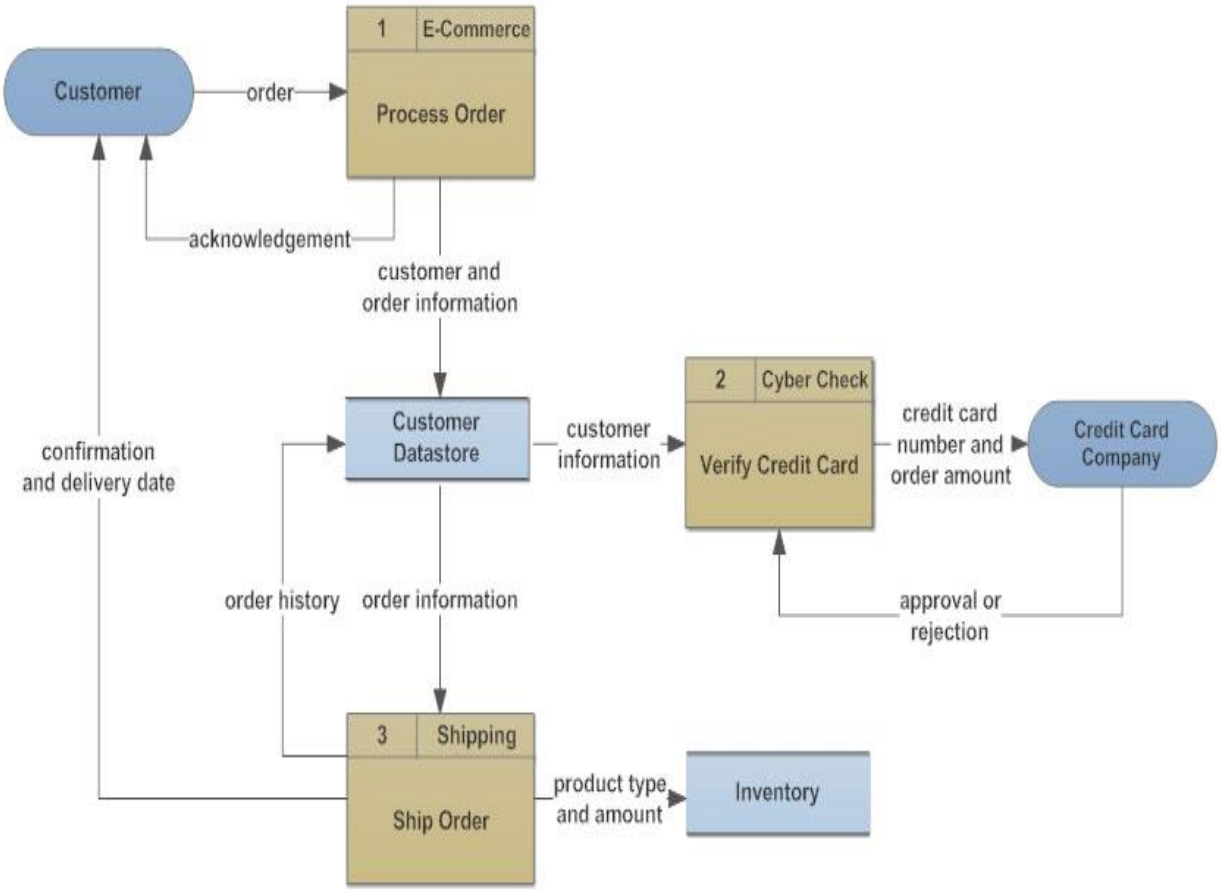
Brainstorming computer algorithms is often accomplished using an SDL diagram. SDL stands for Specification and Description Language. This is a flowchart that offers a unique set of symbols that are used to map out real-time systems. The three basic components of an SDL diagram are the system definition,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

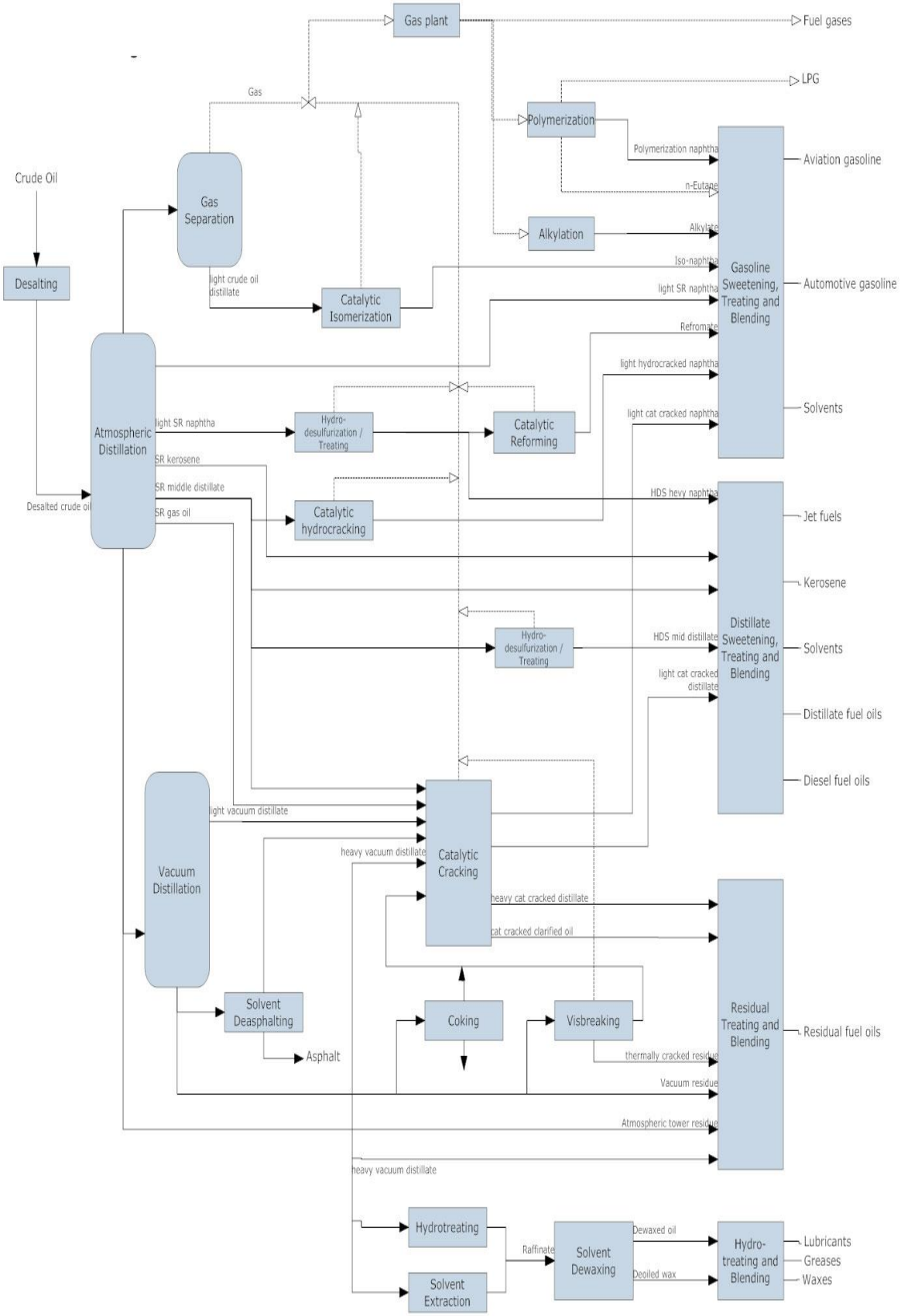
Data flow diagrams (DFD) are an efficient way of bridging the communication gap between system developers and users. They are specialized flowcharts that distill a substantial amount of information into a relatively few symbols and connectors.



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
Aviation gasoline
n-Eutane
Crude Oil
Gas Separation
Alkylation
Alkylate
Desalting
light crude oil distillate
Catalytic Isomerization
Iso-naphtha
Gasoline Sweetening, Treating and Blending
Automotive gasoline
light SR naphtha
Reformate
light hydrocracked naphtha
Atmospheric Distillation
light SR naphtha
Hydro-desulfurization / Treating
Catalytic Reforming
light cat cracked naphtha
Solvents
SR kerosene
SR middle distillate
SR gas oil
Desalted crude oil
Catalytic hydrocracking
HDS hevy naphtha
Jet fuels
Kerosene
Hydro-desulfurization / Treating
HDS mid distillate
Distillate Sweetening, Treating and Blending
Solvents
light cat cracked distillate
Distillate fuel oils
Diesel fuel oils
light vacuum distillate
Vacuum Distillation
heavy vacuum distillate
Catalytic Cracking
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
Hydro-treating and Blending
Lubricants
Greases
Waxes
Solvent Extraction
Deoiled wax