KOTLIN
CODING
EXAMPLI

PROGRAMMI
FOR BEGINNE

J KING

JAVA
BASICS

PROGRAMMING
FOR BEGINNERS

J KING

JAVA BASICS

AND

KOTLIN CODING EXAMPLES

PROGRAMMING FOR BEGINNERS

J KING

JAVA INTRODUCTION

C++ VS JAVA

JAVA HELLO WORLD

JDK | JAVA DEVELOPMENT KIT

JRE (JAVA RUNTIME ENVIRONMENT)

DIFFERENCE BETWEEN JDK, JRE, JVM

OPERATORS IN JAVA

JAVA KEYWORDS

DATA TYPES IN JAVA

JAVA VARIABLES

CONTROL STATEMENTS

JAVA IF-ELSE STATEMENT

JAVA SWITCH STATEMENT

KOTLIN CODING EXAMPLES

BASIC PROGRAMS IN KOTLIN

KOTLIN PROGRAM TO PERFORM ARITHMETIC OPERATIONS ON TWO NUMBERS

PROGRAM TO PERFORM SIMPLE CALCULATION

PROGRAM TO INPUT NUMBERS (INTEGER, FLOAT, DOUBLE) AT RUN TIME

KOTLIN PROGRAM TO FIND LARGEST OF THREE NUMBERS

KOTLIN PROGRAM TO SWAP TWO NUMBERS

KOTLIN PROGRAM TO CHECK WHETHER A NUMBER IS EVEN OR ODD

KOTLIN PROGRAM TO PRINT THE MULTIPLICATION TABLE

PROGRAM TO CALCULATE AND DISPLAY STUDENT GRADES

KOTLIN PROGRAM TO CONVERT TEMPERATURE

KOTLIN PROGRAM TO DISPLAY FIBONACCI SERIES

KOTLIN PROGRAM TO FIND AREA OF CIRCLE

KOTLIN PROGRAM TO FIND AREA OF SQUARE

PROGRAM TO FIND AREA OF A CYLINDER

KOTLIN PROGRAM TO GENERATE 4 DIGITS OTP

KOTLIN PROGRAM TO FIND AREA OF PENTAGON

JAVA

BASICS

PROGRAMMING FOR BEGINNERS

J KING

JAVA INTRODUCTION

The Java programming language was designed with portability, stability, and simplicity in mind.

Despite this, Java has a range of features that make it a common programming language.

We structure our application as a set of different types of objects that contain both data and actions, which is known as object-oriented programming.

Java is a WORA (Write Once, Run Anywhere) programming language because of this.

Java is architecture-neutral because it has no implementation-dependent aspects, making the compiled code executable on a wide range of processors using JRE.

Evolution of Java:

Version	Release Date	Features
JDK Beta	1995	—
JDK 1.0	January 1996	This is the first stable version.
JDK 1.1	February 1997	In this version, added many new library elements, redefined the way of handling events, and reconfigured most of the libraries of 1.0 and deprecated some features defined by 1.0. Added inner class, JavaBeans, JDBC, RMI, JIT (Just In time) compiler.
J2SE 1.2	December 1998	Added support for many features, such as Swing and Collection Framework. The methods suspend(), resume() and stop() of Thread class were deprecated.
J2SE 1.3	May 2000	A very small improvement, as it just improved the development environment.

J2SE 1.4	February 2002	It added some upgrades such as the new keyword *assert*, chained exception and a channel-based I/O subsystem. Also added some feature to the collection framework and the Networking classes.
J2SE 5.0	September 2004	The significant new features added to this version are – Generics, Annotation, Autoboxing and Auto-unboxing, Enumeration, for-each, variable-length argument, Static import, Formatted I/O, Concurrency utilities.
Java SE 6	December 2006	In this version the API libraries and several new packages got enhanced and offered improvements to the run time. It supports JDBC 4.0.
Java SE 7	July 2011	Added JVM support for dynamic language, String in the switch, Automatic resource management in try-statement, support for underscore in integers, binary integer literals etc...
Java SE 8	March 2014	Added Date and time API, Repeating annotation, JavaFX.
Java SE 9	September 2017	Added Java platform module system update, jshell, XML Catalog, jlink, and the JavaDB was removed from JDK
Java SE 10	March 2018	Added features are local variable type interface, Application class data sharing, Garbage collector interface, etc...
Java SE 11	September 2018	Feature added: Dynamic class file loader, HTTP client, and Transport layer security. JavaFX, Java EE, and CORBA modules have been removed from JDK.
Java SE 12	March 2019	Added Microbenchmark Suite, JVM Constant API, One AArch64 Port, Default CDS Archives etc.

C++ vs Java

	C++	Java	C
	C++ is a general programming language created by Bjarne Stroustrup as an extension of c language	Java is class-based, object-based, and designed to have as few implementation dependencies as possible.	C is considered as all programming language father, and It is a general-purpose, procedural programming language supporting structured programming, lexical variable scope, and recursion.
Developer	C++ programming language was invented by **Bjarne Stroustrup** in 1979	Java language was invented by **James Gosling** in 1991	C language was invented by **Dennis Ritchie** in 1972
Programming	In C++ the programming paradigm follows the	The programming supports pure	C is entirely a procedural

Paradigm	Object-Oriented programming	Object-Oriented Programming	language.
Origin	The C++ programming language is based upon C language.	Java is based on both C++ and the C language.	C language is based on assembly level language.
Translator	C++ follows compiler to translate the code into binary-level code.	Java uses an interpreter to translate the code in binary codes.	C also uses a compiler to translate the code into binary level code.
Platform dependency	C++ programming language is platform dependent.	Java is platform-independent as the code written on one platform can run on any platform.	C is also platform dependent.
Code Execution	C++ supports direct code execution.	Java follows JVM which Is Java Virtual Machine. Code will be sent to JVM for translation and code execution.	C language follows direct code execution.
		Java also	C language

Approach	C++ supports the bottom-up approach.	supports bottom up approach.	supports top down approach.
File generation	It generates a .exe file after compiling the program.	It generates a .class file after compilation of a program	It also creates a .exe file.
Pre-processor directives	C++ follows #header and #define to include the pre-processor directives.	Java uses the import option to import the packages it needs to compile the program it is running	C being the predecessor of CPP it also follows #header and #define to include
Key words	C++ language has 63 defined keywords.	Java language has 50 defined keywords.	C language has the support for 52 keywords.
Data Types	Data types like union and structure are supported in C++ programming.	Structure and Union are not supported in Java.	Data types like union and structure are supported in C language also.
In heritance	Cpp and Java both support inheritance	But Java does not support multiple inheritances. To support Java has to use	Does not support inheritance

		a unique feature called interface	
Overloading	C++ support overloading through polymorphism.	Operator overloading is not supported by Java.	Overloading is not supported by C language.
Pointers	C++ supports pointers.	Java does not support pointers.	C supports pointers.
Memory Allocation	Cpp uses the new keyword to allocate memory to a program and to delete it uses “delete” keyword	Java uses the new keyword to allocate memory and uses a unique feature called garbage collector to remove the unwanted memory space.	When comes to C language we use calloc(), malloc() and realloc() function to allocate the memory to a particular program and we use the free() function if we wanted to delete or free the memory space from the individual application.
Exception Handling	C++ supports exception	Java also supports	C does not support

	handling.	exception handling.	exception handling.
Templates	C++ supports templates.	Java does not support templates.	C does not support templates.
Destructors	Supported by C++	Not endorsed by Java	C does not support the constructor nor destructors.
Multi-threading/ Interfaces	Multi-threading is not supported by C++ programming language.	Java supports multithreading to overcome the multiple inheritance problem.	Being the predecessor of C++, it also does not support multi-threading.
Database connectivity	C++ do not support DataBase connectivity.	Java supports DataBase connectivity.	C language does not support database connectivity.
Storage classes	C++ supports the storage class	Java does not supports storage class	C supports storage class.

Java Hello World

To begin, write a simple program that prints "Hello World" to the output window.

Write the software in any text editor or integrated development environment (IDE) (Eclipse, Netbeans, etc.) and save it as a.java file.

You can compile the file by typing “javac filename.java>” in the command prompt or terminal in Linux, or by clicking the play button on the toolbar if you're using an IDE.

Run your program by typing "java filename>" in the terminal/command prompt.

```
public class MyClass //class
{
        //this is the main method
public static void main(String[] args)
        {
             //displaying Hello World
                    System.out.println("Hello World");
        }
}
```

Every line of code that runs in Java must be written within the class. In our example, we named the class MyClass and wrote the code within it.

The entry point or starting point is the key method in the above program. The following is an overview of the keywords used in the main method:

- **Public -** This approach is available to everyone.
- **Static -** The method will run without having to build an instance of the class that contains the main method.
- **Void -** There is no value returned by the system.
- **Main -** The process is known as Primary.
- **String [] -** The method only takes one argument: an array of String elements.

Comments : In the program, there are two ways to compose comments. It can be a single-line or multiple-line statement.

Single line comment : The double forward slash (//) at the beginning of any line defines it.

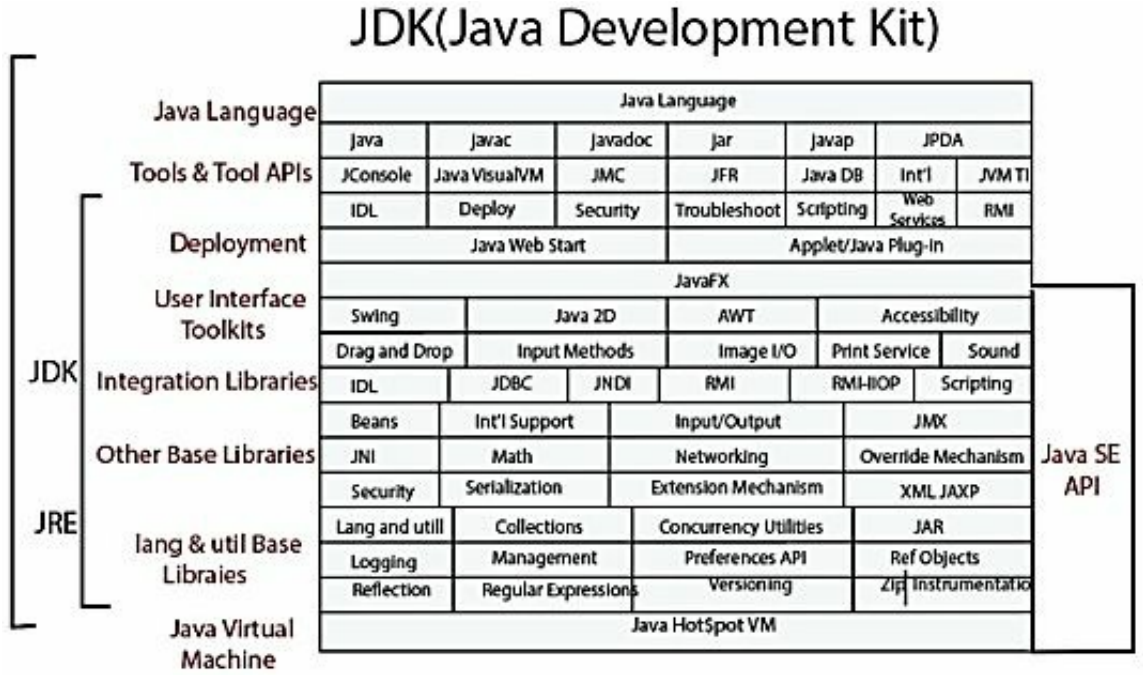
Multiline Comment: At both ends of the pargraph, a single forward slash and an asterisk (/* comment */) characterize it.

JDK | Java Development Kit

The Java Development Kit is a software environment for developing Java applications.

JRE (Java Runtime Environment), java (interpreter), javac (compiler), archiver, documentation generator, and other tools for java application development are all included in the JDK.

Oracle Corporation is making the platform available as a binary product for Java developers operating on Solaris, Linux, Windows, or macOS.

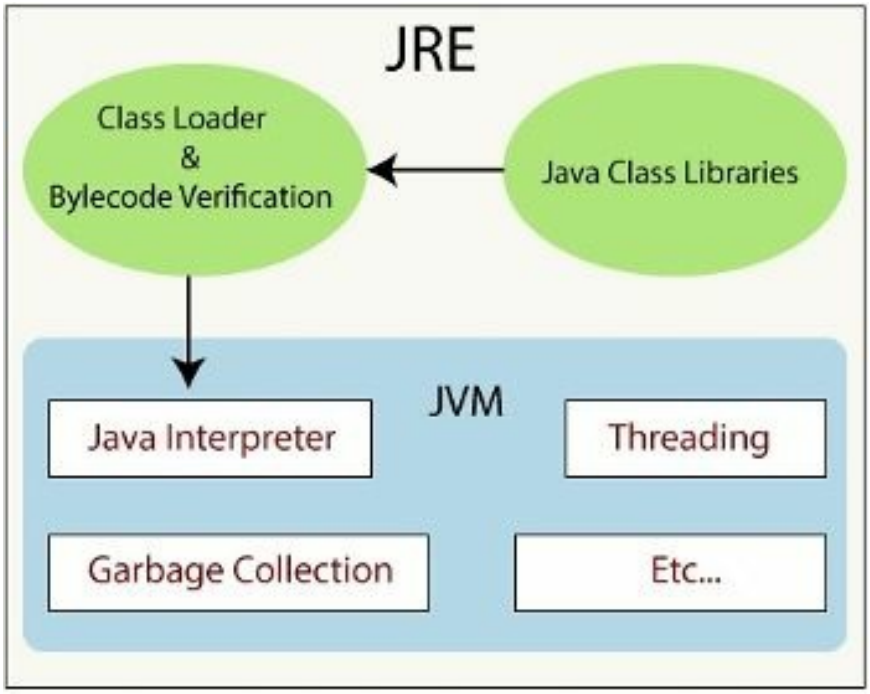


JRE (Java Runtime Environment)

JRE is a Installation package that installs on any operating system and allows you to run Java programs. It has little to do with the application creation process.

It's included in the Java Development Kit (JDK).

It's a software kit that includes the Java Class Library, tools, and a standalone JVM.



JRE is the most popular environment for running Java programs on mobile devices.

The source code is compiled and then converted into Java bytecode.

If we want to run this bytecode on any Operating System, we simply need a JRE for that OS, which is freely available on the Oracle website.

The JRE fetches classes, verifies memory access, and retrieves system resources that the generated program needs.

Working of JRE

All source code that we write must be saved with the.java extension.

Next, we use the “javac” command in the command prompt to compile our software (no need in any IDE). It converts the code to bytecode, which is platform-agnostic.

When we compile our software, it produces a.class file that contains the bytecode for the equivalent java file.

And that bytecode can be run on any device that has the JRE enabled.

The procedure is outlined below.

ClassLoader-

It dynamically loads the various classes needed for program execution in the JVM (Java Virtual Machine). Following are the three class loaders that are used after JVM starts:

- Bootstrap class loader
- Extension class loader
- System class loader

Byte code verifier –

It verifies the bytecode during execution to ensure that the code does not cause the interpreter any problems. Only if the codes pass the bytecode verifier's checks, which search for formatting and illegal code, are they interpreted.

Interpreter-

When the class is loaded and the code is tested, it reads the assembly code

line by line and executes the two functions below:

It is responsible for executing the Byte Code.

Make the requisite calls to the built-in hardware.

Difference between JDK, JRE, JVM

JDK	JRE	JVM
JDK stands for Java Development Kit. It provides development tools and execution environment.	JRE stands for Java Runtime Environment. It provides the set of tools only to execute our program.	JVM stands for Java Virtual Machine. It is responsible for the program to execute line by line.
JDK is the superset of the JRE. It contains JRE with Java compiler, debugger, and core classes.	JRE contains a set of libraries and other files that JVM uses at runtime.	It contains JIT (Just in Time Compiler), Interpreter, and set of APIs. JIT optimizes bytecode to machine specific language compilation by compiling similar bytecodes.

Uses JVM, interpreter/loader, a compiler, an archiver, a documentation generator (Javadoc), and set of APIs to complete the development of Java application.	Uses a set of libraries, other jar files, and set of API to run the program. It doesn't play any role during development. It only works to run the program.	JVM works to Load code, Verifies code, and Execute code and provides a runtime environment.
It physically exits	It physically exits	It doesn't physically exist.

Operators in Java

In Java, an operator is a symbol that performs operations. For instance, +, -, *, /, and so on.

In Java, there are several different types of operators, which are described below:

- Unary Operator,
- Arithmetic Operator,
- Shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Java Operator Precedence

Operator Type	Category	Precedence
Unary	postfix	*expr++ expr--*
	prefix	*++expr --expr +expr -expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Java Unary Operator

Only one operand is necessary for the Java unary operators. Unary operators are used to perform a variety of functions, for example:

- by one incrementing/decrementing a value
- negating a statement
- a boolean's value is inverted

Java Unary Operator Example: ++ and --

```
class OperatorExample{
public static void main(String args[]){
int x= 10 ;
System.out.println(x++); //10 (11)
System.out.println(++x); //12
System.out.println(x--); //12 (11)
System.out.println(--x); //10
}}
```

Output:

```
10

12

12

10
```

Java Unary Operator Example: ~ and !

```
class  OperatorExample{
public  static  void  main(String args[]){
int  a= 10 ;
int  b=- 10 ;
boolean  c= true ;
boolean  d= false ;
System.out.println(~a); //-11 (minus of total positive value which starts from 0
System.out.println(~b); //9 (positive of total minus, positive starts from 0)
System.out.println(!c); //false (opposite of boolean value)
System.out.println(!d); //true
}}
```

Output:

-11

9

false

true

Java Arithmetic Operators

Addition, subtraction, multiplication, and division are all done using Java arithmatic operators.

They work as fundamental mathematical operations.

Java Arithmetic Operator Example

```
class  OperatorExample{
```

```
public  static  void  main(String args[]){
int  a= 10 ;
int  b= 5 ;
System.out.println(a+b); //15
System.out.println(a-b); //5
System.out.println(a*b); //50
System.out.println(a/b); //2
System.out.println(a%b); //0
}}
```

Output:

15

5

50

2

0

Java Arithmetic Operator Example: Expression

```
class  OperatorExample{
public  static  void  main(String args[]){
System.out.println( 10 * 10 / 5 + 3 - 1 * 4 / 2 );
}}
```

Output:

21

Java Left Shift Operator

The Java left shift operator << moves all the bits in a value to the left by a given number of times.

EXAMPLE

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10<<2);//10*2^2=10*4=40
System.out.println(10<<3);//10*2^3=10*8=80
System.out.println(20<<2);//20*2^2=20*4=80
System.out.println(15<<4);//15*2^4=15*16=240
}}
```

Output:

40

80

80

240

Java Right Shift Operator

The Java right shift operator >> shifts the value of the left operand by the number of bits defined by the right operand to the right.

EXAMPLE

```
class OperatorExample{
```

```
public static void main(String args[]){
System.out.println(10>>2);//10/2^2=10/4=2
System.out.println(20>>2);//20/2^2=20/4=5
System.out.println(20>>3);//20/2^3=20/8=2
}}
```

Output:

2

5

2

Java AND Operator Example: Logical && and Bitwise &

If the first condition is false, the logical && operator does not verify the second condition. It only tests the second condition if the first is true.

If the first condition is true or false, the bitwise & operator always tests both conditions.

```
class  OperatorExample{
public  static  void  main(String args[]){
int  a= 10 ;
int  b= 5 ;
int  c= 20 ;
System.out.println(a<b&&a<c); //false && true = false
System.out.println(a<b&a<c); //false & true = false
}}
```

Output:

false

false

Java AND Operator Example: Logical && vs Bitwise &

```
class  OperatorExample{
public  static  void  main(String args[]){
int  a= 10 ;
int  b= 5 ;
int  c= 20 ;
System.out.println(a<b&&a++<c); //false && true = false
System.out.println(a); //10 because second condition is not checked
System.out.println(a<b&a++<c); //false && true = false
System.out.println(a); //11 because second condition is checked
}}
```

Output:

false

10

false

11

Java OR Operator Example: Logical || and Bitwise |

If the first condition is true, the logical || operator does not verify the second condition. It only tests the second condition if the first is false.

If the first condition is true or false, the bitwise | operator always tests both conditions.

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a>b||a<c);//true || true = true
System.out.println(a>b|a<c);//true | true = true
//|| vs |
System.out.println(a>b||a++<c);//true || true = true
System.out.println(a);//10 because second condition is not checked
System.out.println(a>b|a++<c);//true | true = true
System.out.println(a);//11 because second condition is checked
}}
```

Output:

```
true
true
true
10
```

true

11

Java Ternary Operator

In Java programming, the Ternary operator is a one-liner substitute for the if-then-else argument. It's the only conditional operator that accepts three operands.

EXAMPLE 1

```
class OperatorExample{
public static void main(String args[]){
int a=2;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```

Output:

2

EXAMPLE 2

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```

Output:

5

Java Assignment Operator

One of the most common operators in Java is the assignment operator. It's used to allocate the operand on the left to the value on the right.

EXAMPLE

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=20;
a+=4;//a=a+4 (a=10+4)
b-=4;//b=b-4 (b=20-4)
System.out.println(a);
System.out.println(b);
}}
```

Output:

14

16

Java Assignment Operator Example: Adding short

```
class  OperatorExample{
public   static   void  main(String args[]){
short  a= 10 ;
short  b= 10 ;
//a+=b;//a=a+b internally so fine
```

```
a=a+b; //Compile time error because 10+10=20 now int
System.out.println(a);
}}
```

Output:

Compile time error

After type cast:

```
class  OperatorExample{
public   static   void  main(String args[]){
short  a= 10 ;
short  b= 10 ;
a=( short )(a+b); //20 which is int now converted to short
System.out.println(a);
}}
```

Output:

20

Java Keywords

In the Java programming language, there are 50 specific keywords that are used for internal processes or to describe predefined actions.

These java keywords are case sensitive and have special definitions.

Keywords aren't allowed to be used as identifiers, such as names for variables, classes, or methods.

True, wrong, null, const, and goto are all reserved keywords in Java.

These keywords should not be included in the names of variables, classes, or other objects.

Keywords	Description
abstract	**It is used to declare an abstract class that provides the implementation of the interface.**
boolean	**It is used to declare a variable as a boolean type. It only holds true and false values.**
break	**It breaks the current flow of the code (loop or switch statement).**
byte	**This keyword declares a variable that can hold 8-bit data values.**

case	**It is used with the switch statement to define different cases.**
catch	**This keyword is used to catch the exceptions generated by try statements.**
char	**It declares a variable that can hold unsigned 16-bit Unicode characters**
class	**It is used to declare a class.**
continue	**This keyword is used to continue the loop (current flow of the program and skips the remaining code at the specified condition).**
default	**The default keyword is used to define the default block of code in a switch statement.**
do	**This keyword is used in the control statement to declare a loop.**
double	**This keyword declares a variable that can hold a 64-bit floating-point number.**
else	**It is used to indicate the alternative branches in an “if” statement.**

enum	**This keyword defines a fixed set of constants. This constructor is always private or default.**
extends	**It indicates that a class is derived from another class or interface.**
final	**It is used to indicate that a variable holds a constant value. The value of the variable always remains constant if we declare it with the final keyword.**
finally	**It indicates a block of code in a try-catch structure. It is always executed whether an exception is handled or not.**
float	**This keyword declares a variable that can hold a 32-bit floating-point number.**
for	**It executes a set of instructions or functions repeatedly when some conditions become true. If the number of iteration is fixed, it is recommended to use “for loop”.**
if	**It tests the condition and executes the “if block” if the condition is true.**

implements	**It is used to implement an interface.**
import	**This keyword makes classes and interfaces available and accessible to the current source code.**
instanceof	**It tests whether the object is an instance of the specified class or implements an interface.**
int	**It is used to declare a variable that can hold a 32-bit signed integer.**
interface	**It declares an interface. It can have only abstract methods.**
long	**It declares a variable that holds a 64-bit integer.**
native	**This keyword specifies that a method is implemented in native code using JNI (Java Native Interface).**
new	**It creates a new object.**

null	**This keyword indicates that a reference does not refer to anything. It removes the garbage value.**
package	**It declares a Java package that includes the classes.**
private	**It is an access modifier that indicates that a method or variable may be accessed only in the class where it was declared.**
protected	**It is an access modifier that can be accessed within the package and outside the package but through inheritance only. It can’t be applied to the class.**
public	**It is an access modifier that is used to indicate that an item can be accessed anywhere. It has the widest scope among all other modifiers.**
return	**This keyword is used to return from a method when its execution is complete.**
short	**This keyword declares a variable that can hold a 16-bit integer.**

static	**This keyword indicates that a variable or method is a class method. It is used for memory management mainly.**
strictfp	**It is used to restrict the floating-point calculations to ensure the portability of the code.**
super	**This keyword is a reference variable that is used to refer to the parent class object. It can also be used to invoke an immediate parent class method.**
switch	**It contains a switch statement which executes code based on test value. It tests the equality of a variable against multiple values.**
synchronized	**This keyword is used to specify the critical sections or methods in a multithreaded code.**
this	**It refers to the current object in a method or constructor.**
throw	**It explicitly throws an exception. The “throw” keyword is used to throw a custom**

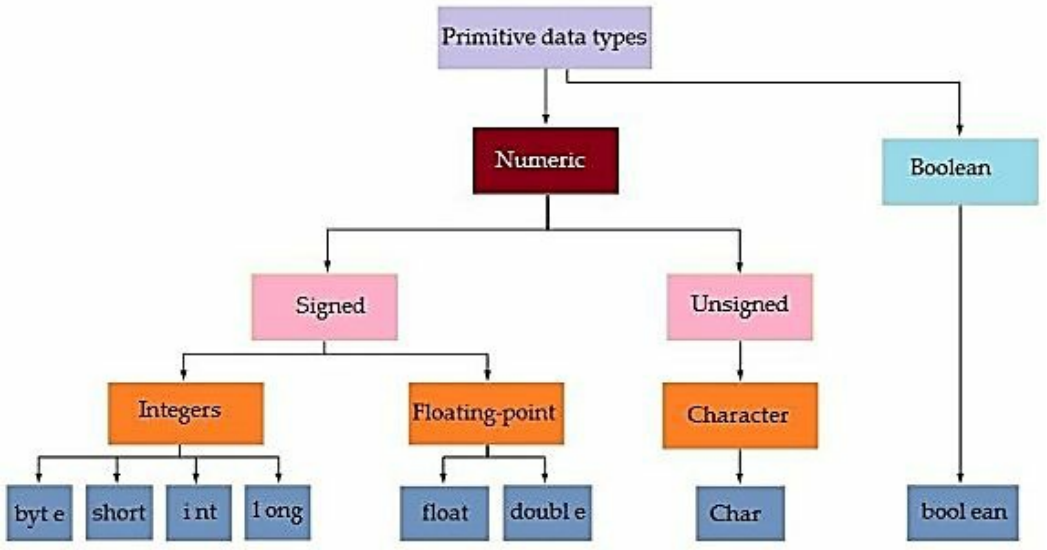
	exception, followed by an instance.
throws	**It declares an exception. A checked exception can be piped with throws.**
transient	**This keyword is used in serialization. If we define any data member as transient, then it can’t be serialized.**
try	**It starts a block of code that will be tested for exceptions. It must be followed by either catch or finally block.**
void	**The void keyword specifies that a method does not have a return value.**
volatile	**This keyword is used to indicate that a variable may change asynchronously.**
while	**It starts a loop that iterates a part of the program number of times.**

Data Types in Java

The different sizes and values that can be stored in the variable are described by data types. In Java, there are two kinds of data types:

Primitive data types: Boolean, char, byte, short, int, long, float, and double are examples of primitive data types.

Non-primitive data types: Classes, Interfaces, and Arrays are examples of non-primitive data forms.



Java Primitive Data Types

Primitive data types are the building blocks of data manipulation in the Java programming language. These are the most basic data types in the Java programming language.

- boolean data type
- byte data type
- char data type

- short data type
- int data type
- long data type
- float data type
- double data type

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

Boolean Data Type

Only two possible values are stored in the Boolean data type: true and false. Simple flags that track true/false conditions are stored in this data form.

The Boolean data type defines a single bit of data, but its "size" cannot be precisely specified.

EXAMPLE :

Boolean one = false

Byte Data Type

The primitive data form byte is an example of this. It's an 8-bit two-s complement signed integer. It has a value range of -128 to 127. (inclusive).

It has a minimum of -128 and a maximum of 127. It has a value of 0 by default.

The byte data form is used to conserve memory in wide arrays where space is at a premium.

Since a byte is four times smaller than an integer, it saves space. It can also be substituted for the "int" data sort.

Example,

byte a = 10 and

byte b = -20.

Short Data Type

A 16-bit signed two's complement integer is the short data form. It has a value range of -32,768 to 32,767. (inclusive). It has a minimum of -32,768 and a maximum of 32,767. It has a value of 0 by default.

The short data type, like the byte data type, can be used to save memory. A short data form is twice the size of an integer.

EXAMPLE

short s = 10000,

short r = -5000

Int Data Type

A 32-bit signed two's complement integer is represented by the int data form.

Its range of values is - 2,147,483,648 (-231 -1) to 2,147,483,647 (231 -1). (inclusive).

It has a minimum of 2,147,483,648 and a limit of 2,147,483,647.

It has a value of 0 by default.

If there is a memory limit, the int data type is commonly used as the default data type for integral values.

EXAMPLE

int a = 100000,

int b = -200000

Long Data Type

A 64-bit two's complement integer is the long data form. It has a value range of -9,223,372,036,854,775,808(-263 -1) to 9,223,372,036,854,775,807(263 -1). (inclusive).

Its minimum and maximum values are 9,223,372,036,854,775,808 and 9,223,372,036,854,775,807.

It has a value of 0 by default. When you need a wider set of values than int can provide, you can use the long data form.

EXAMPLE

long a = 100000L,

long b = -200000L

Float Data Type

The float data form is a 32-bit IEEE 754 floating point with single precision.

It has an infinite value set. If you need to conserve memory in huge arrays of floating point numbers, use a float (rather than a double).

For precise values, such as money, the float data form can never be used. 0.0F is the default value.

EXAMPLE

float f1 = 234.5f

Double Data Type

A double data form is a 64-bit IEEE 754 floating point with double precision.

It has an infinite value set. Like float, the double data form is commonly used for decimal values.

For precise values, such as money, the double data form can never be used. 0.0d is the default value.

EXAMPLE

double d1 = 12.3

Char Data Type

A single 16-bit Unicode character is represented by the char data type. It has a meaning range of 'u0000' (or 0) to 'uffff' (or 65,535 inclusive).

Characters are stored using the char data form.

EXAMPLE

```
char letterA = 'A'
```

Java Variables

A variable is a container that contains the value during the execution of a Java program.

A data type is allocated to a variable.

The word "variable" refers to the name of a memory spot. Local, instance, and static variables are the three types of variables in Java.

The variable is the name of a memory-allocated reserved field.

To put it another way, it's the name of a memory place.

It is made up of the words "variy + able," which means that its meaning can be modified.

```
int data=50;//Here data is variable
```

Types of Variables

- local variable
- instance variable
- static variable

1) Local Variable

A local variable is a variable declared within the method's body.

This variable can only be used within that method, and the other methods in the class are unaware that it exists.

The keyword "static" cannot be used to describe a local variable.

2) Instance Variable

An instance variable is a variable declared within the class but outside the method body.

It hasn't been declared static.

It's called an instance variable because its value is exclusive to each instance and isn't shared across them.

3) Static variable

A static variable is one that has been declared as such.

It's not possible for it to be local.

You may make a single copy of a static variable and share it across all of the class's instances.

When the class is loaded into memory, memory allocation for static variables

occurs only once.

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 Variable Example: Widening

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

```
int  a= 10 ;
float  f=a;
System.out.println(a);
System.out.println(f);
}}
```

Output:

10

10.0

Java Variable Example: Narrowing (Typecasting)

```
class  Simple{
public  static  void  main(String[] args){
float  f= 10 .5f;
//int a=f;//Compile time error
int  a=( int )f;
System.out.println(f);
System.out.println(a);
}}
```

Output:

10.5

10

Java Variable Example: Overflow

```
class  Simple{
public  static  void  main(String[] args){
//Overflow
int  a= 130 ;
```

```
byte  b=( byte )a;
System.out.println(a);
System.out.println(b);
}}
```

Output:

```
130

-126
```

Java Variable Example: Adding Lower Type

```
class  Simple{
public  static  void  main(String[] args){
byte  a= 10 ;
byte  b= 10 ;
//byte c=a+b;//Compile Time Error: because a+b=20 will be int
byte  c=( byte )(a+b);
System.out.println(c);
}}
```

Output:

```
20
```

Control Statements

Java If-else Statement

The condition is evaluated using the Java if statement.

It determines whether a boolean condition is true or false.

In Java, there are several types of if statements.

- if statement
- if-else statement
- if-else-if ladder
- nested if statement

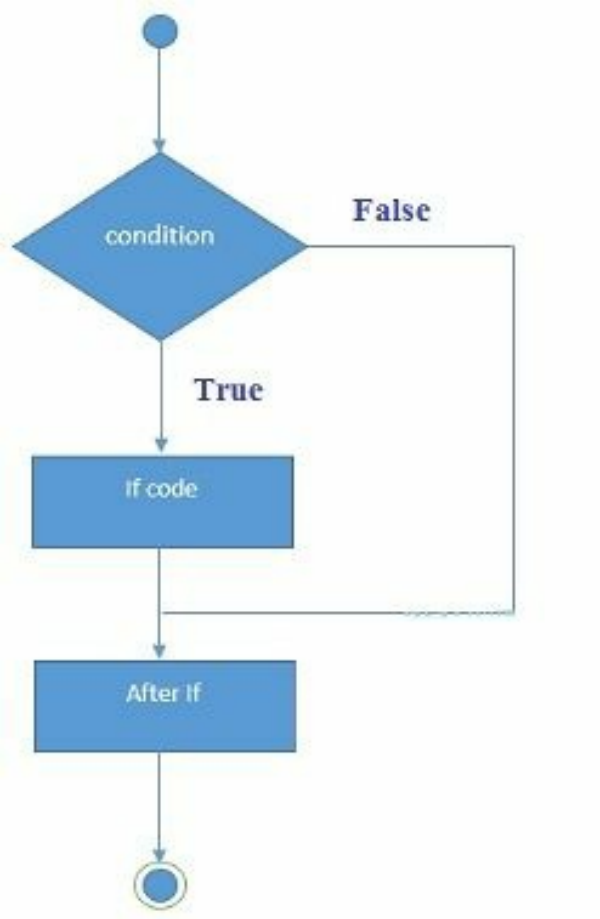
Java if Statement

The if statement in Java is used to evaluate the condition.

If the condition is valid, the if block is executed.

Syntax:

```
if(condition){
//code to be executed
}
```



Example:

```
//Java Program to demonstate the use of if statement.
public class IfExample {
public static void main(String[] args) {
   //defining an 'age' variable
   int age=20;
   //checking the age
   if(age>18){
      System.out.print("Age is greater than 18");
   }
}
}
```

Output:

```
Age is greater than 18
```

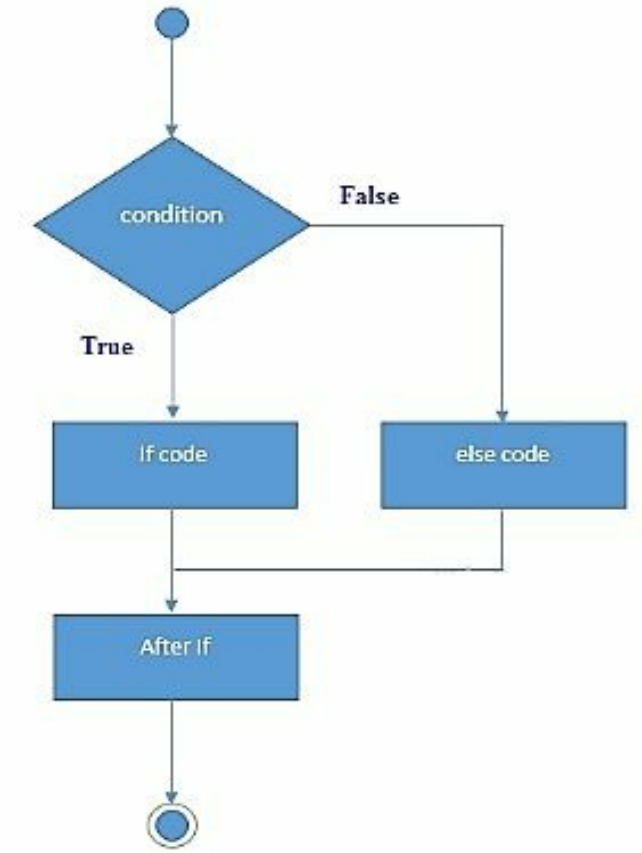
Java if-else Statement

The condition is also evaluated by the Java if-else argument.

If the condition is true, the if block is executed; otherwise, the else block is executed.

Syntax:

```
if(condition){
//code if condition is true
}else{
//code if condition is false
}
```



Example:

```
//A Java Program to demonstrate the use of if-else statement.
//It is a program of odd and even number.
public class IfElseExample {
public static void main(String[] args) {
   //defining a variable
   int number=13;
   //Check if the number is divisible by 2 or not
   if(number%2==0){
      System.out.println("even number");
   }else{
      System.out.println("odd number");
   }
}
}
```

Output:

```
odd number
```

Using Ternary Operator

The ternary operator (? :) can also be used to execute the if...else expression.

It's a fast way to evaluate the situation.

The outcome of? is returned if the condition is true.

The consequence of : is returned if the condition is incorrect.

Example:

```
public class IfElseTernaryExample {
public static void main(String[] args) {
    int number=13;
    //Using ternary operator
    String output=(number%2==0)?"even number":"odd number";
    System.out.println(output);
}
}
```

Output:

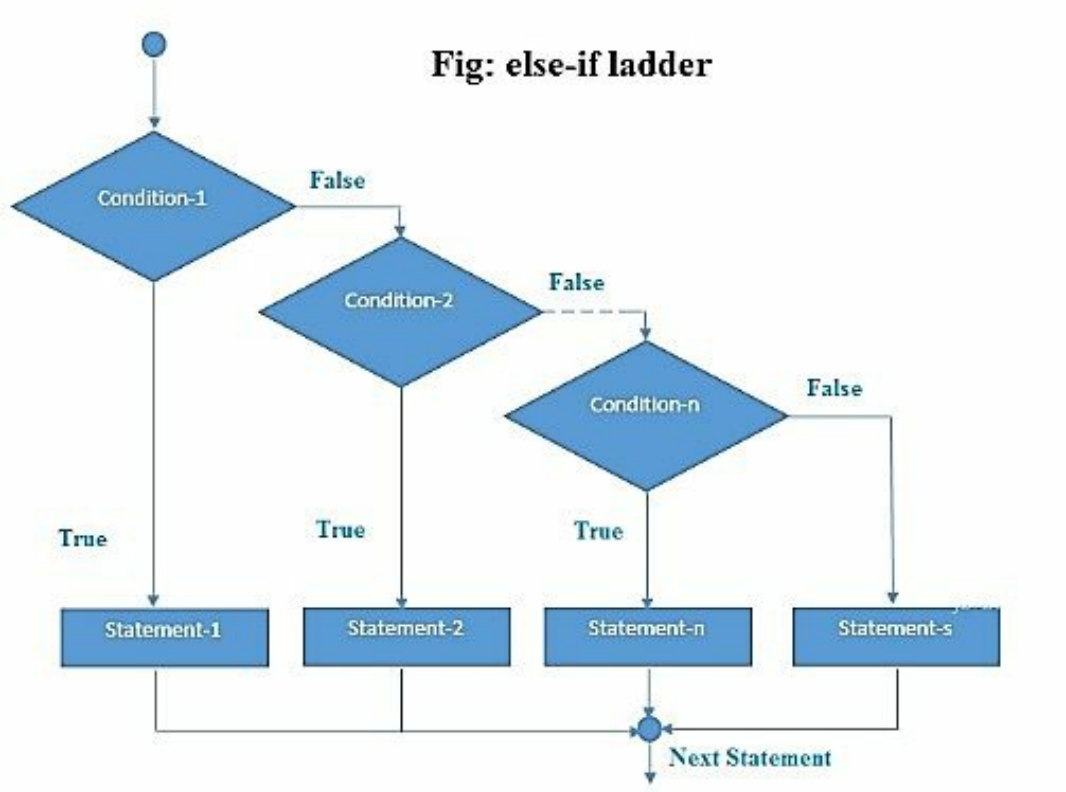
```
odd number
```

Java if-else-if ladder Statement

The if-else-if ladder statement incorporates several statements into one condition.

Syntax:

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```



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
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

Program to check POSITIVE, NEGATIVE or ZERO:

```
public class PositiveNegativeExample {
public static void main(String[] args) {
    int number=-13;
    if(number>0){
    System.out.println("POSITIVE");
    }else if(number<0){
    System.out.println("NEGATIVE");
```

```
    }else{
    System.out.println("ZERO");
   }
}
}
```

Output:

NEGATIVE

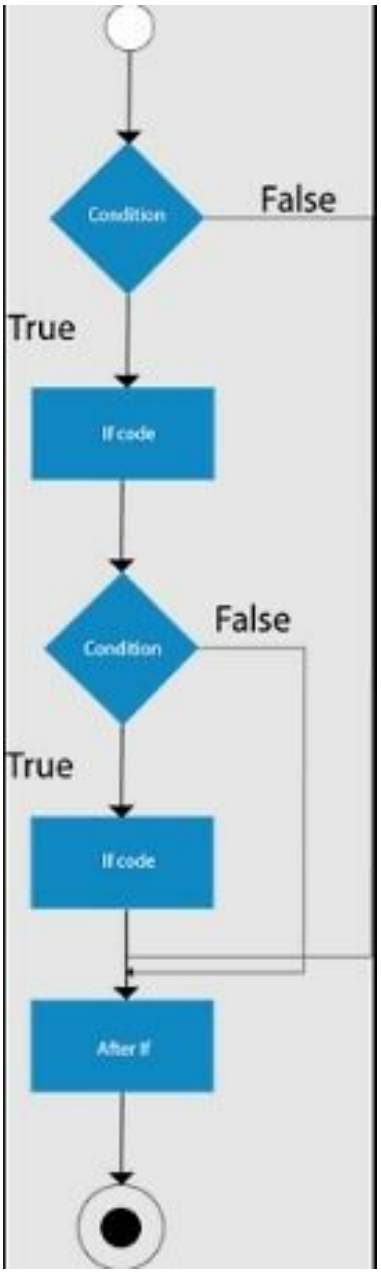
Java Nested if statement

The if block inside another if block is represented by the nested if statement.

Only when the outer if block condition is valid does the inner if block condition execute.

Syntax:

```
if(condition){
    //code to be executed
        if(condition){
          //code to be executed
   }
}
```



Example:

```
//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

Java Switch Statement

The Java switch statement incorporates several conditions into a single statement.

It's similar to an if-else-if conditional sentence.

Byte, short, int, long, enum types, String, and some wrapper types including Byte, Short, Int, and Long are all supported by the switch argument.

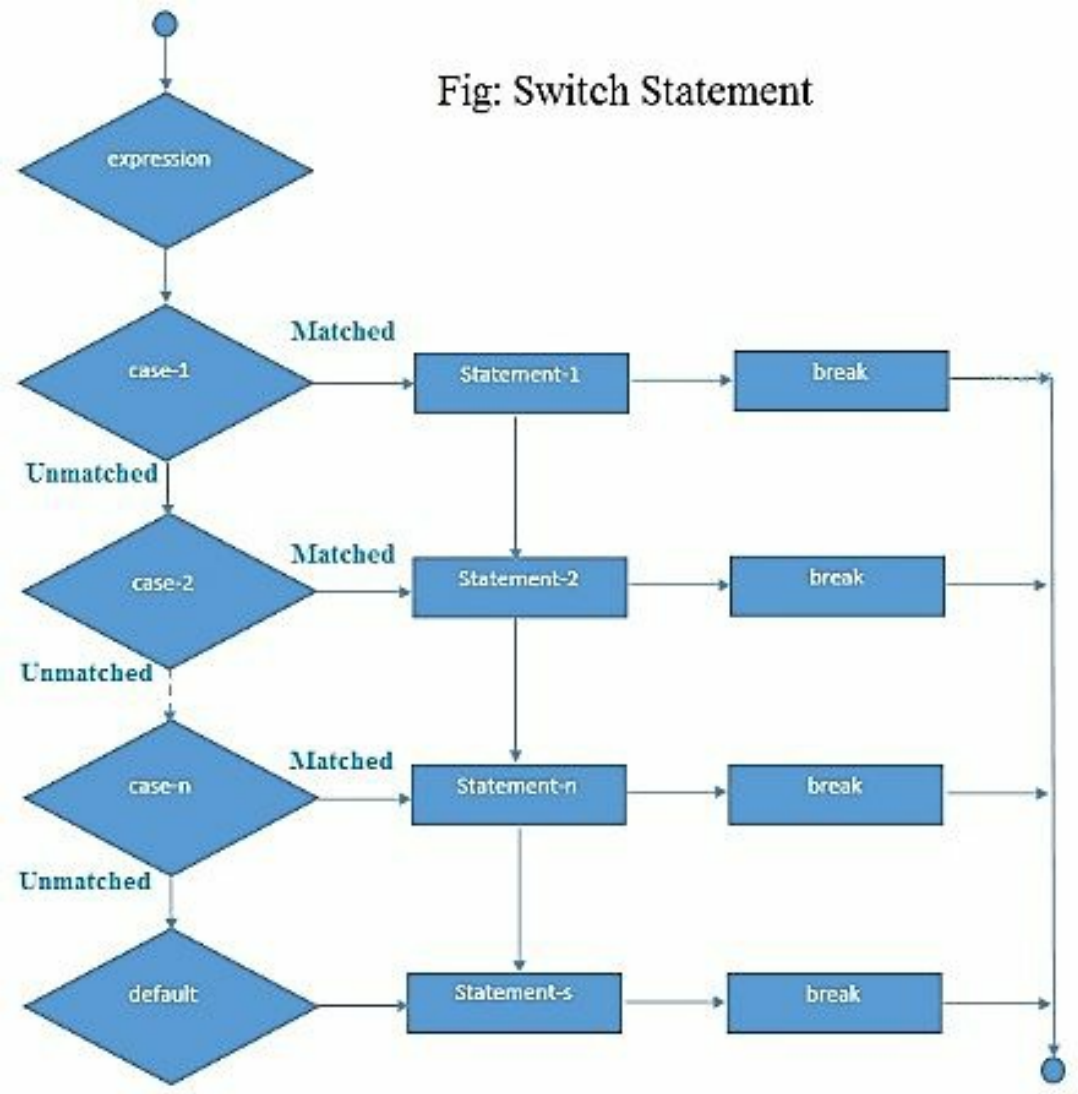
Strings can be included in the switch statement since Java 7.

In other words, the switch statement compares different values to see if they are equal.

Syntax:

```
switch(expression){
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
......

default:
 code to be executed if all cases are not matched;
}
```



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

KOTLIN CODING

EXAMPLES

PROGRAMMING FOR BEGINNERS

J KING

KOTLIN CODING EXAMPLES

Basic Programs in Kotlin

Kotlin program to perform arithmetic operations on two numbers

Example:

Input:

a = 13

b = 5

Output:

a + b = 18

a - b = 8

a * b = 65

a / b = 2

a % b = 3

PROGRAM

```
// Main Method Entry Point of Program
```

```
fun main(args:Array<String>){
    val a = 13
    val b = 5

    val sum  = a + b  // Perform Addition
    val sub  = a - b  // Perform Subtraction
    val muti = a * b  // Perform Multiplication
    val div  = a / b  // Perform Division
    val rem  = a % b  // Perform remainder

    // Print  on Console
    println("Addison of $a and $b is       : $sum")
    println("Subtraction of $a and $b is   : $sub")
    println("Multiplication of $a and $b is: $muti")
    println("Division of $a and $b is      : $div")
    println("Remainder of $a and $b is     : $rem")
}
```

Output

```
Addison of 13 and 5 is       : 18
Subtraction of 13 and 5 is   : 8
Multiplication of 13 and 5 is: 65
Division of 13 and 5 is      : 2
Remainder of 13 and 5 is     : 3
```

Program to perform simple calculation

Example:

Input:

first = 30

second = 15

choice = '+'

Output:

30 + 15 = 45

PROGRAM

```
/**
* Kotlin Program to Perform Simple Calculation on Two integer numbers
* User have to  ( +, -, *, /, % ) put any choice to perform operation
*/

package com.includehelp.basic

import java.util.*


// Main Method Entry Point of Program
fun main(args: Array<String>) {
```

```
// InputStream to get Input
val reader = Scanner(System.`in`)

//Input First Integer Value
print("Enter Integer Value : ")
var first = reader.nextInt()

//Input Second Integer Value
print("Enter Integer Value : ")
var second = reader.nextInt()

//Input Any Character as Choice
print("Enter any Action( +, -, *, /, % ) :  ")
val choice = reader.next()[0]

try{
    var result = when(choice){
        '+' -> first+second
        '-' -> first-second
        '*' -> first*second
        '/' -> first/second
        '%' -> first%second
        else -> {
            System.err.println("Not a Valid Operation choice")
            return
        }
    }
```

```
        //Print Result
        println("Result is : $result")
    }catch (E:Exception){
        System.err.println("Exception  : ${E.toString()}")
    }
}
```

Output

RUN 1:

Enter Integer Value : 35

Enter Integer Value : 8

Enter any Action(+, -, *, /, %) : %

Result is : 3

RUN 2:

Enter Integer Value : 456

Enter Integer Value : 67

Enter any Action(+, -, *, /, %) : /

Result is : 6

Program to input numbers (integer, float, double) at run time

PROGRAM

```
import java.util.*


// Main Method Entry Point of Program
fun main(args: Array<String>) {
    // InputStream to get Input
    var reader = Scanner(System.`in`)

    // Input Integer Value
    println("Enter Integer Value : ")
    val intValue = reader.nextInt()
    println("Integer Number is : $intValue")

    // Input Long Value
    println("Enter Long Value : ")
    val longValue = reader.nextLong()
    println("Long Number is : $longValue")
```

```
    // Input Float Value
    println("Enter Float Value : ")
    val floatValue = reader.nextFloat()
    println("Float Number is : $floatValue")

    // Input Double Value
    println("Enter Double Value : ")
    val doubleValue = reader.nextDouble()
    println("Double Number is : $doubleValue")
}
```

Output

```
Run 1:
Enter Integer Value :
123
Integer Number is : 123
Enter Long Value :
34355566
Long Number is : 34355566
Enter Float Value :
45.34
Float Number is : 45.34
Enter Double Value :
```

456.78

Double Number is : 456.78

Kotlin program to find largest of three numbers

Example:

Input:

First number: 10

Second number: 20

Third number: 30

Output:

Largest number: 30

PROGRAM

```
package com.includehelp.basic

import java.util.*

// Main Method Entry Point of Program
fun main(args: Array<String>) {
    // InputStream to get Input
    val reader = Scanner(System.`in`)

    // Input Integer values
    println("Enter First Value : ")
    var first = reader.nextInt()
```

```
    println("Enter Second Value : ")
    var second = reader.nextInt()
    println("Enter Third Value : ")
    var third = reader.nextInt();

    // condition to find largest of three numbers
    val largest =   if(first>second && first>third)
                        first
                    else if(second>third)
                        second
                    else
                        third

    println("Largest Number is: $largest")
}
```

Output

Enter First Value :

23

Enter Second Value :

45

Enter Third Value :

89

Largest Number is: 89

Kotlin program to swap two numbers

Example:

Input:

First number: 10

Second number: 20

Output:

First number: 20

Second number: 10

PROGRAM

```
import java.util.*

// Main Method Entry Point of Program
fun main(arg: Array<String>) {
    // InputStream to get Input
    var reader = Scanner(System.`in`)

    // Input two values
    println("Enter First Value : ")
    var first = reader.nextInt();
    println("Enter Second Value : ")
    var second = reader.nextInt();
```

```
    println("Numbers Before Swap : \n first = $first \n second = $second ")

    //Code for Swap Numbers
    var temp = first
    first=second
    second=temp

    println("Numbers After  Swap : \n first = $first \n second = $second ")
}
```

Output

```
Run 1:
Enter First Value :
45
Enter Second Value :
12
Numbers Before Swap :
first = 45
second = 12
Numbers After  Swap :
first = 12
second = 45
```

Kotlin program to check whether a number is EVEN or ODD

Example:

Input:

N = 13

Output:

"ODD"

Input:

N = 24

Output:

"EVEN"

PROGRAM

```
/*
 * Kotlin program to Input Integer
 * Numbers and check Number is Even or ODD
*/
```

```
import java.util.*

// Main Method Entry Point of Program
fun main(args: Array<String>) {
    // InputStream to get Input
    val reader = Scanner(System.`in`)

    //Input Integer Value
    println("Enter Integer Value : ")
    var number = reader.nextInt();

    val str = if(number%2==0) "EVEN" else "ODD"

    println("Number is $str")
}
```

Output

```
RUN 1:
Enter Integer Value :
34
Number is EVEN
---
Run 2:
```

Enter Integer Value :

15

Number is ODD

Kotlin program to print the Multiplication table

PROGRAM

```
import java.util.*

// Main function entry point of program
fun main(args: Array<String>) {

    val sc = Scanner(System.`in`)
    println("Enter Number: ")
    val num: Int = sc.nextInt()

    // for loop to print multiplication table of given number
    for (i in 1..10) {
        val result = num * i
        println("$num*$i = $result")
    }
}
```

Output

Enter Number:

5

5*1 = 5

5*2 = 10

5*3 = 15

5*4 = 20

5*5 = 25

5*6 = 30

5*7 = 35

5*8 = 40

5*9 = 45

5*10 = 50

Program to calculate and display student grades

Percentage >=80 -> A

Percentage >=60 -> B

Percentage r>=40 -> C

else D

Note: Maximum Marks 100 in Each Subject

Example:

Input:

English = 98

Physics = 34

Chemistry = 67

Maths = 90

Output:

Total Marks = 356.0

Percentage = 71.2

Grade = B

PROGRAM

```
import java.util.*

//Main Function , Entry point of Program
fun main(args: Array<String>) {

    //array of subjects Names
    val subjectName = arrayOf<String>
("English","Physics","Chemistry","Maths")

    //Input Stream
    val scanner = Scanner(System.`in`)

    //Declare Array to Contain Subjects marks
    val marksArray = DoubleArray(5)

    //Start Input Subjects Marks
    println("Input Marks->")
    for(i in marksArray.indices){
        print("${subjectName[i]} : ")
        marksArray[i] = scanner.nextDouble()
    }
```

```
    //Calculate Total Marks in All Subjects
    val total = marksArray.sum()

    //Calculate Percentage
    val percentage = marksArray.average()

    //Print Total and Percentage
    println("Total of All subjects Marks : $total")
    println("Percentage : $percentage")


    //To find out Grade based on Percentage
    when{
        percentage>80 -> println("Grade : A")
        percentage>60 -> println("Grade : B")
        percentage>40 -> println("Grade : C")
        else -> println("Grade : D")
    }
}
```

Output

Run 1:

English : 98

Physics : 42

Chemistry : 67

Maths : 90

Total of All subjects Marks : 297

Percentage : 74.25

Grade : B

Kotlin program to convert temperature

Example:

Input:

Fahrenheit = 67

Output:

Celsius = 19.444444444444443

PROGRAM

```
package com.includehelp

import java.util.*

//Main Function , Entry point of Program
fun main(args: Array<String>) {

    //Input Stream
    val scanner = Scanner(System.`in`)

    //Input temperature in Fahrenheit
    print("Enter temperature into Fahrenheit : ")
    val fahrenheit = scanner.nextDouble()
```

```
    //Convert  Fahrenheit to Celsius
    val celsius =( (fahrenheit  -  32  ) *  5)/9

    //Print temperature in Celsius
    println("Temperature in Fahrenheit ($fahrenheit) = Celsius ($celsius)")
}
```

Output

Output will be:

Run 1:

Enter temperature into Fahrenheit : 67

Temperature in Fahrenheit (67.0) = Celsius (19.444444444444443)

Run 2:

Enter temperature into Fahrenheit : 78

Temperature in Fahrenheit (78.0) = Celsius (25.555555555555557)

Kotlin program to display Fibonacci series

Example:

Input:

term1 = 0

term2 = 1

N = 10

Output:

Fibonacci series: 0 1 1 2 3 5 8 13 21 34

Input:

term1 = 0

term2 = 1

N = 5

Output:

Fibonacci series: 0 1 1 2 3

PROGRAM

```
/**
        * Display Fibonacci Series up to a Given number of terms
        * e.g.  0 1 1 2 3 5 8 13....n
*/

import java.util.*

//Main Function entry Point of Program
fun main(args: Array<String>) {
        // Input Stream
        val scanner = Scanner(System.`in`)

        // input total number of terms
        println("Enter terms  : ")
        val n: Int = scanner.nextInt()

        var term1 = 0
        var term2 = 1
        var count = 1

        // Iterate Loop to print fibonacci Series upto given terms
        while (count <= n){
        print("$term1 ")
        val s = term1+term2
        term1 = term2;
        term2 = s
        count++
        }
```

}

Output

RUN 1:

Enter terms :

10

0 1 1 2 3 5 8 13 21 34

Run 2:

Enter terms :

5

0 1 1 2 3

Kotlin program to find area of circle

Example:

Input:

Radius: 3

Output:

Area of circle: 28.259999999999998

PROGRAM

```
import java.util.*

// PI Constant Value
val PI = 3.14

/* function to calculate area of circle of given radius */
fun getAreaOfCirlce(radius: Double): Double {
    return PI * radius * radius
}

// Main Method Entry Point of Program
fun main(args:Array<String>){
    val sc = Scanner(System.`in`)
```

```
    // Input Radius
    println("Enter Circle Radius  : ")
    val radius = sc.nextDouble()

    // Call function to get Area of Circle
    val areaOfCircle = getAreaOfCirlce(radius)

    // Print Area of Circle
    println("Area of Circle : $areaOfCircle")
}
```

Output

Run 1:

Enter Circle Radius :

3

Area of Circle : 28.259999999999998

Run 2:

Enter Circle Radius :

8

Area of Circle : 200.96

Kotlin program to find area of Square

Example:

Input:

side = 6

Output:

area = 36.0

PROGRAM

```
import java.util.*

//Main Function , Entry point of Program
fun main(args: Array<String>) {

    //Input Stream
    val scanner = Scanner(System.`in`)

    //Input Side of Square
    print("Enter Side of Square : ")
    val side = scanner.nextDouble()

    //Calculate Area of square
    val squareArea = side*side;
```

```
    //Print Area
    println("Area of Square is : $squareArea")
}
```

Output

Run 1:

Enter Side of Square : 6

Area of Square is : 36.0

Run 2:

Enter Side of Square : 4.5

Area of Square is : 20.25

Program to find area of a cylinder

Example:

Input:

Radius = 5

Height = 7

Output:

Surface Area of Cylinder is :75.39822368615503

PROGRAM

```
package com.includehelp

import java.util.*

//Main Function , Entry point of Program
fun main(args: Array<String>) {

    //Input Stream
    val scanner = Scanner(System.`in`)

    //Input Radius
    print("Enter Radius of Cylinder : ")
    val radius = scanner.nextDouble()

    //Input Height
```

```
    print("Enter Height of Cylinder : ")
    val height = scanner.nextDouble()

    //Cylinder Surface Area
    val areaCylinder = 2*Math.PI*(radius+height)

    //Print Area
    println("Surface Area of Cylinder is :$areaCylinder")
}
```

Output

Run 1:

Enter Radius of Cylinder : 5

Enter Height of Cylinder : 7

Surface Area of Cylinder is :75.39822368615503

Run 2:

Enter Radius of Cylinder : 29

Enter Height of Cylinder : 4

Surface Area of Cylinder is :207.34511513692635

Kotlin program to generate 4 digits OTP

PROGRAM

```
/**
   * Kotlin Program to Generate 4 digit OTP
*/
/**
   * Function for Generate 4 digit OTP String
   * @return
*/
fun generateOTP(): String {
   val randomPin = (Math.random() * 9000).toInt() + 1000
   return randomPin.toString()
}

// Main Method Entry Point of Program
fun main(args: Array<String>) {
   // Call function for Generate OTP
   val otp1 = generateOTP()
   // Print OTP
   println("OTP : $otp1")
}
```

Output

Run 1:

OTP : 7997

Run 2:

OTP : 7682

Run 3:

OTP : 6934

Run 4:

OTP : 4189

Kotlin program to find area of Pentagon

Formula to find area of Pentagon: **= (5/2)sa**, where

s - Side of Pentagon

a - Apothem of Pentagon

Example:

Input:

side= 5

apothem = 6

Output:

area = 75.0

PROGRAM

```
import java.util.*

//Main Function , Entry point of Program
fun main(args: Array<String>) {

    //Input Stream
    val scanner = Scanner(System.`in`)
```

```
    //Input Side
    print("Enter Side of Pentagon : ")
    val s = scanner.nextDouble()

    //Input Apothem
    print("Enter Apothem of Pentagon : ")
    val a = scanner.nextDouble()

    //Area of Pentagon
    val areaPentagon = (5.0/2.0)*s*a

    //Print Area
    println("Pentagon Area is :$areaPentagon")
}
```

Output

Run 1:

Run 1:

Enter Side of Pentagon : 5

Enter Apothem of Pentagon : 6

Pentagon Area is :75.0

Run 2:

Enter Side of Pentagon : 8

Enter Apothem of Pentagon : 6

Pentagon Area is :120.0

Kotlin program to find prime numbers

Example:

Input:

start = 1

end = 100

Output:

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29,

31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]

PROGRAM

```
/**
* Kotlin Program to find out Prime Numbers between
* given Range(include START and END)
* A prime number is a whole number greater than 1
* whose only factors are 1 and itself.
* e.g 7, 11, 13, 17
*/

import java.util.*


//Function to check Prime Number
```

```
fun findPrimeNo(number: Long): Boolean {
    if(number<2) return false
    for (i in 2.toLong()..number/2) {
        if (number % i == 0.toLong()) {
            return false
        }
    }
    return true
}

//Main Function, Entry Point of Program
fun main(arg: Array<String>) {
    //Input Stream
    val sc = Scanner(System.`in`)

    //Input Start of Range
    print("Enter Start of Range  : ")
    val start: Long = sc.nextLong()

    //Input End of Range
    print("Enter End of Range  : ")
    val end: Long = sc.nextLong()

    //Declare Mutable List to hold factors
    val list: MutableList<Long> = ArrayList()

    //iterate through loop start to end to find Prime  number in Range
    for (i in start..end) {
```

```
        if (findPrimeNo(i)) {
            list.add(i)
        }
    }

    println("Prime Numbers from $start to $end  : $list")
}
```

Kotlin program to convert decimal to binary

Example:

Input:

Decimal number = 12

Recursive division:

Number	Quotient	Remainder
12	6	0
6	3	0
3	1	1
1	0	1

Read the remainders in reverse order

1100 is the **binary conversion** for **12**.

PROGRAM

```
import java.util.*

/* function to convert given decimal number into Binary */
fun getBinaryNumber(decimalNumber: Int): String {
    var decimalNumber = decimalNumber
```

```
    val binaryStr = StringBuilder()

    while (decimalNumber > 0) {
        val r = decimalNumber % 2
        decimalNumber /= 2
        binaryStr.append(r)
    }

    return binaryStr.reverse().toString()
}

// Main Method Entry Point of Program
fun main(arg: Array<String>) {
    val sc = Scanner(System.`in`)

    println("Enter Decimal Number  : ")
    //Input Decimal Number
    val decimalNumber: Int = sc.nextInt()

    // Call function to Convert Decimal into binary
    val binaryNumber = getBinaryNumber(decimalNumber)
    // Print Binary Number
    println("Binary Number : $binaryNumber")
}
```

Output

Run 1:

Enter Decimal Number :

15

Binary Number : 1111

Run 2:

Enter Decimal Number :

12345

Binary Number : 11000000111001

Run 3:

Enter Decimal Number :

545654

Binary Number : 10000101001101110110

OUR OTHER PUBLISHING

TAM SEL

APACHE SPARK AND SCALA FOR BEGINNERS: 2 BOOKS IN 1 - Learn Coding Fast!

PYTHON AND JAVASCRIPT CODING BASICS: FOR ABSOLUTE BEGINNERS

LARAVEL FOR BEGINNERS: Learn Coding Fast

HTML FOR BEGINNERS: Html CSS Programming Language Crash Course, Web Design Quick Start Script Tutorial Book in Easy Steps

DART PROGRAMMING BASICS: FOR ABSOLUTE BEGINNERS

RUST AND GOLANG FOR BEGINNERS: 2 BOOKS IN 1 - Learn Coding Fast! RUST AND GOLANG

APACHE SPARK AND HADOOP FOR BEGINNERS: 2 BOOKS IN 1 - Learn Coding Fast!

TENSORFLOW FOR BEGINNERS: LEARN CODING FAST

DJANGO AND PYTHON FOR BEGINNERS: 2 BOOKS IN 1 - Learn Coding Fast!

JAVASCRIPT AND DART CODING BASICS: FOR ABSOLUTE BEGINNER

TENSORFLOW FOR BEGINNERS AND PYTHON CODING BASICS: FOR ABSOLUTE BEGINNERS

RUST Programming Language: For Beginners, Learn Coding Fast!

HADOOP AND PYTHON FOR BEGINNERS: 2 BOOKS IN 1 - Learn Coding Fast!

SELENIUM AND JAVA FOR BEGINNERS: 2 BOOKS IN 1 - Learn Coding Fast!

ANDROID AND JAVA FOR BEGINNERS: 2 BOOKS IN 1 - Learn Coding Fast!

J KING

PYTHON AND HADOOP BASICS

PYTHON BASICS AND PYTHON CODING EXAMPLES

C# AND C++ CODING EXAMPLES

PYTHON CODING AND C PROGRAMMING EXAMPLES

C++ AND JAVA CODING EXAMPLES

ANDROID BASICS AND PYTHON CODING EXAMPLES

C# BASICS AND PYTHON CODING EXAMPLES

ANGULAR AND PYTHON BASICS

PYTHON BASICS AND SCALA CODING EXAMPLES

JAVA BASICS AND PYTHON CODING EXAMPLES

PYTHON BASICS AND C# CODING EXAMPLES

ANDROID BASICS AND JAVA CODING EXAMPLES

C# BASICS AND C# CODING EXAMPLES

JAVASCRIPT AND HTML BASICS

HTML AND JAVASCRIPT CODING EXAMPLES