

Cryptography in Practice

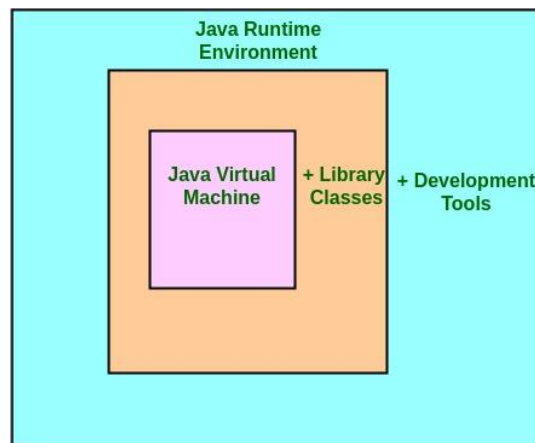
Dr. Ayad Ibrahim
2019-2020

Principles of Java



- Java is one of the most **popular** programming languages.
- Java is **Object Oriented**.
- The Java codes are first compiled into **byte code** (machine independent code).
 - Then the byte code is run on **Java Virtual Machine (JVM)** regardless of the underlying architecture.
- Java syntax is similar to C/C++.
- Java **does not** provide low level programming functionalities like **pointers**.
- Java codes are always written in the form of **classes** and **objects**.
- Java is used in all kind of applications like **Mobile Applications** (Android is Java based), desktop applications, web applications, client server applications, enterprise applications and many more.

Java environment



JDK = JRE + Development Tool
JRE = JVM + Library Classes

JVM, JRE and JDK all three are platform dependent because configuration of each Operating System is different. But, Java is platform independent.

1. **JDK**(Java Development Kit) : JDK is intended for software developers and includes development tools such as the Java compiler, Javadoc, Jar, and a debugger.
2. **JRE**(Java Runtime Environment) : JRE contains the parts of the Java libraries required to run Java programs and is intended for end users.
3. **JVM**: JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed. JVMs are available for many hardware and software platforms.

Steps for Setting up Java Environment for Windows

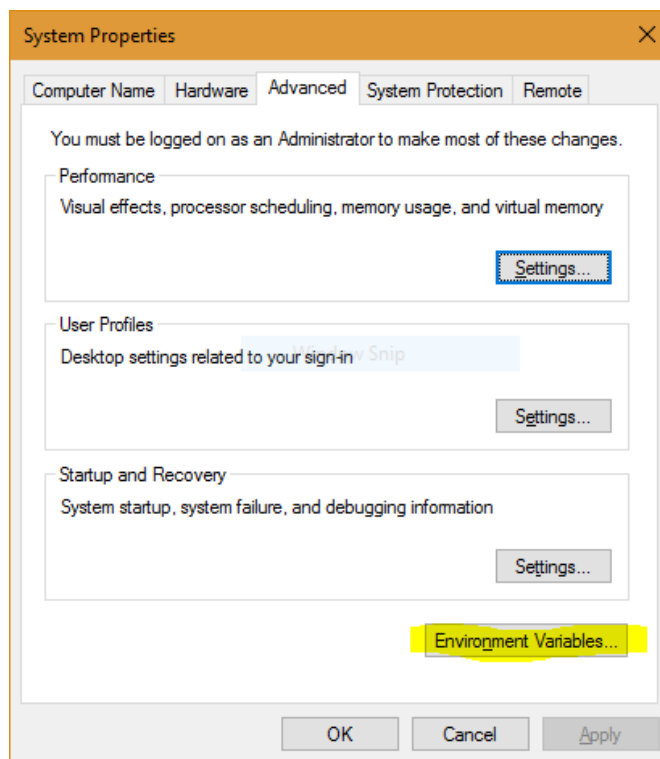
1- Java8 JDK is available at:

- <https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

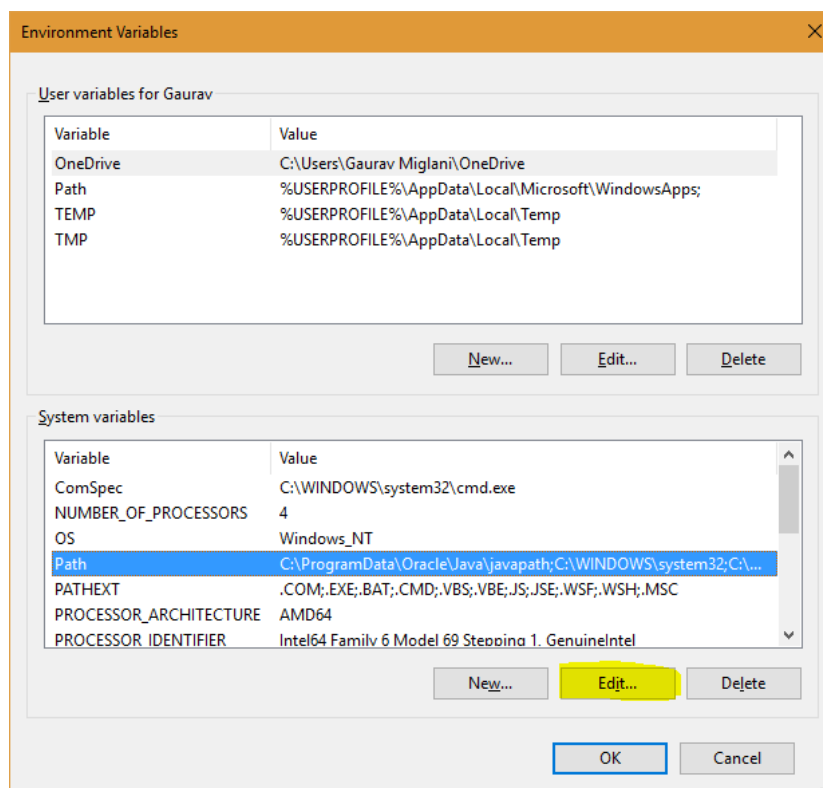
Click second last link for Windows(32 bit) and last link for Windows(64 bit) as highlighted below.

Java SE Development Kit 8u121		
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.		
<input type="radio"/> Accept License Agreement <input checked="" type="radio"/> Decline License Agreement		
Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.86 MB	jdk-8u121-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.83 MB	jdk-8u121-linux-arm64-vfp-hflt.tar.gz
Linux x86	162.41 MB	jdk-8u121-linux-i586.rpm
Linux x86	177.13 MB	jdk-8u121-linux-i586.tar.gz
Linux x64	159.96 MB	jdk-8u121-linux-x64.rpm
Linux x64	174.76 MB	jdk-8u121-linux-x64.tar.gz
Mac OS X	223.21 MB	jdk-8u121-macosx-x64.dmg
Solaris SPARC 64-bit	139.64 MB	jdk-8u121-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.07 MB	jdk-8u121-solaris-sparcv9.tar.gz
Solaris x64	140.42 MB	jdk-8u121-solaris-x64.tar.Z
Solaris x64	96.9 MB	jdk-8u121-solaris-x64.tar.gz
Windows x86	189.36 MB	jdk-8u121-windows-i586.exe
Windows x64	195.51 MB	jdk-8u121-windows-x64.exe

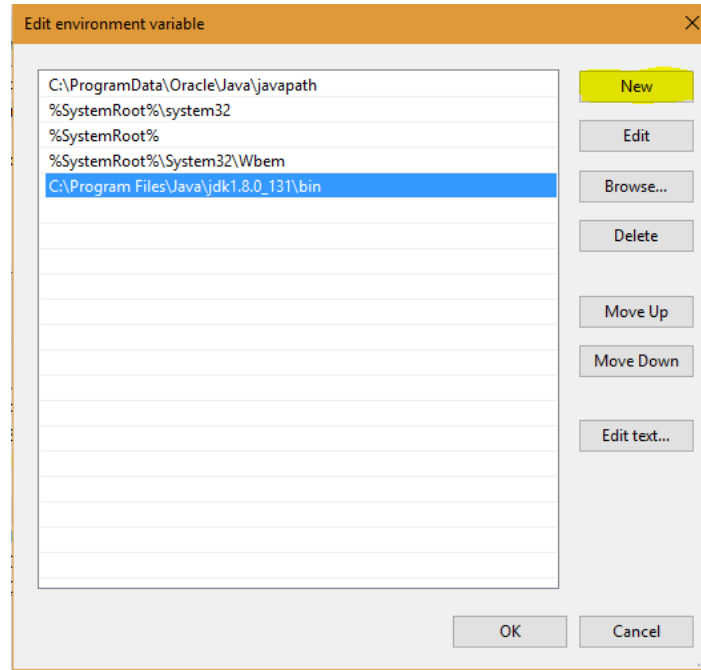
- 2- After download, run the .exe file and follow the instructions to install Java on your machine. Once you installed Java on your machine, you have to setup environment variable.
- 3- Go to **Control Panel -> System and Security -> System**. Under Advanced System Setting option click on **Environment Variables** as highlighted below.



- 4- Now, you have to alter the “**Path**” variable under System variables so that it also contains the path to the Java environment. Select the “Path” variable and click on Edit button as highlighted below.



- 5- You will see list of different paths, click on New button and then add path where java is installed. By default, java is installed in “C:\Program Files\Java\jdk\bin” folder OR “C:\Program Files(x86)\Java\jdk\bin”. In case, you have installed java at any other location, then add that path.



6. Click on OK, Save the settings and you are done !! Now to check whether installation is done correctly, open command prompt and type *javac -version*. You will see that java is running on your machine.
7. In order to make sure whether compiler is setup, type *javac* in command prompt. You will see a list related to javac.

Beginning Java programming

Create the program by typing it into a text editor and saving it to a file – HelloWorld.java.

```
/* This is a simple Java program.
   FileName : "HelloWorld.java". */
class HelloWorld
{
    // Your program begins with a call to main().
    // Prints "Hello, World" to the terminal window.
    public static void main(String args[])
    {
        System.out.println("Hello, World");
    }
}
```

Java Identifiers

In Java, an **identifier** can be a class name, method name, variable name or a label.

For example :

```
public class Test
{
    public static void main(String[] args)
    {
        int a = 20;
    }
}
```

In the above java code, we have 5 identifiers:

- **Test** : class name.
- **main** : method name.
- **String** : predefined class name.
- **args** : variable name.
- **a** : variable name.

Rules for defining Java Identifiers

- The only allowed characters for identifiers are all alphanumeric characters([**A-Z**],[**a-z**],[**0-9**]), '\$'(dollar sign) and '_' (underscore).
 - For example “geek@” is not a valid java identifier as it contain '@' special character.
- Identifiers should **not** start with digits([**0-9**]).
 - For example “123geeks” is a not a valid java identifier.
- Java identifiers are **case-sensitive**. i.e. I <> i
- There is no limit on the length of the identifier but it is advisable to use an optimum length of 4 – 15 letters only.
- **Reserved Words** can't be used as an identifier.
 - For example “int while = 20;” is an invalid statement as while is a reserved word.
 - There are **53** reserved words in Java.

Important Points:

- The name of the **class** defined by the program is HelloWorld, **should be** the same name of **file**(HelloWorld.java).
- In Java, all codes must reside inside a class and there is at most one public class which contain main() method.

Java Class file

- A **Java class file** is a file containing Java bytecode and having **.class extension** that can be executed by **JVM**.
- A Java class file is created by a Java compiler from *.java* files as a result of successful compilation.
- if a *.java* file has more than one class then each class will compile into a separate class files.

For Example: Test.java

```
// Compiling this Java program would
// result in multiple class files.

class Sample
{

}

// Class Declaration
class Student
{

}

// Class Declaration
class Test
{
    public static void main(String[] args)
    {
        System.out.println("Class File Structure");
    }
}
```

After compilation there will be **3 class** files in corresponding folder named as:

- Sample.class
- Student.class
- Test.class

Comments in Java

Single – line comments.

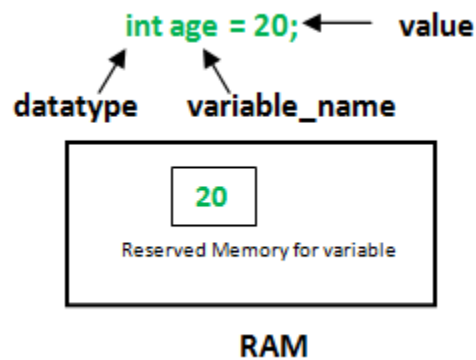
```
//Java program to show single line comments
class Scomment
{
    public static void main(String args[])
    {
        // Single line comment here
        System.out.println("Single line comment above");
    }
}
```

Multi – line comments.

```
//Java program to show multi line comments
class Scomment
{
    public static void main(String args[])
    {
        System.out.println("Multi line comments below");
        /*Comment line 1
        Comment line 2
        Comment line 3*/
    }
}
```

Variables in Java

A variable is a name given to a memory location. It is the basic unit of storage in a program.



Types of variables

There are three types of variables in Java:

- **Local Variables**
- **Instance Variables**
- **Static Variables**

Local Variables: A variable defined within a block or method or constructor is called local variable. These variable are created when the block is entered or the function is called and destroyed after exiting from the block or when the call returns from the function.


```

public class StudentDetails {
    public void StudentAge()
    {
        // local variable age
        int age = 0;
        age = age + 5;
        System.out.println("Student age is : " + age);
    }

    public static void main(String args[])
    {
        StudentDetails obj = new StudentDetails();
        obj.StudentAge();
    }
}

```

Instance Variables: Instance variables are **non-static** variables and are declared in a class outside any method, constructor or block.

- These variables are created when an object of the class is created and destroyed when the object is destroyed.
- We may use access specifiers for instance variables.
- Initilisation of Instance Variable is not Mandatory. Its default value is 0
- Instance Variable can be accessed only by creating objects.

```

import java.io.*;
class Marks {
    // These variables are instance variables.
    // These variables are in a class
    // and are not inside any function
    int engMarks;
    int mathsMarks;
    int phyMarks;
}

class MarksDemo {
    public static void main(String args[])
    {
        // first object
        Marks obj1 = new Marks();
        obj1.engMarks = 50;
        obj1.mathsMarks = 80;
        obj1.phyMarks = 90;

        // second object
        Marks obj2 = new Marks();
        obj2.engMarks = 80;
        obj2.mathsMarks = 60;
        obj2.phyMarks = 85;

        // displaying marks for first object
        System.out.println("Marks for first object:");
        System.out.println(obj1.engMarks);
        System.out.println(obj1.mathsMarks);
        System.out.println(obj1.phyMarks);

        // displaying marks for second object
        System.out.println("Marks for second object:");
        System.out.println(obj2.engMarks);
        System.out.println(obj2.mathsMarks);
        System.out.println(obj2.phyMarks);
    }
}

```

3. Static Variables: Static variables are also known as Class variables.

- These variables are declared using the **static keyword** within a class outside any method constructor or block.
- We can only have one copy of a static variable per class irrespective of how many objects we create.
- Static variables are created at the start of program execution and destroyed automatically when execution ends.
- Initialisation of Static Variable is not Mandatory. Its default value is 0

```
import java.io.*;
class Emp {

    // static variable salary
    public static double salary;
    public static String name = "Harsh";
}

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

        // accessing static variable without object
        Emp.salary = 1000;
        System.out.println(Emp.name + "'s average salary:"
                           + Emp.salary);
    }
}
```

Notes:

- Each object will have its **own copy** of **instance** variable whereas We can only have **one copy** of a static variable per class irrespective of how many objects we create.
- Changes made in an **instance** variable using one object will **not be reflected** in other objects as each object has its own copy of instance variable. In case of static, changes **will be reflected** in other objects as static variables are common to all object of a class.

Data types in Java

There are 8 primitive data types

TYPE	DESCRIPTION	DEFAULT	SIZE	EXAMPLE LITERALS
boolean	true or false	false	1 bit	true, false
byte	twos complement integer	0	8 bits	(none)
char	unicode character	\u0000	16 bits	'a', '\u0041', '\101', '\', '\', '\n', '\b'
short	twos complement integer	0	16 bits	(none)
int	twos complement integer	0	32 bits	-2, -1, 0, 1, 2
long	twos complement integer	0	64 bits	-2L, -1L, 0L, 1L, 2L
float	IEEE 754 floating point	0.0	32 bits	1.23e100f, -1.23e-100f, .3f, 3.14F
double	IEEE 754 floating point	0.0	64 bits	1.23456e300d, -1.23456e-300d, 1e1d

boolean: boolean data type represents only one bit of information **either true or false** .

byte: The byte data type is an 8-bit signed two's complement integer. The byte data type is useful for saving memory in large arrays.

- Size: 8-bit
- Value: -128 to 127

short: The short data type is a 16-bit signed two's complement integer. Similar to byte, use a short to save memory in large arrays, in situations where the memory savings actually matters.

- **Size:** 16 bit
- **Value:** -32,768 to 32,767 (inclusive)

int

It is a 32-bit signed two's complement integer.

- **Size:** 32 bit
- **Value:** -2^{31} to $2^{31}-1$

long: The long data type is a 64-bit two's complement integer.

- Size: 64 bit
- Value: -2^{63} to $2^{63}-1$.

float: The float data type is a single-precision 32-bit [IEEE 754](#) floating point. Use a float (instead of double) if you need to save memory in large arrays of floating point numbers.

- **Size:** 32 bits
- **Suffix :** F/f Example: 9.8f

double: The double data type is a double-precision 64-bit IEEE 754 floating point. For decimal values, this data type is generally the default choice.

Char: The char data type is a single 16-bit Unicode character. A char is a single character.

- Value: '\u0000' (or 0) to '\uffff' 65535

```
class GeeksforGeeks
{
    public static void main(String args[])
    {
        // declaring character
        char a = 'G';

        // Integer data type is generally
        // used for numeric values
        int i=89;

        // use byte and short if memory is a constraint
        byte b = 4;

        // this will give error as number is
        // larger than byte range
        // byte b1 = 7888888955;

        short s = 56;

        // this will give error as number is
        // larger than short range
        // short s1 = 87878787878;

        // by default fraction value is double in java
        double d = 4.355453532;

        // for float use 'f' as suffix
        float f = 4.7333434f;

        System.out.println("char: " + a);
        System.out.println("integer: " + i);
        System.out.println("byte: " + b);
        System.out.println("short: " + s);
        System.out.println("float: " + f);
        System.out.println("double: " + d);
    }
}
```

Operators in Java

Java provides many types of operators which can be used according to the need. They are classified based on the functionality they provide. Some of the types are-

1. Arithmetic Operators
2. Unary Operators
3. Assignment Operator
4. Relational Operators
5. Logical Operators
6. Ternary Operator
7. Bitwise Operators

Arithmetic Operators: They are used to perform simple arithmetic operations on primitive data types.

- *: Multiplication
- /: Division
- %: Modulo
- +: Addition
- -: Subtraction

```
// Java program to illustrate
// arithmetic operators
public class operators {
    public static void main(String[] args)
    {
        int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
        String x = "Thank", y = "You";

        // + and - operator
        System.out.println("a + b = " + (a + b));
        System.out.println("a - b = " + (a - b));

        // + operator if used with strings
        // concatenates the given strings.
        System.out.println("x + y = " + x + y);

        // * and / operator
        System.out.println("a * b = " + (a * b));
        System.out.println("a / b = " + (a / b));

        // modulo operator gives remainder
        // on dividing first operand with second
        System.out.println("a % b = " + (a % b));

        // if denominator is 0 in division
        // then Arithmetic exception is thrown.
        // uncommenting below line would throw
        // an exception
        // System.out.println(a/c);
    }
}
```

2. Unary Operators: Unary operators need only one operand. They are used to increment, decrement or negate a value.

- **- :Unary minus**, used for negating the values.
- **+ :Unary plus**, used for giving positive values.
- **++ :Increment operator**, used for incrementing the value by 1. There are two varieties of increment operator.
 - **Post-Increment** : Value is first used for computing the result and then incremented.
 - **Pre-Increment** : Value is incremented first and then result is computed.
- **-- : Decrement operator**, used for decrementing the value by 1. There are two varieties of decrement operator.
 - **Post-decrement** : Value is first used for computing the result and then decremented.
 - **Pre-Decrement** : Value is decremented first and then result is computed.
- **! : Logical not operator**, used for inverting a boolean value.

```
// Java program to illustrate
// unary operators
public class operators {
    public static void main(String[] args)
    {
        int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
        boolean condition = true;

        // pre-increment operator
        // a = a+1 and then c = a;
        c = ++a;
        System.out.println("Value of c (++a) = " + c);

        // post increment operator
        // c=b then b=b+1
        c = b++;
        System.out.println("Value of c (b++) = " + c);

        // pre-decrement operator
        // d=d-1 then c=d
        c = --d;
        System.out.println("Value of c (--d) = " + c);

        // post-decrement operator
        // c=e then e=e-1
        c = e--;
        System.out.println("Value of c (e--) = " + c);

        // Logical not operator
        System.out.println("Value of !condition ="
                           + !condition);
    }
}
```

3. Assignment Operator : '=' Assignment operator is used to assign a value to any variable.

- `variable = value;`

In many cases assignment operator can be combined with other operators to build a shorter version of statement called **Compound Statement**.

For example, instead of `a = a+5`, we can write `a += 5`.

- **+=**, for adding left operand with right operand and then assigning it to variable on the left.
- **-=**, for subtracting left operand with right operand and then assigning it to variable on the left.
- ***=**, for multiplying left operand with right operand and then assigning it to variable on the left.
- **/=**, for dividing left operand with right operand and then assigning it to variable on the left.
- **%=**, for assigning modulo of left operand with right operand and then assigning it to variable on the left.

4. Relational Operators :

- **==, Equal to** : returns true if left hand side is equal to right hand side.
- **!=, Not Equal to** : returns true if left hand side is not equal to right hand side.
- **<, less than** : returns true if left hand side is less than right hand side.
- **<=, less than or equal to** : returns true if left hand side is less than or equal to right hand side.
- **>, Greater than** : returns true if left hand side is greater than right hand side.
- **>=, Greater than or equal to** : returns true if left hand side is greater than or equal to right hand side.

```
// Java program to illustrate
// relational operators
public class operators {
    public static void main(String[] args)
    {
        int a = 20, b = 10;
        String x = "Thank", y = "Thank";
        int ar[] = { 1, 2, 3 };
        int br[] = { 1, 2, 3 };
        boolean condition = true;

        // various conditional operators
        System.out.println("a == b :" + (a == b));
        System.out.println("a < b :" + (a < b));
        System.out.println("a <= b :" + (a <= b));
        System.out.println("a > b :" + (a > b));
        System.out.println("a >= b :" + (a >= b));
        System.out.println("a != b :" + (a != b));

        // Arrays cannot be compared with
        // relational operators because objects
        // store references not the value
        System.out.println("x == y : " + (ar == br));

        System.out.println("condition==true : "
            + (condition == true));
    }
}
```

5. Logical Operators :

- **&&, Logical AND** : returns true when both conditions are true.
- **||, Logical OR** : returns true if at least one condition is true.

```
// Java program to illustrate
// logical operators

import java.util.*;

public class operators {
    public static void main(String[] args)
    {
        String x = "Sher";
        String y = "Locked";

        Scanner s = new Scanner(System.in);
        System.out.print("Enter username:");
        String uid = s.next();
        System.out.print("Enter password:");
        String upwd = s.next();

        // Check if user-name and password match or not.
        if ((uid.equals(x) && upwd.equals(y))
            || (uid.equals(y) && upwd.equals(x))) {
            System.out.println("Welcome user.");
        }
        else {
            System.out.println("Wrong uid or password");
        }
    }
}
```

6. Ternary operator : Ternary operator is a shorthand version of if-else statement. It has three operands and hence the name ternary. General format is-

condition ? if true : if false

```
// Java program to illustrate
// max of three numbers using
// ternary operator.
public class operators {
    public static void main(String[] args)
    {
        int a = 20, b = 10, c = 30, result;

        // result holds max of three
        // numbers
        result = ((a > b)
                  ? (a > c)
                    ? a
                    : c
                  : (b > c)
                    ? b
                    : c);
        System.out.println("Max of three numbers = "
                           + result);
    }
}
```


7. **Bitwise Operators:** These operators are used to perform manipulation of individual bits of a number.

- **&, Bitwise AND operator:** returns bit by bit AND of input values.
- **|, Bitwise OR operator:** returns bit by bit OR of input values.
- **^, Bitwise XOR operator:** returns bit by bit XOR of input values.
- **~, Bitwise Complement Operator:** This is a unary operator which returns the one's complement representation of the input value, i.e. with all bits inverted.

```
// Java program to illustrate
// bitwise operators
public class operators {
    public static void main(String[] args)
    {
        // if int a = 010
        // Java considers it as octal value
        // of 8 as number starts with 0.
        int a = 0x0005;
        int b = 0x0007;

        // bitwise and
        // 0101 & 0111=0101
        System.out.println("a&b = " + (a & b));

        // bitwise and
        // 0101 | 0111=0111
        System.out.println("a|b = " + (a | b));

        // bitwise xor
        // 0101 ^ 0111=0010
        System.out.println("a^b = " + (a ^ b));

        // bitwise and
        // ~0101=1010
        System.out.println("~a = " + ~a);

        // can also be combined with
        // assignment operator to provide shorthand
        // assignment
        // a=a&b
        a &= b;
        System.out.println("a= " + a);
    }
}
```

Decision Making in Java (if, if-else, switch, break, continue, jump)

if: if statement used to decide whether a certain statement or block of statements will be executed or not.

Syntax:

```
if(condition)
{
    // Statements to execute if
    // condition is true
}

// Java program to illustrate If statement
class IfDemo
{
    public static void main(String args[])
    {
        int i = 10;

        if (i > 15)
            System.out.println("10 is less than 15");

        // This statement will be executed
        // as if considers one statement by default
        System.out.println("I am Not in if");
    }
}
```

if-else: if a condition is true it will execute a block of statements and if the condition is false else block will be executed.

Syntax:

```
if (condition)
{
    // Executes this block if condition is true
}
else
{
    // Executes this block if condition is false
}
```

```
// Java program to illustrate if-else statement
class IfElseDemo
{
    public static void main(String args[])
    {
        int i = 10;

        if (i < 15)
            System.out.println("i is smaller than 15");
        else
            System.out.println("i is greater than 15");
    }
}
```

nested-if: A nested if is an if statement that is the target of another if or else. Nested if statements means an if statement inside an if statement.

Syntax:

```
if (condition1)
{
    // Executes when condition1 is true
    if (condition2)
    {
        // Executes when condition2 is true
    }
}
```

```
// Java program to illustrate nested-if statement
class NestedIfDemo
{
    public static void main(String args[])
    {
        int i = 10;

        if (i == 10)
        {
            // First if statement
            if (i < 15)
                System.out.println("i is smaller than 15");

            // Nested - if statement
            // Will only be executed if statement above
            // it is true
            if (i < 12)
                System.out.println("i is smaller than 12 too");
            else
                System.out.println("i is greater than 15");
        }
    }
}
```

if-else-if ladder: Here, a user can decide among multiple options. The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

```
if (condition)
    statement;
else if (condition)
    statement;
.
.
else
    statement;
```

Example:

```
// Java program to illustrate if-else-if ladder
class ifelseifDemo
{
    public static void main(String args[])
    {
        int i = 20;

        if (i == 10)
            System.out.println("i is 10");
        else if (i == 15)
            System.out.println("i is 15");
        else if (i == 20)
            System.out.println("i is 20");
        else
            System.out.println("i is not present");
    }
}
```

switch-case: The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.

Syntax:

```
switch (expression)
{
    case value1:
        statement1;
        break;
    case value2:
```

```
    statement2;
    break;
case valueN:
    statementN;
    break;
default:
    statementDefault;
}
```

Example:

```
// Java program to illustrate switch-case
class SwitchCaseDemo
{
    public static void main(String args[])
    {
        int i = 9;
        switch (i)
        {
            case 0:
                System.out.println("i is zero.");
                break;
            case 1:
                System.out.println("i is one.");
                break;
            case 2:
                System.out.println("i is two.");
                break;
            default:
                System.out.println("i is greater than 2.");
        }
    }
}
```

Return: The return statement is used to explicitly return from a method. That is, it causes a program control to transfer back to the caller of the method.

```
// Java program to illustrate using return
class Return
{
    public static void main(String args[])
    {
        boolean t = true;
        System.out.println("Before the return.");

        if (t)
            return;

        // Compiler will bypass every statement
        // after return
        System.out.println("This won't execute.");
    }
}
```

Break: In Java, break is majorly used for:

- Terminate a sequence in a switch statement (discussed above).
- To exit a loop.

```
// Java program to illustrate using
// break to exit a loop
class BreakLoopDemo
{
    public static void main(String args[])
    {
        // Initially loop is set to run from 0-9
        for (int i = 0; i < 10; i++)
        {
            // terminate loop when i is 5.
            if (i == 5)
                break;

            System.out.println("i: " + i);
        }
        System.out.println("Loop complete.");
    }
}
```

Loops in Java

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

Syntax :

```
while (boolean condition)
```

```
{
    loop statements...
}
```

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

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

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

- 2. for loop:** for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

Syntax:

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

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

- **Enhanced For loop:** Enhanced for loop provides a simpler way to iterate through the elements of a collection or array.

Syntax:

```
for (T element:Collection obj/array)
{
    statement(s)
}
```

```
// Java program to illustrate enhanced for loop
public class enhancedforloop
{
    public static void main(String args[])
    {
        String array[] = {"Ron", "Harry", "Hermoine"};

        //enhanced for loop
        for (String x:array)
        {
            System.out.println(x);
        }

        /* for loop for same function
        for (int i = 0; i < array.length; i++)
        {
            System.out.println(array[i]);
        }
        */
    }
}
```

3. **do while:** do while loop is similar to while loop with only difference that it checks for condition after executing the statements

Syntax:

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

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


Packages In Java

Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces.

Built-in Packages

These packages consist of a large number of classes which are a part of Java **API**. Some of the commonly used built-in packages are:

- 1) **java.lang:** Contains language support classes (e.g. `Class` which defines primitive data types, math operations). This package is automatically imported.
- 2) **java.io:** Contains classes for supporting input / output operations.
- 3) **java.util:** Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.
- 4) **java.applet:** Contains classes for creating Applets.
- 5) **java.awt:** Contains classes for implementing the components for graphical user interfaces (like button , ; menus etc).
- 6) **java.net:** Contains classes for supporting networking operations.

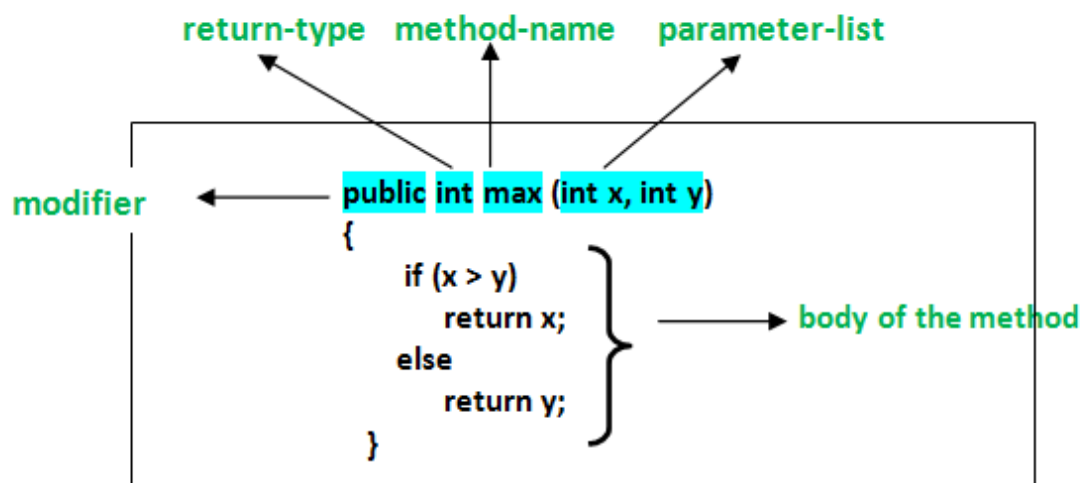
Note: to use package inside your Java program use **import** keywords.

Example: `import java.util.*;`

Methods in Java

A method is a collection of statements that perform some specific task and return the result to the caller. A method can perform some specific task without returning anything.

Method Declaration



Modifier:- Defines **access type** of the method i.e. from where it can be accessed in your application.

In Java, there 4 type of the **access specifiers**.

- ✓ **public:** accessible in all class in your application.

- ✓ **protected**: accessible within the class in which it is defined and in its **subclass(es)**
- ✓ **private**: accessible only within the class in which it is defined.
- ✓ **default** (declared/defined without using any modifier) : accessible within same class and package within which its class is defined.

The return type : The data type of the value returned by the method or void if does not return a value.

Method Name : the rules for field names apply to method names as well, but the convention is a little different.

Parameter list : Comma separated list of the input parameters are defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, you must use empty parentheses ().

Method body : it is enclosed between braces. The code you need to be executed to perform your intended operations.

Calling a method

```
// Program to illustrate methods in java
import java.io.*;

class Addition {

    int sum = 0;

    public int addTwoInt(int a, int b){

        // adding two integer value.
        sum = a + b;

        //returning summation of two values.
        return sum;
    }

}

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

        // creating an instance of Addition class
        Addition add = new Addition();

        // calling addTwoInt() method to add two integer using instance created
        // in above step.
        int s = add.addTwoInt(1,2);
        System.out.println("Sum of two integer values :"+ s);

    }

}
```

Arrays in Java

An array is a group of like-typed variables that are referred to by a common name.

- In Java all arrays are dynamically allocated.(discussed below)
- We can find array's length using member **length**.
- A Java array variable can also be declared like other variables with [] after the data type.
- The variables in the array are ordered and each have an index beginning from 0.
- Java array can be also be used as a static field, a local variable or a method parameter.
- The **size** of an array must be specified by an int value and not long or short.

40	55	63	17	22	68	89	97	89
0	1	2	3	4	5	6	7	8

<- Array Indices

Array Length = 9

First Index = 0

Last Index = 8

One-Dimensional Arrays :

The general form of a one-dimensional array declaration is

```
type var-name[];
```

OR

```
type[] var-name;
```

Example:

```
// both are valid declarations
```

```
int intArray[];
```

```
or int[] intArray;
```

Instantiating an Array in Java

When an array is declared, only a reference of array is created. To actually create or give memory to array, you create an array like this:

```
var-name = new type [size];
```

Example:

```
int intArray[];    //declaring array
```

```
intArray = new int[20]; // allocating memory to array
```

OR

```
int[] intArray = new int[20]; // combining both statements in one
```

Array Literal

In a situation, where the size of the array and variables of array are already known, array literals can be used.

```
int[] intArray = new int[]{ 1,2,3,4,5,6,7,8,9,10 };
```

Accessing Java Array Elements using for Loop

// Java program to illustrate creating an array
// of integers, puts some values in the array,
// and prints each value to standard output.

```
class GFG
{
    public static void main (String[] args)
    {
        // declares an Array of integers.
        int[] arr;

        // allocating memory for 5 integers.
        arr = new int[5];

        // initialize the first elements of the array
        arr[0] = 10;

        // initialize the second elements of the array
        arr[1] = 20;

        //so on...
        arr[2] = 30;
        arr[3] = 40;
        arr[4] = 50;

        // accessing the elements of the specified array
        for (int i = 0; i < arr.length; i++)
            System.out.println("Element at index " + i +
                               " : "+ arr[i]);
    }
}
```

Multidimensional Arrays

Multidimensional arrays are **arrays of arrays** with each element of the array holding the reference of other array.

```
int[][] intArray = new int[10][20]; //a 2D array or matrix
int[][][] intArray = new int[10][20][10]; //a 3D array
```

```
class multiDimensional
{
    public static void main(String args[])
    {
        // declaring and initializing 2D array
        int arr[][] = { {2,7,9},{3,6,1},{7,4,2} };

        // printing 2D array
        for (int i=0; i < 3 ; i++)
        {
            for (int j=0; j < 3 ; j++)
                System.out.print(arr[i][j] + " ");

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

Passing Arrays to Methods

```
// Java program to demonstrate
// passing of array to method

class Test
{
    // Driver method
    public static void main(String args[])
    {
        int arr[] = {3, 1, 2, 5, 4};

        // passing array to method m1
        sum(arr);
    }

    public static void sum(int[] arr)
    {
        // getting sum of array values
        int sum = 0;

        for (int i = 0; i < arr.length; i++)
            sum+=arr[i];

        System.out.println("sum of array values : " + sum);
    }
}
```

BigInteger Class in Java

BigInteger class is used for mathematical operation which involves very big integer calculations that are outside the limit of all available primitive data types.

- For example factorial of 100 contains 158 digits in it so we can't store it in any primitive data type available.

```
// Java program to find large factorials using BigInteger
import java.math.BigInteger;
import java.util.Scanner;

public class Example
{
    // Returns Factorial of N
    static BigInteger factorial(int N)
    {
        // Initialize result
        BigInteger f = new BigInteger("1"); // Or BigInteger.ONE

        // Multiply f with 2, 3, ...N
        for (int i = 2; i <= N; i++)
            f = f.multiply(BigInteger.valueOf(i));

        return f;
    }

    // Driver method
    public static void main(String args[]) throws Exception
    {
        int N = 20;
        System.out.println(factorial(N));
    }
}
```

Declaration:

```
BigInteger A, B;
A = BigInteger.valueOf(54);
B = BigInteger.valueOf(37);
```

Or

```
A = new BigInteger("54");
B = new BigInteger("123456789123456789");
```

We can create BigInteger for One or Zero.

```
A = BigInteger.ONE;  
A = BigInteger.Zero;
```

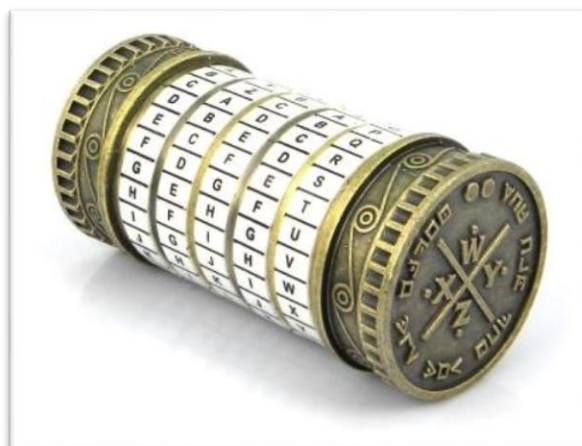
Mathematical operations:

```
BigInteger C = A.add(B);
```

Extraction of value from BigInteger:

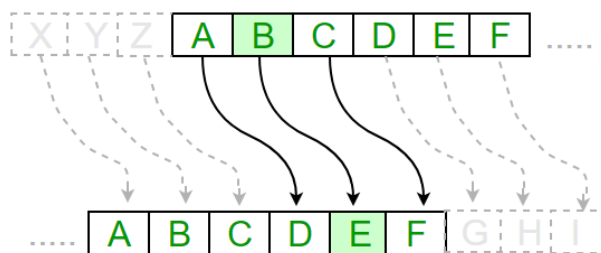
```
int x    = A.intValue();    // value should be in limit of int x  
long y   = A.longValue();   // value should be in limit of long y  
String z = A.toString();
```

Classical Ciphers



Shift cipher

The encryption can be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, $A = 0, B = 1, \dots, Z = 25$. Encryption of a letter by a shift n can be described mathematically as.




```
class Shift_cipher
{
    public static StringBuffer encrypt(String text, int s)
    {
        StringBuffer result= new StringBuffer();

        for (int i=0; i<text.length(); i++)
        {
            if (Character.isUpperCase(text.charAt(i)))
            {
                char ch = (char)((((int)text.charAt(i) +
                                   s - 65) % 26 + 65);
                result.append(ch);
            }
            else
            {
                char ch = (char)((((int)text.charAt(i) +
                                   s - 97) % 26 + 97);
                result.append(ch);
            }
        }
        return result;
    }
    public static void main(String[] args)
    {
        String text = "ATTACKATONCE";
        int s = 1;
        System.out.println("Text : " + text);
        System.out.println("Shift : " + s);
        System.out.println("Cipher: " + encrypt(text, s));
    }
}
```

Note: ASCII code for A is 65. ASCII code for a is 97.

H.W1: write the decrypt code of shift cipher.

H.W2. Apply brute-force attack on shift cipher.

Monoalphabetic cipher

Each plaintext letter is mapped to different cipher text letter according to key permutation.

```
import java.util.Scanner;

public class Monoalphabetic_cipher
{
    public static char p[] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
        'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
        'w', 'x', 'y', 'z' };
    public static char ch[] = { 'Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O',
        'P', 'A', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L', 'Z', 'X', 'C',
        'V', 'B', 'N', 'M' };

    public static String doEncryption(String s)
    {
        char c[] = new char[(s.length())];
        for (int i = 0; i < s.length(); i++)
        {
            for (int j = 0; j < 26; j++)
            {
                if (p[j] == s.charAt(i))
                {
                    c[i] = ch[j];
                    break;
                }
            }
        }
        return (new String(c));
    }
}
```

```
public static String doDecryption(String s)
{
    char p1[] = new char[(s.length())];
    for (int i = 0; i < s.length(); i++)
    {
        for (int j = 0; j < 26; j++)
        {
            if (ch[j] == s.charAt(i))
            {
                p1[i] = p[j];
                break;
            }
        }
    }
    return (new String(p1));
}

public static void main(String args[])
{
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the message: ");
    String en = doEncryption(sc.next().toLowerCase());
    System.out.println("Encrypted message: " + en);
    System.out.println("Decrypted message: " + doDecryption(en));
    sc.close();
}
```

Playfair Cipher

1- Use the key to generate 5*5 matrix.

The key is "monarchy"
 Thus the initial entries are
 'm', 'o', 'n', 'a', 'r', 'c', 'h', 'y'
 followed by remaining characters of
 a-z(except 'j') in that order.

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

2- Split plaintext into pairs of two letters (digraphs). If there is an odd number of letters, a Z is added to the last letter.

For example:

PlainText: "instruments"

After Split: 'in' 'st' 'ru' 'me' 'nt' 'sz'

3- Rules for Encryption:

- If both the letters are in the same column:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

If both the letters are in the same row:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

If neither of the above rules is true:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

Playfair Cipher program

```
import java.util.Scanner;

public class Playfair_Cipher_Decryption
{
    private String KeyWord      = new String();
    private String Key          = new String();
    private char  matrix_arr[][] = new char[5][5];

    public void setKey(String k)
    {
        String K_adjust = new String();
        boolean flag = false;
        K_adjust = K_adjust + k.charAt(0);
        for (int i = 1; i < k.length(); i++)
        {
            for (int j = 0; j < K_adjust.length(); j++)
            {
                if (k.charAt(i) == K_adjust.charAt(j))
                {
                    flag = true;
                }
            }
            if (flag == false)
                K_adjust = K_adjust + k.charAt(i);
            flag = false;
        }
        KeyWord = K_adjust;
    }

    public void KeyGen()
    {
        boolean flag = true;
        char current;
        Key = KeyWord;
        for (int i = 0; i < 26; i++)
        {
            current = (char) (i + 97);
            if (current == 'j')
                continue;
            for (int j = 0; j < KeyWord.length(); j++)
            {
                if (current == KeyWord.charAt(j))
                {
                    flag = false;
                    break;
                }
            }
            if (flag)
                Key = Key + current;
            flag = true;
        }
        System.out.println(Key);
    }
}
```

```
        matrix();
    }

    private void matrix()
    {
        int counter = 0;
        for (int i = 0; i < 5; i++)
        {
            for (int j = 0; j < 5; j++)
            {
                matrix_arr[i][j] = Key.charAt(counter);
                System.out.print(matrix_arr[i][j] + " ");
                counter++;
            }
            System.out.println();
        }
    }

    private String format(String old_text)
    {
        int i = 0;
        int len = 0;
        String text = new String();
        len = old_text.length();
        for (int tmp = 0; tmp < len; tmp++)
        {
            if (old_text.charAt(tmp) == 'j')
            {
                text = text + 'i';
            }
            else
            {
                text = text + old_text.charAt(tmp);
            }
        }
        len = text.length();
        for (i = 0; i < len; i = i + 2)
        {
            if (text.charAt(i + 1) == text.charAt(i))
            {
                text = text.substring(0, i + 1) + 'x' + text.substring(i + 1);
            }
        }
        return text;
    }

    private String[] Divid2Pairs(String new_string)
    {
        String Original = format(new_string);
        int size = Original.length();
        if (size % 2 != 0)
        {
            size++;
            Original = Original + 'x';
        }
        String x[] = new String[size / 2];
        int counter = 0;
```

```
        for (int i = 0; i < size / 2; i++)
        {
            x[i] = Original.substring(counter, counter + 2);
            counter = counter + 2;
        }
        return x;
    }

    public int[] GetDiminsions(char letter)
    {
        int[] key = new int[2];
        if (letter == 'j')
            letter = 'i';
        for (int i = 0; i < 5; i++)
        {
            for (int j = 0; j < 5; j++)
            {
                if (matrix_arr[i][j] == letter)
                {
                    key[0] = i;
                    key[1] = j;
                    break;
                }
            }
        }
        return key;
    }

    public String encryptMessage(String Source)
    {
        String src_arr[] = Divid2Pairs(Source);
        String Code = new String();
        char one;
        char two;
        int part1[] = new int[2];
        int part2[] = new int[2];
        for (int i = 0; i < src_arr.length; i++)
        {
            one = src_arr[i].charAt(0);
            two = src_arr[i].charAt(1);
            part1 = GetDiminsions(one);
            part2 = GetDiminsions(two);
            if (part1[0] == part2[0])
            {
                if (part1[1] < 4)
                    part1[1]++;
                else
                    part1[1] = 0;
                if (part2[1] < 4)
                    part2[1]++;
                else
                    part2[1] = 0;
            }
            else if (part1[1] == part2[1])
            {

```

```

        if (part1[0] < 4)
            part1[0]++;
        else
            part1[0] = 0;
        if (part2[0] < 4)
            part2[0]++;
        else
            part2[0] = 0;
    }
    else
    {
        int temp = part1[1];
        part1[1] = part2[1];
        part2[1] = temp;
    }
    Code = Code + matrix_arr[part1[0]][part1[1]]
        + matrix_arr[part2[0]][part2[1]];
}
return Code;
}

public String decryptMessage(String Code)
{
    String Original = new String();
    String src_arr[] = Divid2Pairs(Code);
    char one;
    char two;
    int part1[] = new int[2];
    int part2[] = new int[2];
    for (int i = 0; i < src_arr.length; i++)
    {
        one = src_arr[i].charAt(0);
        two = src_arr[i].charAt(1);
        part1 = GetDiminsions(one);
        part2 = GetDiminsions(two);
        if (part1[0] == part2[0])
        {
            if (part1[1] > 0)
                part1[1]--;
            else
                part1[1] = 4;
            if (part2[1] > 0)
                part2[1]--;
            else
                part2[1] = 4;
        }
        else if (part1[1] == part2[1])
        {
            if (part1[0] > 0)
                part1[0]--;
            else
                part1[0] = 4;
            if (part2[0] > 0)
                part2[0]--;
            else
                part2[0] = 4;
        }
    }
}

```



```
        part2[0] = 4;
    }
    else
    {
        int temp = part1[1];
        part1[1] = part2[1];
        part2[1] = temp;
    }
    Original = Original + matrix_arr[part1[0]][part1[1]]
        + matrix_arr[part2[0]][part2[1]];
    }
    return Original;
}

public static void main(String[] args)
{
    Playfair_Cipher_Decryption x = new Playfair_Cipher_Decryption();
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter a keyword:");
    String keyword = sc.next();
    x.setKey(keyword);
    x.KeyGen();
    System.out
        .println("Enter word to encrypt: (Make sure length of message is
even)");
    String key_input = sc.next();
    if (key_input.length() % 2 == 0)
    {
        System.out.println("Encryption: " + x.encryptMessage(key_input));
        System.out.println("Decryption: "
            + x.decryptMessage(x.encryptMessage(key_input)));
    }
    else
    {
        System.out.println("Message length should be even");
    }
    sc.close();
}
}
```

Hill Cipher

We have to encrypt the message 'ACT' (n=3). The key is 'GYBNQKURP' which can be written as the nxn matrix:

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix}$$

The message 'ACT' is written as vector:

$$\begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix}$$

The enciphered vector is given as:

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix} = \begin{bmatrix} 67 \\ 222 \\ 319 \end{bmatrix} \equiv \begin{bmatrix} 15 \\ 14 \\ 7 \end{bmatrix} \pmod{26}$$

which corresponds to ciphertext of 'POH'.

To decrypt the message:

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix}^{-1} \equiv \begin{bmatrix} 8 & 5 & 10 \\ 21 & 8 & 21 \\ 21 & 12 & 8 \end{bmatrix} \pmod{26}$$

for the previous Ciphertext 'POH':

$$\begin{bmatrix} 8 & 5 & 10 \\ 21 & 8 & 21 \\ 21 & 12 & 8 \end{bmatrix} \begin{bmatrix} 15 \\ 14 \\ 7 \end{bmatrix} \equiv \begin{bmatrix} 260 \\ 574 \\ 539 \end{bmatrix} \equiv \begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix} \pmod{26}$$

```

// Java code to implement Hill Cipher
class Hill_cipher
{
    // Following function generates the
    // key matrix for the key string
    static void getKeyMatrix(String key, int keyMatrix[][])
    {
        int k = 0;
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                keyMatrix[i][j] = (key.charAt(k)) % 65;
                k++;
            }
        }
    }

    // Following function encrypts the message
    static void encrypt(int cipherMatrix[][],
                        int keyMatrix[][],
                        int messageVector[][])
    {
        int x, i, j;
        for (i = 0; i < 3; i++)
        {
            for (j = 0; j < 1; j++)
            {
                cipherMatrix[i][j] = 0;

                for (x = 0; x < 3; x++)
                {
                    cipherMatrix[i][j] +=
                        keyMatrix[i][x] * messageVector[x][j];
                }

                cipherMatrix[i][j] = cipherMatrix[i][j] % 26;
            }
        }
    }

    // Function to implement Hill Cipher
    static void HillCipher(String message, String key)
    {
        // Get key matrix from the key string
        int [][]keyMatrix = new int[3][3];
        getKeyMatrix(key, keyMatrix);

        int [][]messageVector = new int[3][1];

        // Generate vector for the message
        for (int i = 0; i < 3; i++)
            messageVector[i][0] = (message.charAt(i)) % 65;
    }
}

```

```
int [][]cipherMatrix = new int[3][1];

// Following function generates
// the encrypted vector
encrypt(cipherMatrix, keyMatrix, messageVector);

String CipherText="";

// Generate the encrypted text from
// the encrypted vector
for (int i = 0; i < 3; i++)
    CipherText += (char)(cipherMatrix[i][0] + 65);

// Finally print the ciphertext
System.out.print(" Ciphertext:" + CipherText);
}

// Driver code
public static void main(String[] args)
{
    // Get the message to be encrypted
    String message = "ACT";

    // Get the key
    String key = "GYBNQKURP";

    HillCipher(message, key);
}
}
```

Vigenere Cipher

to encrypt use Vigenere table:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Example:

Input : Plaintext : GEEKSFORGEES

Keyword : AYUSH

Output : Ciphertext : GCYCZFMLEIM

For generating key, the given keyword is repeated in a circular manner until it matches the length of the plain text.

The keyword "AYUSH" generates the key "AYUSHAYUSHAYU"

The plain text is then encrypted using the process explained below.

Encryption

The first letter of the plaintext, G is paired with A, the first letter of the key. So use row G and column A of the Vigenère square, namely G. Similarly, for the second letter of the plaintext, the second letter of the key is used, the letter at row E and column Y is C. The rest of the plaintext is enciphered in a similar fashion.

```
// Java code to implement Vigenere Cipher

class Vigenere
{
    // This function generates the key in a cyclic manner until it's length isn't
    // equal to the length of original text
    static String generateKey(String str, String key)
    {
        int x = str.length();

        for (int i = 0; ; i++)
        {
            if (x == i)
                i = 0;
            if (key.length() == str.length())
                break;
            key+=(key.charAt(i));
        }
        return key;
    }

    // This function returns the encrypted text
    // generated with the help of the key
    static String cipherText(String str, String key)
    {
        String cipher_text="";

        for (int i = 0; i < str.length(); i++)
        {
            // converting in range 0-25
            int x = (str.charAt(i) + key.charAt(i)) %26;

            // convert into alphabets(ASCII)
            x += 'A';

            cipher_text+=(char)(x);
        }
        return cipher_text;
    }

    // This function decrypts the encrypted text
    // and returns the original text
    static String originalText(String cipher_text, String key)
    {
        String orig_text="";

        for (int i = 0 ; i < cipher_text.length() &&
            i < key.length(); i++)
        {
            // converting in range 0-25
            int x = (cipher_text.charAt(i) -
                key.charAt(i) + 26) %26;

            // convert into alphabets(ASCII)

```

```
        x += 'A';
        orig_text+=(char)(x);
    }
    return orig_text;
}

// Driver code
public static void main(String[] args)
{
    String str = "GEEKSFORGEEKS";
    String keyword = "AYUSH";

    String key = generateKey(str, keyword);
    String cipher_text = cipherText(str, key);

    System.out.println("Ciphertext : "
        + cipher_text + "\n");

    System.out.println("Original/Decrypted Text : "
        + originalText(cipher_text, key));
}
```

XOR Cipher

```
class Xor_cipher
{
    // The same function is used to encrypt and
    // decrypt
    static String encryptDecrypt(String inputString)
    {
        // Define XOR key
        // Any character value will work
        char xorKey = 'P';

        // Define String to store encrypted/decrypted String
        String outputString = "";

        // calculate length of input string
        int len = inputString.length();

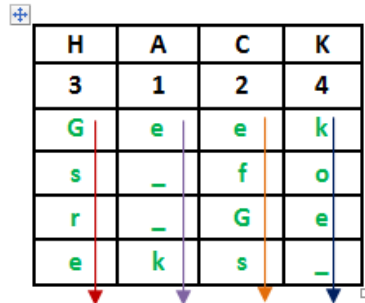
        // perform XOR operation of key
        // with every character in string
        for (int i = 0; i < len; i++)
        {
            outputString = outputString +
                Character.toString((char) (inputString.charAt(i) ^ xorKey));
        }

        System.out.println(outputString);
        return outputString;
    }

    // Driver code
    public static void main(String[] args)
    {
        String sampleString = "GeeksforGeeks";

        // Encrypt the string
        System.out.println("Encrypted String");
        String encryptedString = encryptDecrypt(sampleString);

        // Decrypt the string
        System.out.println("Decrypted String");
        encryptDecrypt(encryptedString);
    }
}
```


Transposition Cipher**Encryption****Given text** = Geeks for Geeks**Keyword** = HACK**Length of Keyword** = 4 (no of rows)**Order of Alphabets in HACK** = 3124


H	A	C	K
3	1	2	4
G	e	e	k
s	_	f	o
r	_	G	e
e	k	s	_

Print Characters of column 1,2,3,4

Encrypted Text = e kefGsGsrekeo_

```

public class Transposition_cipher
{
    public static String selectedKey;
    public static char  sortedKey[];
    public static int   sortedKeyPos[];

    // default constructor define the default key
    public Transposition_cipher()
    {
        selectedKey = "megabuck";
        sortedKeyPos = new int[selectedKey.length()];
        sortedKey = selectedKey.toCharArray();
    }

    // Parameterized constructor define the custom key
    public Transposition_cipher(String myKey)
    {
        selectedKey = myKey;
        sortedKeyPos = new int[selectedKey.length()];
        sortedKey = selectedKey.toCharArray();
    }

    // To reorder data do the sorting on selected key
    public static void doProcessOnKey()
    {
        // Find position of each character in selected key and arrange it on
        // alphabetical order
        int min, i, j;
        char originalKey[] = selectedKey.toCharArray();
        char temp;
    }
}

```

```

// First Sort the array of selected key
for (i = 0; i < selectedKey.length(); i++)
{
    min = i;
    for (j = i; j < selectedKey.length(); j++)
    {
        if (sortedKey[min] > sortedKey[j])
        {
            min = j;
        }
    }
    if (min != i)
    {
        temp = sortedKey[i];
        sortedKey[i] = sortedKey[min];
        sortedKey[min] = temp;
    }
}
// Fill the position of array according to alphabetical order
for (i = 0; i < selectedKey.length(); i++)
{
    for (j = 0; j < selectedKey.length(); j++)
    {
        if (originalKey[i] == sortedKey[j])
            sortedKeyPos[i] = j;
    }
}
}

// to encrypt the targeted string
public static String doEncryption(String plainText)
{
    int min, i, j;
    char originalKey[] = selectedKey.toCharArray();
    char temp;
    doProcessOnKey();
    // Generate encrypted message by doing encryption using Transpotion
    // Cipher
    int row = plainText.length() / selectedKey.length();
    int extrabit = plainText.length() % selectedKey.length();
    int exrow = (extrabit == 0) ? 0 : 1;
    int rowtemp = -1, coltemp = -1;
    int totallen = (row + exrow) * selectedKey.length();
    char pmat[][] = new char[(row + exrow)][(selectedKey.length())];
    char encry[] = new char[totallen];
    int tempcnt = -1;
    row = 0;
    for (i = 0; i < totallen; i++)
    {
        coltemp++;
        if (i < plainText.length())
        {
            if (coltemp == (selectedKey.length()))
            {
                row++;
            }
        }
    }
}

```

```

        coltemp = 0;
    }
    pmat[row][coltemp] = plainText.charAt(i);
}
else
{ // do the padding ...
    pmat[row][coltemp] = '*';
}
}
int len = -1, k;
for (i = 0; i < selectedKey.length(); i++)
{
    for (k = 0; k < selectedKey.length(); k++)
    {
        if (i == sortedKeyPos[k])
        {
            break;
        }
    }
    for (j = 0; j <= row; j++)
    {
        len++;
        encry[len] = pmat[j][k];
    }
}
String p1 = new String(encry);
return (new String(p1));
}

// to decrypt the targeted string
public static String doDecryption(String s)
{
    int min, i, j, k;
    char key[] = selectedKey.toCharArray();
    char encry[] = s.toCharArray();
    char temp;
    doProcessOnKey();
    // Now generating plain message
    int row = s.length() / selectedKey.length();
    char pmat[][] = new char[row][(selectedKey.length())];
    int tempcnt = -1;
    for (i = 0; i < selectedKey.length(); i++)
    {
        for (k = 0; k < selectedKey.length(); k++)
        {
            if (i == sortedKeyPos[k])
            {
                break;
            }
        }
        for (j = 0; j < row; j++)
        {
            tempcnt++;
            pmat[j][k] = encry[tempcnt];
        }
    }
}

```

```
    }
    // store matrix character in to a single string
    char p1[] = new char[row * selectedKey.length()];
    k = 0;
    for (i = 0; i < row; i++)
    {
        for (j = 0; j < selectedKey.length(); j++)
        {
            if (pmat[i][j] != '*')
            {
                p1[k++] = pmat[i][j];
            }
        }
    }
    p1[k++] = '\0';
    return (new String(p1));
}

@SuppressWarnings("static-access")
public static void main(String[] args)
{
    Transposition_cipher tc = new Transposition_cipher();
    System.out.println("Encrypted Message is: "
        + tc.doEncryption("Sanfoundry"));
    System.out.println("Decrypted Message is: "
        + tc.doDecryption(tc.doEncryption("Sanfoundry")));
}
}
```

Java Cryptography

The Java cryptography API is divided between the following Java **packages**:

- java.security
- java.security.cert
- java.security.spec
- java.security.interfaces
- javax.crypto
- javax.crypto.spec
- javax.crypto.interfaces

The core **classes** and interfaces of these packages are:

- SecureRandom
- Cipher
- MessageDigest
- Signature
- Mac
- SecretKeyFactory
- KeyPairGenerator
- KeyGenerator

Cipher class

The Cipher (javax.crypto.Cipher) class represents a cryptographic algorithm. A cipher can be used to both encrypt and decrypt data.

Example to create instance of cipher

```
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
```

Cipher modes:

- EBC - Electronic Codebook
- CBC - Cipher Block Chaining
- CFB - Cipher Feedback
- OFB - Output Feedback
- CTR - Counter

Cipher Initialization

Here is an example of initializing a Cipher instance in **encryption** mode:

```
Key key = ... // get / create symmetric encryption key
cipher.init(Cipher.ENCRYPT_MODE, key);
```

Here is an example of initializing a Cipher instance in **decryption** mode:

```
Key key = ... // get / create symmetric encryption key
cipher.init(Cipher.DECRYPT_MODE, key);
```

Encrypting and Decrypting Data

In order encrypt or decrypt data with a Cipher instance you call one of these two methods:

- `update()`
- `doFinal()`

Here is an encryption example:

```
byte[] plainText = "abcdefghijklmnopqrstuvwxyz".getBytes("UTF-8");
byte[] cipherText = cipher.doFinal(plainText);
```

If you have to encrypt or decrypt multiple blocks of data, e.g. multiple blocks from a large file, you call the **update()** once for each block of data, and finish with a call to **doFinal()** with the last data block. Here is an example of encrypting multiple blocks of data:

```
byte[] data1 = "abcdefghijklmnopqrstuvwxyz".getBytes("UTF-8");
byte[] data2 = "zyxwvutsrqponmlkjihgfedcba".getBytes("UTF-8");
byte[] data3 = "01234567890123456789012345".getBytes("UTF-8");

byte[] cipherText1 = cipher.update(data1);
byte[] cipherText2 = cipher.update(data2);
byte[] cipherText3 = cipher.doFinal(data3);
```

KeyGenerator class

The *Java KeyGenerator* class (`javax.crypto.KeyGenerator`) is used to generate symmetric encryption keys.

We create a *KeyGenerator* instance by calling the static method **getInstance()** passing as parameter the name of the encryption algorithm to create a key for.

```
KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
```

After creating the KeyGenerator instance you must **initialize** it. Initializing a KeyGenerator instance is done by calling its **init()** method.

```
SecureRandom secureRandom = new SecureRandom();  
int keyBitSize = 256;  
  
keyGenerator.init(keyBitSize, secureRandom);
```

Once the Java KeyGenerator instance is initialized you can use it to generate keys. **Generating** a key is done by calling the KeyGenerator **generateKey()** method.

```
SecretKey secretKey = keyGenerator.generateKey();
```

DES in Java

```
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;

public class DES
{
    public static void main(String[] argv) {

        try{

            KeyGenerator keygenerator = KeyGenerator.getInstance("DES");
            SecretKey myDesKey = keygenerator.generateKey();

            Cipher desCipher;

            // Create the cipher
            desCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");

            // Initialize the cipher for encryption
            desCipher.init(Cipher.ENCRYPT_MODE, myDesKey);

            //sensitive information
            byte[] text = "No body can see me".getBytes();

            System.out.println("PLainText [Byte Format] : " + text);
            System.out.println("PLainText [String Format] : " + new
String(text));

            // Encrypt the text
            byte[] textEncrypted = desCipher.doFinal(text);

            System.out.println("Text Encryted : " + textEncrypted);

            // Initialize the same cipher for decryption
            desCipher.init(Cipher.DECRYPT_MODE, myDesKey);

            // Decrypt the text
            byte[] textDecrypted = desCipher.doFinal(textEncrypted);

            System.out.println("Text Decryted : " + new String(textDecrypted));

        }catch(NoSuchAlgorithmException e){
            e.printStackTrace();
        }catch(NoSuchPaddingException e){
            e.printStackTrace();
        }catch(InvalidKeyException e){
```



```
        e.printStackTrace();
    }catch(IllegalBlockSizeException e){
        e.printStackTrace();
    }catch(BadPaddingException e){
        e.printStackTrace();
    }
}
}
```

AES in JAVA

```
import java.util.Base64;

import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;

public class AES {
    static Cipher cipher;

    public static void main(String[] args) throws Exception {
        KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
        keyGenerator.init(128);
        SecretKey secretKey = keyGenerator.generateKey();
        cipher = Cipher.getInstance("AES");

        String plainText = "AES Symmetric Encryption Decryption";
        System.out.println("Plain Text Before Encryption: " + plainText);

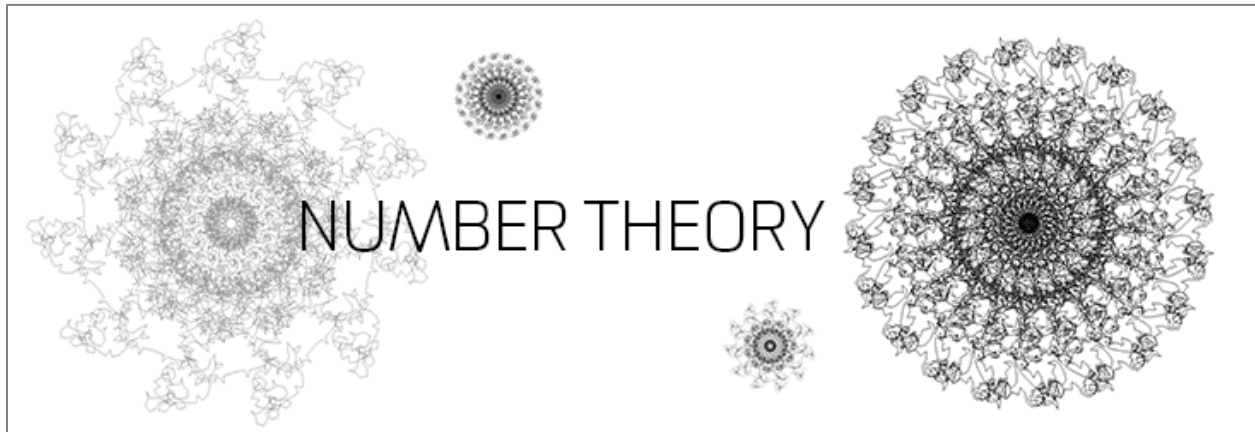
        String encryptedText = encrypt(plainText, secretKey);
        System.out.println("Encrypted Text After Encryption: " + encryptedText);

        String decryptedText = decrypt(encryptedText, secretKey);
        System.out.println("Decrypted Text After Decryption: " + decryptedText);
    }

    public static String encrypt(String plainText, SecretKey secretKey)
        throws Exception {
        byte[] plainTextByte = plainText.getBytes();
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedByte = cipher.doFinal(plainTextByte);
        Base64.Encoder encoder = Base64.getEncoder();
        String encryptedText = encoder.encodeToString(encryptedByte);
        return encryptedText;
    }

    public static String decrypt(String encryptedText, SecretKey secretKey)
        throws Exception {
        Base64.Decoder decoder = Base64.getDecoder();
        byte[] encryptedTextByte = decoder.decode(encryptedText);
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decryptedByte = cipher.doFinal(encryptedTextByte);
        String decryptedText = new String(decryptedByte);
        return decryptedText;
    }
}
```

Number theory



Euclidean algorithms (Basic and Extended)

GCD of two numbers is the largest number that divides both of them.

$$\begin{aligned} 36 &= 2 \times 2 \times 3 \times 3 \\ 60 &= 2 \times 2 \times 3 \times 5 \end{aligned}$$

$$\begin{aligned} \text{GCD} &= \text{Multiplication of common factors} \\ &= 2 \times 2 \times 3 \\ &= 12 \end{aligned}$$

```
// Java program to demonstrate working of extended
// Euclidean Algorithm

import java.util.*;
import java.lang.*;

class GCD_computation
{
    // extended Euclidean Algorithm
    public static int gcd(int a, int b)
    {
        if (a == 0)
            return b;

        return gcd(b%a, a);
    }
}

// Driver Program
public static void main(String[] args)
{
    int a = 10, b = 15, g;
    g = gcd(a, b);
    System.out.println("GCD(" + a + " , " + b + ") = " + g);
}
```

```

    a = 35; b = 10;
    g = gcd(a, b);
    System.out.println("GCD(" + a + " , " + b + ") = " + g);

    a = 31; b = 2;
    g = gcd(a, b);
    System.out.println("GCD(" + a + " , " + b + ") = " + g);

}
}

```

Extended Euclidean Algorithm:

Extended Euclidean algorithm also finds integer coefficients x and y such that:

$$ax + by = \gcd(a, b)$$

Examples:

Input: a = 30, b = 20

Output: gcd = 10

$$x = 1, y = -1$$

(Note that $30 \cdot 1 + 20 \cdot (-1) = 10$)

Input: a = 35, b = 15

Output: gcd = 5

$$x = 1, y = -2$$

(Note that $35 \cdot 1 + 15 \cdot (-2) = 5$)

```

// Java program to demonstrate working of extended Euclidean Algorithm

import java.util.*;
import java.lang.*;

class gcd_extended
{
    // extended Euclidean Algorithm
    public static int gcdExtended(int a, int b, int x, int y)
    {
        // Base Case
        if (a == 0)
        {
            x = 0;
            y = 1;
            return b;
        }

        int x1=1, y1=1; // To store results of recursive call

```

```

        int gcd = gcdExtended(b%a, a, x1, y1);

        // Update x and y using results of recursive call
        x = y1 - (b/a) * x1;
        y = x1;

        return gcd;
    }

// Driver Program
public static void main(String[] args)
{
    int x=1, y=1;
    int a = 35, b = 15;
    int g = gcdExtended(a, b, x, y);
    System.out.print("gcd(" + a + " , " + b + ") = " + g);
}
}

```

Modular Exponentiation (Power in Modular Arithmetic)

Given three numbers x , y and p , compute $(x^y) \% p$.

Examples :

Input: $x = 2, y = 3, p = 5$

Output: 3

Explanation: $2^3 \% 5 = 8 \% 5 = 3$.

Input: $x = 2, y = 5, p = 13$

Output: 6

Explanation: $2^5 \% 13 = 32 \% 13 = 6$.

```

// Iterative Java program to compute modular power
import java.io.*;

class Modular_power {

    /* Iterative Function to calculate (x^y)%p in O(log y) */
    static int power(int x, int y, int p)
    {
        // Initialize result
        int res = 1;

        // Update x if it is more than or equal to p
        x = x % p;

        while (y > 0)
        {
            // If y is odd, multiply x with result

```

```

        if((y & 1)==1)
            res = (res * x) % p;

        // y must be even now y = y / 2
        y = y >> 1;
        x = (x * x) % p;
    }
    return res;
}

// Driver Program to test above functions
public static void main(String args[])
{
    int x = 2;
    int y = 5;
    int p = 13;
    System.out.println("Power is " + power(x, y, p));
}
}

```

Modular multiplicative inverse

Given two integers ‘a’ and ‘m’, find modular multiplicative inverse of ‘a’ under modulo ‘m’.

The modular multiplicative inverse is an integer ‘x’ such that.

$$a x \equiv 1 \pmod{m}$$

The value of x should be in $\{0, 1, 2, \dots, m-1\}$, i.e., in the range of integer modulo m.

The multiplicative inverse of “a modulo m” exists if and only if a and m are relatively prime (i.e., if $\gcd(a, m) = 1$).

Examples:

Input: a = 3, m = 11

Output: 4

Since $(4*3) \bmod 11 = 1$, 4 is modulo inverse of 3

One might think, 15 also as a valid output as $(15*3) \bmod 11$ is also 1, but 15 is not in ring $\{0, 1, 2, \dots, 10\}$, so not valid.

Input: a = 10, m = 17

Output: 12

Since $(10*12) \bmod 17 = 1$, 12 is modulo inverse of 3

The idea is to use [Extended Euclidean algorithms](#) that takes two integers 'a' and 'b', finds their gcd and also find 'x' and 'y' such that

$$ax + by = \gcd(a, b)$$

To find multiplicative inverse of 'a' under 'm', we put $b = m$ in above formula. Since we know that a and m are relatively prime, we can put value of gcd as 1.

$$ax + my = 1$$

If we take modulo m on both sides, we get

$$ax + my \equiv 1 \pmod{m}$$

We can remove the second term on left side as 'my (mod m)' would always be 0 for an integer y.

$$ax \equiv 1 \pmod{m}$$

So the 'x' that we can find using [Extended Euclid Algorithm](#) is multiplicative inverse of 'a'

```
class Multiplicative_inverse
{
    // Returns modulo inverse of a with
    // respect to m using extended Euclid
    // Algorithm Assumption: a and m are
    // coprimes, i.e., gcd(a, m) = 1
    static int modInverse(int a, int m)
    {
        int m0 = m;
        int y = 0, x = 1;

        if (m == 1)
            return 0;

        while (a > 1)
        {
            // q is quotient
            int q = a / m;

            int t = m;

            // m is remainder now, process
            // same as Euclid's algo
            m = a % m;
            a = t;
            t = y;

            // Update x and y
            y = x - q * y;
            x = t;
        }

        // Make x positive
        if (x < 0)
            x += m0;
    }
}
```

```
        return x;
    }

    public static void main(String args[])
    {
        int a = 3, m = 11;

        System.out.println("Modular multiplicative "+
                           "inverse is " + modInverse(a, m));
    }
}
```

Euler's Totient Function

Euler's Totient function for an input n is count of numbers in $\{1, 2, 3, \dots, n\}$ that are relatively prime to n , i.e., the numbers whose GCD (Greatest Common Divisor) with n is 1.

```
// A simple java program to calculate Euler's Totient Function
import java.io.*;

class Euler_function {

    // Function to return GCD of a and b
    static int gcd(int a, int b)
    {
        if (a == 0)
            return b;
        return gcd(b % a, a);
    }

    // A simple method to evaluate Euler Totient Function
    static int phi(int n)
    {
        int result = 1;
        for (int i = 2; i < n; i++)
            if (gcd(i, n) == 1)
                result++;
        return result;
    }

    // Driver code
    public static void main(String[] args)
    {
        int n;

        for (n = 1; n <= 10; n++)
            System.out.println("phi(" + n + ") = " + phi(n));
    }
}
```

Multiplicative order

In number theory, given an integer A and a positive integer N with $\gcd(A, N) = 1$, the multiplicative order of A modulo N is the smallest positive integer k with $A^k \pmod{N} = 1$. ($0 < K < N$)

Examples :

Input : $A = 4$, $N = 7$

Output : 3

explanation : $\text{GCD}(4, 7) = 1$

$A^k \pmod{N} = 1$ (smallest positive integer K)

$4^1 \pmod{7} = 4$

$4^2 \pmod{7} = 2$

$4^3 \pmod{7} = 1$

$4^4 \pmod{7} = 4$

$4^5 \pmod{7} = 2$

$4^6 \pmod{7} = 1$

smallest positive integer $K = 3$

```
// Java program to implement multiplicative order
import java.io.*;

class Multiplicative_order {

    // fuction for GCD
    static int GCD(int a, int b) {

        if (b == 0)
            return a;

        return GCD(b, a % b);
    }

    // Function return smallest +ve integer that holds condition  $A^k \pmod{N} = 1$ 
    static int multiplicativeOrder(int A, int N) {

        if (GCD(A, N) != 1)
            return -1;

        // result store power of A that rised to the power N-1
        int result = 1;

        int K = 1;

        while (K < N) {

            // modular arithmetic
            result = (result * A) % N;

            // return samllest +ve integer
            if (result == 1)
                return K;
        }
    }
}
```

```
        // increment power
        K++;
    }

    return -1;
}

// driver program to test above function
public static void main(String args[]) {

    int A = 4, N = 7;

    System.out.println(multiplicativeOrder(A, N));
}
}
```

Primality Test (Fermat Method)

Given a number n , check if it is prime or not.

Fermat's Little Theorem:

If n is a prime number, then for every a , $1 < a < n-1$,

$$a^{n-1} \equiv 1 \pmod{n}$$

OR

$$a^{n-1} \% n = 1$$

Example: Since 5 is prime, $2^4 \equiv 1 \pmod{5}$ [or $2^4 \% 5 = 1$],
 $3^4 \equiv 1 \pmod{5}$ and $4^4 \equiv 1 \pmod{5}$

Since 7 is prime, $2^6 \equiv 1 \pmod{7}$,

$3^6 \equiv 1 \pmod{7}$, $4^6 \equiv 1 \pmod{7}$

$5^6 \equiv 1 \pmod{7}$ and $6^6 \equiv 1 \pmod{7}$

Refer [this](#) for different proofs.

```
import java.io.*;
import java.math.*;

class Fermat_primary_test {

    /* Iterative Function to calculate
    // (a^n)%p in O(logy) */
    static int power(int a,int n, int p)
    {
        // Initialize result
        int res = 1;

        // Update 'a' if 'a' >= p
        a = a % p;

        while (n > 0)
        {
            // If n is odd, multiply 'a' with result
            if ((n & 1) == 1)
                res = (res * a) % p;

            // n must be even now
            n = n >> 1; // n = n/2
            a = (a * a) % p;
        }
        return res;
    }

    // If n is prime, then always returns true,
    // If n is composite than returns false with
    // high probability Higher value of k increases
    // probability of correct result.
    static boolean isPrime(int n, int k)
    {
        // Corner cases
        if (n <= 1 || n == 4) return false;
        if (n <= 3) return true;

        // Try k times
        while (k > 0)
        {
            // Pick a random number in [2..n-2]
            // Above corner cases make sure that n > 4
            int a = 2 + (int)(Math.random() % (n - 4));

            // Fermat's little theorem
            if (power(a, n - 1, n) != 1)
                return false;

            k--;
        }

        return true;
    }
}
```

```
}

// Driver Program
public static void main(String args[])
{
    int k = 3;
    if(isPrime(11, k))
        System.out.println(" true");
    else
        System.out.println(" false");
    if(isPrime(15, k))
        System.out.println(" true");
    else
        System.out.println(" false");
}
}
```

Efficient method to print all prime factors of a given number

Given a number n , write an efficient function to print all prime factors of n .

- For example, if the input number is 12, then output should be “2 2 3”.
- And if the input number is 315, then output should be “3 3 5 7”.
- Following are the steps to find all prime factors.
 - 1) While n is divisible by 2, print 2 and divide n by 2.
 - 2) After step 1, n must be odd. Now start a loop from $i = 3$ to square root of n . While i divides n , print i and divide n by i . After i fails to divide n , increment i by 2 and continue.
 - 3) If n is a prime number and is greater than 2, then n will not become 1 by above two steps. So print n if it is greater than 2.

```
// Program to print all prime factors
import java.io.*;
import java.lang.Math;

class Factors
{
    // A function to print all prime factors of a given number n
    public static void primeFactors(int n)
    {
        // Print the number of 2s that divide n
        while (n%2==0)
        {
            System.out.print(2 + " ");
            n /= 2;
        }

        // n must be odd at this point. So we can skip one element (Note i = i + 2)
        for (int i = 3; i <= Math.sqrt(n); i+= 2)
        {
            // While i divides n, print i and divide n

```

```
        while (n%i == 0)
        {
            System.out.print(i + " ");
            n /= i;
        }
    }
    if (n > 2)
        System.out.print(n);
}
public static void main (String[] args)
{
    int n = 315;
    primeFactors(n);
}
}
```

RSA Algorithm

```
import java.io.DataInputStream;
import java.io.IOException;
import java.math.BigInteger;
import java.util.Random;

public class RSA
{
    private BigInteger p;
    private BigInteger q;
    private BigInteger N;
    private BigInteger phi;
    private BigInteger e;
    private BigInteger d;
    private int bitlength = 1024;
    private Random r;

    public RSA()
    {
        r = new Random();
        p = BigInteger.probablePrime(bitlength, r);
        q = BigInteger.probablePrime(bitlength, r);
        N = p.multiply(q);
        phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
        e = BigInteger.probablePrime(bitlength / 2, r);
        while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi) < 0)
        {
            e.add(BigInteger.ONE);
        }
        d = e.modInverse(phi);
    }

    public RSA(BigInteger e, BigInteger d, BigInteger N)
    {
        this.e = e;
        this.d = d;
        this.N = N;
    }

    @SuppressWarnings("deprecation")
    public static void main(String[] args) throws IOException
    {
        RSA rsa = new RSA();
        DataInputStream in = new DataInputStream(System.in);
        String teststring;
        System.out.println("Enter the plain text:");
        teststring = in.readLine();
        System.out.println("Encrypting String: " + teststring);
        System.out.println("String in Bytes: "
            + bytesToString(teststring.getBytes()));
        // encrypt
        byte[] encrypted = rsa.encrypt(teststring.getBytes());
        // decrypt
        byte[] decrypted = rsa.decrypt(encrypted);
        System.out.println("Decrypting Bytes: " + bytesToString(decrypted));
        System.out.println("Decrypted String: " + new String(decrypted));
    }
}
```

```
private static String bytesToString(byte[] encrypted)
{
    String test = "";
    for (byte b : encrypted)
    {
        test += Byte.toString(b);
    }
    return test;
}

// Encrypt message
public byte[] encrypt(byte[] message)
{
    return (new BigInteger(message)).modPow(e, N).toByteArray();
}

// Decrypt message
public byte[] decrypt(byte[] message)
{
    return (new BigInteger(message)).modPow(d, N).toByteArray();
}
}
```

Message authentication codes and hash functions

MAC in JAVA

The *Java Mac* (javax.crypto. **Mac** class can create a Message Authentication Code (MAC) from binary data. A MAC is a [message digest](#) which has been encrypted with a secret key.

- Creating a Mac instance is done using the `getInstance()` method.

```
Mac mac = Mac.getInstance("HmacSHA256");
```

- Once created, the Java Mac instance must be initialized. You initialize the Mac instance by calling its **init()** method passing as parameter the secret key to be used by the Mac instance.

```
byte[] keyBytes = new byte[]{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
String algorithm = "RawBytes";
SecretKeySpec key = new SecretKeySpec(keyBytes, algorithm);

mac.init(key);
```

- To calculate a MAC value you call the Mac **update()** or **doFinal()** method.
- If you only have a single block of data to calculate the MAC for, you can call `doFinal()` directly, like this:

```
byte[] data = "abcdefghijklmnopqrstuvxyz".getBytes("UTF-8");
byte[] macBytes = mac.doFinal(data);
```

If you have multiple blocks of data to calculate the MAC for, e.g. if you are reading a file block by block, then you must call the `update()` method with each block, and finish with a call to `doFinal()`.

```
byte[] data = "abcdefghijklmnopqrstuvxyz".getBytes("UTF-8");
byte[] data2 = "0123456789".getBytes("UTF-8");

mac.update(data);
mac.update(data2);

byte[] macBytes = mac.doFinal();
```



```
import java.security.Key;
import java.security.SecureRandom;

import javax.crypto.KeyGenerator;
import javax.crypto.Mac;

public class MAC {
    public static void main(String args[]) throws Exception{
        //Creating a KeyGenerator object
        KeyGenerator keyGen = KeyGenerator.getInstance("DES");

        //Creating a SecureRandom object
        SecureRandom secRandom = new SecureRandom();

        //Initializing the KeyGenerator
        keyGen.init(secRandom);

        //Creating/Generating a key
        Key key = keyGen.generateKey();

        //Creating a Mac object
        Mac mac = Mac.getInstance("HmacSHA256");

        //Initializing the Mac object
        mac.init(key);

        //Computing the Mac
        String msg = new String("hellow my name is ayad");
        byte[] bytes = msg.getBytes();
        byte[] macResult = mac.doFinal(bytes);

        System.out.println("Mac result:");
        System.out.println(new String(macResult));
    }
}
```

Hashing (message digest) in Java

To calculate cryptographic hashing value in Java, **MessageDigest Class** is used, under the package **java.security**.

- ✓ **MessageDigest** Class provides following cryptographic hash function to find hash value of a text, they are:

To create a Java MessageDigest instance you call the static **getInstance()** method of the MessageDigest class.

```
MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");
```

The Java Cryptography API supports the following message digest algorithms:

Algorithm Name
MD2
MD5
SHA-1
SHA-256
SHA-384
SHA-512

If you have a single block of data to calculate a message digest from, use the **digest()** method.

```
byte[] data1 = "0123456789".getBytes("UTF-8");  
  
MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");  
byte[] digest = messageDigest.digest(data1);
```

If you have multiple blocks of data to include in the same message digest, call the **update()** method and finish off with a call to **digest()**.

```
byte[] data1 = "0123456789".getBytes("UTF-8");  
byte[] data2 = "abcdefghijklmnopqrstuvwxyz".getBytes("UTF-8");  
  
MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");  
messageDigest.update(data1);  
messageDigest.update(data2);  
  
byte[] digest = messageDigest.digest();
```

MD5 Hash

```
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

// Java program to calculate MD5 hash value
public class MD5 {
    public static String getMd5(String input)
    {
        try {

            // Static getInstance method is called with hashing MD5
            MessageDigest md = MessageDigest.getInstance("MD5");

            // digest() method is called to calculate message digest
            // of an input digest() return array of byte
            byte[] messageDigest = md.digest(input.getBytes());

            // Convert byte array into signum representation
            BigInteger no = new BigInteger(1, messageDigest);

            // Convert message digest into hex value
            String hashtext = no.toString(16);
            while (hashtext.length() < 32) {
                hashtext = "0" + hashtext;
            }
            return hashtext;
        }

        // For specifying wrong message digest algorithms
        catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
    }

    // Driver code
    public static void main(String args[]) throws NoSuchAlgorithmException
    {
        String s = "GeeksForGeeks";
        System.out.println("Your HashCode Generated by MD5 is: " + getMd5(s));
    }
}
```

SHA-512 Hash In Java

```
// Java program to calculate SHA-512 hash value

import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class GFG {
    public static String encryptThisString(String input)
    {
        try {
            // getInstance() method is called with algorithm SHA-512
            MessageDigest md = MessageDigest.getInstance("SHA-512");

            // digest() method is called
            // to calculate message digest of the input string
            // returned as array of byte
            byte[] messageDigest = md.digest(input.getBytes());

            // Convert byte array into signum representation
            BigInteger no = new BigInteger(1, messageDigest);

            // Convert message digest into hex value
            String hashtext = no.toString(16);

            // Add preceding 0s to make it 32 bit
            while (hashtext.length() < 32) {
                hashtext = "0" + hashtext;
            }

            // return the HashText
            return hashtext;
        }

        // For specifying wrong message digest algorithms
        catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
    }

    // Driver code
    public static void main(String args[]) throws NoSuchAlgorithmException
    {
        System.out.println("HashCode Generated by SHA-512 for: ");

        String s1 = "GeeksForGeeks";
        System.out.println("\n" + s1 + " : " + encryptThisString(s1));

        String s2 = "hello world";
        System.out.println("\n" + s2 + " : " + encryptThisString(s2));
    }
}
```

Java Signature

The *Java **Signature** class* (`java.security.Signature`) can create a digital signature for binary data.

- A digital signature is a message digest encrypted with a private key of a private / public key pair. Anyone in possession of the public key can verify the digital signature.
- You create a `Signature` instance by calling the static **`getInstance()`** method.

```
Signature signature = Signature.getInstance("SHA256WithDSA");
```

- We can initialize a `Signature` instance by calling its **`init()`** method.

```
SecureRandom secureRandom = new SecureRandom();
KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("DSA");
KeyPair keyPair = keyPairGenerator.generateKeyPair();

signature.initSign(keyPair.getPrivate(), secureRandom);
```

- The *Java `KeyPairGenerator` class* (`java.security.KeyPairGenerator`) is used to generate asymmetric encryption / decryption key pairs.
- Creating a `KeyPairGenerator` instance is done by calling the method **`getInstance()`** method.

```
KeyPairGenerator keyPairGenerator =
    KeyPairGenerator.getInstance("DSA");
```

- To generate a `KeyPair` with a `KeyPairGenerator` you call the `generateKeyPair()` method.

```
KeyPair keyPair = keyPairGenerator.generateKeyPair();
```

- Create a digital signature by calling the **`update()`** method one or more times, finishing with a call to **`sign()`**.

```
byte[] data = "abcdefghijklmnopqrstuvwxyz".getBytes("UTF-8");
signature.update(data);

byte[] digitalSignature = signature.sign();
```

- If you want to **verify** a digital signature created by someone else, you must initialize a `Signature` instance into **verification** mode (instead of signature mode).

```
Signature signature = Signature.getInstance("SHA256WithDSA");
```

```
signature.initVerify(keyPair.getPublic());
```

- Once initialized into verification mode you can use the Signature instance to verify a digital signature.

```
byte[] data2 = "abcdefghijklmnopqrstuvxyz".getBytes("UTF-8");  
signature2.update(data2);  
  
boolean verified = signature2.verify(digitalSignature);
```

Digital Signature program:

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.Signature;

import java.util.Scanner;

public class SignatureValidation {
    public static void main(String args[]) throws Exception{
        //Creating KeyPair generator object
        KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("DSA");

        //Initializing the key pair generator
        keyPairGen.initialize(2048);

        //Generate the pair of keys
        KeyPair pair = keyPairGen.generateKeyPair();

        //Getting the privatekey from the key pair
        PrivateKey privKey = pair.getPrivate();

        //Creating a Signature object
        Signature sign = Signature.getInstance("SHA256withDSA");

        //Initializing the signature
        sign.initSign(privKey);
        byte[] bytes = "Hello how are you".getBytes();

        //Adding data to the signature
        sign.update(bytes);

        //Calculating the signature
        byte[] signature = sign.sign();

        //Initializing the signature
        sign.initVerify(pair.getPublic());
        sign.update(bytes);

        //Verifying the signature
        boolean bool = sign.verify(signature);

        if(bool) {
            System.out.println("Signature verified");
        } else {
            System.out.println("Signature failed");
        }
    }
}
```

RC4 Cipher

```
import javax.crypto.spec.*;
import java.security.*;
import javax.crypto.*;

public class RC4
{
    private static String algorithm = "RC4";

    public static void main(String []args) throws Exception {
        String toEncrypt = "The shorter you live, the longer you're dead!";

        System.out.println("Encrypting...");
        byte[] encrypted = encrypt(toEncrypt, "password");

        System.out.println("Decrypting...");
        String decrypted = decrypt(encrypted, "password");

        System.out.println("Decrypted text: " + decrypted);
    }

    public static byte[] encrypt(String toEncrypt, String key) throws Exception {
        SecureRandom sr = new SecureRandom(key.getBytes());
        KeyGenerator kg = KeyGenerator.getInstance(algorithm);
        kg.init(sr);
        SecretKey sk = kg.generateKey();

        // create an instance of cipher
        Cipher cipher = Cipher.getInstance(algorithm);

        // initialize the cipher with the key
        cipher.init(Cipher.ENCRYPT_MODE, sk);

        // encrypt!
        byte[] encrypted = cipher.doFinal(toEncrypt.getBytes());

        return encrypted;
    }

    public static String decrypt(byte[] toDecrypt, String key) throws Exception {
        // create a binary key from the argument key (seed)
        SecureRandom sr = new SecureRandom(key.getBytes());
        KeyGenerator kg = KeyGenerator.getInstance(algorithm);
        kg.init(sr);
        SecretKey sk = kg.generateKey();

        // do the decryption with that key
        Cipher cipher = Cipher.getInstance(algorithm);
        cipher.init(Cipher.DECRYPT_MODE, sk);
        byte[] decrypted = cipher.doFinal(toDecrypt);

        return new String(decrypted);
    }
}
```


Diffie-Hellman Exchange protocol

```
import java.util.*;

class DH
{
    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter modulo(p)");
        int p=sc.nextInt();
        System.out.println("Enter primitive root of "+p);
        int g=sc.nextInt();
        System.out.println("Choose 1st secret no(Alice)");
        int a=sc.nextInt();
        System.out.println("Choose 2nd secret no(BOB)");
        int b=sc.nextInt();

        int A = (int)Math.pow(g,a)%p;
        int B = (int)Math.pow(g,b)%p;

        int S_A = (int)Math.pow(B,a)%p;
        int S_B = (int)Math.pow(A,b)%p;

        if(S_A==S_B)
        {
            System.out.println("ALice and Bob can communicate with each other!!!");
            System.out.println("They share a secret no = "+S_A);
        }

        else
        {
            System.out.println("ALice and Bob cannot communicate with each other!!!");
        }
    }
}
```