Unit – 2 Basics of JAVA

2.1 Variables, Data Types, Casting, Operators

2.2 Compiling and running Java program

2.3 Command line arguments

2.4 Accepting input from console

2.5Arrays

Prof. A. P. Chaudhari (M.Sc., SET)
HOD, Department of Computer Science
SVS's Dadasaheb Rawal College, Dondaicha

2.1) – a) Variables:

A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.

Variable is name of reserved area allocated in memory. In other words, it is a name of memory location. Variable name can be chosen by the programmer in a meaningful way so as reflect what it represents in the program.

It is a combination of "vary + able" that means its value can be changed. It may take different value at different times during the execution of the program.

In Java, there are different types of variables, for example:

String - stores text, such as "Hello". String values are surrounded by double quotes

int - stores integers (whole numbers), without decimals, such as 123 or -123

float - stores floating point numbers, with decimals, such as 19.99 or -19.99

2.1) – a) Variables:

char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes

boolean - stores values with two states: true or false

Declaring Variable:

To create a variable, you must specify the type and assign it a value:

Syntax:

data_type variable_name = value;

Where *data_type* is one of Java's data types and *variable_name* is the name of the variable. The *equal sign* is used to assign *values* to the variable.

e.g.: string name = “Atharv”;

int rollno = 101;

float per = 92.67;

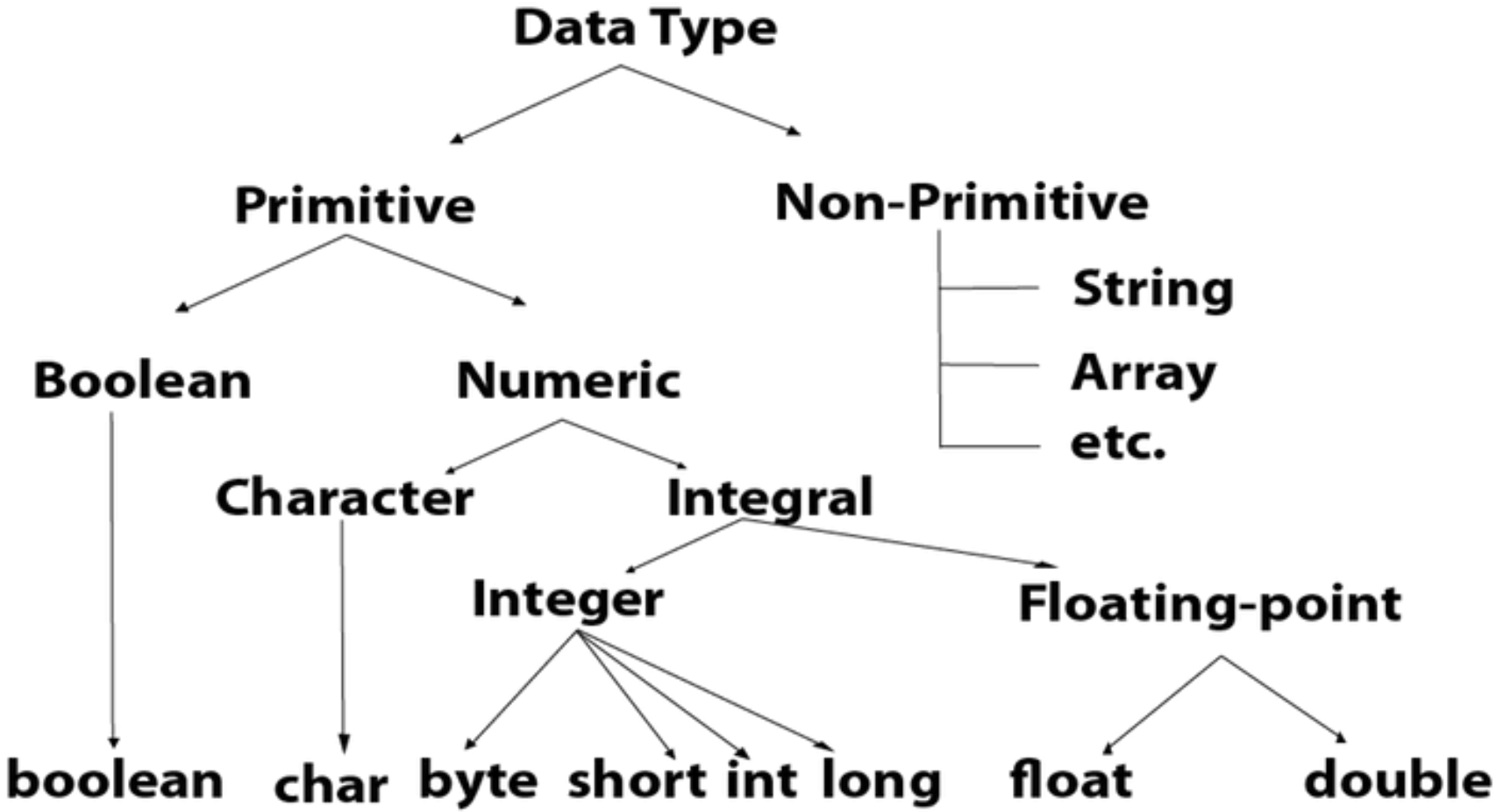
2.1) – b) Data Types

The type of value that a variable can hold is called data type. Every variable in Java has a data type. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. Data types specify the different sizes and values that can be stored in the variable.

There are two types of data types in Java:

1) **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2) **Non-primitive data types:** The non-primitive data types include Classes, Interfaces and Arrays.

2.1) – b) Data Types



2.1) – b) Data Types

Primitive data types: There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named by a keyword. These are the most basic data types available in Java language.

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

2.1) – c) Casting

It is a process of converting from one data type to another data type. Casting of data types in Java is also known as type casting in Java. There are two types of casting-

i) Implicit Casting
ii) Explicit Casting

i) Implicit Casting (Widening)

In case of implicit casting, source is smaller than destination and no need to do casting, JVM (Java Virtual Machine) will do required casting.

byte → short → int → long → float → double

e.g. float a;

int b;

a = b;

2.1) – c) Casting

ii) Explicit Casting (Narrowing)

In case of explicit casting, destination is smaller than source, we need to do casting explicitly. JVM (Java Virtual Machine) will not do any casting.

double → float → long → int → short → byte

Syntax: data_type variable1 = (data_type) variable2;

e.g. float a;

int b = (int) a;

2.1) – d) Operators

Operator in Java is a symbol which is used to perform operations on variables and values. It takes one or more arguments and operates on them to produce a result. The constants, variables or expression on which operator operates are called as operands. e.g. 6 + 8 here + is the operator and 6 and 8 are operands.

There are many types of operators in Java which are given below:

Operator Type	Category	Precedence
Unary	postfix	*expr*++ *expr*--
	prefix	++*expr* --*expr* +*expr* -*expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=

2.1) – d) Operators

Operator Type	Category	Precedence
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=
Special Operator	Instanceof Operator	
	Member Selection Operator	

2.1) – d) Operators

1) **Unary Operator:** The Java unary operators require only one operand.

Unary operators are used to perform various operations i.e.:

- incrementing/decrementing a value by one
- inverting the value of a boolean

e.g:

```
int x=10;
System.out.println(x++); //10 (11)
System.out.println(++x); //12
System.out.println(x--);  //12 (11)
System.out.println(--x);  //10
boolean c=true;
boolean d=false;
System.out.println(!c);   //false (opposite of boolean value)
System.out.println(!d);   //true
```

2.1) – d) Operators

2) **Arithmetic Operator:** Java arithmetic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.

e.g:

```
int a=25;
int b=4;
System.out.println(a + b);          //29
System.out.println(a - b);          //21
System.out.println(a * b);          //100
System.out.println(a / b);          //6
System.out.println(a % b);          //1       (% - Modulo Operator)
```

2.1) – d) Operators

3) Shift Operator: Shift operator works on bits and performs bit-by-bit operation. Assume if a = 60; now in binary format they will be as follows −

a = 0011 1100

<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

2.1) – d) Operators

4) **Bitwise Operator:** Java defines several bitwise operators, which can be applied to the integer types- long, int, short, char, and byte. Bitwise operator works on bits and performs bit-by-bit operation.

Assume if a = 60 and b = 13; now in binary format they will be as follows −

a = 0011 1100

b = 0000 1101

& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001

2.1) – d) Operators

5) Relational Operator: There are following relational operators supported by Java language. Assume variable **A = 10** and variable **B = 20**, then −

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

2.1) – d) Operators

6) Logical Operator: The following table lists the logical operators −

Assume Boolean variables **A holds true** and variable **B holds false**, then -

Operator	Description	Example
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
|| (logical or)	Called Logical OR Operator. If any one of the two operands are non-zero, then the condition becomes true.	(A || B) is true
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

2.1) – d) Operators

7) **Ternary Operator:** Ternary operator is also known as the Conditional operator. Java Ternary operator is used as one liner replacement for if-then-else statement and used a lot in Java programming. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide, which value should be assigned to the variable. The operator is written as −

variable x = (expression) ? value if true : value if false

e.g:

```
int a=2;
int b=5;
int min = (a < b) ? a : b;
System.out.println(min);
```

2.1) – d) Operators

8) Assignment Operator: Following are the assignment operators supported by Java language −

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C – A
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A

2.1) – d) Operators

8) Assignment Operator: Following are the assignment operators supported by Java language −

Operator	Description	Example
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

2.1) – d) Operators

9) Special Operators:

Java supports special operators such as instance of operator and member selection operator (Dot operator).

a) Instanceof Operator:

The instanceof operator compares an object to a specified type. It is used to test whether the object is an instance of the specified type (class, subclass or interface). This is also known as comparison operator it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that have null value, it returns false.

e.g.: String name = “Vivekanand”;

Boolean result = name instanceof String;

2.1) – d) Operators

b) Member Selection (Dot) Operator:

The dot operator is used to access class members. i.e. Instance variable and methods of class object.

e.g.:

```
s1.roll_no;
s1.putdata();
```

It is also used to access classes and sub packages from a package.

2.2 Compiling and running Java program

Before compiling the program, user first has to write Java program using any text editor called source file. Save the file with .java extension. Now, compile the source file using the compiler ‘javac’ with the name of source file as below –

javac filename.java

The Java compiler translates the source file into instructions that the Java Virtual Machine can understand. The instructions contained within this file are known as byte code. After compilation it creates the .class file.

To run the Java program we need to use Java interpreter ‘java’ at the command prompt. We type as following:

java filename

The interpreter starts the execution from the main method in the program.

2.2 Compiling and running Java program

Step – I) Open command prompt

Step – II) Write command as follows:

Microsoft Windows [Version 6.1.7600]

Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Vivekanand>cd..

C:\Users>cd..

C:\>cd jdk1.4

C:\jdk1.4>cd bin

C:\jdk1.4\bin>edit

2.2 Compiling and running Java program

Step – III) Write program in editor (Source Program)

```
import java.io.*;
class demo33
{
	public static void main(String args[])
	{
		System.out.println("Atharv");
		System.out.println("Dondaicha");
	}
}
```

Step – IV) Compile source code by following command-

```
C:\jdk1.4\bin>javac demo33.java
```

Step – V) Run program by following command-

```
C:\jdk1.4\bin>java demo33
```

Output -

```
Atharv
Dondaicha
```

2.3 Command Line Arguments

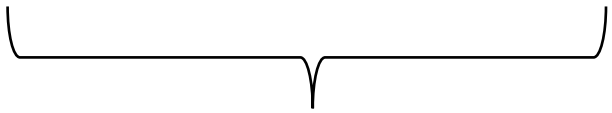
Command line arguments are the parameters that are passed through the command prompt at the time of execution of program. By default every command line argument will be treated as String value and those are stored in a String array of main () method.

public static void main (String args[ ])

Here, args[ ] is declared as an array of string. Any arguments supplied in the command line at the time of execution are passed to the array args[ ] as its element. We can simply access the array elements and use them in a program.

Syntax:

java filename value1, value2, ……. Value_n



Command line arguments

2.3 Command Line Arguments

Program – 1

```
import java.io.*;
class demo34
{
	public static void main(String args[])
	{
		System.out.println("Number of arguments are:"+args.length);
		for(int i=0; i<args.length; i++)
		{
			System.out.println(args[i]);
		}
	}
}
```

Compile: javac demo34.java

Run: java demo34 C C++ Java VB

Output: Number of arguments are:4

C

C++

Java

VB

2.3 Command Line Arguments

Program – 2

```
import java.io.*;
class add
{
	public static void main(String args[])
	{
		int a, b, c;
		a = Integer.parseInt(args[0]);
		b = Integer.parseInt(args[1]);
		c = a + b;
		System.out.println("Adition is: "+c);
	}
}
```

Compile: javac add.java

Run: java add 25 15

Output: Addition is: 40

2.3 Command Line Arguments

Program – 3

```
import java.io.*;
class student
{
	public static void main(String args[])
	{
		int a;
		String b,c;
		float d;
		a = Integer.parseInt(args[0]);
		b = args[1];
		c = args[2];
		d = Float.parseFloat(args[3]);

		System.out.println("Roll No:"+a);
		System.out.println("Name:"+b);
		System.out.println("Class:"+c);
		System.out.println("Percentage:"+d);
	}
}
```

Compile: javac student.java

Run: java student 101 Atharv TYBSc 92.67

Output: Roll No: 101

Name: Atharv

Class: TYBSc

Percentage: 92.67

2.4 Accepting input from console

Java **DataInputStream** class allows an application to read primitive data from the input stream in a machine-independent way.

Program – 1

```
import java.io.*;
class demo35
{
	public static void main(String args[])throws IOException
	{
		String sname;
		DataInputStream d = new DataInputStream(System.in);
		System.out.println("Enter your name:");
		sname = d.readLine();
		System.out.println("Your Name:"+sname);
	}
}
```

Compile: javac demo35.java

Run: java demo35

Output: Enter your name: Vivekanand

Your Name: Vivekanand

2.4 Accepting input from console

Program – 2

```
import java.io.*;
class stud1
{
	public static void main(String args[])throws IOException
	{
		int rno;
		String sname;
		float per;
		DataInputStream d = new DataInputStream(System.in);
		System.out.println("Enter Roll Number:");
		rno = Integer.parseInt(d.readLine());
		System.out.println("Enter Name:");
		sname = d.readLine();
		System.out.println("Enter Percentage:");
		per = Float.parseFloat(d.readLine());
		System.out.println("Roll No:"+rno);
		System.out.println("Name:"+sname);
		System.out.println("Percentage:"+per);
	}
}
```

Compile: javac stud1.java
Run: java stud1
Output:

```
Enter Roll Number: 101
Enter Name: ABC
Enter Percentage: 88.96
Roll No: 101
Name: ABC
Percentage: 88.96
```

2.4 Accepting input from console

The **Scanner** class is used to get user input, and it is found in the java.util package. To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation. In our example, we will use the nextLine() method, which is used to read Strings:

Program – 3

```
import java.util.*;
class demo37
{
        public static void main(String args[])
        {
                String name;
                Scanner in = new Scanner(System.in);
                System.out.print("Enter your name: ");
                name = in.nextLine();
                System.out.println("Name is: " + name);
                in.close();
        }
}
```

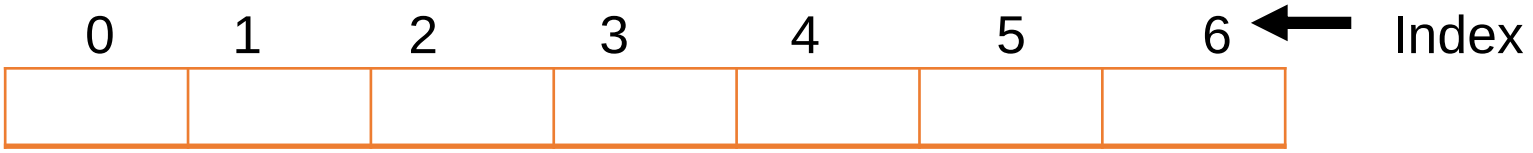
Compile: javac demo37.java

Run: java demo37

Output: Enter your name: MNO
Name is: MNO

2.5 Arrays

An array is collection of continuous similar item that share the common name. Array is a container object that holds value of homogeneous type. This means that all elements in the array have the same data type. A position in an array is indicated by a non negative integer value called as index. An element at the given position is accessed by using this index. The size of array is fixed and cannot increase to accommodate more elements.



Types of Array:

1) One dimensional array

2) Two dimensional array

2.5 Arrays

1) One Dimensional Array

Array which requires only one index/subscript to refer its elements is called as one dimensional array.

Syntax: **data_type array_name [ ] = new data_type[size];**

e.g.: int marks [ ] = new int [10];

Initializing Array:

An element of an array must be given a value before it is used. With Java compilers all variables including array elements are given default initial values. All number elements are initialized to 0. We can obtain the length of array using array_name.length

e.g.: marks[0]=86;

We can also initialize arrays automatically in the same way as the ordinary variables when they are declared as below:

data_type array_name [ ] = {list_of_values};

e.g.: int marks [ ] = {70,86,93,48,66};

2.5 Arrays

Program 1

```
import java.io.*;
class demo38
{
	public static void main(String args[])
	{
		int marks[]={70,86,93,48,66};
		System.out.println("Number of elements in array are:“ +marks.length);
		System.out.println("Elements in array marks:");
		for(int i=0; i<marks.length; i++)
		{
			System.out.println(marks[i]);
		}
	}
}
```

Output:

```
Number of elements in array are:5
Elements in array marks:
70  86  93  48  66
```

2) Two Dimensional Array (Multi-dimensional Array)

Array which requires two index/subscripts to refer its element is called as two dimensional array. Many times it is required to manipulate the data in table format or in matrix format which contains row and columns. In these cases, we have to use two dimensional array.

Syntax: data_type array_name [ ] [ ] = new data_type [row] [column];

OR

data_type [ ] [ ] array_name = new data_type [row] [column];

e.g. int mat [ ] [ ] = new int [3] [4];

This will create total 12 storage locations for two dimensional arrays. We can store the values in each of these memory locations by referring their respective row and column number as follows:

mat [2] [3] = 25;

2.5 Arrays

Like one dimension arrays, two dimensional array can also be initialize at compile time as follows:

```
int mat [2] [2] = {4,7,3,9};
```

We can also initialization done row by row as follows:

```
int mat [2] [2] = { {4,2}, {3,9} };
```

2.5 Arrays

Program 1

```
import java.io.*;
class demo39
{
	public static void main(String args[])
	{
		int mat [][] = {{1,2,3}, {4,5,6}, {7,8,9}};
		System.out.println("The given matrix is:");
		for(int i=0; i<3; i++)
		{
			for(int j=0; j<3; j++)
			{
				System.out.print(mat[i][j]+"\t");
			}
			System.out.println();
		}
	}
}
```

Output:

The given matrix is:

1	2	3
4	5	6
7	8	9