Chapter 3

BASIC ELEMENTS OF JAVA PROGRAMMING

Disclaimer:
This lecture material is prepared using the book *JAVA: The Complete Reference* by *Herbert Schildt*

CONTENTS

1. **First Java Program**
2. **Data Types**
3. **Identifiers, Literals, Keywords**
4. **Operators**
   - Arithmetic, Assignment, Comparison, Logical, Conditional, Bit wise, InstanceOf, arrow.
5. **Type casting**
   - Implicit or widening casting
   - Explicit or narrowing casting
6. **Type promotion**
   - Type promotion rules
7. **Control statements**
   - Conditional statement (if, if else, if else if, nested if else, switch case etc.)
   - Loop statement(for, while, do while)
   - Jump (break, continue, return)
8. **Taking input from keyboard**
   - Command Line Argument method
   - Using Scanner Class
   - Using BufferedReader class

Chapter 3 BASIC ELEMENTS OF JAVA PROGRAMMING

First JAVA Program

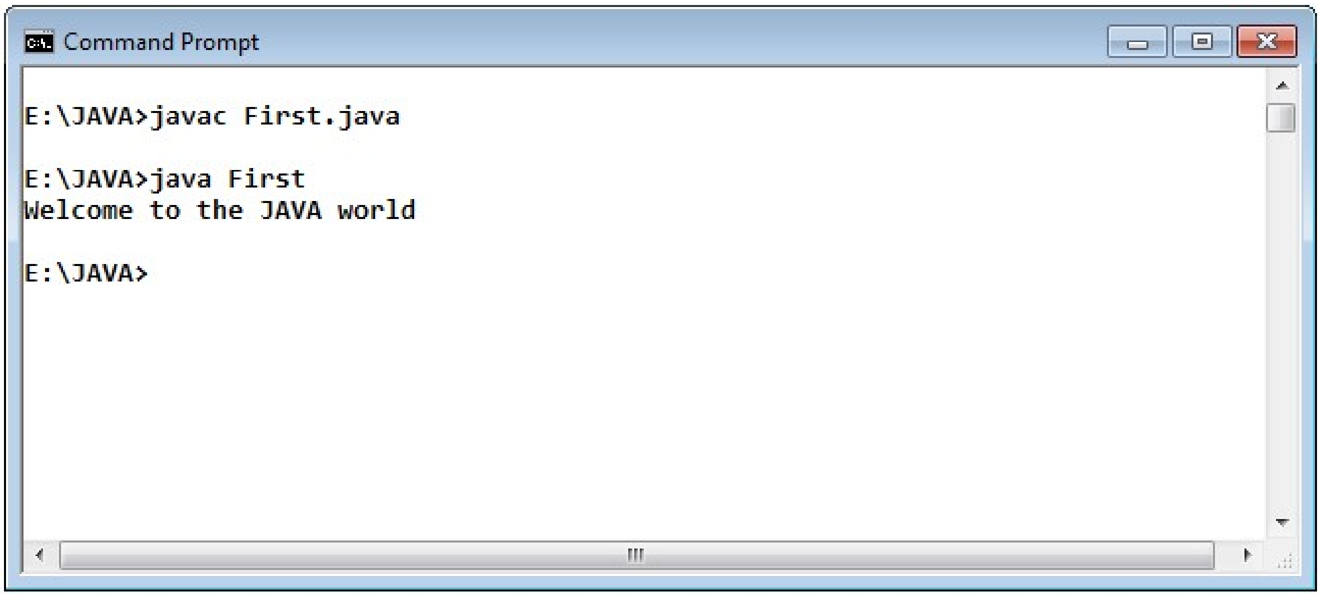
Let us write our first JAVA program which will display the message: *Welcome to the JAVA world.*

Program 1 Write a program to print a message "Welcome to the JAVA world".

Solution:

```
// First.java
class First
{
        public static void main (String args[])
        {
        System.out.println("Welcome to the JAVA world ");
        }
}
```

Output:



```
E:\JAVA>javac First.java

E:\JAVA>java First
Welcome to the JAVA world

E:\JAVA>
```

How to compile & execute the above program

To compile the above program, use the following syntax on the command line:

javac *java_filename*.java

Here we will use the following command to compile our program:

javac First.java

If successfully compiled, the compiler creates a file called **First.class** that contains byte code version of our java program.

Now to run a java program, use the following syntax:

javac *java_filename*

Here we will use following command to run the program:

java First

Explanation of the above program

class First

In a purely object oriented programming system, all our codes remain inside a class definition. Therefore, our program begins with a class definition. Here class is a keyword used to define a class & First is the name of our class.

{

Here this opening brace indicates the beginning of the class definition block.

public static void main(String args[])

All Java applications begin execution from main ( ) function. (This is just like C/C++.)

- The **public** keyword is an access specifier, which allows the programmer to control the visibility of class members. When a class member is preceded by public, then that member may be accessed by code outside the class in which it is declared.
- The keyword **static** allows main ( ) to be called without having to instantiate a particular instance of the class.
- The keyword **void** simply tells the compiler that main ( ) does not return a value. As stated, main () is the method called when a Java application begins. Keep in mind that Java is case-sensitive. Thus, Main is different from main.
- **String** args**[ ]** declares a parameter named args, which is an array of instances of the class String. This args parameter is essential when we enter some input through the command line while executing a program.

{

Here this opening brace indicates the beginning of main method definition block.

System.out.println("Welcome to the JAVA world ");

This line outputs the string “Welcome to the JAVA world” followed by a new line on the screen. Output is actually accomplished by the built-in println() method. In this case, println( ) displays the string which is passed to it.

```
}
```

Here this closing brace indicates the end of main method definition block.

```
}
```

Here this closing brace indicates the end of the class definition block.

Java Is a Strongly Typed Language

It is important to state at the outset that Java is a strongly typed language. Main reasons are-

- Every variable has a type, every expression has a type, and every type is strictly defined.
- All assignments, whether explicit or via parameter passing in method calls, are checked for type compatibility.

Data Types

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

- **Primitive data types:** Java defines eight primitive types of data. The primitive data types (or simple data types) include **boolean, char, byte, short, int, long, float** and **double** which are put in four distinct groups as shown below.
- **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

Java Primitive Data Types

Table. Primitive data types in JAVA

Category	Data Type	Default Value	Size	Range
Boolean	boolean	false	1 bit	
Character	char	'\u0000'	2 byte	
Integer	byte	0	1 byte	-128 to 127 (-2^7 to 2^7-1)
	short	0	2 byte	-32768 to 32767 (-2^{15} to 2^{15}-1)

	int	0	4 byte	-2,147,483,648 to -2,147,483,647 (-2^{31} to 2^{31}-1)
	long	0L	8 byte	-9,223,372,036,854,775,808 to -9,223,372,036,854,775,807 (-2^{63} to 2^{63}-1)
Float/ Real	float	0.0f	4 byte	
	double	0.0d	8 byte	

Note: In general, if n bits are required for a data type, then its range is -2^{n-1} to $2^{n-1}-1$.

Question 1. Although using a byte or short would be more efficient than using an int in situations in which the larger range of an int is not needed but this is not the case. Justify your answer.

Answer- The reason is that when byte and short values are used in an expression, they are promoted to int when the expression is evaluated.

Question 2. Why char uses 2 byte in java unlike C/C++ which uses 1 byte of memory only and what is \u0000?

Answer- It is because java uses Unicode system, not ASCII (American Standard Code for Information Interchange) code system. Unicode system represents a character set which can be used to represent all languages available world. \u0000 is the lowest range of Unicode system.

JAVA Tokens

The smallest unit of code appearing in a program which is recognized by the compiler is called as token. A token in a java program are of following types:

- Identifiers
- Literals/ Constant
- Keywords
- Operators
- Comment
- Special characters

Let us discuss all the tokens in details.

Identifiers

These are names assigned to variables, methods, classes, packages & interfaces. Following are the rules to construct an identifier:

- An identifier contains alphabets $(\mathrm{A}-\mathrm{Z}, \mathrm{a}-\mathrm{z})$, digits $(0-9)$, underscore (_) or dollar ($) only.
- The first character in an identifier must be an alphabet or underscore (_) or dollar ($).
- Blank spaces are not allowed within an identifier.
- Keywords can't be identifiers.
- Identifier name are case sensitive.
- There is no limit on the length of a Java identifier.

Examples of some valid identifiers are $x, a1, \$a,\ India, \$myVariable$ etc.

Literals or constant

Literals in Java are a sequence of characters (digits, letters, and other characters) that represent constant values to be stored in variables. Java offers following types of literals:

- **Integer literals:** These are numeric values without a decimal point. Examples of integer literals are 1, -34, 1083 (in decimal) etc. Two other bases that can be used in integer literals are **octal** (base eight) and **hexadecimal** (base 16). Octal values are denoted in Java by a leading **0** and hexadecimal hexadecimal values are denoted by leading **0x** or **0X**. It is possible to assign an integer literal to one of Java's other integer types, such as byte or long, without causing a type mismatch error. When a literal value is assigned to a byte or short variable, no error is generated if the literal value is within the range of the target type. An integer literal can always be assigned to a long variable. However, to specify a long literal, we need to explicitly tell the compiler that the literal value is of type long. We do this by appending an upper- or lowercase L to the literal. For example, 0x7fffffffffffffL or 9223372036854775807L is the largest long. We can embed one or more underscores in an integer literal. Doing so makes it easier to read large integer literals. When the literal is compiled, the underscores are discarded. For example, given int x = 123_456_789; the value given to x will be 123,456,789.

  Note- In octal system permissible characters are 0-7 digits and in hexadecimal 0-9 digits, A, B, C, D, E, F are used to represent a number.

- **Floating literals:** These are numeric values with a decimal point and fractional component. Examples of floating literals are 1.97, -34.07, 2009.06 (standard notation), 1.56e+78, 1.09e-9 (scientific notation) etc. Floating-point literals in Java default to double precision. To specify a float literal, we must append an F or f to the constant. We can also explicitly specify a double literal by appending a D or d. Doing so is, of course, redundant. The default double type consumes 64 bits of storage, while the smaller float type requires only 32 bits.
- **Character literals:** Any single character enclosed in a single quote (‘ ’) is termed as a character literal. Examples of character literals are ‘A’, ‘5’, ‘ ’, ‘.’ etc.
- **String literals:** One or more character enclosed in a double quote (“ ”) is termed as a string literal. Examples of string literals are “Java Programming”, “5” etc.
- **Boolean literals:** These are literals which acquire Boolean values *true* or *false*.
- **Null literals:** The *null* in Java is a literal of the null type. It cannot be cast to a primitive type such as int. float etc, but can be cast to a reference type. Also, null does not have the value 0 necessarily.

Keywords

Keywords or Reserved words are the words that represent some predefined actions. These words are therefore not allowed to use as a variable names or objects. There are 49 keywords in java which are listed below:

abstract	Java abstract keyword is used to declare abstract class. Abstract class can provide the implementation of interface. It can have abstract and non-abstract methods.
boolean	Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.
break	Java break keyword is used to break loop or switch statement. It breaks the current flow of the program at specified condition.
byte	Java byte keyword is used to declare a variable that can hold an 8-bit data values.
case	Java case keyword is used to with the switch statements to mark blocks of text.
catch	Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.

char	Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters
class	Java class keyword is used to declare a class.
continue	Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.
default	Java default keyword is used to specify the default block of code in a switch statement.
do	Java do keyword is used in control statement to declare a loop. It can iterate a part of the program several times.
double	Java double keyword is used to declare a variable that can hold a 64-bit floating-point numbers.
else	Java else keyword is used to indicate the alternative branches in an if statement.
enum	Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.
extends	Java extends keyword is used to indicate that a class is derived from another class or interface.
final	Java final keyword is used to indicate that a variable holds a constant value. It is applied with a variable. It is used to restrict the user.
finally	Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether exception is handled or not.
float	Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.
for	Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some conditions become true. If the number of iteration is fixed, it is recommended to use for loop.
if	Java if keyword tests the condition. It executes the if block if condition is true.
implements	Java implements keyword is used to implement an interface.
import	Java import keyword makes classes and interfaces available and accessible to the current source code.

instanceof	Java instanceof keyword is used to test whether the object is an instance of the specified class or implements an interface.
int	Java int keyword is used to declare a variable that can hold a 32-bit signed integer.
interface	Java interface keyword is used to declare an interface. It can have only abstract methods.
long	Java long keyword is used to declare a variable that can hold a 64-bit integer.
Native	Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).
new	Java new keyword is used to create new objects.
null	Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.
package	Java package keyword is used to declare a Java package that includes the classes.
private	Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.
protected	Java protected keyword is an access modifier. It can be accessible within package and outside the package but through inheritance only. It can't be applied on the class.
public	Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.
return	Java return keyword is used to return from a method when its execution is complete.
short	Java short keyword is used to declare a variable that can hold a 16-bit integer.
static	Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is used for memory management mainly.
strictfp	Java strictfp is used to restrict the floating-point calculations to ensure portability.
super	Java super keyword is a reference variable that is used to refer parent class object. It can be used to invoke immediate parent class method.
switch	The Java switch keyword contains a switch statement that executes code based on test value. The switch statement tests the equality of a variable against multiple values.
synchronized	Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.

this	Java this keyword can be used to refer the current object in a method or constructor.
throw	The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exception. It is followed by an instance.
throws	The Java throws keyword is used to declare an exception. Checked exception can be propagated with throws.
transient	Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.
try	Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.
void	Java void keyword is used to specify that a method does not have a return value.
volatile	Java volatile keyword is used to indicate that a variable may change asynchronously.
while	Java while keyword is used to start a while loop. This loop iterates a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

Operators in Java

Operator in java is a symbol that is used to perform operations on one or more operands. Depending upon number of operands, operators are classified as shown in the below table.

Type of operator	Number of operands	Example
Unary	1	−, ++ , −− etc
Binary	2	+, − , ∗ , %, / etc
Ternary	3	?:

Various types of operators in java are

- Arithmetic Operator
- Relational Operator
- Logical Operator
- Increment & decrement Operator
- Assignment Operator

- Ternary Operator
- Bitwise and Shift Operator
- instanceof operator
- arrow operator

.

Arithmetic Operator

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The operands of the arithmetic operators must be of a numeric type. We cannot use them on boolean types, but we can use them on char types, since the char type in Java is, essentially, a subset of int.The following table lists various arithmetic operators.

Arithmetic operator	Yields	Example
Addition (+)	addition result	$2 + 9 = 11$
Subtraction (−)	difference result	$23 - 7 = 16$
Multiplication (*)	product result	$23 * 7 = 161$
Division (/)	division result	$23/7 = 3, 6/4 = 1$
Modular Division (%)	remainder result	$23\%6 = 5,\ 23.4\%10 = 3.4$

Relational Operator

These operators are binary operators that take two operands, whose values are being compared. Comparison operators are used in conditional statements, especially in loops, where the result of the comparison decides whether execution should proceed or not. They form the key to program flow control, known as conditional processing.

The following table lists various relational operators:

Relational operator	Meaning	Example
>	Strictly greater than	2>5 yields false
>=	Greater than or equal to	5>=4 yields true
<	Strictly less than	5<100 yields true
<=	Smaller than or equal to	5<=5 yields true
==	Double equal to	6==7 yields false

!=	Not equal to	6!=7 yields true

Logical operator

Logical operators are used when we want to check multiple conditions together. The logical operators are given below:

Operator	Meaning
&&	Logical AND, a && b is true when both a and b are true, false otherwise.
||	Logical OR, a || b is false when both a and b are false, true otherwise.
!	Logical NOT, !a is true if a is false and vice-versa.

Logical AND Operator

We can combine many relational expressions using "Logical And" operators .The result will be a boolean type. This operator is represented by the symbol "&&".

Consider the operation "operand1 && operand2"

Here operand1 and operand2 can be relational expressions or boolean types. The result of the operation "operand1 && operand2" will be

- "true" only if both operand1 and operand2 are "true"
- "false" if any of the operands (operand1 or operand2) is "false" or both the operands (operand1 or operand2) are "false".

[Truth Table of AND (&&)]

A	B	A&&B
true	true	true
true	false	false
false	true	false
false	false	false

Logical OR Operator

We can combine many relational expressions using "Logical OR" operators .The result will be a boolean type. This operator is represented by the symbol "||".

Consider the operation "operand1 || operand2"

Here operand1 and operand2 can be relational expressions or boolean types. The result of the operation "operand1 || operand2" will be

- "true" if any of the operands (operand1 or operand2) is "true" or both the operands (operand1 or operand2) are "true".
- "false" only if both operand1 and operand2 are "false"

[Truth Table of OR (||)]

A	B	A||B
true	true	true
true	false	true
false	true	true
false	false	false

Logical Not Operator

Logical Operator can be used with a boolean type variable or with a relational expression or with a logical expression. Logical not operator is denoted by the symbol "!".

- Assume the original value is "true". If we use this operator, the value will be changed to "false".
- Assume the original value is "false". If we use this operator, the value will be changed to "true".

[Truth Table of NOT (!)]

A	!A
false	true
true	false

Increment and Decrement Operator

The increment operator increases its operand by one. The decrement operator decreases its operand by one. For example, this statement:

x = x + 1;

can be rewritten like this by use of the increment operator:

x++;

Similarly, this statement:

x = x - 1;

is equivalent to

x--;

These operators are unique in that they can appear both in **postfix** form(x++), where they *follow the operand* as just shown, and **prefix** form(++x), where they *precede the operand*.

Expression	Meaning
y=x++	Value of x is assigned to y before it is incremented
Y=++x	Value of x is assigned first and then assigned to y

Example:

```
public class Test
{
        public static void main(String args[])
        {
            int a, b, c;
            a=7;
            b = a++;
            c=++b;
            System.out.println("a= " + a);
            System.out.println("b= " + b);
            System.out.println("c= " + c);
        }
}
```

Output:

```
a=8
b=8
c=8
```

Assignment Operator

Assignment operators are used in Java to assign values to variables. For example,

int age;

The assignment operator assigns the value on its right to the variable on its left. Here, 5 is assigned to the variable age using = operator.

Shorthand assignment operator: Operators like +=, -=, *=, /= etc are known as shorthand assignment operator.

x+=1 is equivalent to x=x+1

y/=9 is equivalent to y=y/9 and so on.

Conditional operator

The conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide; which value should be assigned to the variable. The operator is written as:

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

Example:

```
public class Test
{
        public static void main(String args[])
        {
            int a, b;
            a = 10;
            b = (a == 1) ? 20: 30;
            System.out.println("Value of b is: " + b);
            b = (a == 10) ? 20: 30;
            System.out.println("Value of b is: " + b);
        }
}
```

Output:

```
Value of b is: 30
Value of b is: 20
```

Bitwise and shift 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 the bit-by-bit operation. Assume if **a = 60** and **b = 13**; now in binary format they will be as follows −

```
a          = 0011 1100
b          = 0000 1101
           ------------------
a&b        = 0000 1100
a|b        = 0011 1101
a^b        = 0011 0001
~a         = 1100 0011
~b         = 1111 0010
```

The following table lists the bitwise operators −

Operator	Description	Example
& **(bitwise and)**	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| **(bitwise or)**	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^ **(bitwise XOR)**	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ **(bitwise compliment)**	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.

Operator	Description	Example
<< **(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

instanceof operator

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

Example:

```
class Simple
{
         public static void main(String args[])
        {
                Simple s=new Simple();
                System.out.println(s instanceof Simple);
         }
}
```

Output:

True

Arrow (->) operator

A **lambda expression** is, essentially, an anonymous (that is, unnamed) method. However, this method is not executed on its own. Instead, it is used to implement a method defined by a functional interface. Thus, a lambda expression results in a form of anonymous class. Lambda expressions are also commonly referred to as *closures*.

A **functional interface** is an interface that contains one and only one abstract method. Normally, this method specifies the intended purpose of the interface. Thus, a functional interface typically represents a single action. For example, the standard interface Runnable is a functional interface because it defines only one method: run( ). Therefore, run( ) defines the action of Runnable.

Operator Precedence

Below table shows the order of precedence for Java operators, from highest to lowest. Operators in the same row are equal in precedence. In binary operations, the order of evaluation is left to right (except for assignment, which evaluates right to left). Although they are technically separators, the [ ], ( ), and . can also act like operators. In that capacity, they would have the highest precedence. Also, notice the arrow operator (->). It is used in lambda expressions.

Highest
++(postfix), --(postfix)
++(prefix), --(prefix), ~, !,+(unary), -(unary), (*type_cast*)
., /(division), %(Modular division)
+ (Addition), - (subtraction)
>>, <<<, <<
>, >=, <, <=, instanceof()
==, !=
& (Bitwise and)
^ (Bitwise not)

| (Bitwise or)
&& (logical and)
||(logical or)
?: (conditional operator)
->(arrow operator)
= (assignment), op= (short hand assignment)
Lowest

Comments in JAVA

The java comments are statements that are not executed by the compiler and interpreter. The comments can be used to provide information or explanation about the variable, method, class or any statement. It can also be used to hide program code for specific time.

In Java there are three types of comments:

- Single line comments
- Multi line comments
- Documentation comments

Single line comments

A beginner level programmer uses mostly single-line comments for describing the code functionality. It is the easiest type of comments.

```
//
```

Multi line comments

To describe a full method in a code or a complex snippet single line comments can be tedious to write, since we have to give '//' at every line. So to overcome this multi line comments can be used.

```
/*………………..
………………….
*/


/**
….
```

*/

Documentation comments

Documentation comments starts with a forward slash followed by two asterisks, and ends with an asterisk followed by a forward slash. This comment contains descriptions about the code and is used to create API document. This comment is usually added in the beginning of the java file explaining why this program is designed and how to use it.

Typecasting

The method of converting one primitive data type into another data type is called type casting. There are two types of typecasting: *implicit type casting* and *explicit type casting*.

Implicit type casting or widening casting

This is the process of converting a smaller type to a larger type. When one type of data is assigned to another type of variable, an automatic type conversion will take place if the following two conditions are met:

1) The two types are compatible.
2) The destination type is larger than the source type.

$\boldsymbol{byte \rightarrow short \rightarrow char \rightarrow int \rightarrow long \rightarrow float \rightarrow double}$

For widening conversions, the numeric types, including integer and floating point types, are compatible with each other. However, there are no automatic conversions from the numeric types to char or boolean. Also, char and boolean are not compatible with each other.

The implicit type casting is demonstrated below.

```
// ImplicitTC.java
public class ImplicitTC
{
        public static void main(String[] args)
        {
           int myInt = 9;
           double myDouble = myInt;              // Automatic casting: int to double
           System.out.println(myInt);            // Outputs 9
           System.out.println(myDouble);         // Outputs 9.0
        }
}
```

Explicit type casting or narrowing casting

This type casting is done manually by placing the type in parentheses in front of the value. This is done manually by the programmer. Hence it is also known as explicit type casting.

For example, the following fragment casts an int to a byte. If the integer’s value is larger than the range of a byte, it will be reduced modulo (the remainder of an integer division by the) byte’s range.

```
int a;
byte b;
// …
b = (byte) a;
```

A different type of conversion will occur when a floating-point value is assigned to an integer type: truncation. As you know, integers do not have fractional components. Thus, when a floating-point value is assigned to an integer type, the fractional component is lost.

The explicit type casting is demonstrated below.

```
// ExplicitTC.java
public class ExplicitTC
{
        public static void main(String[] args)
        {
           double myDouble = 9.78;
           int myInt = (int) myDouble;          // Manual casting:double to int
           System.out.println(myDouble);        // Outputs 9.78
           System.out.println(myInt);           // Outputs 9
        }
}
```

Automatic Type Promotion in Expressions

In an expression, the precision required of an intermediate value will sometimes exceed the range of either operand. For example, examine the following expression:

```
        byte a = 40;
        byte b = 50;
        byte c = 100;
        int d = a * b / c;
```

The result The result of the intermediate term a * b easily exceeds the range of either of its byte operands. To handle this kind of problem, Java automatically promotes each **byte**, **short**, or **char** operand to **int** when evaluating an expression. This means that the sub expression **a*b** is performed using integers—not bytes. Thus, 2,000, the result of the intermediate expression, 50 * 40, is legal even though a and b are both specified as type byte.

As useful as the automatic promotions are, they can cause confusing compile-time errors. For example, this seemingly correct code causes a problem:

```
                byte b = 50;
                b = b * 2;          // Error! Cannot assign an int to a byte without cast!
```

The code is attempting to store 50 * 2, a perfectly valid byte value, back into a byte variable. However, because the operands were automatically promoted to int when the expression was evaluated, the result has also been promoted to int. Thus, the result of the expression is now of type int, which cannot be assigned to a byte without the use of a cast.

The Type Promotion Rules

Java defines several type promotion rules that apply to expressions. They are

3) First, all **byte**, **short**, and **char** values are promoted to **int**.
4) Then, if one operand is a **long**, the whole expression is promoted to **long**.
5) If one operand is a **float**, the entire expression is promoted to **float**.
6) If any of the operands are **double**, the result is **double**.

Arrays

An array is a group of like-typed variables that are referred to by a common name. Arrays of any type can be created and may have one or more dimensions. A specific element in an array is accessed by its index. Arrays offer a convenient means of grouping related information.

One-Dimensional Arrays

A *one-dimensional* array is, essentially, a list of like-typed variables. To create an array, we must create an array variable of the desired type. The general form of a one-dimensional array declaration is:

type var-name[ ];

For example, the following declares an array named month_days with the type "array of int":

int month_days[];

To allocate memory for arrays, new operator is applied to one-dimensional arrays as follows:

array-var = **new** type [size];

Therefore, obtaining an array is a **two-step** process.

a) First, we must declare a variable of the desired array type.

b) Second, we must allocate the memory that will hold the array, using **new**, and assign it to the array variable.

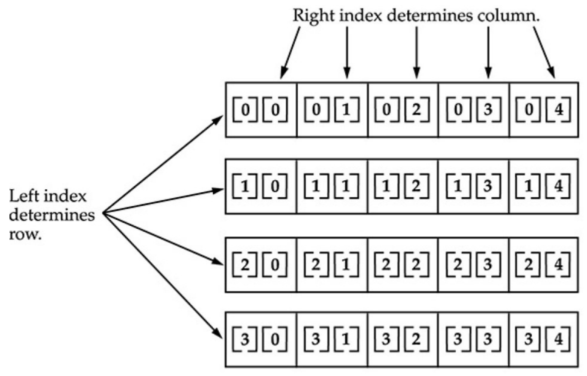
Thus, in Java all arrays are dynamically allocated.

Multidimensional Arrays

In Java, multidimensional arrays are implemented as arrays of arrays. To declare a multidimensional array variable, specify each additional index using another set of square brackets. For example, the following declares a two dimensional array variable called twoD:

int twoD[][] = **new int**[4][5];

This allocates a 4 by 5 array and assigns it to twoD. Internally, this matrix is implemented as an array of arrays of int. Conceptually, this array will look like the one shown in below figure.



Alternative Array Declaration Syntax

There is a second form that may be used to declare an array:

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

Here, the square brackets follow the type specifier, and not the name of the array variable. For example, the following two declarations are equivalent:

```
int al[] = new int[3];
int[] a2 = new int[3];
```

The following declarations are also equivalent:

```
char twod1[][] = new char[3][4];
char[][] twod2 = new char[3][4];
```

This alternative declaration form offers convenience when declaring several arrays at the same time. For example,

```
int[] nums, nums2, nums3;          // create three arrays
```

creates three array variables of type int. It is the same as writing

```
int nums[], nums2[], nums3[];      // create three arrays
```

Java's Control Statements

The JAVA control statements inside a program are usually executed sequentially. Sometimes a programmer wants to break the normal flow and jump to another statement or execute a set of statements repeatedly. The statements which break the normal sequential flow of the program are called control statements. Java's program control statements can be put into the following categories: *selection or decision making statement, iteration or loop statement, and jump*.

- **A Selection or decision making statement** allows program to choose different paths of execution based upon the truth value of an expression.
- **An iteration or loop statement** allows program execution to repeat one or more statements many number of time based upon some condition.
- **Jump statements** allows program to jump out of a loop before the normal exit of loop body.

Selection or decision making statements

Different types of selection or decision statements in java are:

if Statement

This statement is used when the decision is made taking only one condition into consideration. The syntax of if statement is as follows:

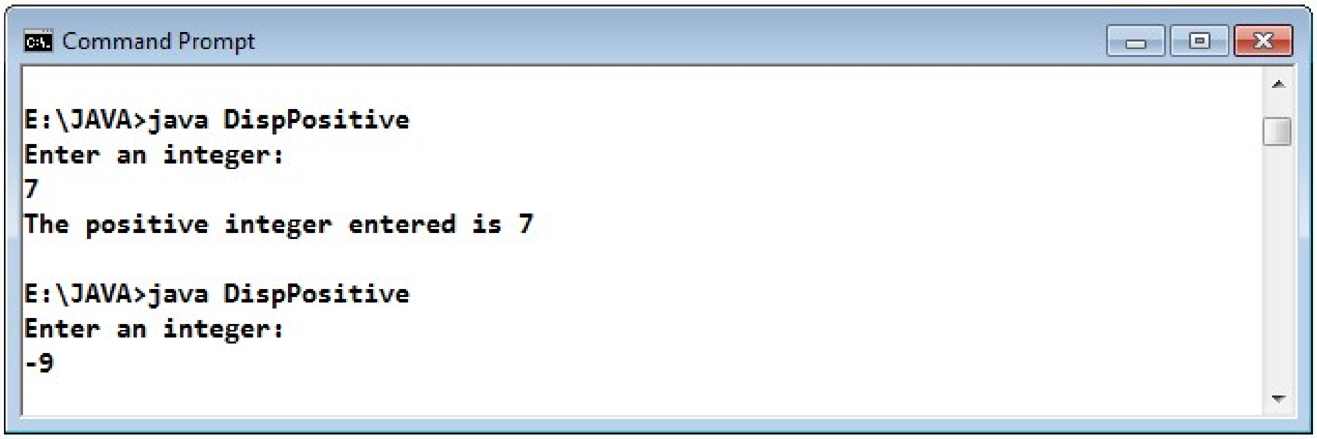
```
        if (condition)
        {
                //These codes are executed when condition is true
        }
```

Program❷ Write a program to input an integer from keyboard and display it if it is positive.

Solution:

```
import java.util.*;
class DispPositive
{
  public static void main(String args[])
  {
    int n;
    System.out.println("Enter an integer:");
    Scanner input = new Scanner(System.in);
    n = input.nextInt();
       if(n>0)
       {
       System.out.println("The integer = " + n);
       }
  }
}
```

Output



```
E:\JAVA>java DispPositive
Enter an integer:
7
The positive integer entered is 7

E:\JAVA>java DispPositive
Enter an integer:
-9
```

if else statement

This statement is used when the decision is made taking exactly two conditions into consideration. The syntax of if else statement is as follows:

```
if (condition)
{
//These codes are executed when condition is true
}
else
{
//These codes are executed when condition is false
}
```

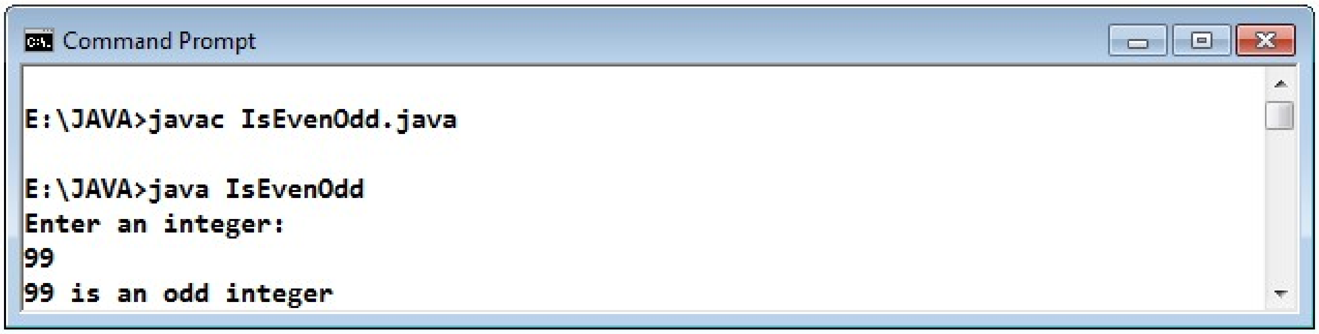
Program❷ Write a program to input an integer from keyboard and check whether it is even or odd.

Solution:

```
import java.util.*;
class IsEvenOdd
{
  public static void main(String args[])
  {
    int n;
    System.out.println("Enter an integer:");
    Scanner input = new Scanner(System.in);
```

```
      n = input.nextInt();
         if(n%2==0)
         {
         System.out.println(n+" is an even integer");
         }
         else
         {
         System.out.println(n+" is an odd integer");
         }
   }
}
```

Output:



```
E:\JAVA>javac IsEvenOdd.java

E:\JAVA>java IsEvenOdd
Enter an integer:
99
99 is an odd integer
```

if-else-if statement

This statement is used when the decision is made taking more than two conditions into consideration.

The syntax of if else if statement is as follows:

```
if (condition1)
{
//These codes are executed when condition1 is true
}
else if (condition2)
{
//These codes are executed when condition2 is true
}
…………………………
…………………………
```

```
        else
        {
        //These codes are executed when all conditions are false
        }
```

Program❸ Write a program to calculate division obtained by a student with his average mark entered through the keyboard as per following rules:

$\text{percentage mark} \geq 60$, **1st division**

$50 \leq \text{percentage mark} < 60$, **2nd division**

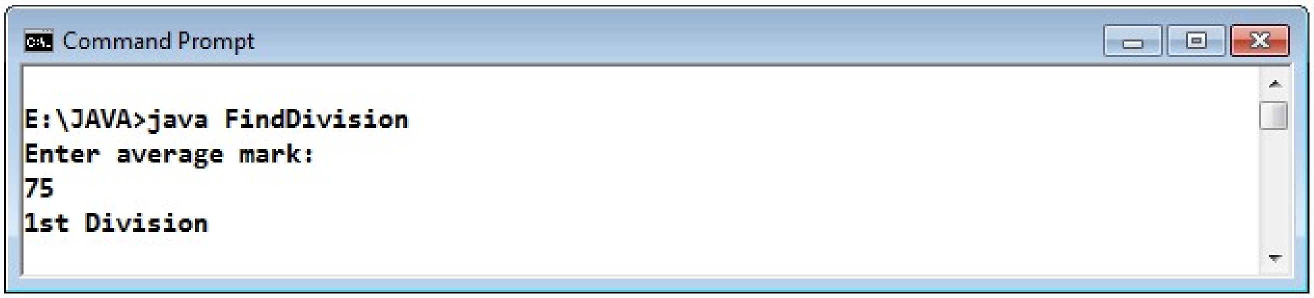
$40 \leq \text{percentage mark} < 50$, **3rd division**

$20 \leq \text{percentage mark} < 50$, **Fail**

Solution:

```
import java.util.*;
class FindDivision
{
        public static void main(String args[])
        {
                int avg_mark;
                System.out.println("Enter average mark:");
                Scanner input = new Scanner(System.in);
                avg_mark = input.nextInt();
                if(avg_mark >=60)
                {
                System.out.println("1st Division");
                }
                else if(avg_mark >=50 && avg_mark <60)
                {
                System.out.println("2nd Division");
                }
                else if(avg_mark >=40 && avg_mark <50)
                {
                }
                else
                {
                System.out.println("3rd  Division");
                }
        }
}
```

Output



```
E:\JAVA>java FindDivision
Enter average mark:
75
1st Division
```

Switch case statements

The switch statement is Java's multi way branch statement. It provides an easy way to dispatch execution to different parts of your code based on the value of an expression. As such, it often provides a better alternative to a large series of if-else-if statements. The syntax of switch case statement is as follows:

```
switch (expression)
{
case value1:
// These codes are executed when value of expression is value1
break;
case value2:
// These codes are executed when value of expression is value2
break;
……………….
……………….
case valueN:
// These codes are executed when value of expression is valueN
break;
```

```
        default:
        // These codes are executed when value of expression is other than above case values
        }
```

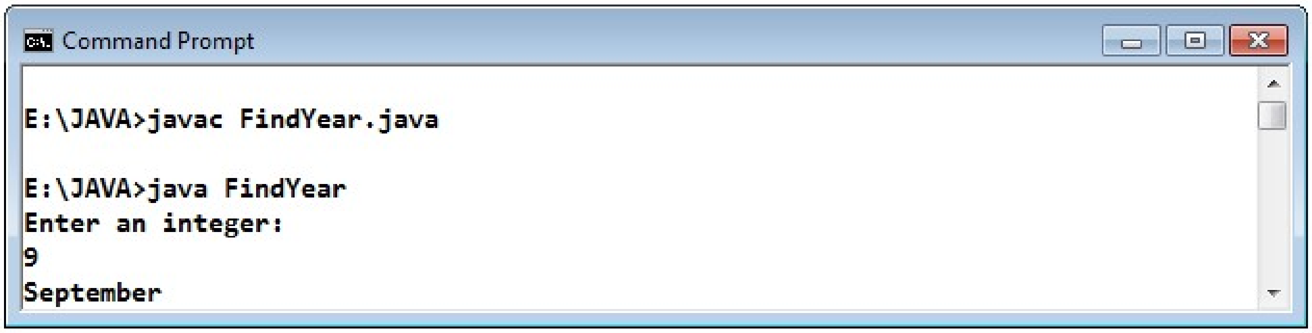
Program❹ Write a program to display a month name, if its corresponding integer value is entered through the keyboard. Integer 1 corresponds to January, 2 corresponds to February and so on.

Solution:

```
// FindYear.java
import java.util.*;
class FindYear
{
        public static void main(String args[])
        {
                int m;
                System.out.println("Enter an integer:");
                Scanner input = new Scanner(System.in);
                m = input.nextInt();
                switch(m)
                {
                case 1:
                System.out.println("January");
                break;
                case 2:
                System.out.println("February");
                break;
                case 3:
                System.out.println("March");
                break;
                case 4:
                System.out.println("April");
                break;
                case 5:
                System.out.println("May");
                break;
```

```
            case 6:
            System.out.println("June");
            break;
            case 7:
            System.out.println("July");
            break;
            case 8:
            System.out.println("August");
            break;
            case 9:
            System.out.println("September");
            break;
            case 10:
            System.out.println("October");
            break;
            case 11:
            System.out.println("November");
            break;
            case 12:
            System.out.println("December");
            break;
            default:
            System.out.println("Invalid month");
            }
        }
}
```

Output:



```
E:\JAVA>javac FindYear.java

E:\JAVA>java FindYear
Enter an integer:
9
September
```

Iteration or loop statements

Java's iteration statements are for, while, and do-while. These statements create what we commonly call loops. As you probably know, a loop repeatedly executes the same set of instructions until a termination condition is met.

while loop

The while loop is Java's most fundamental looping statement. It repeats a statement or block while its controlling expression is true. The general form of for while loop statement is:

```
        Initialization;
        while (condition)
        {
                // these codes are executed in each iteration of the loop
                //increment or decrement statement which advances the loop;
        }
```

The condition can be any Boolean expression. The body of the loop will be executed as long as the conditional expression is true. When condition becomes false, control passes to the next line of code immediately following the loop.

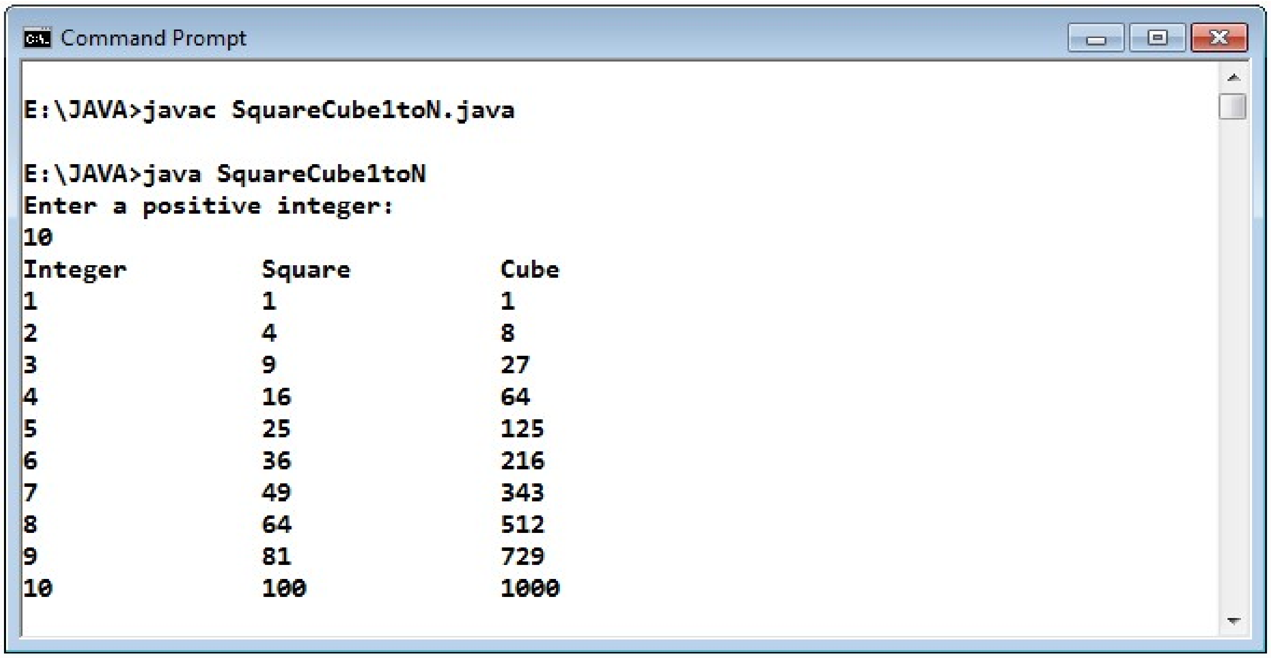
Note: The curly braces are unnecessary if only a single statement is being repeated.

Program❺ Write a program to display the square and cube of all integers from 1 to n, for any positive integer.

Solution:

```
//SquareCube1toN.java
import java.util.*;
class SquareCube1toN
{
        public static void main(String args[])
        {
                int n, i=1, s, c;
                System.out.println("Enter a positive integer:");
                Scanner input = new Scanner(System.in);
                n = input.nextInt();
                System.out.println("Integer\t\tSquare\t\tCube");
                while(i<=n)
                {
                s=i*i;
                c=i*i*i;
                System.out.println(i+"\t\t"+s+"\t\t"+c);
                i++;
                }
        }
}
```

Output



```
E:\JAVA>javac SquareCube1toN.java

E:\JAVA>java SquareCube1toN
Enter a positive integer:
10
Integer         Square          Cube
1               1               1
2               4               8
3               9               27
4               16              64
5               25              125
6               36              216
7               49              343
8               64              512
9               81              729
10              100             1000
```

do while Loop

If the conditional expression controlling a while loop is initially false, then the body of the loop will not be executed at all. However, sometimes it is desirable to execute the body of a while loop at least once, even if the conditional expression is false to begin with. The do-while loop always executes its body at least once, because its conditional expression is at the bottom of the loop. Its general form is:

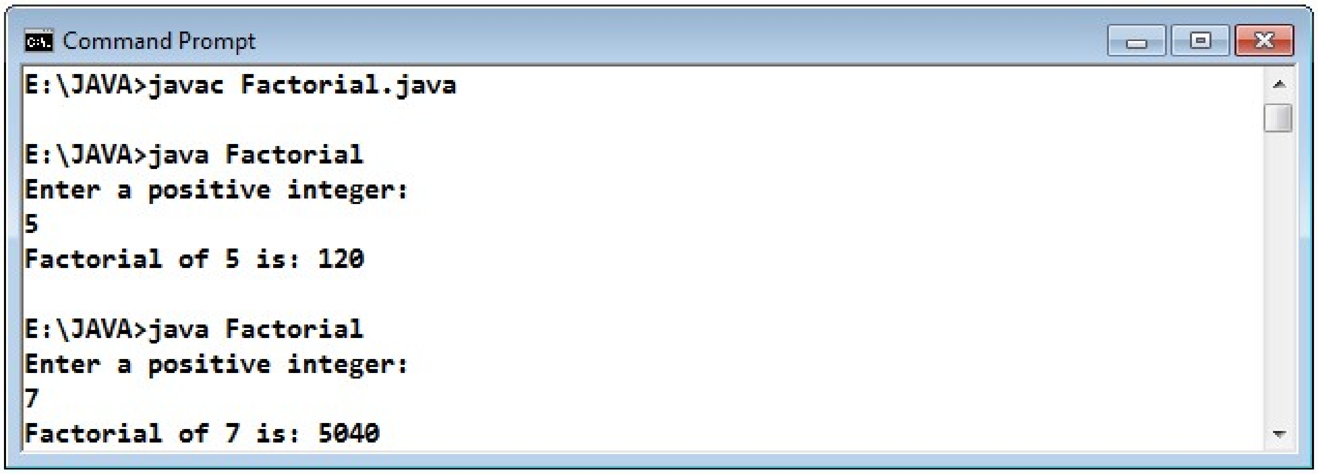
```
        Initialization;
        do
        {
        // body of loop
        Increment/ decrement;
        } while (condition);
```

Program❻ Write a program to find factorial of a positive integer given from the keyboard.

Solution:

```
//Factorial.java
import java.util.*;
class Factorial
{
    public static void main(String[] args)
    {
                int n, i=1;
                long fact=1;
                System.out.println("Enter a positive integer:");
                Scanner input = new Scanner(System.in);
                n = input.nextInt();
            do
        {
          fact = fact * i;
          i++;
        } while (i <= n);
        System.out.println("Factorial of " + n + " is: " + fact);
    }
}
```

Output


Command Prompt

```
E:\JAVA>javac Factorial.java

E:\JAVA>java Factorial
Enter a positive integer:
5
Factorial of 5 is: 120

E:\JAVA>java Factorial
Enter a positive integer:
7
Factorial of 7 is: 5040
```

for loop

The general form of for loop statement is:

```
for (initialization; condition; iteration)
{
        // these codes are executed in each iteration of the loop
}
```

The for loop operates as follows:

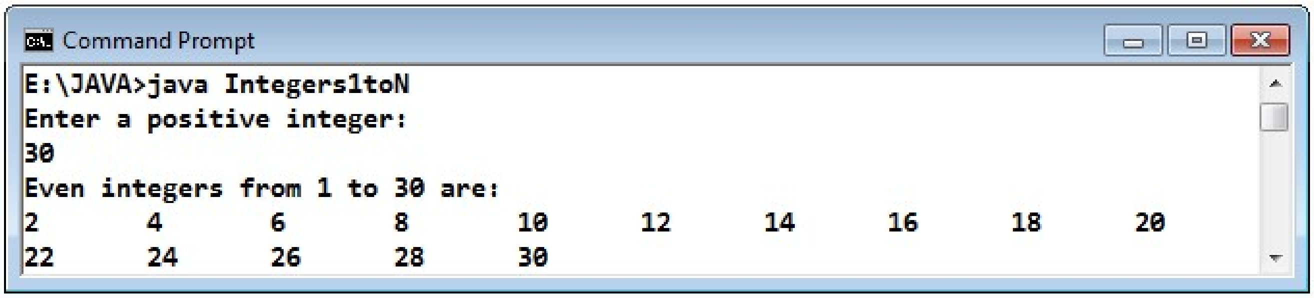
- When the loop first starts, the initialization part of the loop is executed. Generally, this is an expression that sets the value of the loop control variable, which acts as a counter. It is important to understand that the initialization expression is executed once.
- Next, the conditional expression is evaluated. This must be a Boolean expression. It usually tests the loop control variable against a target value. If this expression is true, then the body of the loop is executed. If it is false, the loop terminates.
- Next, the iteration part of the loop is executed. This is usually an expression that increments or decrements the loop control variable. The loop then iterates, first evaluating the conditional expression, then executing the body of the loop, and then executing the iteration expression with each pass. This process repeats until the controlling expression is false.

Program❼ Write a program to print all even integers from 1 to n, for any positive integer n input through the keyboard.

Solution:

```
//Integers1toN.java
import java.util.*;
class Integers1toN
{
        public static void main(String args[])
        {
                int n, i;
                System.out.println("Enter a positive integer:");
                Scanner input = new Scanner(System.in);
                n = input.nextInt();
                System.out.println("Even integers from 1 to "+n+" are:");
                for(i=1; i<=n; i++)
                {
                        if(i%2==0)
                        {
                        System.out.print(i+"\t");
                        }
                }
        }
}
```

Output



```
E:\JAVA>java Integers1toN
Enter a positive integer:
30
Even integers from 1 to 30 are:
2       4       6       8       10      12      14      16      18      20
22      24      26      28      30
```

Jump Statements

Java offers following jump statements:

break statement: A break statement takes the program control out of the loop. When break statement is used, program control immediately jump to the statement immediately follows the loop statement.

continue statement: A continue statement takes control to the beginning of the loop.

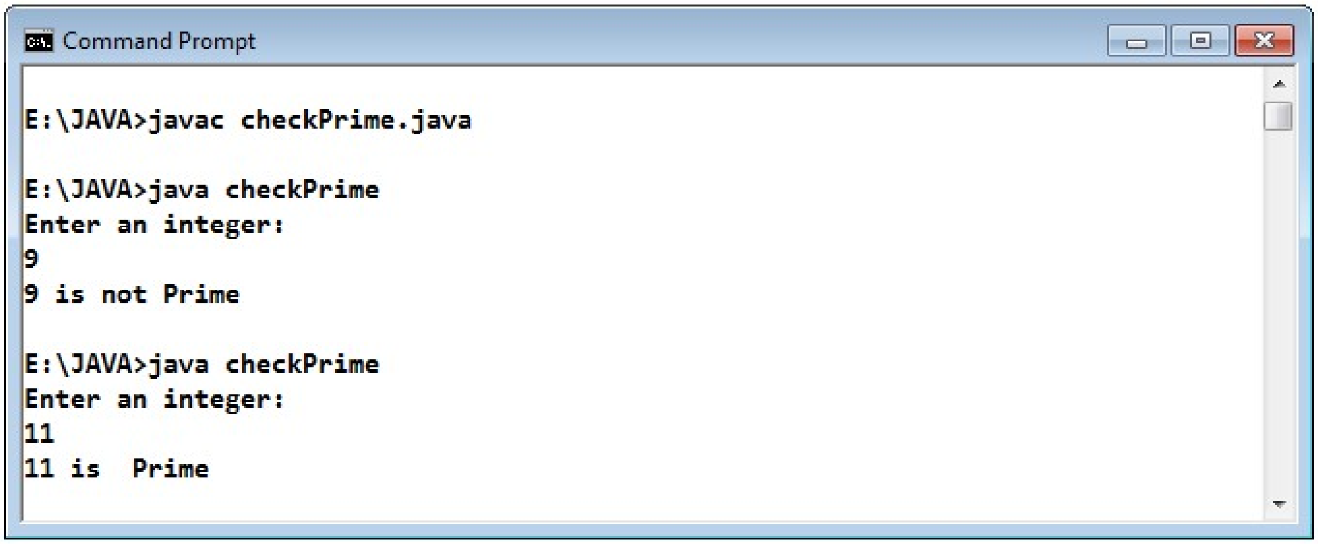
Note: goto statement is not supported by java.

Program❽ Write a program to check whether an integer entered through keyboard is prime or not.

Solution:

```
//checkPrime.java
import java.util.*;
class checkPrime
{
        public static void main(String args[])
        {
                int n, i;
                System.out.println("Enter an integer:");
                Scanner input = new Scanner(System.in);
                n = input.nextInt();
                for(i=2; i<n; i++)
                {
                        if(n%i==0)
                        {
                        System.out.println(n+" is not Prime");
                        break;
                        }
                }
                if(i==n)
                System.out.println(n+" is  Prime");
        }
}
```

Output



```
E:\JAVA>javac checkPrime.java

E:\JAVA>java checkPrime
Enter an integer:
9
9 is not Prime

E:\JAVA>java checkPrime
Enter an integer:
11
11 is  Prime
```

Taking input from keyboard

Taking input from keyboard is one of the most essential parts of a programming language. In java, input can be taken from the keyboard using either of these following methods:

(i) **Command line arguments**

(ii) **Scanner class**

(iii) **BufferedReader class**

Command line argument method

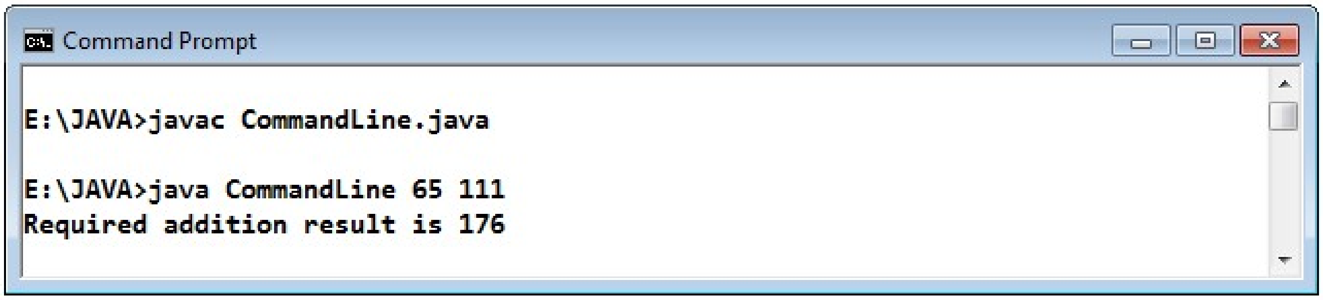
The command line arguments are the arguments that we pass during the execution of a program. Accessing command line arguments inside a java program is quite easy. They are stored in the String array (named args[] in our examples. 1st argument is stored in args[0], 2nd argument is stored in args[1] and so on) which is passed as a parameter to the main() method. It is clearly explained in the following example.

Program❾ Write a program to find addition of two integers, where the inputs are given as command line arguments.

Solution:

```
//CommandLine.java
class CommandLine
{
        public static void main(String args[])
        {
                int x, y, res;
                x=Integer.parseInt(args[0]);
                y=Integer.parseInt(args[1]);
                res=x+y;
                System.out.println("Required addition result is "+res);
        }
}
```

Output



```
E:\JAVA>javac CommandLine.java

E:\JAVA>java CommandLine 65 111
Required addition result is 176
```

Explanation:

The inputs given during program execution get stored as string value in args[0] & args[1] respectively. Here Integer.parseInt() method is used to convert string to its corresponding integer.

Scanner class method

Scanner is a class in java.util package used for obtaining the input of the primitive types like int, double, etc. and strings. It is the easiest way to read input in a Java program, though not very efficient if we want an input method for scenarios where time is a constraint like in competitive programming.

- To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream. We may pass an object of class File if we want to read input from a file.
- To read numerical values of a certain data type xyz, the function to use is nextXyz(). For example, to read a value of type short, we can use **nextShort()**. To read an integer value, we use **nextInt()**. To read a long value we use **nextShort()**. To read a double value, we use **nextDouble()** and so on.
- To read strings, we use **nextLine().**

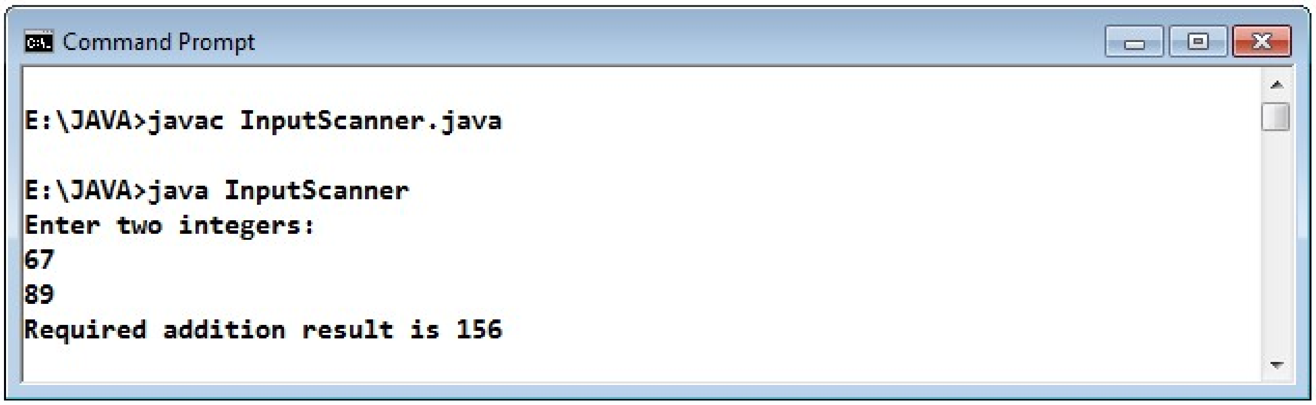
- To read a single character, we use next().charAt(0). next() function returns the next token/word in the input as a string and charAt(0) function returns the first character in that string.

Program ⓾ Write a program to find addition of two integers, where the inputs are given from the keyboard.

Solution:

```
//InputScanner.java
import java.util.*;
class InputScanner
{
        public static void main(String args[])
        {
                int x, y, res;
                System.out.println("Enter two integers:");
                Scanner input = new Scanner(System.in);
                x = input.nextInt();
                y = input.nextInt();
                res=x+y;
                System.out.println("Required addition result is "+res);
        }
}
```

Output



```
E:\JAVA>javac InputScanner.java

E:\JAVA>java InputScanner
Enter two integers:
67
89
Required addition result is 156
```

BufferedReader class method

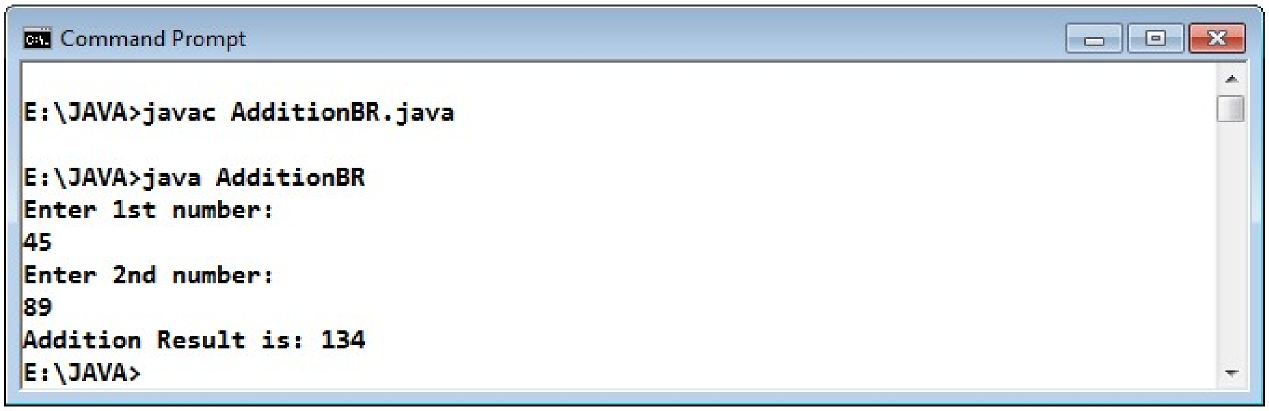
It is the most efficient method of reading inputs from the keyboard. This is the Java classical method to take input, Introduced in JDK1.0. This method is used by wrapping the **System.in** (standard input stream) in an **InputStreamReader** which is wrapped in a **BufferedReader**, we can read input from the user in the command line. The main demerit is that wrapping code is hard to remember and we have to import built in package **java.io**.

Program ⓫ Write a program to find addition of two numbers given from the keyboard, using BufferedReader class method.

Solution:

```
import java.io.*;
class BufferedReaderAddition
{
        public static void main(String args[])
        {
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
                String str;
                int x, y, z;
                try
                {
                        System.out.println("Enter 1st number:");
                        str = br.readLine();
                        x=Integer.parseInt(str);
                        System.out.println("Enter 2nd number:");
                        str = br.readLine();
                        y=Integer.parseInt(str);
                        z=x+y;
                        System.out.print(“Addition Result is: ”+z);
                }
                catch (Exception e)
                {
                        System.out.println("Invalid Number ..Try Again!!!!!!!");
                }
        }
}
```

Output



```
E:\JAVA>javac AdditionBR.java

E:\JAVA>java AdditionBR
Enter 1st number:
45
Enter 2nd number:
89
Addition Result is: 134
E:\JAVA>
```

Review Questions of Chapter 3

MCQ Type

1. **Choose the most appropriate answer:**

a) JAVA characters take __________ of memory.

a) 2 Bytes b) 1 Byte
c) 3 Bytes d) 4 Bytes

b) Which of the following is a class in java?

a) int b) float
c) char d) String

c) In java, which of the following data type groups are of same size?

a) int and long b) long and float
c) float and double d) int and float

d) Why an array is called "*a homogeneous collection of data*"?

a) It stores different types of data
b) Size of an array is limited
c) An array can store only one type of data
d) An array uses indices for addressing an item stored in it

e) Find the output of below code:

```
class Test
{
        public static void main(String[] args)
        {
        for(int i = 0; true; i++)
        {
        System.out.println("Hello");
        break;
        }
        }
}
```

a) Hello b) Hello infinite number of times
c) Compilation Error d) None of these

f) Find the output of below code:

```
class Test
{
        public static void main(String[] args)
        {
        for(int i = 0; 1; i++)
        {
        System.out.println("Hello");
        break;
        }
        }
}
```

a) Hello b) Hello infinite times
c) Compilation Error d) None of these

g) **Which of these can be returned by the operator &?**
a) Integer b) Boolean
c) Character d) Integer or Boolean

h) **Which of these is long data type literal?**
a) 0x99fffL b) ABCDEFG
c) 0x99fffa d) 99671246

i) **Which of these operators is used to allocate memory to array variable in Java?**
a) malloc b) alloc
c) new d) new malloc

j) **Which of these is an incorrect array declaration?**
a) int arr[] = new int[5] b) int [] arr = new int[5].
c) int arr[] = new int[5] d) int arr[] = int [5] new

k) **What will this code print?**
int arr[] = new int [5];
System.out.print(arr);
a) 0 b) value stored in arr[0].
c) 00000 d) Class name@ hashcode in hexadecimal form

l) **Which of these is necessary to specify at time of array initialization?**
a) Row b) Column
c) Both Row and Column d) None of the mentioned

m) **Which one is a valid declaration of a boolean?**
a) boolean b1 = 1; b) boolean b2 = 'false';
c) boolean b3 = false; d) boolean b4 = 'true'

Short Type

2. Answer briefly

a) Why a java character takes 2 bytes of memory unlike C or C++?
b) Write the syntax to accept an integer input using scanner class.
c) Although using a byte or short would be more efficient than using an int in situations in which the larger range of an int is not needed but this is not the case. Justify your answer.
d) What are different methods to accept input through keyboard in java?
e) Name the data structure used to maintain recursive function calls in a program.
f) Define an array. How memory is allocated to an array dynamically?
g) Differentiate between = and ==.
h) Define conditional operator in java. Give its syntax.
i) What are different comment lines in java?
j) Define literals in java.
k) What are bit wise operators?

Long Type

3. Answer in details

a) Define tokens. Explain various types of tokens available in java in details.
b) Explain various loop control statements in java with examples.
c) Explain various selection control statement in java with examples.
d) Define operator. Explain various types of operator with examples.

Programming Exercise

4.

a) Write a program to find sum of first n natural numbers, where value of integer is supplied from the keyboard.
b) Write a program to find sum of digits of an integer given through the keyboard.
c) Write a program to print first n terms of a Fibonacci sequence using a recursive function. Assume first two terms of the Fibonacci sequence are 0 and 1.
d) Write a program to find GCD of two integers using recursive function. Your program should able to pass the following test cases:

Input		Output
5	7	1
20	0	20
0	7	7
9	6	3
5	h	Invalid input

e) Write a program to print sum of even factors of an integer.
f) Write a program to check whether an integer is prime number or not.
g) Write a program to check whether integer is Armstrong or not.
h) Write a program to print the sum of all diagonal elements of a square matrix containing numeric elements.
i) Write a program to multiply two n × n matrices.
j) Write a program to find maximum value from a list containing n numeric values.

****