SCHOOL OF FUTURE TECH

ITM SKILLS UNIVERSITY

COURSE:- Java Programming

Sem:- II

Module -1

By:
Dr. Ritu Jain,
Associate Professor,
Computer Science-ISU

History of Java

- Java was conceived by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems, Inc. in 1991.
- Primary motivation was the need for a **platform-independent (that is, architecture-neutral) language** that could be used to create embedded software to be embedded in various consumer electronic devices, such as microwave ovens and remote controls.
- Initially, it was called **Oak** and was developed as a part of the Green project.
- In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies.
- JDK 1.0 was released on January 23, 1996. After the first release of Java, there have been many additional features added to the language.
- Now Java is being used in desktop applications, web applications, enterprise applications, mobile applications, etc.
- **James Gosling** is known as the Father of Java.

Bytecode

- Java is **platform independent** as "Output of Java compiler is not executable code. Rather, it is bytecode".
- *Bytecode is a highly optimized set of instructions designed to be executed by the Java run-time system, which is called the Java Virtual Machine (JVM).*
- Translating a Java program into bytecode makes it much easier to run a program in a wide variety of environments because only the JVM needs to be implemented for each platform. Once the run-time package exists for a given system, any Java program can run on it.
- The fact that Java program is executed by JVM also helps to make it secure. Because execution in under the control of the JVM, it ensures that no side effects outside the system are generated.
- In this way, Java addressed **portability and security** concerns

Source Code
Sample.java
javac
Compiler
List of Errors
.....
ByteCode
Sample.class
java
java
java
java
JVM
Machine Code
Windows OS
JVM
Machine Code
UNIX OS
JVM
Machine Code
MAC OS
JVM
Machine Code
Mobile OS

Java Development Kit (JDK)

- It is a software development environment used for developing Java applications and applets.

- JDK is a kit(or package) that includes two things:
  - Development Tools(to provide an environment to develop your java programs)
  - JRE (to execute your java program).
- We need to install JDK on our computer in order to create, compile and run the java program.

Java Runtime Environment

JRE stands for **"Java Runtime Environment"** The Java Runtime Environment provides the minimum requirements for executing a Java application; it consists of the *Java Virtual Machine (JVM), core classes*, and *supporting files*.

JRE (Java Runtime Environment) is an installation package that provides an environment to **only run(not develop)** the java program(or application)onto your machine. JRE is only used by those who only want to run Java programs that are end-users of your system.

3. JVM (Java Virtual Machine): Function of Java Virtual Machine is to execute the bytecode produced by the compiler. Every Operating System has a different JVM but the output they produce after the execution of bytecode is the same across all the operating systems. This is why Java is known as a **platform-independent language.**

Created by Dr. Ritu Jain

Types of Java Applications

There are mainly 4 types of applications that can be created using Java programming:

- **Standalone Application**: Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc.
- **Web Application:** java is perfect for developing web applications because of its ability to interact with a large number of systems.
- **Enterprise Application:** Java is a popular choice for building enterprise-level applications due to its scalability, reliability, and security. Java's powerful libraries and frameworks enable developers to create robust applications for managing business processes.
- **Mobile Application:** Java is also used to build applications that run across smartphones and other small-screen devices.

Features of Java

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

- **Simple**
- **Object oriented**
- **Distributed**
- **Interpreted**
- **Robust**
- **Secure**
- **Architecture neutral**
- **Portable**
- **High performance**
- **Multithreaded**
- **Dynamic**

Features of Java

- **Simple**
  - It's simple and easy to learn if you already know the basic concepts of OOP.
  - C++ programmer can move to JAVA with very little effort to learn.
  - does not have complex features like pointers, operator overloading, multiple inheritances, and explicit memory allocation.

Features of Java

- **Object Oriented**
  - Almost “Everything is an Object” paradigm. All program code and data reside within objects and classes.
  - Java comes with an extensive set of classes, arranged in packages that can be used in our programs through inheritance.
  - Organizing the program in the terms of a collection of objects is a way of object-oriented programming.
  - The four main concepts of Object-Oriented programming are:
    - **Abstraction**: refers to hiding the implementation details of a code and exposing only the necessary information to the user.
    - **Encapsulation**: Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse.
    - **Inheritance**: is the mechanism in Java by which one class is allowed to inherit the features(fields and methods) of another class.
    - **Polymorphism**: is a feature that allows one interface to be used for a general class of actions. The specific action is determined by the exact nature of the situation.

Features of Java

- **Distributed**
  - Java is designed for distributed environment of the Internet. In Java, accessing a resource using a URL is not much different from accessing a file.
- **Compiled and Interpreted**
  - Usually a computer language is either compiled or Interpreted. Java combines both this approach and makes it a two-stage system.
  - Compiled : Java enables creation of a cross platform programs by compiling into an intermediate representation called Java Bytecode.
  - Interpreted : Bytecode is then interpreted, which generates machine code that can be directly executed by the machine that provides a Java Virtual machine.

Created by Dr. Ritu Jain

Features of Java

- **Robust**
  - Java language is robust which means reliable.
  - It is developed in such a way that it puts a lot of effort into **checking errors** as early as possible, that is why the java compiler is able to detect even those errors that are not easy to detect by another programming language.
  - The main features of java that make it robust are *garbage collection, exception handling, and memory allocation.*
- **Secure**
  - In java, we don't have pointers, so we cannot access out-of-bound arrays i.e it shows ArrayIndexOutOfBound Exception if we try to do so. That's why several security flaws are not possible in Java.
  - Also, java programs run in an environment that is independent of the OS (operating system) environment which makes java programs more secure. Java achieves this protection by confining a Java program to the java execution environment and not allowing it to access other parts of the computer.

Java Virtual Machine (JVM)

Features of Java

- **Architecture Neutral**
  - Java language and Java Virtual Machine has been designed to achieve the goal of "**write once; run anywhere**" (WORA).
  - Compiler converts source code to bytecode and then the JVM executes the bytecode generated by the compiler.
  - This bytecode can run on any platform be it Windows, Linux, or macOS which means if we compile a program on Windows, then we can run it on Linux and vice versa.
  - Each operating system has a different JVM, but the output produced by all the OS is the same after the execution of the bytecode. That is why we call java a platform-independent language.
  - Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

- **Portable**
  - As we know, java code written on one machine can be run on another machine. The platform-independent feature of java in which its platform-independent bytecode can be taken to any platform for execution makes java portable.

Features of Java

- **High Performance**
  - Java architecture is defined in such a way that it reduces overhead during the runtime and at some times java uses Just In Time (JIT) compiler where the compiler compiles code on-demand basics where it only compiles those methods that are called making applications to execute faster.
- **Multithreaded**
  - Java supports multithreading. It is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of the CPU.
- **Dynamic**
  - Java supports the dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

A First Simple Program

```
/* This is a simple Java program. Call this file "HelloWorld.java". */
class HelloWorld
{
        public static void main(String args[])
        {
                System.out.println("Hello World");
        }
}
```

Created by Dr. Ritu Jain

Entering the Program

- The first thing that you must learn about Java is that the name you give to a source file is very important. For this example, the name of the source file should be **HelloWorld.java**.
- In Java, all code must reside inside a class. By convention, the name of that class should match the name of the file that holds the program. It should also have same case as Java is case-sensitive.

Typical cycle in Java

Figure 13: Typical cycle in Java programming

Lexical Issues

- Java programs are a collection of
  - *whitespace -space, tab, newline*
  - *identifiers*
  - *literals- constant, eg: 500, “Hello”, ‘a’, 45.5, 55.5f, 5555555l*
  - *separators- (),[],{},semicolon(;), period(.), comma(,),Ellipse(...)*
  - *operators*
  - *comments- Single line(//), multiple line (/* */), documentation (/** */)*
  - *keywords*

Created by Dr. Ritu Jain

Identifiers

- Identifiers are used for class names, method names, variable, constants or label names.
- An identifier can contain:
  - uppercase letters (A-Z),
  - lowercase letters (a-z),
  - numbers (0-9),
  - underscore (_)
  - and dollar-sign character ($) (not intended for general use)
- Identifier must not begin with a number
- Java is case-sensitive, so **VALUE** is a different identifier than **Value**.
- Some examples of valid identifiers are:

  AvgTemp, count, a4, $test, this_is_ok, _num
- Invalid identifier names include these:

  2count, a+c, high-temp, Not/ok, roll num

Created by Dr. Ritu Jain

Java Keywords

A keyword is **reserved word, that have a predefined meaning in the Java language**.

- These keywords cannot be used as an identifier.
- Java is a case sensitive language so, int and Int are not same.
- All keywords are in lower case.
- The keywords **const** and **goto** are reserved but not used.
- In addition to keywords, Java reserves following values: true, false, null . These values can't be used as identifier.

```
abstract   continue   for          new          switch
assert     default    if           package      synchronized
boolean    do         goto         private      this
break      double     implements   protected    throw
byte       else       import       public       throws
case       enum       instanceof   return       transient
catch      extends    int          short        try
char       final      interface    static       void
class      finally    long         strictfp     volatile
const      float      native       super        while
_ (underscore)
```

Created by Dr. Ritu Jain

Primitive Data Types

ain

Primitive Data Types in Java

ITM SKILLS UNIVERSITY

DATA TYPES	SIZE	DEFAULT	EXPLAINATION
boolean	1 bit	false	Stores true or false values
byte	1 byte/ 8bits	0	Stores whole numbers from -128 to 127
short	2 bytes/ 16bits	0	Stores whole numbers from -32,768 to 32,767
int	4 bytes/ 32bits	0	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes/ 64bits	0L	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes/ 32bits	0.0f	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes/ 64bits	0.0d	Stores fractional numbers. Sufficient for storing 15 decimal digits
char	2 bytes/ 16bits	'\u0000'	Stores a single character/letter or ASCII values

Primitive Data Types in Java

- **Primitive data types:** *Represent single values – not objects*
- *All data types have strictly defined range.*
- *Eight* primitive data types include
  - **Integer**: all integers (byte, short, int, long) are signed
    - byte – 8 bits
      - Examples: byte a = 10; byte b = -20;
    - short – 16 bits
      - Examples: short s = 1000;
    - int – 32 bits
      - Example: int num=565;
    - long – 64 bits
- Note: **byte and short values in exp. With binary op. *promoted* to int.**

Created by Dr. Ritu Jain

Primitive Data Types in Java

- **Floating point numbers**:
  - **float** - 32 bits; when you don't require a large degree of precision.
  - **double** – 64 bits; All math functions in Java return double values

- **char** – 16 bits; range:0-65536
  - Java uses Unicode to represent characters.
- **boolean**: true, false
  - Example: boolean b=true;

Created by Dr. Ritu Jain

Literals

- A constant value in Java is created by using a literal representation of it. E.g: 100, 98.6, 'X' "This is a test"

- **Integer Literals:** Any whole number value
  - Eg. 345, 600,1
  - When a literal value is assigned to a **byte** or **short** variable, no error is generated if the literal value is within the range.
  - Example:
    - byte b1=10;
    - short s=5;
    - byte b2= 500; //DNC
  - Integer literal can be assigned to char as long as it is within range.
    - char c1= 97;
    - char c2= 67000; //DNC
  - Long literal is expressed by appending l or L at the end of the literal.
    - Example:
      - long l1=5000;
      - long l2=999999999999; //DNC

Created by Dr. Ritu Jain

Floating-Point Literals

- represent decimal values with a fractional component.
- They can be expressed in either
  - **Standard notation:** *whole number component . fractional component*
    - E.g.: 2.45, 3.1456
  - **Scientific notation:** floating point number E a suffix specifies a power of 10 by which the number is to be multiplied.
    - E.g.: 6.022E23, 31459E-05, 2e+100
- *default to **double** precision.*
- To specify a **float** literal, append an *F* or *f*
- Can also specify d or D for double literal but it is redundant.

boolean Literals

- There are only two logical values that a **boolean** value can have, **true** and **false**.
- The **true** does not equal 1,
- **false** does not equal 0.
- In Java, they can only be assigned to variables declared as **boolean**, or used in expressions with boolean operators.

Created by Dr. Ritu Jain

character Literals

- Represented by Unicode character set.
- Can be manipulated with integer operations, such as addition, subtraction operators.
- E.g: ‘a’
- Escape sequences: there are several escape sequences that allow you to enter the character you need:

\'	Single quote
\"	Double quote
\\	Backslash
\n	New line (also known as line feed)

String Literals

- A sequence of characters between a pair of double quotes.
- E.g:
  - “Hello World”,
  - “two\nlines”,
  - “\”This is in quotes\”“

Created by Dr. Ritu Jain

Variable

- The variable is the basic unit of storage in a Java program.
- Eg.
  - int x, y, z;
  - int d=3, e, f=5;
  - double pi=3.14159;
  - char c=‘a’;

Created by Dr. Ritu Jain

Type Conversion and Casting

Java's Automatic Conversions

- When one type of data is assigned to another type of variable, an *automatic type conversion (widening conversion)* will take place if:
  - The two types are compatible.
  - The destination type is larger than the source type.

Byte –> Short –> Int –> Long – > Float –> Double

Widening or Automatic Conversion

•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.**
- boolean is not compatible with any datatype

Created by Dr. Ritu Jain

- Java also performs an automatic type conversion when storing a literal integer constant into variables of type **byte**, **short**, **long**, or **char**.

Ex: Automatic Type conversion

```
class AutoTypeConvEx1 {
    public static void main(String[] args)
    {
        int i = 100;
        // Automatic type conversion:int to long type
        long l = i;

        // Automatic type conversion: long to float type
        float f = l;

        System.out.println("Int value " + i);
        System.out.println("Long value " + l);
        System.out.println("Float value " + f);
    }
}
```

Created by Dr. Ritu Jain

Ex: numeric to char: implicit type conversion not allowed, but explicit conversion allowed

```
public class CharIntIncompat {
   public static void main(String[] argv)
   {
      char ch1 = ‘a’;
      char ch2 =98;
      int num = 99;
      // ch = num;          //DNC:incompatible type, automatic conversion not allowed
      ch= (char)num; //this is ok
   }
}
```

Created by Dr. Ritu Jain

Narrowing or Explicit Conversion

- If we want to assign a value of larger data type to a smaller data type we need to perform **explicit type casting or narrowing conversion**.

Double –> Float –> Long –> Int –> Short –> Byte

Narrowing or Explicit Conversion

- **Syntax: (target type) value**
- Example:
  - int i=10;
    byte b=i; //DNC
    byte b=(byte) i; //ok

- A different type of conversion will occur when a floating-point value is assigned to an integer type: *truncation*.
  - Example:
    - double d=55.8;
      int i=(int)d; //i=55

.

Created by Dr. Ritu Jain

Narrowing or Explicit Conversion

- If the size of the whole number component is too large to fit into the target type, then that value will be reduced modulo the target type's range.
- Ex:

```
int a=257
byte b=(byte)a;
```

- If int value is larger than the range of byte, it will be reduced modulo byte's range
- Here, value of b will be 1 i.e. 257% 256 (range of byte)
- Fractional part will be lost, if floating point is converted to integer type.

Java program to illustrate explicit type conversion

```
//Java program to illustrate explicit type conversion
class TypeCastingEx
{	public static void main(String[] args)
	{
		double d = 100.23;

		long l = (long)d;		//explicit type casting :truncation,
		int i = (int)l;		//explicit type casting

		System.out.println("Long value "+l); //fractional part lost ; value: 100

		System.out.println("Int value "+i);	//fractional part lost; value: 100

	}
}
```

Created by Dr. Ritu Jain

Type promotion in Expressions

- While evaluating expressions, conditions for type promotion are:
  - Java automatically promotes each byte, short, or char operand to int when evaluating an expression having binary operator.
  - If one operand is a long, float or double the whole expression is promoted to long, float or double respectively.

- What would be the output?
  - byte b = 50;
  - b = b * 2;
  - Correct way: b=(byte) (b*2);

Created by Dr. Ritu Jain

Format Specifiers

%d : for all integral types
%f: for all floating point types
%c: for characters
%s : for string
%b : for boolean

Created by Dr. Ritu Jain

Operators in java

- **Operator** in java is a symbol that is used to perform operations.
  - Arithmetic Operator
  - Bitwise Operator
  - Relational Operator
  - Boolean Logical Operator
  - Assignment Operator
  - ? Operator

Created by Dr. Ritu Jain

Arithmetic Operator

Operator	Result
+	Addition
–	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
–=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
– –	Decrement

- The operands of the arithmetic operators must be of a numeric type. You cannot use them on **boolean** types, but you can use them on **char** types, since the **char** type in Java is, essentially, a subset of **int**.
- When the division operator is applied to an integer type, there will be no fractional component attached to the result (truncation).

Created by Dr. Rubi Jain

```
// Demonstrate the basic arithmetic operators.
class BasicMath {
public static void main(String args[]) {
        int a = 1 + 1;   //2
        int b = a * 3;    //6
        int c = b / 4;    //1
        int d = c - a;    //-1
        int e = -d;       //1
        System.out.printf("a = %d, b = %d, c= %d, d = %d,  e= %d“, a, b, c, d, e);
         System.out.println("\nFloating Point Arithmetic");
        double da = 1 + 1;     //2.0
        double db = da * 3;    //6.0
        double dc = db / 4;    //1.5
        double dd = dc - a;     //-0.5
        double de = -dd;        //0.5
        System.out.printf(“a = %f, b = %f, c= %f, d = %f,  e= %f”, da, db, dc, dd,
de);
        }
}
```

Created by Dr. Ritu Jain

The Modulus Operator

It can be applied to floating-point types as well as integer types.

```
// Demonstrate the % operator.
class Modulus
{
        public static void main(String args[])
         {
                int x = 42;
                double y = 42.25;
                System.out.println("x mod 10 = " + x % 10);   //2
                System.out.println("y mod 10 = " + y % 10);   //2.25
        }
}
```

Created by Dr. Ritu Jain

Compound assignment operators

- Java provides special operators that can be used to combine an arithmetic operation with an assignment.

  a = a + 4; can be rewritten as a += 4;

  a = a % 4; can be rewritten as a %= 4;

  var = var op expression; can be rewritten as

  var op= expression;

- The compound assignment operators provide two benefits:
  - First, they save you a bit of typing,
  - Second, they are implemented more efficiently by the Java run-time system than are their equivalent long forms.

Created by Dr. Ritu Jain

WAP to demonstrate the use of compound assignment operators.

Increment and Decrement

- The increment operator increases its operand by one.
- The decrement operator decreases its operand by one.
- x = x + 1; can be rewritten as x++;
- x = x - 1 is equivalent to x--

Created by Dr. Ritu Jain

Increment and Decrement Operator

- In the **prefix form**, the operand is incremented or decremented before the value is obtained for use in the expression.
- In **postfix form**, the previous value is obtained for use in the expression, and then the operand is modified.
- For example: x = 42;
  y = ++x; // y= 43 x = 43
- when written like this,
  x = 42;
  y = x++; // y = 42, x= 43

Created by Dr. Ritu Jain

WAP to demonstrate the use of increment(pre and post) and decrement (press and post) operators.

Relational Operators

Operator	Result
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Created by Dr. Ritu Jain

Note: The outcome of these operations is **boolean** value.

Relational Operators

- Most frequently used in the expressions that control the **if** statement and the various **loop** statements.
- Any type in Java, including integers, floating-point numbers, characters, and boolean can be compared using the equality test, **==**, and the inequality test, **!=**.
- Only numeric types can be compared using <, >, <=, >= operators. That is, only integer, floating-point, and character operands may be compared to see which is greater or less than the other.
- int a = 4;
  int b = 1;
  boolean c = a < b;
- In this case, the result of **a<b** (which is **false**) is stored in **c**
- In Java, these statements must be written like this:

```
int done=0;
if(done == 0)... // This is Java-style.
if(done != 0)...
```

Created by Dr. Ritu Jain

Boolean Logical Operators

Operator	Result
&	Logical AND
|	Logical OR
^	Logical XOR (exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else

- The Boolean logical operators shown here operate only on **boolean** operands.
- Result will be **boolean** value.

Boolean Logical Operators

A	B	A | B	A & B	A ^ B	!A
False	False	False	False	False	True
True	False	True	False	True	False
False	True	True	False	True	True
True	True	True	True	False	False

Created by Dr. Ritu Jain

```
// Demonstrate the boolean logical operators.
class BoolLogic
{
    public static void main(String args[])
    {
        boolean a = true;
        boolean b = false;
        boolean c = a | b;
        boolean d = a & b;
        boolean e = a ^ b;
        boolean f = (!a & b) | (a & !b);
        boolean g = !a;

        System.out.println(" a|b = " + c);              //true
        System.out.println(" a&b = " + d);              //false
        System.out.println(" a^b = " + e);              //true
        System.out.println("(!a&b)|(a&!b) = " + f);    //true
        System.out.println(" !a = " + g);               //false
    }
}
```

Created by Dr. Ritu Jain

Short-Circuit Logical Operators

- If you use the || and **&&** forms, Java will not bother to evaluate the right-hand operand when the outcome of the expression can be determined by the left operand alone.
- This is very useful when the right-hand operand depends on the value of the left one in order to function properly.
- if (denom != 0 && num / denom > 10)

- WAP to Demonstrate the boolean logical operators.

Created by Dr. Ritu Jain

The Assignment Operator

Chain of assignments:

```
int x, y, z;
x = y = z = 100; // set x, y, and z to 100
```

Created by Dr. Ritu Jain

The ? Operator

- ***expression1 ? expression2 : expression3***
- *expression1 can be any expression that evaluates to a* ***boolean*** *value.*
- If *expression1* is **true**, then *expression2* is evaluated; otherwise, *expression3* is evaluated.
- E.g1 : ratio = denom == 0 ? 0 : num / denom;

Created by Dr. Ritu Jain

Operator Precedence (Highest to lowest)

Separators [],(), . have highest precedence

Highest						
++ (postfix)	– – (postfix)					
++ (prefix)	– – (prefix)	~	!	+ (unary)	– (unary)	(*type-cast*)
*	/	%				
+	–					
>>	>>>	<<				
>	>=	<	<=	instanceof		
==	!=					
&						
^						
|						
&&						
||						
?:						
->						
=	op=					
Lowest						

Operator Precedence

- Operators with higher precedence are evaluated before operators with relatively lower precedence.
- Associativity: When operators of equal precedence appear in the same expression, associativity govern which is evaluated first.
- unary operators, ternary operator, assignment operator: right to left for all other operators, it is left to right.
- Separators [], (), . have highest precedence.

Created by Dr. Ritu Jain

How to get Java Scanner

To get the instance of Java Scanner which reads input from the user, we need to pass the input stream (System.in) in the constructor of Scanner class. For Example:

```
Scanner in = new Scanner(System.in);
```

Created by Dr. Ritu Jain

Methods of Scanner Class

boolean	nextBoolean()	It scans the next token of the input into a boolean value and returns that value.
byte	nextByte()	It scans the next token of the input as a byte.
double	nextDouble()	It scans the next token of the input as a double.
float	nextFloat()	It scans the next token of the input as a float.
int	nextInt()	It scans the next token of the input as an Int.
String	nextLine()	This method can read the input till the end of line.
long	nextLong()	It scans the next token of the input as a long.
short	nextShort()	It scans the next token of the input as a short.
String	next()	This method can read the input only until a space(” “) is encountered.

```
//WAP for Accepting single character, int, float, string and double value from the //keyboard
import java.util.Scanner;
public class UserInput1
{public static void main(String args[])
   {
            Scanner sc=new Scanner(System.in);
            System.out.println("Enter an integer number”);
            int i=sc.nextInt();
            System.out.println("int i:  "+i);
            System.out.println("Enter an float number");
            float f=sc.nextFloat();
            System.out.println("float f:  "+f);
            System.out.println("Enter an double number");
            double d=sc.nextDouble();
            System.out.println("double d:  "+d);
            System.out.println("Enter a String");
            String s=sc.next();
            System.out.println("String s:  "+s);
            System.out.println("Enter a character");
            char c=sc.next().charAt(0);
            System.out.println("char c:  “+c);
            sc.close();
   }}
```

Control Statements

- Control statements are divided into three groups:
  1. Selection (if and switch)
  2. Iteration (do while, while, for, for each)
  3. Jump (break and continue)

Created by Dr. Ritu Jain

Selection statements

- allow to control the flow of program's execution based upon conditions known only during run-time.
- Java provides four selection statements:
    1) if
    2) if-else
    3) if-else-if
    4) switch

Created by Dr. Ritu Jain

if statement

if (*condition*) {

***statement block1*;**

}

Statement_2;

- If condition is true, statement_block1 is executed and the statement_2 is executed.
- If condition is false, statement_block1 is skipped and the statement_2 is executed.

```
int a, b;
//...
if(a < b) a = 0;
else b = 0;
```

Created by Dr. Ritu Jain

if else statement

```
if (condition) {
        statement block1;
}
else {
   statement block2;
}
```

- Note that the **else** clause is optional.
- If the condition is **true**, then *statement1* is executed, else *statement2* is executed.
- Remember that both *statement1* and *statement2* will not be executed in any situation.

nested if

```
if(condition)
{
    if(condition)
        statement;
}
else
    statement;

statement 2;
```

Nested if

```
if(i == 10) {
  if(j < 20) a = b;
  if(k > 100) c = d;
  else a = c;
}
else a = d;
```

Created by Dr. Ritu Jain

else if ladder

- The general form of the **if-else-if** ladder is :

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

Created by Dr. Ritu Jain

```
// Demonstrate if-else-if statements.
class IfElse {
  public static void main(String args[]) {
    int month = 4; // April
    String season;

    if(month == 12 || month == 1 || month == 2)
      season = "Winter";
    else if(month == 3 || month == 4 || month == 5)
      season = "Spring";
    else if(month == 6 || month == 7 || month == 8)
      season = "Summer";
    else if(month == 9 || month == 10 || month == 11)
      season = "Autumn";
    else
      season = "Bogus Month";

    System.out.println("April is in the " + season + ".");
  }
}
```

switch statement

- The **switch** statement provides multiway branching and is often a better alternative to nested **if-else-if** statements. The general form of the switch statement is :

```
switch(expression) {
        case value1: statement block;
                     break;
        case value2: statement block;
                     break;
        :
        case value n: statement block;
                      break;
        default:    default statement block;
        }
```

Created by Dr. Ritu Jain

Note: The expression must be of type **byte**, **short**, **int, char or String**

switch statement

- case statements must be of a type compatible with the expression.
- Each case value must be a literal.
- Duplicate case values not allowed.
- When a match is found, the set of statements following that **case** statement is executed. If none of the values match, then the default statement is executed.
- Note however, that the **default** statement is optional.
- **break** statement: to terminate the statement block
- The **break** statement is optional. If you do not use the **break** statement, execution continues with the next **case**.

Created by Dr. Ritu Jain

switch

- *The **switch** statement differs from **if** statement in that, the switch can test only for equality, whereas **if** can evaluate any type of **Boolean** expression.*
- *No two case constants in the same **switch** can have identical values.*
- *A **switch** statement is generally more efficient than a set of nested **if** statements.*

Created by Dr. Ritu Jain

```
// A simple example of the switch.
class SampleSwitch {
  public static void main(String args[]) {
    for(int i=0; i<6; i++)
      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;
        case 3:
          System.out.println("i is three.");
          break;
        default:
          System.out.println("i is greater than 3.");
      }
  }
}
```

Jain

Using a String to control Switch

```
class StringSwitch
{
  public static void main(String args[])
   {
      String str = “two”;
       switch(str)
        {
          case “one”:
                System.out.println(“one”);
                              break;
          case “two”:
                              System.out.println(“two”);
                              break;
             case “three”:
                              System.out.println(“three”);
          default:
                             System.out.println(“no match”);
             }
        }
}
```

Created by Dr. Ritu Jain