

Dayananda Sagar College of Engineering



Department of Information Science and
Engineering



Department of Information Science and Engineering



Course Name: INTRODUCTION TO JAVA

Course Code:18IS6IEJVA

Semester:6

Faculty : Mrs.Radhika T V
Assistant Professor , Dept of ISE
DSCE

INTRODUCTION

Creation of Java

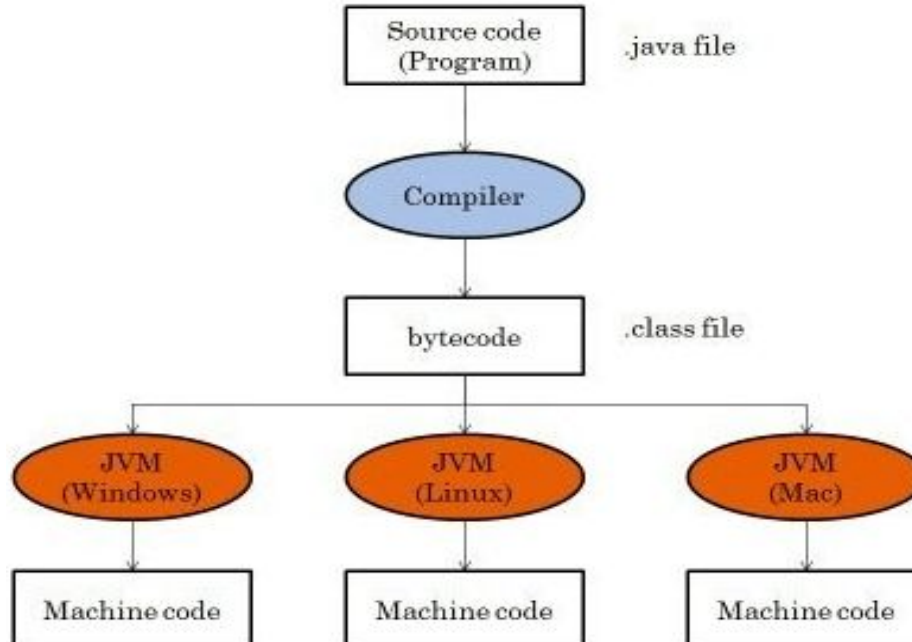
- Java was invented by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan (Green Team) at Sun Microsystems, Inc. in 1991.
- Initially designed for small, embedded systems in electronic appliances like set-top boxes.
- The language was initially called “Oak,” but was renamed “Java” in 1995.
- The primary motivation was the need for a platform-independent (that is, architecture- neutral) language.
- The trouble with C and C++ (and most other languages) is that they are designed to be compiled for a specific target.

Creation of Java(Cntd.)

- With the emergence of the World Wide Web, Java was propelled to the forefront of computer language design, because the Web, too, demanded portable programs.
 - Internet ultimately led to Java's large-scale success.
- Java shares some of the other attributes that helped make C and C++ successful.
 - Java is a programmer's language.
 - Java is cohesive and logically consistent.
 - Java gives the programmer full control
- While it is true that Java was influenced by C++, it is not an enhanced version of C++.

BYTECODE

- Bytecode (.class file) 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
 - The fact that a Java program is executed by the JVM also helps to make it secure.



Advantage of Java Bytecode

- Platform independence
- Add portability to Java which is lacking in languages like C or C++.
- Sun Microsystems captioned JAVA as "write once, read anywhere" or "WORA" in resonance to the bytecode interpretation.

The Java Buzzwords

- **Simple**
 - Java inherits the C/C++ syntax and many of the object-oriented features of C++.
- **Secure**
 - Allowing Java programs to be executed by the JVM makes Java program fully secured under the control of the JVM.
- **Portable**
 - Ability to run the program on any platform and no dependency on the underlying hardware / operating system.
- **Object-oriented**
 - The object oriented model in Java is simple and easy to extend

The Java Buzzwords(cntd..)

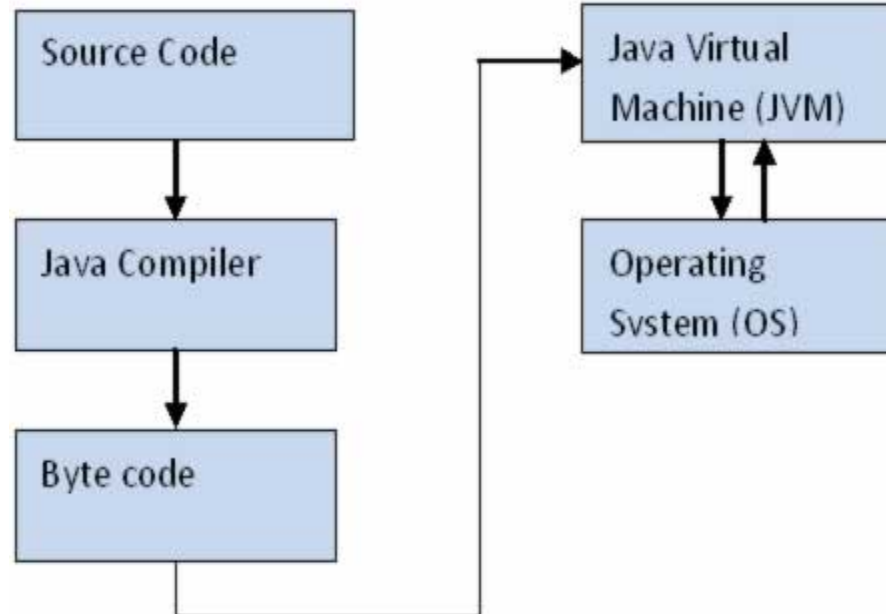
- **Robust**
 - Platform Independent:
 - Object Oriented Programming Language:
 - Memory Management:
 - Exception Handling:
- **Multithreaded**
 - Allows to develop program that can do multiple task simultaneously.
- **Architecture-neutral**
 - Java code does not depend on the underlying architecture and only depends on it JVM

The Java Buzzwords(cntd..)

- **Interpreted**
 - JVM interprets the ByteCode into Machine instructions during runtime.
- **High performance**
 - JVM was intelligently developed to interpret only the piece of the code that is required to execute and untouch the rest of the code
- **Distributed**
 - Java has a feature called Remote Method Invocation (RMI)
- **Dynamic**
 - Java programs access various runtime libraries and information

Java Architecture

Compilation and interpretation in Java



Object-Oriented Programming

- Everything in java is an object.
- **Two Paradigms**
 - *process-oriented model.*
 - *object-oriented programming,*

Abstraction

- Hiding implementation details from user and providing only the functionality.
- An essential element of object-oriented programming is *abstraction*.
- Humans manage complexity through abstraction.
- For example, people do not think of a car as a set of tens of thousands of individual parts- ***car is a single object***
- Abstraction lets you focus on what the object does instead of how it does it.

Abstraction (cntd..)

- Data Abstraction is the property by virtue of which only the essential details are displayed to the user.
- There are two ways to achieve abstraction in java
 - Abstract class
 - Interface

The Three OOP Principles

- 1. Encapsulation**
- 2. Inheritance**
- 3. Polymorphism**

Encapsulation

- The process of binding data and corresponding methods (behavior) together into a single unit is called **encapsulation in Java**.
- Encapsulation is a programming technique that binds the class members together and prevents them from being accessed by other classes.
- Every Java class is an example of encapsulation
- Another example of encapsulation is a capsule.
- Other examples are school bag, login to gmail account etc.

Encapsulation(cntd..)

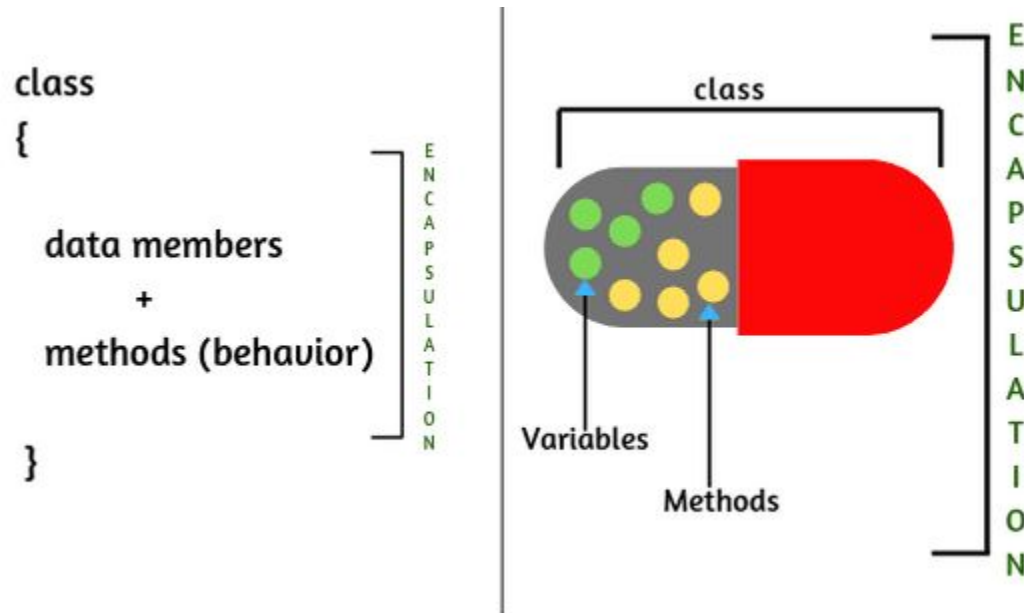


Fig: Encapsulation

Inheritance in Java

- The process of obtaining the data members and methods from one class to another class is known as **inheritance**.
- It is one of the fundamental features of object-oriented programming.
- **Important points**
 - Class which is give data members is known as **base or super or parent class**.
 - Class which is taking the data members and methods is known as **sub or derived or child class**.

Inheritance in Java

- The concept of inheritance is also known as re-usability .

Use of Inheritance

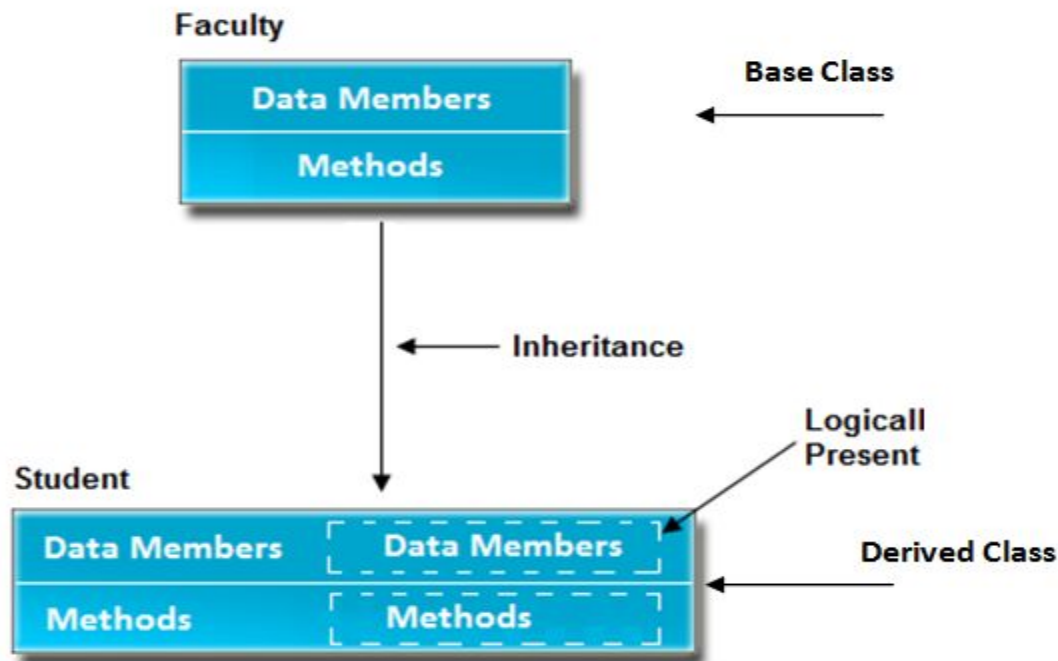
- For Method Overriding (used for Runtime Polymorphism).
- It's main uses are to enable polymorphism
- For code Re-usability

Syntax of Inheritance

```
class Subclass-Name extends Superclass-Name
{
//methods and fields
}
```

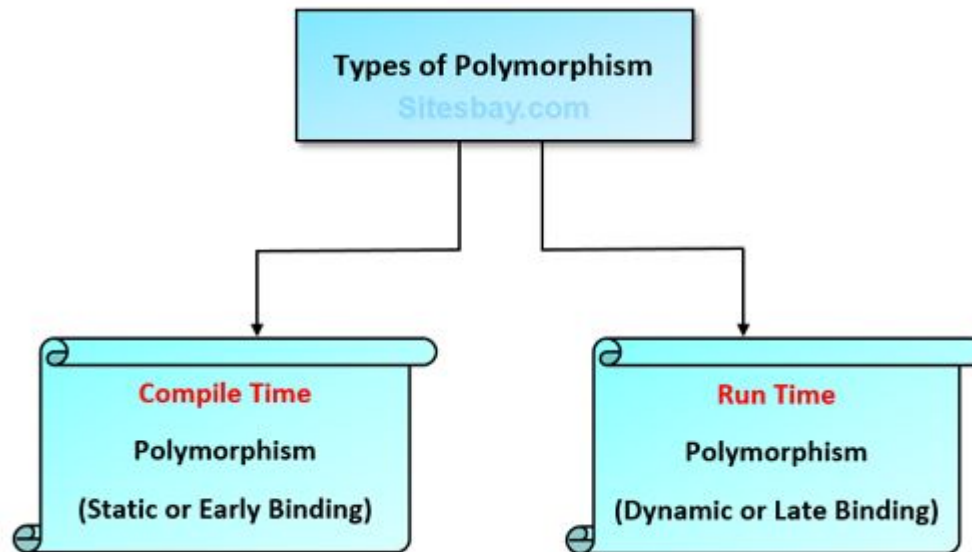
Inheritance in Java(cntd..)

- The real life example of inheritance is child and parents, all the properties of father are inherited by his son
- Another example is illustrated in diagram below



Polymorphism

- The process of representing one form in multiple forms is known as Polymorphism
- Polymorphism is derived from 2 greek words: poly and morphs.
- Polymorphism is not a programming concept but it is one of the principal of OOPs.



Real life example of polymorphism in Java



In Shopping malls behave like Customer

In Bus behave like Passenger

In School behave like Student

At Home behave like Son

Real life example of polymorphism in Java (cntd..)



A First Simple Program

```
/*  
This is a simple Java program. Call this file  
    "Example.java".  
*/  
  
    class Example {  
// Your program begins with a call to main(). public static  
    void main(String args[]) {  
System.out.println("This is a simple Java program.");  
    }  
    }
```

Steps of Execution

- Save the filename with classname. Here it is **Example.java**
- **Compiling:**

C:\>javac Example.java

- The **javac** compiler creates a file called **Example.class** that contains the bytecode version of the program.
- To actually run the program, you must use the Java application launcher, called **java**.

C:\>java Example

- When the program is run, the following output is displayed:

This is a simple Java program.

Example 2

```
/*  
Here is another short example. Call this file "Example2.java".  
*/  
class Example2 {  
public static void main(String args[])  
{ int num;  
num = 100;  
System.out.println("This is num: " + num);  
num = num * 2;  
System.out.print("The value of num * 2 is ");  
System.out.println(num);  
}  
}
```

Output: When you run this program, you will see the following output:

This is num: 100

The value of num * 2 is 200

Two Control Statements

The if Statement

Syntax: **if**(*condition*) *statement*;

- Here, *condition* is a Boolean expression
- **Example:** if(num < 100)

System.out.println("num is less than 100");

Operator	Meaning
<	Less than
>	Greater than
==	Equal to

Program that illustrates the if statement:

```
/* Demonstrate the if. Call this file "IfSample.java".*/
```

```
Class IfSample {  
    public static void main(String args[])  
{  
    int x, y;  
    x = 10;  
    y = 20;  
    if(x < y)  
        System.out.println("x is less than y");  
    x = x * 2;  
    if(x == y)  
        System.out.println("x now equal to y");  
    x = x * 2;  
    if(x > y) System.out.println("x now greater than y");  
    // this won't display anything  
    if(x == y) System.out.println("you won't see this");  
}  
}
```

The output :

x is less than y
x now equal to y
x now greater than y

The for Loop

- **Syntax:**

for(initialization; condition; iteration) statement;

- The initialization portion of the loop sets a loop control variable to an initial value.
- The condition is a Boolean expression that tests the loop control variable

Program that illustrates the for statement:

```
/*
```

Demonstrate the for loop. Call this file "ForTest.java".

```
*/
```

```
Class ForTest {  
public static void main(String args[])  
{  
    int x;  
    for(x = 0; x<10; x = x+1)  
        System.out.println("This is x: " + x);  
}  
}
```

The output :

```
This is x: 0  
This is x: 1  
This is x: 2  
This is x: 3  
This is x: 4  
This is x: 5  
This is x: 6  
This is x: 7  
This is x: 8  
This is x: 9  
y
```

Lexical Issues

1. Whitespace

- Java programs are a collection of whitespace, identifiers, literals, comments, operators, separators, and keywords.
- In Java, whitespace is a space, tab, or newline

2. Identifiers

- Identifiers are used for class names, method names, and variable names.
- It is descriptive sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters.
- They must not begin with a number. Java is case-sensitive, so **VALUE** is a different identifier than **Value**.

Example: *Valid identifiers are*

AvgTemp count, a4,\$test,this_is_ok

Invalid identifiers are

2Count, high-temp, Not/ok

Lexical Issues

3. Literals

- A constant value in Java is created by using a *literal* representation of it.
For example, here are some literals:

```
100 98.6 'X' "This is a test" int a=100;  
char d="x"
```

4. Comments

There are three types of comments defined by Java

- Single line
- Multiline
- Documentation: This type of comment is used to produce an HTML file that documents your program. The documentation comment begins with a `/**` and ends with a `*/`.

Lexical Issues

5. Separators

In Java, there are a few characters that are used as separators.

Symbol Name	Purpose
()	Used to contain lists of parameters in method definition and invocation.
{ }	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[]	Used to declare array types. Also used when dereferencing array values.
;	Terminates statements.
, int a=0, b=8,c;	Separates consecutive identifiers in a variable declaration.
. Import java.util.*;	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.

The Primitive Types

- Java defines eight *primitive* types of data: **byte**, **short**, **int**, **long**, **char**, **float**, **double**, **Boolean**.
- **Integers**-This group includes **byte**, **short**, **int**, and **long**, which are for whole-valued signed numbers.
- **Floating-point numbers**-This group includes **float** and **double**, which represent numbers with fractional precision.
- **Characters**-This group includes **char**, which represents symbols in a character set, like letters and numbers.
- **Boolean**-This group includes **boolean**, which is a special type for representing true/false values.

Integers

- Java defines four integer types: **byte**, **short**, **int**, and **long**. All of these are signed, positive and negative values.

Name	Width	Range	Example
long	64	−9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	long days ; long seconds;
int	32	−2,147,483,648 to 2,147,483,647	int a
short	16	−32,768 to 32,767	short s; short t;
byte	8	−128 to 127	byte b, c;

Floating-Point Types

- Floating-point numbers, also known as *real* numbers, are used when evaluating expressions that require fractional precision.
- Example: calculations such as square root. 0.4567×10^2
- Java implements the standard (IEEE–754) set of floating-point types and operators.
- There are two kinds of floating-point types, **float** and **double**,

Name	Width in Bits	Approximate Range	Example
Double	64	$4.9\text{e}-324$ to $1.8\text{e}+308$	double a;
float	32	$1.4\text{e}-045$ to $3.4\text{e}+038$	Float hightemp, lowtemp;

Characters

- In Java, the data type used to store characters is **char**.
- **char** in Java is not the same as **char** in C or C++.
- In C/C++, **char** is 8 bits wide.
- Java uses Unicode to represent characters.
- *Unicode* defines a fully international character set that can represent all of the characters found in all human languages. The range of a **char** is 0 to 65,536.

Characters (cntd..)

- **Program that demonstrates char variables:**

```
Class CharDemo {  
    public static void main(String args[])  
    {  
        char ch1, ch2;  
        ch1 = 88; // code for X ch2 = 'Y';  
        System.out.print("ch1 and ch2: ");  
        System.out.println(ch1 + " " + ch2);  
    }  
}
```

Output:

ch1 and ch2: X Y

Boolean

- Java has a primitive type, called **boolean**, for logical values. It can have only one of two possible values, **true** or **false**.

Program:

```
Class BoolTest
```

```
{
```

```
public static void main(String args[])
```

```
{ boolean b;
```

```
b = false;
```

```
System.out.println("b is " + b);
```

```
b = true;
```

```
System.out.println("b is " + b);
```

```
// a boolean value can control the if statement
```

```
if(b) System.out.println("This is executed.");
```

```
b = false;
```

```
if(b) System.out.println("This is not executed.");}}
```

Output:

b is false

b is true

This is executed.

Variables

- A variable is defined by the combination of an identifier, a type, and an optional initializer.

Declaring a Variable

- All variables must be declared before they can be used.
- **Syntax** : *type identifier [= value]*;
- Example: `int a, b, c; // declares three ints, a, b, and c.`
- `int d = 3, e, f = 5; // declares three more ints, initializing e and f`

Dynamic Initialization

- Java allows variables to be initialized dynamically, using any expression valid at the time the variable is declared.

Variables(cntd..)

// Program to Demonstrate dynamic initialization.

```
Class DynInit {  
    public static void main(String args[])  
    {  
        double a = 3.0, b = 4.0;  
        // c is dynamically initialized double  
        c = Math.sqrt(a * a + b * b);  
        System.out.println("Hypotenuse is " + c);  
    }  
}
```

Type Conversion and Casting

- Assigning a value of one type to a variable of another type is possible .
- If the two types are compatible, then Java will perform the conversion automatically.
- **For example**, it is always possible to assign an **int** value to a **long** variable.
- But not all types are compatible, and thus, not all type conversions are **implicitly** allowed. For instance, there is no automatic conversion defined from **double** to **byte**.
- It is still possible to obtain a conversion between incompatible types. *Casting should be done*,
- *Casting* performs an **explicit conversion** between incompatible types.

Type Conversion and Casting(cntd..),

Java's Automatic Conversions

- 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:
 - The two types are compatible.
 - The destination type is larger than the source type.
- When these two conditions are met, a ***widening conversion*** takes place.
- Widening conversions works between numeric types. But there are no automatic conversions from the numeric types to **char** or **boolean**

Type Conversion and Casting(cntd..),

Casting Incompatible Types

- What if we want to assign an **int** value to a **byte** variable?
- This kind of conversion is sometimes called a ***narrowing conversion***, since we are explicitly making the value narrower so that it will fit into the target type.
- To create a conversion between two incompatible types, you must use a cast.
- A *cast* is simply an explicit type conversion.
- **General form:**
- *(target-type) value*
- **target-type* specifies the desired type to convert the specified value to.
- If the integer's value is larger than the range of a **byte**, it will be reduced to modulo (the remainder of an integer division) **byte**'s range.

Type Conversion and Casting(cntd.,)

Example 1:

```
int a;
```

```
byte b;
```

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

Example 2:

```
int i = 257 % 127=3
```

```
byte b;
```

```
b = (byte) i;  00000000 00000000 00000001 00000001
```

- A different type of conversion will occur when a floating-point value is assigned to an integer type: ***truncation***.
- Example, if the value 1.23 is assigned to an integer, the resulting value will simply be 1.

Cntd..

//Program to Demonstrate casts.

```
class Conversion {  
public static void main(String args[]) {  
    byte b;  
    int i = 257;  
    double d = 323.142;  
    System.out.println("\nConversion of int to byte.");  
    b = (byte) i;  
    System.out.println("i and b " + i + " " + b);  
    System.out.println("\nConversion of double to int."); i =  
        (int) d;  
    System.out.println("d and i " + d + " " + i);  
    System.out.println("\nConversion of double to byte."); b =  
        (byte) d;  
    System.out.println("d and b " + d + " " + b);  
}  
}
```

Output:

Conversion of int to
byte.

i and b 257 1

Conversion of double to
int.

d and i 323.142 323

Conversion of double to
byte.

d and b 323.142 67

Cntd..,

Automatic Type Promotion in Expressions

```
byte a = 40;
```

```
byte b = 50;
```

```
byte c = 100;
```

```
int d = a * b / c;
```

- 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.

```
byte b = 50;
```

```
b = b * 2; // Error! Cannot assign an int to a byte!
```

Cntd..,

The Type Promotion Rules

- Java defines several *type promotion* rules that apply to expressions.
- All **byte**, **short**, and **char** values are promoted to **int**.
- If one operand is a **long**, the whole expression is promoted to **long**.
- If one operand is a **float**, the entire expression is promoted to **float**.
- If any of the operands is **double**, the result is **double**.

Cntd..,

Program:

```
class Promote {  
public static void main(String args[])  
{  
byte b = 42;  
char c = 'a';  
short s = 1024;  
int i = 50000;  
float f = 5.67f;  
double d = .1234;  
double result = (f * b) + (i / c) - (d * s);  
System.out.println((f * b) + " + " + (i / c) + " - " + (d * s));  
System.out.println("result = " + result);  
}  
}
```

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**.

One-Dimensional Arrays

- A one-dimensional array is a list of like-typed variables.
- To create an array, you first 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 link `month_days` with an actual, physical array of integers, you must allocate one using `new` and assign it to `month_days`. `new` is a special operator that allocates memory.
- `array-var = new type[size];`

One-Dimensional Arrays cntd

- **Accessing Array elements**
- All array indexes start at zero.
- For example, this statement assigns the value 28 to the second element of month_days.
`month_days[1] = 28;`
- The next line displays the value stored at index 3.
- `System.out.println(month_days[3]);`

One-Dimensional Arrays cntd

- **Accessing Array elements**
- All array indexes start at zero.
- For example, this statement assigns the value 28 to the second element of month_days.
`month_days[1] = 28;`
- The next line displays the value stored at index 3.
- `System.out.println(month_days[3]);`

One-Dimensional Arrays cntd

Write a Program to illustrate one dimensional Array

```
class Array {  
public static void main(String args[]) {  
int month_days[];  
month_days = new int[12]; } // int month_days[] = new int[12];  
// int month_days[] = new int[12];  
month_days[0] = 31;  
month_days[1] = 28;  
month_days[2] = 31;  
month_days[3] = 30;  
month_days[4] = 31;  
month_days[5] = 30;  
month_days[6] = 31;  
month_days[7] = 31;  
month_days[8] = 30;  
month_days[9] = 31;  
month_days[10] = 30;  
month_days[11] = 31;  
System.out.println("April has " + month_days[3] + " days.");
```

Output:

April has 30 Days

One-Dimensional Arrays cntd

// An improved version of the previous program.

```
Class AutoArray {  
public static void main(String args[])  
{  
Int month_days[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31,  
30, 31 };  
System.out.println("April has " + month_days[3] + "  
days.");  
}  
}
```

Multidimensional Arrays

- In Java, *multidimensional arrays* are actually 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];`

// Demonstrate a two-dimensional array

Multidimensional Arrays

// Demonstrate a two-dimensional array.

```
Class TwoDArray {  
public static void main(String args[])
```

```
    { int twoD[][]= new int[4][5];  
    int i, j, k = 0;  
    for(i=0; i<4; i++)  
        for(j=0; j<5; j++)  
        { twoD[i][j] = k;  
        k++;  
        }  
    for(i=0; i<4; i++) { for(j=0; j<5; j++)  
        System.out.print(twoD[i][j] + " ");  
        System.out.println();  
    }  
}
```

This program generates the following output:

```
0 1 2 3 4  
5 6 7 8 9  
10 11 12 13 14  
15 16 17 18 19
```

CHAPTER 2: OPERATORS

Arithmetic Operators

The operands of the arithmetic operators must be of a numeric type.
The following table lists the arithmetic operators:

Operator	Result
-----------------	---------------

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

Arithmetic Operators

```
// Demonstrate the basic arithmetic operators.
classBasicMath {
public static void main(String args[])
{
// arithmetic using integers
System.out.println("Integer Arithmetic");
int a = 1 + 1;
int b = a * 3;
int c = b / 4;
int d = c - a;
int e = -d;
System.out.println("a = " + a);
System.out.println("b = " + b);
System.out.println("c = " + c);
System.out.println("d = " + d); System.out.println("e = " + e);
// arithmetic using doubles
System.out.println("\nFloating Point Arithmetic");
```

Cntd..,

```
double da = 1 + 1;  
doubledb = da * 3;  
double dc = db / 4;  
doubledd = dc - a;  
double de = -dd;  
System.out.println("da = " + da);  
System.out.println("db = " + db);  
System.out.println("dc = " + dc);  
System.out.println("dd = " + dd);  
System.out.println("de = " + de);  
}  
}
```

Output:

Integer Arithmetic a = 2

b = 6

c = 1

d = -1

e = 1

Floating Point Arithmetic da = 2.0

db = 6.0

dc = 1.5

dd = -0.5