Java Programming
(MCA 4253)

COURSE OUTCOMES

At the end of the course, the student should be able to

CO1:	Implement basic concepts of OOP using Java.
CO2:	Create packages, interfaces, object cloning.
CO3:	Explain and use the concept of files, exception handling, generic programming
CO4:	implement event handling, develop multithreaded programming.
CO5:	demonstrate Collections, iterators, database programming

References:

1.	Herbert Schildt, Java The Complete Reference, 11th Edition, McGraw Hill, 2019.
2.	Cay S. Horstmann. Core Java: Volume I - Fundamentals. 11th Edition, Pearson Education, 2018.
3.	Cay S.Horstmann, Core Java: Volume II – Advanced Features, 11th Edition, Pearson Education, 2019.
4.	Herbert Schildt and Dale Skrien, Java Fundamentals, Tata McGraw-Hill Education, 2015.

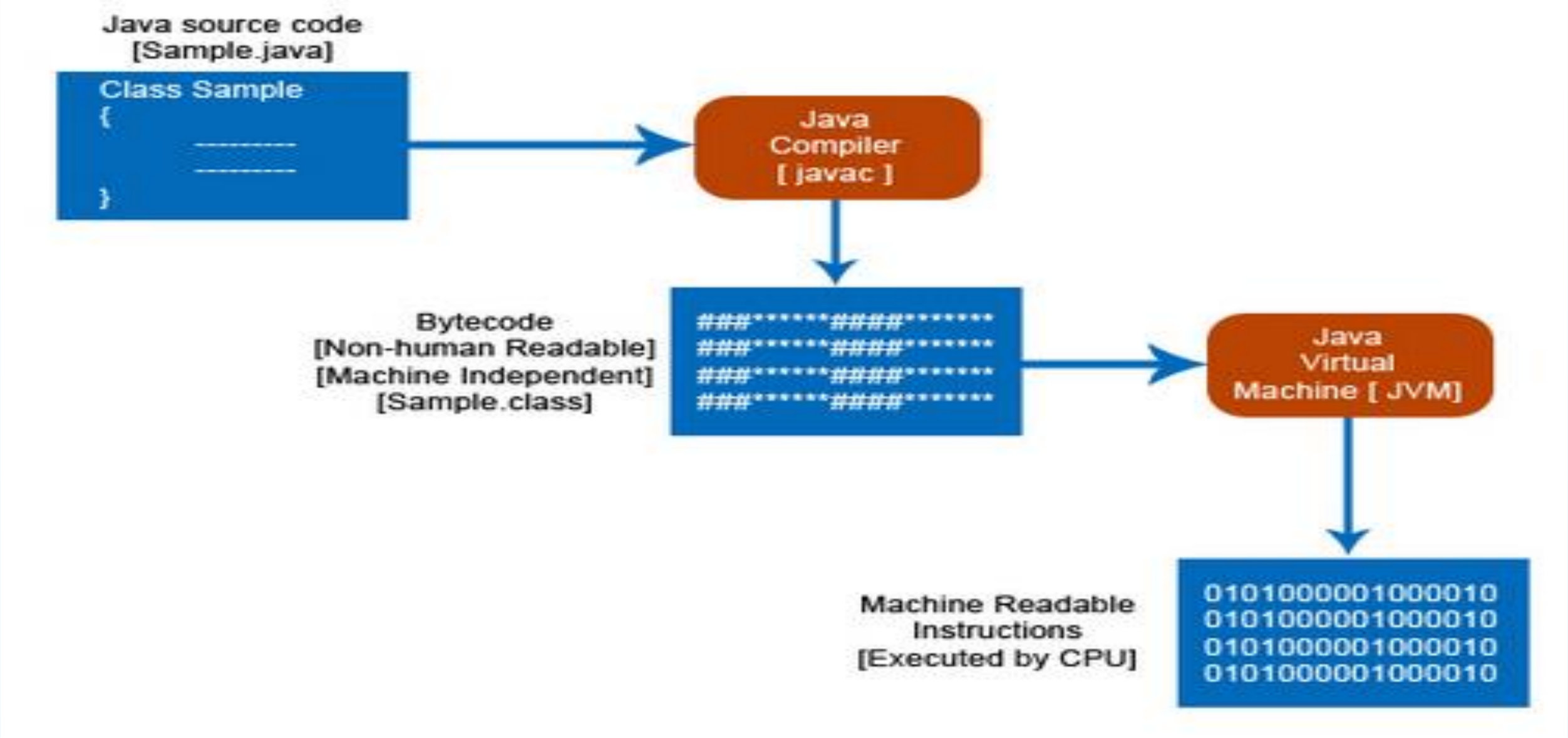
James Gosling

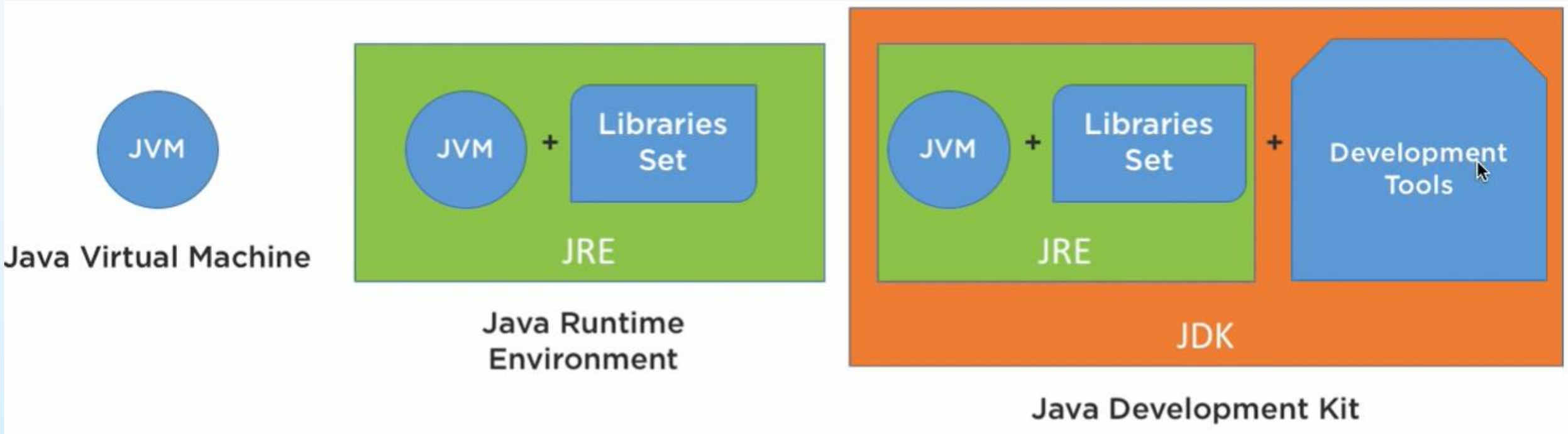
UNIT 1

BASIC CONCEPTS

JAVA

- **Program Life Cycle**




JVM
Java Virtual Machine
JVM
+
Libraries Set
JRE
Java Runtime Environment
JVM
+
Libraries Set
JRE
+
Development Tools
JDK
Java Development Kit

Java Development Kit (JDK)

Java Runtime Environment (JRE)

Java Virtual Machine (JVM)

Jre = jvm + library classes
(It contains set of libraries that jvm uses at run-time)

Jdk = jre + development tools
-A complete package for java application development

Java Development Kit (JDK)

- The JDK comes with a collection of tools that are used for developing and running Java programs.
- The source code is compiled using javac and executed using Java interpreter.
- Java compiler produces an intermediate code known as bytecode. The Java bytecode is the intermediate representation of program that contains instructions the Java interpreter will execute.

Basic Concepts

- **Features of Java**
  - Object Oriented Language
    - True OOPL
    - Reusability (built-in classes in packages)
  - Compiled and Interpreted
    - Compiler generates bytecode instructions
    - Interpreter generates machine instructions executable on the particular machine
  - Platform-independent (Architecture-Neutral) and Portable
    - Change in program not required
  - Robust and Secure
    - Compile-time and run-time check for data types
    - Exception handling mechanism
  - Multi-threaded
    - Do many things concurrently with synchronization

Basic Concepts

- **OOP Principles**
  - Encapsulation
    - Mechanism that binds together code and data it manipulates, and keeps it safe from outside interference and misuse
  - Inheritance
    - Process by which one object acquires the properties of another object
  - Polymorphism
    - A feature that allows one interface to be used for a general class of actions

Program Structure

Documentation Section
Package Statements
Import Statements
Interface Statements
Class Definitions

Main Method Class
{
  Main Method Definition
}

Reading input from keyboard

```
int a;
Scanner sc = new Scanner(System.in);
 a = sc.nextInt();
```

1) int nextInt()

It is used to read an integer value from the keyboard.

2) int nextFloat()

It is used to read a float value from the keyboard.

3) long nextLong()

It is used to read a long value from the keyboard.

4) String next()

It is used to read string value from the keyboard.

```
import java.util.Scanner;
```

Scanner example

```
import java.util.Scanner;

class test

{       public static void main(String args[])

        {       int a,b,c;

                Scanner sc = new Scanner(System.in);

                System.out.println("Enter a first number");

                a = sc.nextInt();

                System.out.println("Enter a second number");

                b = sc.nextInt();

                c = a+ b;

                System.out.println("sum is :"+c);

        }

}
```

Scanner example-2

```
import java.util.Scanner;
class GetInputFromUser{
  public static void main(String args[])   {
        int a;
        float b;
        String s;
        Scanner in = new Scanner(System.in);
        System.out.println("Enter a string");
        s = in.nextLine();
        System.out.println("You entered string "+s);
        System.out.println("Enter an integer");
        a = in.nextInt();
        System.out.println("You entered integer "+a);
        System.out.println("Enter a float");
        b = in.nextFloat();
        System.out.println("You entered float "+b);
  }
```

Data Types and Variables

Java Keywords

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	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

Comments

- // for single line, /* and */ for multiline

Data Types and Variables

Primitive Data Types

- Integers:
  - Java does not support unsigned integers

Name	Width in bits	Range	
long	64	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
int	32	-2,147,483,648	2,147,483,647
short	16	-32,768	32,767
byte	8	-128	127

- Floating-Point:

Name	Width in bits	Approximate Range	
double	64	4.9e-324	1.8e+308
float	32	1.4e-045	3.4e+038

- Character: char 16 (Unicode)
- Boolean: boolean 8 (true, false)

Data Types and Variables

Literals

- A sequence of characters (digits, letters and other characters) that represent constant values to be stored in variables.
- Categories of literals
  - Integer Literals : decimal, binary(0B),octal (0) and hexadecimal (0x)
    - Assign integer literal to byte or short – no error if within range
    - Long literal end with L
  - Floating-Point Literals: Standard and Scientific notation
    - Default is double precision (Optional D or d can be used)
    - Float literal end with F or f
  - Boolean Literals: Values true and false (not same as 1 and 0)
  - Character Literals: Unicode character set
    - Visible ASCII characters used within quotes (E.g., ‘a’, ‘6’)
    - Others using escape sequences (E.g., ‘\n’, ‘\b’, ‘\’’)
  - String Literals: Enclose within double quotes

Data Types and Variables

- **Identifiers**
  - Used for naming variables, methods, classes, interfaces and packages
  - Sequence of uppercase and lowercase letters, numbers, underscore and dollar-sign characters
  - Keywords cannot be used
  - Must not begin with a number
  - Case sensitive
  - Conventions:
    - Variables and Methods: Mixed Case
      - first letter of each word except the first word in uppercase, others in lowercase
    - Interfaces and Classes: Camel Case
      - first letter of each word in uppercase, others in lowercase
    - Packages: Lower Case

Data Types and Variables

- **Variables**
  - Syntax: type identifier [= value][, identifier [=value]…];
  - Examples:

```
int a, b, c;
int d = 3, e, f = 5;
byte z = 10;
double pi = 3.14159;
float k = 3.56f;
char x = 'x';
```

  - Types of variables
    - Local variables
    - Instance variables
    - Class / Static variables

Data Types and Variables

- **Local Variables**
  - Declared in methods, constructors or blocks
  - Created when a method, constructor or block is entered and destroyed when the method, constructor or block is exited
  - Visible only within the method, constructor or block
  - No default values; an initial value should be assigned before its first use
  - Access specifiers cannot be used

Data Types and Variables

- **Instance (Member) Variables**
  - Declared in a class, but outside a method, constructor or block
  - Created when an object is created and destroyed when the object is destroyed
  - Access specifiers can be used
  - Visible in all the methods, constructors and blocks in the class
  - Visibility for the subclasses can be specified using the access modifiers
  - Have default values (Numbers: 0, Boolean: false, Object references: null)
  - Can be accessed directly by using the variable name inside the class; however, within static methods and other classes (when it is given accessibility) should be called using the fully qualified name

```
objectReference.variableName
```

Question:

```
2   class BoolTest
3   {
4   public static void main(String args[])
5   {
6       boolean b;
7       b = false;
8       System.out.println("b is " + b);
9       b = true;
10      System.out.println("b is " + b);
11      if(b)
12          System.out.println("This is executed.");
13      b = false;
14      if(b)
15          System.out.println("This is not executed.");
16      System.out.println("10 > 9 is " + (10 > 9));
17      }
18  }
```

```
b is false
b is true
This is executed.
10 > 9 is true
```

Data Types and Variables

Scope and Lifetime of Variables

- Example:

```
void test()
{
        int x = 10;
        if (x == 10)
        {
                int y = 20;          // y = 20
                x = y * 2;           // x = 40
                System.out.println("x = " + x);
                System.out.println("y = " + y);
        }
        y = 100;                     // cannot find symbol
        x = x + 15;
        System.out.println("x = " + x);
}
```

Question

```
2   class ScopeExample
3   {
4       public static void main(String args[])
5       {
6           int b = 1;
7           {
8               int b = 2;
9               System.out.println(b);
10          }
11      }
12  }
```

Compile-time error, b already defined!

Data Types and Variables

Type Conversion and Casting

- Automatic Conversion: Conditions
  - Two types are compatible
    - Numeric types (integer and floating-point) are compatible with each other
  - Destination type is larger than source type
- Examples:
  - byte to int – possible
  - Any numeric type to char/boolean – not possible
  - char and boolean – not compatible with each other
- Automatic type conversion is done when a literal integer constant is stored into variables of type byte, short, long and char

Type conversion example:

```
1.  int  a = 2;
2.  float  b = 2.5f;
3.  double  c = 11.11;
4.  a = c;        // Error
5.  b = c;        // Error
6.  c = a;        // OK
7.  c = b;        // OK
```

Data Types and Variables

- **Type Conversion and Casting** (Continued …)
  - Casting incompatible types
    - Syntax: (target-type) value
    - Example: int a; byte b; b = (byte) a;
  - Truncation in case of floating-point values
    - Example: 1.23 assigned to integer → 1 (0.23 truncated)
  - Examples:

```
byte b;
int i = 257;
double d = 323.142;
b = (byte) i;          // 1 (remainder of special type of division)
i = (int) d;           // 323 (0.142 lost)
b = (byte) d;          // 67 (0.142 lost and reduced to modulo)
```

Note: If i = 255, then "b = (byte) i" results in -1

Data Types and Variables

Automatic Type Promotion in Expressions

- Depends on precision required
  - Precision required by an intermediate value exceeds the range of either operand
- Example 1:
  - byte a = 40, b = 50, c = 100; int d = a * b / c;
    - Result of a * b exceeds range of byte
    - Promotes byte, short or char operand to int during evaluation
    - a * b is performed using integers and …
- Example 2:
  - byte b = 50; b = b * 2;
    - Error, cannot assign an int to a byte (operands promoted to int)
    - Use explicit type-casting

```
byte b = 50;
b = (byte) (b * 2);
```

Data Types and Variables

Type Promotion Rules

- All byte, short and char promoted to int
- If one operand is a long, whole expression is promoted to long
- If one operand is a float, entire expression is promoted to float
- If any one of the operands is double, the result is double
- Example:

```
byte b = 42;        char c = 'a';        short s = 1024;
int i = 50000;      float f = 5.67f;     double d = 0.1234;
double result = (f * b) + (i / c) – (d * s);
```

  - » In f * b, b promoted to float; result float
  - » In i / c, c promoted to int; result int
  - » In d * s, s promoted to double; result double
  - » float + int → float (int promoted to float)
  - » float – double → double (float promoted to double)

Arrays

Basic Concepts of an Array

- *Arrays* are dynamically created objects.
- An array object contains a number of variables.
  - Number of variables may be zero (i.e., array is empty).
  - Variables contained in an array have no names
    - They are referenced by array access expressions that use non-negative integer index values (from 0 to n-1).
  - Array objects have a *public* instance variable called **length**, which gives the number of elements in the array.

Arrays

One-Dimensional Arrays

- Syntax: type var-name[];
  - Only declaration, memory not allocated
  - Allocate memory using new operator

    array-var = new type[size]
- Example 1:

```
int one[];
one = new int [5];
```

- Example 2:

```
int one[] = new int [12];
```

- Example 3:

```
int one[] = {1, 2, 3, 4, 5};
```

Arrays

Two-Dimensional Arrays

- Syntax: type var-name[][];
  - Only declaration, memory not allocated
  - Allocate memory using new operator

    array-var = new type[size][size]
- Example 1:

```
int two[][];
two = new int [3][2];
```

- Example 2:

```
int two [][] = new int [3][2];
```

- Example 3:

```
int two[][] = {{1, 2}, {3, 4}, {5, 6}};
```

Arrays

- **Two-Dimensional Arrays** (Continued …)
  - Example 4:

```
int two[][] = new int [3][];
two [0] = new int [2];
two [1] = new int [2];
two [2] = new int [2];
```

  - Example 5:

```
int two [][] = new int [3][];
two [0] = new int [1];
two [1] = new int [3];
two [2] = new int [2];
```

(Differing-size second dimension)

Arrays

- **Alternate methods of declaration**
  - Alternate syntax:

    1-D: type [] var-name;

    2-D: type [][] var-name;
  - Example 1:

```
int a[] = new int [3];
int[] a = new int [3];
```
  - Example 2:

```
int b[][] = new int [3][4];
int[][] b = new int [3][4];
```
  - Example 3:

```
int num1[], num2[], num3[];
int[] num1, num2, num3;
```

Array Example:

```
2   class ArrayDemo
3   {
4       public static void main( String []args)
5       {
6           int A[], size;
7           Scanner S = new Scanner(System.in);
8           System.out.println("Enter the size of array:");
9           size = S.nextInt();
10
11          A = new int[size];
12
13          System.out.println("Enter elements: ");
14          for( int i = 0 ; i < size ; i++ )
15              A[i] = S.nextInt();
16
17          System.out.println("Array elements are: ");
18          for( int i = 0 ; i < A.length ; i++ )
19              System.out.println( A[i] );
20      }
21  }
```

OUTPUT

```
Enter the size of array:
4
Enter elements:
11 22 33 44
Array elements are:
11
22
33
44
```

Array Example-1:

```
1   class ArrayTest
2   {
3       public static void main( String []args )
4       {
5           int a[] = { 11 , 22 , 33 };
6           int b[] = a;
7           a[1] = 100;
8           for( int i = 0 ; i < a.length ; i++)
9               System.out.printf("%d\t",b[i]);
10      }
11  }
```

OUTPUT

```
11      100     33
```

Array Example-2:

```
1   class ArrayTest
2   {
3       public static void main( String []args )
4       {
5           int a[] = { 11 , 22 , 33 };
6           int b[] =a.clone();
7           a[1] = 100;
8           for( int i = 0 ; i < a.length ; i++)
9               System.out.printf("%d\t",b[i]);
10      }
11  }
```

OUTPUT

```
11      22      33
```

2-D Array example:

```
1   class Array2DTest
2   {
3       public static void main( String []args )
4       {
5           int M[][] = { { 11 , 22 , 33 } ,
6                         { 44 , 55 , 66 }
7                       };
8           for( int i = 0 ; i < M.length ; i++)
9           {
10              for( int j = 0 ; j < M[i].length ; j++ )
11                  System.out.printf("%d\t", M[i][j] );
12              System.out.println();
13          }
14      }
15  }
```

```
11      22      33
44      55      66
```

2-D Array example-2:

```
1  class Array2DTest
2  {
3      public static void main( String []args )
4      {
5          int M[][] = { { 11 , 22 , 33 } ,
6                        { 44 } ,
7                        { 55 , 66 }
8                      };
9          for( int i = 0 ; i < M.length ; i++)
10         {
11             for( int j = 0 ; j < M[i].length ; j++ )
12                 System.out.printf("%d\t", M[i][j] );
13             System.out.println();
14         }
15     }
16 }
```

```
11      22      33
44
55      66
```

Operators

Arithmetic and Relational Operators

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

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

Operators

Bitwise and Boolean Logical Operators

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

Operator	Result
&	Logical AND
|	Logical OR
^	Logical 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

Operators

Assignment and Ternary Operators

- Assignment operator

  Syntax: var = expression;

  Example: x = 10; y = 4.5; p = q = ‘a’

- Ternary Operator

  Syntax: expression 1 **?** expression 2 **:** expression 3;

  Example: ratio = (den == 0) ? 0 : (num / den);

Operators

Operator Precedence

Highest			
( )	[ ]	•	
++	--	~	!
*	/	%	
+	-		
>>	>>>	<<	
>	>=	<	<=
==	!=		
&			
|			
&&			
||			
?:			
=	op=		
Lowest			

Note:

1. First row contains separators (and not operators), but they act as operators in an expression.
2. Within each row, all operators have the same precedence.
3. Precedence can be changed using parentheses.

Mathematical Functions

Math class (java.lang)

Function	Meaning	Function	Meaning
sin(x)	Sine of x	log10(x)	Logarithm of x base 10
cos(x)	Cosine of x	sqrt(x)	Square root of x
tan(x)	Tangent of x	cbrt(x)	Cube root of x
asin(x)	Arc Sine of x	abs(x)	Absolute of x
acos(x)	Arc Cosine of x	max(a, b)	Maximum of a and b
atan(x)	Arc Tangent of x	min(a, b)	Minimum of a and b
atan2(x, y)	Arc Tangent of x / y	ceil(x)	Ceiling of x
pow(x, y)	x to the power y	floor(x)	Floor of x
exp(x)	Exponential of x	round(x)	Rounded value of x
log(x)	Natural logarithm of x		

Control Statements

- **Selection statements**
  - The *if* statement

    Syntax: Same as in C and C++
    - *if* statements can be nested
    - *if-else-if* ladder can be used
  - The *switch* statement

    Syntax: Same as in C and C++
    - *switch* statements can be nested

Note:

From JDK 7 onwards, strings can be used as case labels in ‘switch’ statement

Control Statements

- **Iteration statements**
  - The *while* statement

    Syntax: Same as in C and C++
  - The *do-while* statement

    Syntax: Same as in C and C++
  - The *for* statement

    Syntax: Same as in C and C++

Control Statements

- **Iteration statements** (Continued …)
  - The *for-each version* of *for* statement (enhanced for)

Syntax:

```
for (type itr-var : collection)
        statement block;
```

Example 1:

```
int nums[] = {1, 2, 3, 4, 5}, sum = 0;
for (int x : nums)
        sum += x;
```

Example 2:

```
int nums[] = {1, 2, 3, 4, 5}, sum = 0;
for (int x : nums)
        {
                sum += x;
                if (x == 2) break;
        }
```

Control Statements

- **Iteration statements** (Continued …)

Example 3:

```
int nums[] = {1, 2, 3, 4, 5}, sum = 0;

for (int x : nums)
{
        sum += x;  x = x + 2;          // no effect on nums
}
// collection is "read-only"
```

Foreach -2D array

```
1   class ForEach2D
2   {
3     public static void main(String args[])
4     {
5       int M[][] = { { 11 , 22 , 33 } ,
6                                { 44 } ,
7                           { 55 , 66 }
8                       };
9
10      for( int x[] : M )
11      {
12          for( int y : x )
13          {
14              System.out.printf( "%d\t", y );
15          }
16          System.out.println();
17      }
18    }
19  }
```

```
11      22      33
44
55      66
```

question

```java
class ForEach3
{
  public static void main(String args[])
  {
    int sum = 0;
    int nums[][] = new int[3][5];

    for(int i = 0; i < 3; i++)
      for(int j=0; j < 5; j++)
        nums[i][j] = (i+1)*(j+1);

    for(int x[] : nums)
    {
      for(int y : x)
      {
        System.out.println("Value is: " + y);
        sum += y;
      }
    }
    System.out.println("Summation: " + sum);
  }
}
```

```
Value is: 1
Value is: 2
Value is: 3
Value is: 4
Value is: 5
Value is: 2
Value is: 4
Value is: 6
Value is: 8
Value is: 10
Value is: 3
Value is: 6
Value is: 9
Value is: 12
Value is: 15
Summation: 90
```

Control Statements

- **Jump (transfer of control) statements**
  - Basic form of the *break* statement

    Syntax: Same as in C and C++

    - Used with switch statement and loops

  - Labeled *break* statement

    Syntax: *break* label;

Control Statements

```java
class break_example
{
  public static void main(String args[])
  {
    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if ( j == 1 )
                break;
            System.out.println ("In");
        }

        System.out.println ("Mid");
    }

    System.out.println ("Out");

  }
}
```

OUTPUT

```
In
Mid
In
Mid
Out
```

Control Statements

- **Jump (transfer of control) statements** (Continued …)
  - Basic form of the *continue* statement

    Syntax: Same as in C and C++

    - Used for early termination of loops

  - Labeled *continue* statement

    Syntax: *continue* label;

Control Statements

```java
class Continue_Demo
{
    public static void main( String args[] )
    {
        for( int i = 0; i < 2; i++ )
        {
            for( int j = 0; j < 3; j++ )
            {
                if( j == 1 )
                    continue;
                System.out.println ("In");
            }
            System.out.println ("Mid");
        }
        System.out.println ("Out");
    }
}
```

OUTPUT

```
In
In
Mid
In
In
Mid
Out
```

Control Statements

- **Jump (transfer of control) statements** (Continued …)
  - The *return* statement

    Syntax: Same as in C and C++

    - Used to return from a method
  - Other jump statements (used with exception handling)
    - try
    - catch
    - throw
    - throws
    - finally

Command Line Arguments

- **Command Line Arguments**
  - Pass information into a program on invoking it
  - Accessed using String array argument of main()
    - args[0], args[1], …
  - All arguments are passed as strings
    - Conversion to other types should be done in the program

Command Line Arguments

Command Line Arguments

```
1  class CommandLine
2  {
3      public static void main( String args[] )
4      {
5          for( int i = 0; i < args.length; i++ )
6              System.out.println("args[" + i + "] = " + args[i] );
7      }
8  }
```

Run this as: java CommandLine **Java 100 -5**

Output:

```
args[0] = Java
args[1] = 100
args[2] = -5
```

Break statement with label

```java
class Break
{
  public static void main(String args[])
  {
    boolean t = true;

    first:
    {
      second:
      {
        third:
        {
          System.out.println("Before the break.");
          if(t)
            break second; // break out of second block
          System.out.println("This won't execute");
        }
        System.out.println("This won't execute");
      }
      System.out.println("This is after second block.");
    }
  }
}
```

OUTPUT

```
Before the break.
This is after second block.
```

Control Statements

```
3     public static void main(String args[])
4     {
5       outerLoop: for (int i = 0; i < 4; i++)
6                  {
7                       for (int j = 0; j < 5; j++)
8                       {
9                            if (j == 3)
10                                break outerLoop;
11                           System.out.println ("In");
12                      }
13                      System.out.println ("Mid");
14                 }
15      System.out.println ("Out");
16    }
```

OUTPUT

```
In
In
In
Out
```

Control Statements

```
3     public static void main(String args[])
4     {
5       outerLoop: for (int i = 0; i < 3; i++)
6                  {
7                       for (int j = 0; j < 5; j++)
8                       {
9                            if (j == 2 )
10                                continue outerLoop;
11                           System.out.println ("In");
12                      }
13                      System.out.println ("Mid");
14                 }
15      System.out.println ("Out");
16    }
```

OUTPUT

```
In
In
In
In
In
In
Out
```

Class Fundamentals

Classes and Objects

- Class defines a new data type; Used to create objects
- Class is a template for an object; Object is an instance of class
- Defining a class

```
class className [extends superClassName]
{
        [fields declaration;]
        [methods declaration;]
}
```

- Class can be empty
- No need of semicolon after closing brace

Class Fundamentals

- **Classes and Objects** (Continued …)
  - Members of a class
    - Instance variables (fields / member variables): Data / variables defined within a class
    - Methods: Contain the code
  - Declaring member variables
    - Normal variable declaration
  - Declaring methods

```
type methodName (parameter-list)
{
        method-body;
}
```

  - Constructor: As in C++
  - Destructor: No destructor in Java
    - Uses the concept of garbage collection and finalize() method

Class Fundamentals

- **Classes and Objects** (Continued …)
  - Creating objects (instantiation)

    Syntax

```
        className objectName;
        objectName = new className();
            OR
        className objectName = new className();
```

    Example

```
        Box myBox;
        myBox = new Box();
            OR
        Box myBox = new Box();
```

Class Fundamentals

- **Classes and Objects** (Continued …)
  - Accessing class members

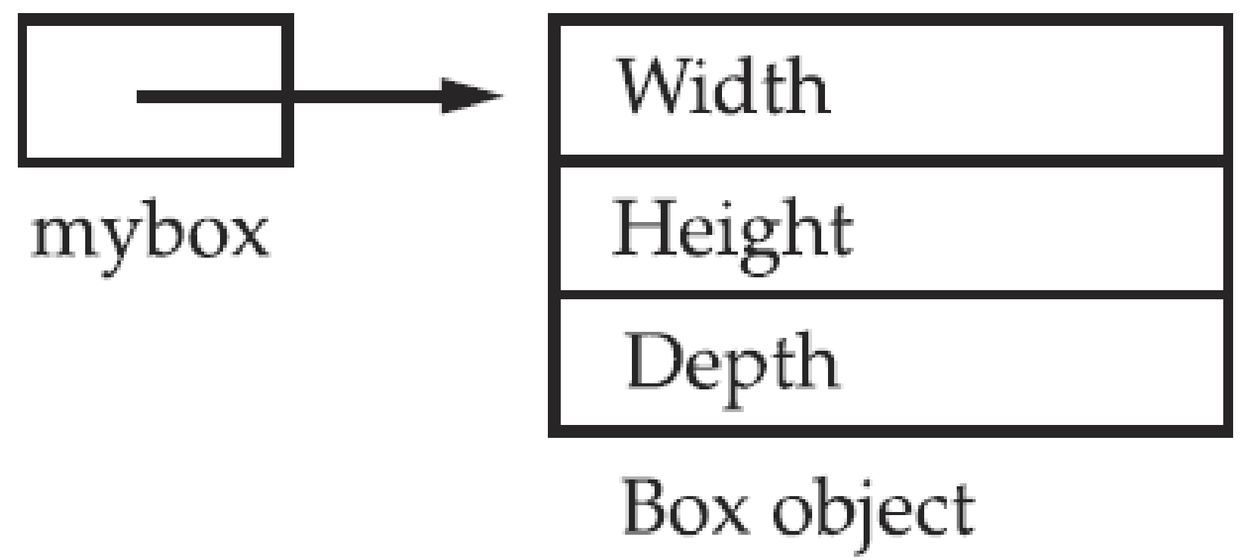
    Syntax

```
objectName.variableName = value;              // set
value = objectName.variableName;              // get
objectName.methodName(parameter-list);        // invoke a method
```

    Example

```
myBox.length = 10;
myBox.getData(10, 15, 5);
myBox.volume();
```

Class Fundamentals

Statement	Effect
Box mybox;	
mybox = new Box();	

Class Ex-1:

```
1   class Box
2   {
3       double width;
4       double height;
5       double depth;
6
7       double volume()
8       {
9           return width * height * depth;
10      }
11
12      void setDim(double w, double h, double d)
13      {
14          width = w;
15          height = h;
16          depth = d;
17      }
18  }
```

```
19  class BoxDemo
20  {
21      public static void main(String args[])
22      {
23          Box mybox1 = new  Box();
24          Box mybox2 = new  Box();
25          double  vol;
26
27          mybox1.setDim(10, 20, 15);
28          mybox2.setDim(3, 6, 9);
29
30          vol = mybox1.volume();
31          System.out.println("Volume is " + vol);
32
33          vol = mybox2.volume();
34          System.out.println("Volume is " + vol);
35      }
36  }
```

width *height* *depth*

mybox1

width	10
height	20
depth	15

mybox2

width	3
height	6
depth	9

OUTPUT

```
Volume is 3000.0
Volume is 162.0
```

Class Ex-2:

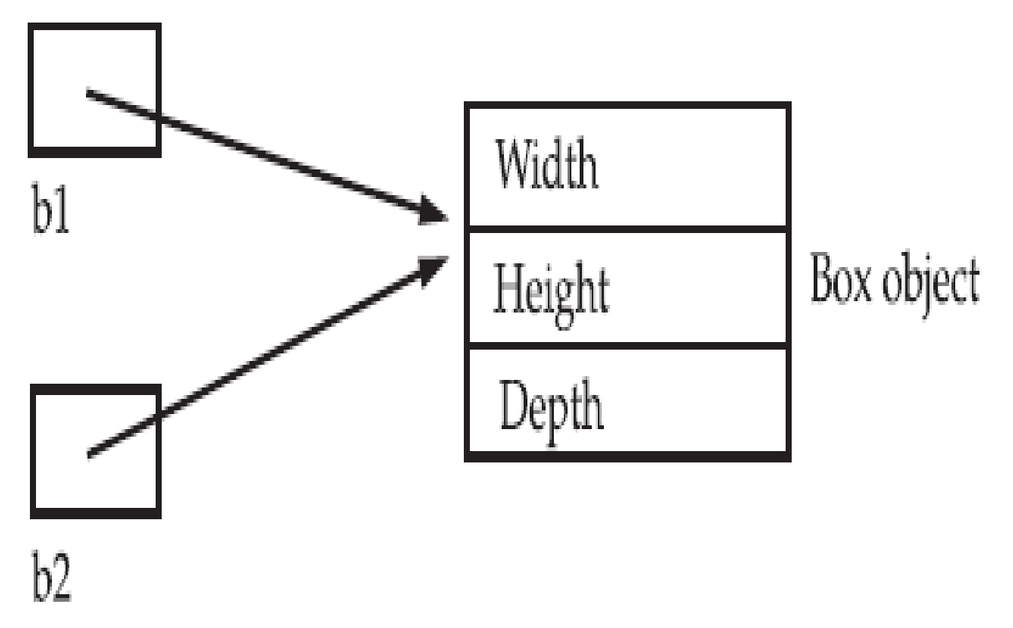
```
18  class BoxDemo
19  {
20      public static void main(String args[])
21      {
22          Box mybox1 = new  Box();
23          Box mybox2 = new  Box();
24          double  vol;
25
26          mybox1.width = 10;
27          mybox1.height = 20;
28          mybox1.depth = 15;
29
30          mybox2.width = 3;
31          mybox2.height = 6;
32          mybox2.depth = 9;
33
34          vol = mybox1.volume();
35          System.out.println("Volume is " + vol);
36
37          vol = mybox2.volume();
38          System.out.println("Volume is " + vol);
39      }
40  }
```

Class Fundamentals

- **Classes and Objects** (Continued …)
  - Object Reference
    - When we assign one object reference variable to another object reference variable, new copy of the object is not created

```
Box b1 = new Box();
Box b2 = b1;
```

    - Reference variables b1 and b2 refer to the same object



When we assign one object reference variable to another object reference variable, we are not creating a copy of the object, we are only making a copy of the reference.

Class Ex-3:

```java
class Box
{
  double width;
  double height;
}

class BoxDemo
{
  public static void main(String args[])
  {
    Box box1 = new Box();

    box1.width = 10;
    box1.height = 20;
    Box box2 = box1;
    box2.height=25;
    System.out.println("Box1: " +box1.width+ ","+box1.height);
    System.out.println("Box2: " +box2.width+ ","+box2.height);
  }
}
```

OUTPUT

```
Box1: 10.0,25.0
Box2: 10.0,25.0
```

Class Ex-4:

```
1   class Box
2   {
3       double width;
4       double height;
5       double depth;
6
7       void setDim(double w, double h, double d)
8       {
9           width = w;
10          height = h;
11          depth = d;
12      }
13      void show()
14      {
15          System.out.println( width+" "+height+" "+depth );
16      }
17  }
```

```
19  class BoxDemo3
20  {
21      public static void main(String args[])
22      {
23          Box mybox1 = new  Box();
24          Box mybox2, mybox3;
25          mybox2 = mybox1;
26          mybox3 = mybox2;
27
28          mybox1.setDim(10, 20, 15);
29          mybox2.setDim(3, 6, 9);
30
31          mybox3.width++;
32          mybox3.height++;
33          mybox3.depth++;
34
35          System.out.println("\nMybox1 :");    mybox1.show();
36
37          System.out.println("\nMybox2 :");    mybox2.show();
38
39          System.out.println("\nMybox3 :");    mybox3.show();
40      }
41  }
```

OUTPUT:

```
Mybox1 :
4.0 7.0 10.0

Mybox2 :
4.0 7.0 10.0

Mybox3 :
4.0 7.0 10.0
```

Class Ex-5:

```
1   class Box
2   {
3       double width;
4       double height;
5       double depth;
6
7       void setDim(double w, double h, double d)
8       {
9           width = w;
10          height = h;
11          depth = d;
12      }
13      void show()
14      {
15          System.out.println( width+" "+height+" "+depth );
16      }
17  }
```

```java
19  class BoxDemo3
20  {
21      public static void main(String args[])
22      {
23          Box mybox1 = new  Box();
24          Box mybox2, mybox3;
25          mybox2 = new Box();
26          mybox3 = mybox2;
27
28          mybox1.setDim(10, 20, 15);
29          mybox2.setDim(3, 6, 9);
30
31          mybox3.width++;
32          mybox3.height++;
33          mybox3.depth++;
34
35          System.out.println("\nMybox1 :");    mybox1.show();
36
37          System.out.println("\nMybox2 :");    mybox2.show();
38
39          System.out.println("\nMybox3 :");    mybox3.show();
40      }
41  }
```

OUTPUT:

```
Mybox1 :
10.0 20.0 15.0

Mybox2 :
4.0 7.0 10.0

Mybox3 :
4.0 7.0 10.0
```

Access Control

- **Access specifiers**
  - public : Can be accessed by any other code
    - main() is declared as public
      (should be accessible to Java run-time system)
  - private : Can be accessed by other members of its class
  - protected : Applies only when inheritance is used

- When no access specifier is used
  - By default, member of a class is public within its own package (but, cannot be accessed outside its package)

Class Ex-6:

```
1   class Test
2   {
3     int a; // default access
4     public int b; // public access
5     private int c; // private access
6
7     // methods to access c
8     void setc(int i)
9     { // set c's value
10      c = i;
11    }
12    int getc()
13    { // get c's value
14      return c;
15    }
16  }
```

```java
class AccessTest
{
  public static void main(String args[])
  {
    Test ob = new Test();

    // These are OK, a and b may be accessed directly
    ob.a = 10;
    ob.b = 20;

    //  ob.c = 100; // Error!

    // You must access c through its methods
    ob.setc(100); // OK

    System.out.println("a, b, and c: " + ob.a + " " +
                       ob.b + " " + ob.getc() );
  }
}
```

OUTPUT:

```
a, b, and c: 10 20 100
```

Class Fundamentals

- **Constructors**
  - A method that
    - Initializes an object immediately upon creation
    - Has same name as class name
    - Has no return type, not even void
    - Can accept parameters (Parameterized constructors)

Constructor example -1

```
1   class Box
2   {
3     private double width;
4     private double height;
5     private double depth;
6
7     Box()       //No-arg constructor
8     {
9       System.out.println("Constructing Box");
10      width = 10;
11      height = 10;
12      depth = 10;
13    }
14
15    public double volume()
16    {
17      return width * height * depth;
18    }
19  }
```

```java
class BoxDemo6
{
  public static void main(String args[])
  {
    Box mybox1 = new Box();
    Box mybox2 = new Box();

    double vol;

    vol = mybox1.volume();
    System.out.println("Volume is " + vol);

    vol = mybox2.volume();
    System.out.println("Volume is " + vol);
  }
}
```

OUTPUT:

```
Constructing Box
Constructing Box
Volume is 1000.0
Volume is 1000.0
```

Constructor Ex -2 :

```
1   class Box
2   {
3     private double width;
4     private double height;
5     private double depth;
6
7     Box()
8     {
9       width = height = depth = 1;
10    }
11
12    Box(double w, double h, double d)
13    {
14      width = w;
15      height = h;
16      depth = d;
17    }
18
19    public double volume()
20    {
21      return width * height * depth;
22    }
23  }
```

Constructor Ex -2…

```
25  class BoxDemo7
26  {
27    public static void main(String args[])
28    {
29      Box mybox1 = new Box( 5 , 9 , 10 );
30      Box mybox2 = new Box( 3 , 6 , 10 );
31      Box mybox3 = new Box();
32
33      double vol;
34
35      vol = mybox1.volume();
36      System.out.println("Volume is " + vol);
37
38      vol = mybox2.volume();
39      System.out.println("Volume is " + vol);
40
41      vol = mybox3.volume();
42      System.out.println("Volume is " + vol);
43    }
44  }
```

OUTPUT:

```
Volume is 450.0
Volume is 180.0
Volume is 1.0
```

Class Fundamentals

The *this* keyword

- Used inside any method to refer to the object that invoked it
- Example:

```
Box (double w, double h, double d)
{
        this.width  = w;          // width  = w;
        this.height = h;          // height = h;
        this.depth  = d;          // depth  = d;
}
```

  - Here, use of *this* keyword is redundant (not necessary)
  - Even without it, the three statements refer to the instance variables of the object that invoked the constructor

Class Fundamentals

- **The *this* keyword** (Continued …)
  - Used for resolving any name space collisions between instance variables and formal arguments
    - Instance variable hiding: When instance variables have the same name as local variables (including formal arguments), local variables hide the instance variables
  - Example

```
Box (double width, double height, double depth)
{
        this.width  = width;
        this.height = height;
        this.depth  = depth;
}
```

Overloading of Methods

Method Overloading

- Defining two or more methods in a class that share the same name, with different parameter declarations
  - A type of Polymorphism
  - Overloaded methods must differ in the type and/or number of parameters
  - Invoking an overloaded method
    - Java uses type and/or number of arguments for determining the required method (and not the return type)
    - Possibility of type promotion
- In C, three versions of abs():
  - abs() for integer
  - fabs() for floating-point
  - labs() for long integer
- In Java, abs() overloaded :

```
public static int abs( int i )
public static double abs( double d )
public static float abs( float  f )
public static long abs( long lng )
```

Method overloading – Eg:

```
1   class OverloadDemo
2   {
3     void test()
4     {
5       System.out.println("No parameters");
6     }
7
8     void test(int a)
9     {
10      System.out.println("a: " + a);
11    }
12
13    void test(int a, int b)
14    {
15      System.out.println("a and b: " + a + " " + b);
16    }
17
18    double test(double a)
19    {
20      System.out.println("double a: " + a);
21      return a*a;
22    }
23  }
```

Method overloading – Eg…

```
25  class Overload
26  {
27    public static void main(String args[])
28    {
29      OverloadDemo ob = new OverloadDemo();
30      double result;
31
32      // call all versions of test()
33      ob.test();
34      ob.test( 10 );
35      ob.test( 10 , 20 );
36      result = ob.test( 2.5 );
37      System.out.println("Result of ob.test(2.5): " + result);
38    }
39  }
```

OUTPUT:

```
No parameters
a: 10
a and b: 10 20
double a: 2.5
Result of ob.test(2.5): 6.25
```

Overloading of Methods

Overloading Constructors

- Example:
  - Constructors

```
Box()
        { height = 0; width = 0; depth = 0; }
Box(double h, double w, double d)          // for a cuboid
        { height = h; width = w; depth = d; }
Box(double len)                            // for a cube
        { height = width = depth = len; }
```

  - Invoke as

```
Box myBox1 = new Box();
Box myBox2 = new Box(10, 8, 6);
Box myBox3 = new Box(5);
```

Passing Objects

- **Passing objects as arguments to methods**
  - Primitive type arguments – passed by value
  - Objects – passed by reference
    - When a variable of class type is created, we are only creating a reference to the object

Call_by_value eg:

```
1   class Test
2   {
3     void meth(int i, int j)
4     {
5       i *= 2;
6       j /= 2;
7     }
8   }
9
10  class CallByValue
11  {
12    public static void main(String args[])
13    {
14      Test ob = new Test();
15      int a = 15, b = 20;
16
17      System.out.println("a and b before call: ");
18      System.out.println( a + " " + b );
19
20      ob.meth(a, b);
21
22      System.out.println("a and b after call: ");
23      System.out.println( a + " " + b);
24    }
25  }
```

OUTPUT:

```
a and b before call:
15 20
a and b after call:
15 20
```

Passing array to a method:

```java
class Test
{
  void arr_test( int a[] )
  {
      a[0]++;
      a[1]--;
      a[2] *= 2;
  }
}
```

```java
class arr_pass
{
  public static void main(String args[])
  {
    Test ob = new Test();
    int arr[] = { 11 ,22 , 33 };

    System.out.println("array contents
    for( int ele : arr )
        System.out.println( ele );

    ob.arr_test( arr );

    System.out.println("array contents before call: ");
    for( int ele : arr )
        System.out.println( ele );
  }
}
```

```
array contents before call:
11
22
33
array contents before call:
12
21
66
```

Passing object to a method:

```
1   class Test
2   {
3     private int a, b;
4
5     Test(int i, int j)
6     {
7       a = i;
8       b = j;
9     }
10
11    void meth(Test o)
12    {
13      o.a *=  2;
14      o.b /= 2;
15    }
16    void show()
17    {
18        System.out.println( a+ " " +b );
19    }
20  }
```

```java
class PassObjRef
{
  public static void main(String args[])
  {
    Test ob1 = new Test( 10,  20 );
    Test ob2 = new Test( 11 , 22 );

    System.out.println("ob1.a and ob1.b before call: ");
    ob1.show();

    ob1.meth( ob2 );

    System.out.println("ob2.a and ob2.b after call: ");
    ob2.show();
  }
}
```

OUTPUT:

```
ob1.a and ob1.b before call:
10 20
ob2.a and ob2.b after call:
22 11
```

Returning Objects

- **Returning objects from methods**
  - Method can return any type of data (including objects)

Returning object – Eg:

```
1   class Test
2   {
3     private int a;
4
5     Test(int i)
6     {
7       a = i;
8     }
9
10    Test incrByTen()
11    {
12        Test temp = new Test(a+10);
13        return temp;
14    }
15    int show()
16    {
17        return a;
18    }
19  }
```

```java
class RetOb
{
  public static void main(String args[])
  {
    Test ob1 = new Test(2);
    Test ob2;

    ob2 = ob1.incrByTen();
    System.out.println("ob1.a: " + ob1.show() );
    System.out.println("ob2.a: " + ob2.show() );

    ob2 = ob2.incrByTen();
    System.out.println("ob2.a after second increase: "
                                + ob2.show() );
  }
}
```

OUTPUT:

```
ob1.a: 2
ob2.a: 12
ob2.a after second increase: 22
```

Example: Array of objects

```
1   class Employee
2   {
3       private String name;
4       private double salary;
5
6       Employee( String n, double s )
7       {
8           name = n;
9           salary = s;
10      }
11
12      String getName()
13      {
14          return name;
15      }
16
17      double getSalary()
18      {
19          return salary;
20      }
22      void raiseSalary( double byPercent )
23      {
24          double raise = salary * byPercent / 100;
25          salary += raise;
26      }
27  }
```

```
29 public class EmployeeTest
30 {
31    public static void main(String[] args)
32    {
33       // fill the staff array with three Employee objects
34       Employee[] staff = new Employee[3];
35
36       staff[0] = new Employee("Anil", 50000 );
37       staff[1] = new Employee("John", 60000 );
38       staff[2] = new Employee("Vinod", 40000 );
39
40       // raise everyone's salary by 5%
41       for ( Employee e : staff )
42          e.raiseSalary( 5 );
43
44       // print out information about all Employee objects
45       for ( Employee e : staff )
46          System.out.println("name=" + e.getName() + ",salary=" + e.getSalary());
47    }
48 }
```

Staff[0]	name	Anil
	salary	50000
Staff[1]	name	John
	salary	60000
Staff[2]	name	Vinod
	salary	40000

OUTPUT:

```
name=Anil,salary=52500.0
name=John,salary=63000.0
name=Vinod,salary=42000.0
```

Recursion

- **Recursion and recursive functions**
  - Recursion: Process of calling a function from itself
  - Recursive function: A function that calls itself

```
long factorial ( int  n )
 {
    if (n ==0)
      return (1);
    return (n * factorial (n-1));
 }
```

120

```
long fact = factorial( 5 );
```

24

5* factorial(4)

6

4* factorial(3)

2

3*factorial(2)

1

2* factorial(1)

1

1*factorial(0)

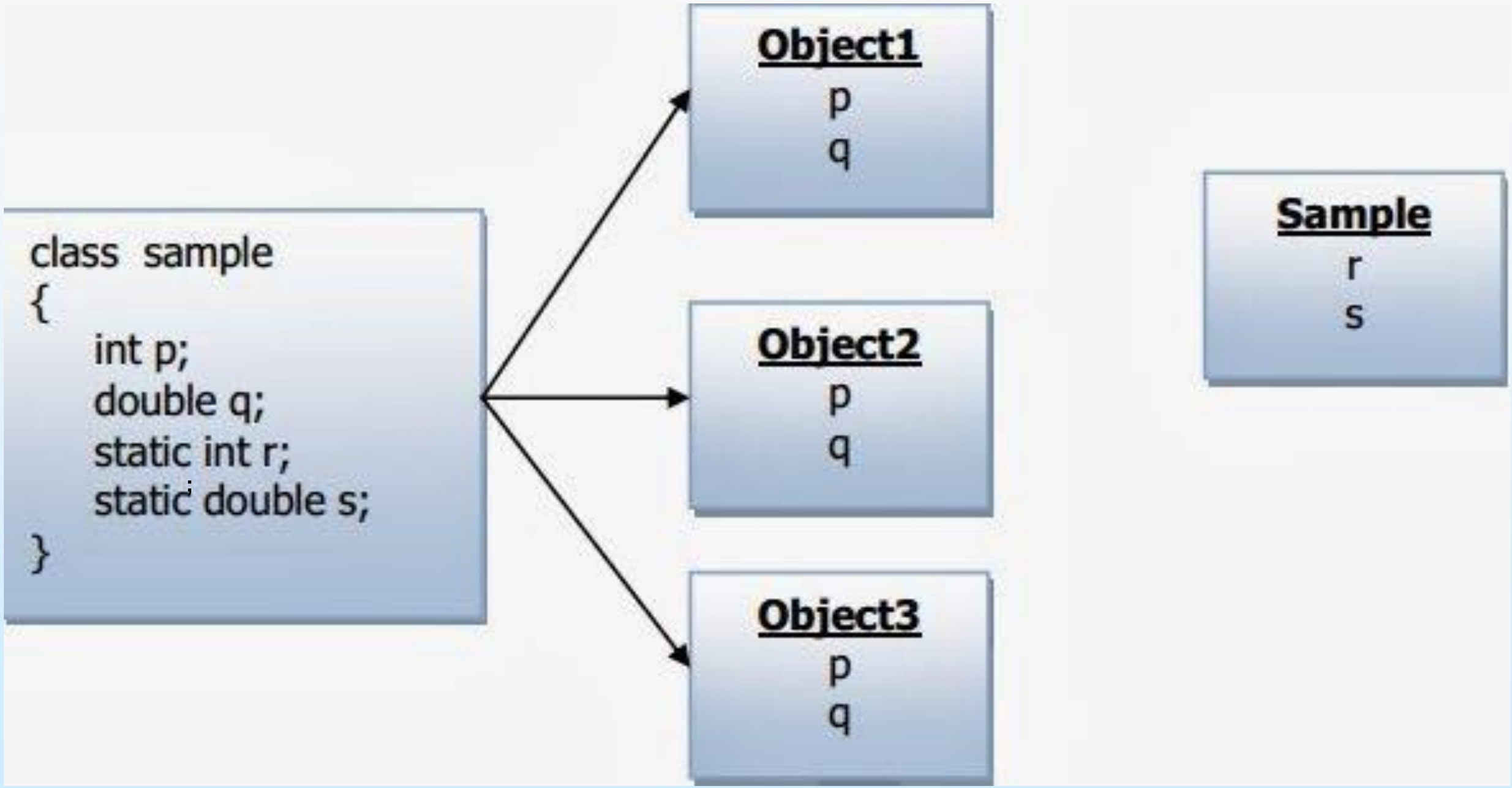
1

Recursion Example-2:

```
1   class RecTest
2   {
3
4       int ComputeSum(int Arr[], int n )
5       {
6           if( n <= 0 )
7               return 0;
8           return ( ComputeSum( Arr, n - 1) + Arr[n - 1] );
9       }
10  }
11
12  class Recursion2
13  {
14    public static void main(String args[])
15    {
16      RecTest R = new RecTest();
17      int Arr[] = { 10 , 20 , 30 , 40 };
18      int sum = R.ComputeSum( Arr , 4 );
19      System.out.println("Sum = "+sum);
20    }
21  }
```

Access Control

- **Static members of a class**
  - Can be accessed without reference to any object
    - E.g., main() method (called before any objects exist)
  - Instance variables can be static
    - Act like global variables
    - When objects are created, no copy of static variables is made
    - All objects share the same static variable
  - Methods can be static
    - They can only call other static methods
    - They can access only static data
    - They cannot refer to *this* or *super* in any way
  - If computation is required to initialize static variable
    - Declare a static block (gets executed only once, when the class is first loaded)


class sample
{
int p;
double q;
static int r;
static double s;
}
Object1
p
q
Object2
p
q
Object3
p
q
Sample
r
s

Example: static

```
1  class StaticDemo
2  {
3    static int a = 42;
4    static int b = 99;
5    static void callme()
6    {
7      System.out.println("a = " + a);
8    }
9  }
10
11 class StaticByName
12 {
13   public static void main(String args[])
14   {
15     StaticDemo.callme();
16     System.out.println("b = " + StaticDemo.b);
17   }
18 }
```

```
a = 42
b = 99
```

```java
class UseStatic
{
   static int a = 3;
   static int b;

   static void meth(int x)
   {
      System.out.println("x = " + x);
      System.out.println("a = " + a);
      System.out.println("b = " + b);
   }

   static
   {
      System.out.println("Static block initialized.");
      b = a * 4;
   }

   public static void main(String args[])
   {
      meth(42);
   }
}
```

```
Static block initialized.
x = 42
a = 3
b = 12
```

Question-1

```
1   class Static_Test
2   {
3     int denom = 3;
4     static int val = 1024;
5
6     static int valDivDenom()
7     {
8       return val/denom;
9     }
10  }
11  class Static_Demo
12  {
13    public static void main(String[] args)
14    {
15      System.out.println("val is " +Static_Test.valDivDenom() );
16    }
17  }
```

Question-1

```
1   class Static_Test
2   {
3     int denom = 3;
4     static int val = 1024;
5
6     static int valDivDenom()
7     {
8       return val/denom; // Error !
9     }
10  }
11  class Static_Demo
12  {
13    public static void main(String[] args)
14    {
15      System.out.println("val is " +Static_Test.valDivDenom() );
16    }
17  }
```

Question-2

```java
class StaticBlock
{
  static double a;
  static double b;

  static
  {
    System.out.println("Inside static block.");
    a = Math.sqrt(25.0);
    b = Math.sqrt(49.0);
  }

  StaticBlock(String msg)
  {
    System.out.println(msg);
  }
}

class SDemo
{
  public static void main(String[] args)
  {
    StaticBlock ob = new StaticBlock("Inside Constructor");

    System.out.println( StaticBlock.a );
    System.out.println( StaticBlock.b );
  }
}
```

```
Inside static block.
Inside Constructor
5.0
7.0
```

Question-3:

```java
class MyClass
{
    int count = 0;

  MyClass()
  {
    count++;
  }
}

class Test
{
  public static void main(String[] args)
  {
    for(int i=0; i < 3; i++)
    {
      MyClass obj = new MyClass();
      System.out.println("Number of objects created: " + obj.count);
    }
  }
}
```

```
Number of objects created: 1
Number of objects created: 1
Number of objects created: 1
```

Question-4: To count the total number of objects created

```
1   class MyClass
2   {
3     static int count = 0;
4
5     MyClass()
6     {
7       count++;
8     }
9   }
10
11  class UseStatic
12  {
13    public static void main(String[] args)
14    {
15      for(int i=0; i < 3; i++)
16      {
17        MyClass obj = new MyClass();
18        System.out.println("Number of objects created: " + MyClass.count);
19      }
20    }
21  }
```

Access Control

- ***final* variable**
  - Prevents the contents of the variable being modified
    - Should be initialized when it is declared
    - Assign it a value within a constructor
    - Variable is essentially a constant

    - Coding convention used: All uppercase identifiers
    - Examples:

      ```
      final double INTEREST_RATE = 10
      final int FILE_NEW = 1
      ```
  - Can also be applied to methods (cannot be inherited)

Example: Final

```
1   class Final
2   {
3       final int a;
4       public static void main(String args[])
5       {
6           Final ob = new Final();
7           System.out.println(ob.a);
8       //  ob.a++; Error
9       }
10      Final()
11      {
12          a = 10;
13          //a++;  Error
14      }
15  }
```

Example -final

```
1   class Final
2   {
3       final int a=11;
4       public static void main(String args[])
5       {
6           Final ob = new Final();
7           System.out.println(ob.a);
8       //  ob.a++; Error
9       }
10      Final()
11      {
12          //a = 10; Error
13          //a++;  Error
14      }
15  }
```

Array as an Object

- ***length* instance variable**
  - An array is implemented as an object
    - Instance variable *length*: Size of the array
      - Number of elements that an array can hold
      - Not number of elements actually used

Example:

```
class Length{
    public static void main(String args[]) {
        int a1[] = new int [10];
        int a2[] = { 1, 2, 3, 4, 5 };
        System.out.println (a1.length);     // 10
        System.out.println (a2.length);     // 5
    }
}
```

Example: length field

```
4   class Length
5   {
6     public static void main(String args[])
7     {
8       int a1[] = new int[10];
9       int a2[] = {3, 5, 7, 1, 8, 99, 44, -10};
10      int a3[] = {4, 3, 2, 1};
11
12      System.out.println("length of a1 is " + a1.length);
13      System.out.println("length of a2 is " + a2.length);
14      System.out.println("length of a3 is " + a3.length);
15    }
16  }
```

```
length of a1 is 10
length of a2 is 8
length of a3 is 4
```

The End