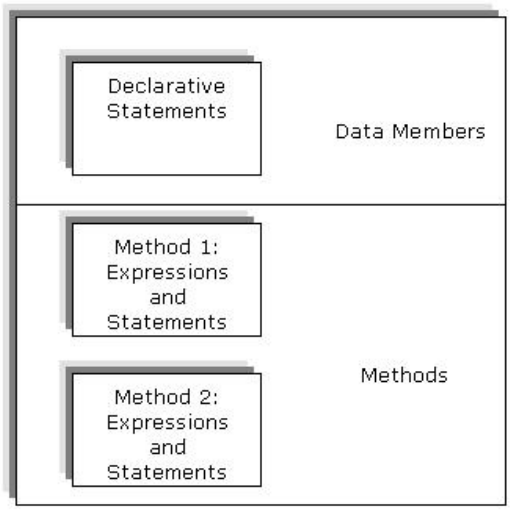
Objectives

- In this lesson, you will learn about:
  - Structure of a Java program
  - Access specifiers and modifiers
  - Creating a Java application
  - Use conditional statements
  - Use looping constructs
  - Enhance methods of a class
  - Pass arguments to methods
  - Create nested classes
  - Casting and conversion in Java
  - Overloading constructors

Structure of Java Application

- Creating Classes and Objects
  - The components of a class consists of
    - Data members (Attributes)
    - Methods



Structure of Java Application (Contd..)

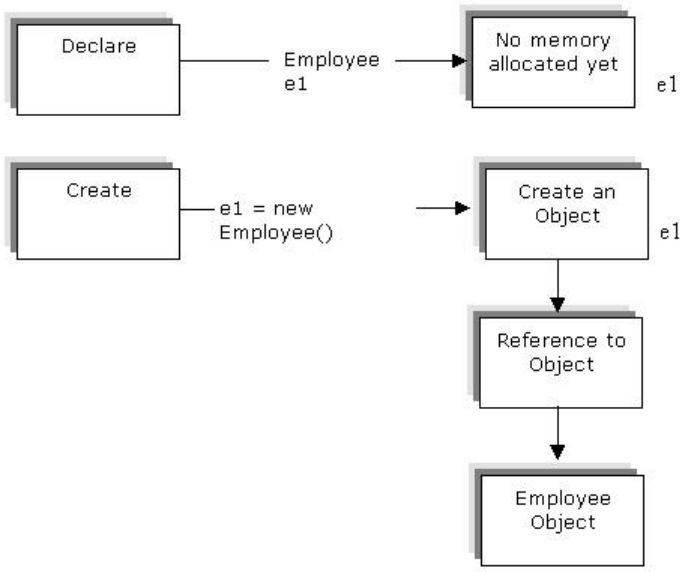
- Creating Classes in Java
  - The statements written in a Java class must end with a semicolon, **;**.

```
class ClassName
{
//Declaration of data members
//Declaration of methods
}
```

- Double slash, **//**, are comment entries. Comments are ignored by compiler.

Structure of Java Application (Contd.)

- Creating Objects of Classes
  - An object is an instance of a class having a unique identity.
  - To create an object, you need to perform the following steps:
    - Declaration
    - Instantiation or creation



Structure of Java Application (Contd.)

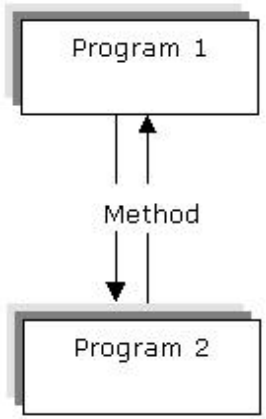
- Accessing Data Members of a Class
  - Assign values to data members of the object before using them.
  - You can access the data members of a class outside the class by specifying the object name followed by the dot operator and the data member name.

```
e1.employeeName="John";
e2.emmployeeName="Andy";

e1.employeeID=1;
e2.employeeID=2;
e1.employeeDesignation = “Manager”;
e2.employeeDesignation = “Director”;
```

Structure of Java Application (Contd.)

- Adding Methods to a Class
  - Accessing data members directly overrules the concept of encapsulation.
  - Advantages of using methods in a Java program:
    - Reusability
    - Reducing Complexity
    - Data Hiding



Structure of Java Application (Contd.)

- The syntax to define a method:

```
void methodName()
{
     // Method body.
}
```

Structure of Java Application (Contd.)

- Declaring the main() Method:
  - The syntax to declare the main()method:

```
public static void main(String args[])
{
// Code for main() method
}
```

  - public: method can be accessed from any object in a Java program.
  - static : associates the method with its class.
  - void: signifies that the `main()` method returns no value.
- The `main()` method can be declared in any class but the name of the file and the class name in which the `main()` method is declared should be the same.
- The `main()` method accepts a single argument in the form of an array of elements of type String.

Structure of Java Application (Contd.)

- Following code snippet creates objects for four employees of an organization:

```
Employee e1 =new Employee();
Employee e2 =new Employee();
Employee e3 =new Employee();
Employee e4 =new Employee();
```

Structure of Java Application (Contd.)

- Defining Constructors
  - A constructor is a method with the same name as the class name.
  - A constructor of a class is automatically invoked every time an instance of a class is created.
- Characteristics of Constructors
  - There is no return type for a constructor.
  - A constructor returns the instance of the class instead of a value.
  - A constructor is used to assign values to the data members of each object created from a class.

Access Specifiers and Modifiers

- Access Specifiers
  - public
  - private
  - protected
  - friendly
- The public access specifier
  - Class members with public specifier can be accessed anywhere in the same class, package in which the class is created, or a package other than the one in which the class is declared.
  - The `public` keyword is used to declare a member as public.

```
public <data type> <variable name>;
```

Access Specifiers and Modifiers

- The private access specifier
  - A data member of a class declared private is accessible at the class level only in which it is defined.
  - The private keyword is used to declare a member as private.

    private float <variableName>; // Private data member of float type

    Private void methodName(); // Private method
- The protected access specifier
  - The variables and methods are accessible only to the subclasses of the class in which they are declared.
  - The protected keyword is used to declare a member as protected.

    protected <data type> <name of the variable>;
- The friendly or package access specifier
  - If you do not specify any access specifier, the scope of data members and methods is friendly.

Access Specifiers and Modifiers(Contd.)

- Types of Permitted Modifiers
  - Modifiers determine or define how the data members and methods are used in other classes and objects.
    - static
    - final
    - abstract
    - native
    - synchronized

Access Specifiers and Modifiers(Contd.)

- static
  - Used to define class variables and methods that belong to a class and not to any particular instance of the class.
  - Associates the data members with a class and not the objects of the class.
- final
  - Indicates that the data member cannot be modified.
  - Does not allow the class to be inherited.
  - A final method cannot be modified in the subclass.
  - All the methods and data members in a final class are implicitly final.
- abstract
  - Used to declare classes that define common properties and behavior of other classes.
  - An `abstract` class is used as a base class to derive specific classes of the same type.

Access Specifiers and Modifiers(Contd.)

- native
  - Used only with methods.
  - Inform the compiler that the method has been coded in a programming language other than Java, such as C or C++ .
  - The `native` keyword with a method indicates that the method lies outside the Java Runtime Environment (JRE).

```
public native void nativeMethod(var1, var2, . . .) ;
```

- synchronized
  - controls the access to a block of code in a multithreaded programming environment.
  - Java supports multithreaded programming and each thread defines a separate path of execution.

Compiling an Application

- Compiling an Application
  - Save the code with a file name that is exactly the same as that of the class name, with a .java extension.
  - The steps to compile the Java file:
    - Open the command window.
    - Set the PATH variable to the bin directory of the installation directory by using the following command at the command prompt:
    - PATH=C:\j2sdk1.4.1_02\bin

Compiling an Application

- Change to the directory where you have saved the .java file. You can also give the complete path of the .java file while compiling it .
- Compile the application
- Execute the Bytecode.

Pre-Assessment Questions

1. Which access specifier is used with the data member of a class that is accessible at the class level only in which it is defined?
   a. public
   b. private
   c. protected
   d. default

2. The _____ keyword is used to define class variables and methods that belong to a class and not to any particular instance of the class.
   a. static
   b. native
   c. synchronized
   d. abstract

Pre-Assessment Questions (Contd.)

3. Consider the statements:
   Statement A: A final method cannot be overridden.
   Statement B: A final class cannot be inherited.
   Identify the correct option.
   a. Statement A is true and statement B is false.
   b. Statement A is false and statement B is true.
   c. Both, statements, A and B, are true.
   d. Both, statements, A and B, are false.

Pre-Assessment Questions (Contd.)

4. When do you get an error message as :
   Exception in thread "main" Java.lang.NoSuchMethodError:main

   a. When the class has been incorrectly declared in a program.
   b. When you try to execute a program that does not have a main() method.
   c. When you have given incorrect arguments to the main() method.
   d. When you save the file with a different name as the name of the class in which the code is written

Pre-Assessment Questions (Contd.)

5. Consider the statements:
   Statement A: An abstract class can be instantiated.
   Statement B: An abstract class cannot be inherited.
   Identify the correct option.

   a. Statement A true and statement B false.
   b. Statement A false and statement B true.
   c. Both, statements, A and B, true.
   d. Both, statements, A and B, false.

Solutions to Pre-Assessment Questions

1. b. private
2. a. static
3. c. Both, statements, A and B are true
4. b. When you try to execute a program that does not have a main() method
5. d. Both, statements, A and B are false

Using Conditional Statements

- Conditional statements allow selective execution of a set of statements depending on the value of expressions associated with them.
- Conditional statements are also known as decision-making statements.
- You can control the flow of a program using conditional statements.
- Two types of conditional statements in Java are:
  - The if-else statement
  - The switch-case construct

Using Conditional Statements (Contd.)

- Using the if-else Statement
  - The if-else statement:
    - Enables you to execute selectively.
    - Is followed by a boolean expression.
    - Executes the set of statements depending upon the result of the boolean expression.

Using Conditional Statements (Contd.)

- Using the if-else Statement (Contd.)
  - Syntax of the `if-else` statement is:

```
if(boolean expression)
    {
        statement(s)
    }
    else
    {
                statement(s)
    }
```

In the preceding syntax, if the boolean expression of the if construct evaluates to true, the Java compiler executes the statements following the `if` construct else executes the statements following the `else` construct.

Using Conditional Statements (Contd.)

- Relational Operators:
  - Used to compare the values of two variables or operands and find the relationship between the two.
  - The relational operators are therefore called comparison operators also.

Using the Arithmetic Assignment Operators

- Arithmetic Assignment Operators
  - Addition(+), subtraction(-), multiplication(*), division(/), and modulo(%) are the arithmetic operators supported by Java.
  - Various arithmetic operators, such as +, -, /, *, and % are combined with the assignment operator (=) and are called arithmetic assignment operators.

Using the Arithmetic Assignment Operators (Contd.)

Operator	Use	Description
+=	op1 += op2	Adds operand, op1 and operand, op2 and assigns the result to op1. This expression is equivalent to op1 = op1+op2.
- =	op1 -= op2	Subtracts operand, op2 from operand, op1 and assigns the result to op1. This expression is equivalent to op1 = op1 – op2.

Using the Arithmetic Assignment Operators (Contd.)

Operator	Use	Description
*=	op1 *= op2	Multiplies operand, op1 and operand, op2 and assigns value of the result to op1. This expression is equivalent to op1 = op1*op2.
/=	op1 /= op2	Divides operand, op1 by operand, op2, and assign the value of the result to op1. This expression is equivalent to op1 = op1/op2.

Using the Arithmetic Assignment Operators (Contd.)

Operator	*Use*	*Description*
%=	op1 %= op2	Assigns the remainder of division of operand, op1 and operand, op2 to op1. This expression is equivalent to op1 = op1% op2.

Using Bit-wise Operators

- Bit-wise operators
  - Operate on the individual bits of their operand.
  - Operands can be various data types like int, short, long, char, and byte.
  - Operands are converted into their binary equivalents before operation.
  - The result in the binary form after the operation is converted back into its decimal equivalent.

Using Bit-wise Operators(Contd.)

- The following table lists the various bit-wise operators in Java:

Operator	Use	Operation
&(AND)	x & y	Performs bit-wise AND operation. It evaluates to 1 if both bits, x and y are 1. If either or both bits are 0, the result is 0.
|(OR)	x | y	Performs bit-wise OR operation. It evaluates to 0 if both bits, x and y are 0. If either or both bits are 1, the result is 1.

Using Bit-wise Operators(Contd.)

Operator	Use	Operation
~(inversion)	~x	Performs unary NOT operation. Converts all the 1s into 0s and all the 0s into 1s.
^(XOR)	x ^ y	Performs bit-wise XOR operation. It evaluates to 1 if bits have different values and 0 if both the bits have the same value.

Using Bit-wise Operators (Contd.)

- Using the Bit-wise AND Operator
  - The Bit-wise AND operator (&) performs AND operation on two operands.
  - Displays 1 if both bits are 1 else 0 in all other cases.

Operation	Result
0 & 0	0
0 & 1	0
1 & 1	1
1 & 0	0

Using Bit-wise Operators (Contd.)

- Using the Bit-wise OR Operator
  - The Bit-wise OR operator (|) performs OR operation on two operands.
  - Displays 0 if both bits are 0 else 1 in all other cases.

Operation	Result
0 | 0	0
0 | 1	1
1 | 1	1
1 | 0	1

Using Bit-wise Operators (Contd.)

- Using the Bit-wise NOT Operator
  - Bit-wise NOT operator (~) is a unary operator and performs NOT operation on each bit of binary number.
  - The NOT operator inverts or complements each of the bits of a binary number.

Operation	Result
~ 0	1
~ 1	0

Using Bit-wise Operators (Contd.)

- Using the Bit-wise XOR Operator
  - The Bit-wise XOR (^) operator performs XOR operation on two operands.
  - The XOR operator applied on two bits results in 1 if exactly one bit is 1 else 0 in all other cases .

Operation	Result
0 ^ 0	0
0 ^1	1
1 ^ 1	0
1 ^ 0	1

Using Shift Operators

- Works on the bits of data.
- Shifts the bits of it's operand either to left or right.

Operator	Use	Operation
>> (Right Shift)	val1 >> val2	Shifts the bits of the val1 operand to the right by the number of positions specified by val2.
<< (Left Shift)	val1<< val2	Shifts the bits of the val1 operand to the left by the number of positions specified by val2.

Using Shift Operators (Contd.)

Operator	Use	Operation
>>> (Unsigned Shift Operator)	val1 >>> val2	Shifts the bits of the val1 operand to the right by the number of positions specified by val2. A zero value is input in the high-order bit irrespective of the value of the high-order bit of val1. The high-order bit is the leftmost bit of the binary number.

Using Shift Operators (Contd.)

- Using the Right Shift and Left Shift Operators
  - The right shift and the left shift operators are binary operators.
    - The right shift operator shifts all the bits of a binary number in the right direction.

      ```
      operand >> num
      ```

    - The left shift operator, <<, shifts all the bits of a binary number in the left direction.

      ```
      operand << num
      ```

- Using the Unsigned Shift Operator
  - Unsigned shift operator (>>>) is used to shift the bits of a binary number to the right.
  - The operator fills the leftmost bits of a binary value with 0 irrespective of whether the number has 0 or 1 at the leftmost bit.

Using instance of Operator

- Used to test whether an object is an instance of a specific class.
- Used at the run time.
- The syntax of the instanceof operator is:

  ```
  op1 instanceof op2
  ```

- op1 is the name of an object and op2 is the name of a class.
- Returns a true value if the op1 object is an instance of the op2 class.

Using Conditional Statements (Contd.)

- The following table lists the various relational operators (Contd.):

Operator	*Use*	*Operation*
>	op1 > op2	Returns true if the value of the op1 operand is greater than the value of the op2 operand.
<	op1<op2	Returns true if the value of the op1 operand is less than the value of the op2 operand.
>=	op1>=op2	Returns true if the value of the op1 operand is greater than or equal to value of op2 operand.

Using Conditional Statements (Contd.)

Operator	Use	Operation
<=	op1<=op2	Returns true if value of op1 operand is less than or equal to value of op2 operand.
==	op1 ==op2	Returns true if value of op1 operand is equal to value of op2 operand.
!=	op1 != op2	Returns true if value of op1 operand is not equal to value of op2 operand.

Using Conditional Statements (Contd.)

- Conditional Operators :
  - Used to combine multiple conditions in one boolean expression.
  - Are of two types :
    - Unary
    - Binary

  - Various conditional operators are:
    - AND(&&)
    - OR(||)
    - NOT(!)

Using Conditional Statements (Contd.)

- The following table lists the various conditional operators and their operations:

Operator	***Use***	***Operation***
AND (&&)	op1 && op2	Returns true if both op1 operand and op2 operand are true.
OR (||)	op1 || op2	Returns true if either op1 operand or op2operand is true.
NOT (!)	! op1	Returns true if op1 operand is not true.

Using Conditional Statements (Contd.)

- Using the multiple if-else Statement:
  - You can replace a single `if` construct with multiple if-else statements to write a compound `if` statement.
  - The multiple if-else construct allows you to check multiple boolean expressions in a single compound `if` statement.

Using Conditional Statements (Contd.)

- Using the multiple if-else Statement (Contd.):
  - The syntax of the multiple if-else statements is:

```
if (Boolean_expression_1)
{
  statements
}
else if (Boolean_expression_2)
{
  statements
}
else (Boolean_expression_3)
{
  statements}
```

Using Conditional Statements (Contd.)

- The switch-case Construct:
  - Successively tests the value of an expression or a variable against a list of case labels with integer or character constants.
  - When a match is found in one of the case labels, the statements associated with that case label get executed.
  - The `switch` keyword in the switch-case construct contains the variable or the expression whose value is evaluated to select a case label.
  - The `case` keyword is followed by a constant and a colon.
  - The data type of case constant should match the switch variable.

Using Conditional Statements (Contd.)

- The switch-case Construct (Contd.)
  - The syntax of the switch-case construct is:

```
switch(expression or variable name)
    {
    case Expr1:
            statements;
            break;
    case Expr2:
            statements;
            break;
    default:
            statements; }
```

Using Conditional Statements (Contd.)

- The switch-case construct (Contd.)
  - Statements associated with the default keyword are executed if the value of the switch variable does not match any of the case constants.
  - The break statement used in the case label causes the program flow to exit from the body of the switch-case construct.

Using Conditional Statements (Contd.)

- The break Statement:
  - Causes the program flow to exit from the body of the switch construct.
  - Control goes to the first statement following the end of the switch-case construct.
  - If not used inside a case construct, the control passes to the next case statement and the remaining statements in the switch-case construct are executed.

Using Looping Statements

- A looping statement causes a section of program to be executed a certain number of times.
- The repetition continues while the condition set in the looping statement remains true.
- When the condition becomes false, the loop ends and the control is passed to the statement following the loop.

Using Looping Statements (Contd.)

- The for Loop:
  - Is a looping statement iterating for a fixed number of times.
  - Consists of the `for` keyword followed by parentheses containing three expressions, each separated by a semicolon.
  - The three expressions in the `for` loop are:
    - Initialization expression
    - Test expression
    - IterationExpr expression
  - Is used when the number of iterations is known in advance.
  - For example, it can be used to determine the square of each of the first ten natural numbers.

Using Looping Statements(Contd.)

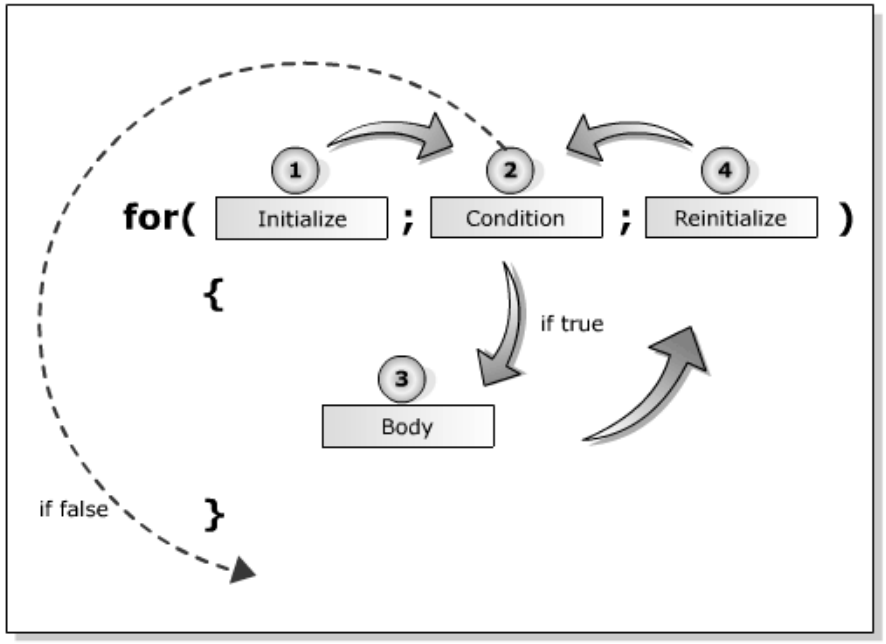
- The for Loop (Contd.)
  - The syntax of the `for` loop is:

```
for(InitializationExpr; TestExpr; IterationExpr)
    {
            statement_1
            statement_2
            …
    }
```

  - In the preceding syntax, the initialization expression is executed only once, when the control is passed to the for loop for the first time. The initialization expression gives the loop variable an initial value.
  - The test expression is executed each time the control passes to the beginning of the loop. If true, the body of the loop is executed, otherwise not.
  - The IterationExpr expression is always executed when the control returns to the beginning of the loop in each loop iteration.

Using Looping Statements(Contd.)

- The for Loop (Contd.)
  - The following figure shows the use of `for` loop:



Using Looping Statements(Contd.)

- The `while` Loop:
  - Executes a statement or a block of statements as long as the evaluating condition remains true.
  - The evaluating condition has to be a boolean expression and must return a boolean value that can be true or false.
  - The syntax of the `while` loop is:

```
while(Bool_Expr)
    {
        statements; //executed as long as Bool_Expr is true
    }
```

In the preceding syntax, the statements in the while loop are executed as long as the Bool_Expr condition is true. When the condition returns false, the statement immediately following the while block is executed.

Using Looping Statements(Contd.)

- The `do-while` Loop:
  - Used when the body of the loop needs to be executed at least once.
  - The `do-while` construct places the test condition at the end of the loop.
  - The `do` keyword marks the beginning of the loop and braces form the body of the loop.
  - The while statement provides the test condition.
  - The test condition checks whether the loop will execute again or not.
  - The syntax of the `do-while` loop is:

```
do
{
        statements;
}
while(Bool_Expr);
```

Using Looping Statements(Contd.)

- The `continue` statement:
  - In the `while` and `do-while` loops:
    - The `continue` statement returns the control to the conditional expression that controls the loop.
    - The control skips any statement following the continue statement in the loop body.
  - In the `for` loop:
    - control goes to the re-initialization expression first and then to the conditional expression.

Enhancing Methods of a Class

- Methods are used to access the variables that are defined in a class.
- A method is an interface to a class.
- Parameterized methods need some extra information for its execution.
- The extra information is passed to the method by using arguments.
- Arguments are also known as parameters of the methods.

Enhancing Methods of a Class (Contd.)

- Defining Parameterized Methods:
  - Parameterized methods use parameters to process data and generate an output.
  - The output of a parameterized method is not static and depends on the value of the parameters passed.
  - Parameters in a parameterized method allow the method to be generalized.

Enhancing Methods of a Class (Contd.)

- Defining a Method that Returns a Value:
  - You can define a method that can return a value instead of just computing results and displaying them.
  - The return type of a method is specified to the Java compiler in the method declaration statement.
  - The return type of a method can be of any primitive data type or abstract data type.
  - The value is returned from a method using the return keyword followed by the value to be returned.
  - The methods that do not return any value have return type void.

Enhancing Methods of a Class (Contd.)

- Overloading Methods
  - Method overloading is defined as the function that enables you to define two or more methods with the same name but with different signatures within the class.
  - The methods that share the same name, but have different signatures are called overloaded methods.
  - The signature of a method consists of:
    - The name of the method.
    - The number of arguments it takes.
    - The data type of the arguments.
    - The order of the arguments.

Enhancing Methods of a Class (Contd.)

- When an overloaded method is invoked, the Java compiler uses the type of arguments or the number of arguments to determine which copy of overloaded method to invoke.
- Overloading methods must differ in the type or the number of parameters.
- The return types of overloaded methods can be different.
- Overloaded methods are used when you need several methods that perform closely related tasks.
- Method overloading is a way to implement polymorphism in Java.

Enhancing Methods of a Class (Contd.)

- Overloading Methods (Contd.)
  - The following code snippet shows an overloaded multiply() method:

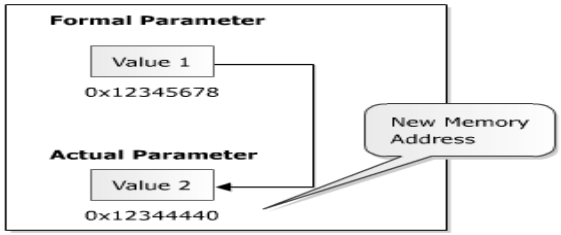
```
public void multiply(int a, int b)    //Multiply two integers
public void multiply(float a, float b)    //Multiply two floats
public void multiply(double a, double b)    //Multiply two doubles
```

Passing Arguments to a Method

- Arguments can be passed to a method using two ways:
  - Call-by-value: Copies the value of the actual parameters to the formal parameters. Therefore, the changes made to the formal parameters have no effect on the actual parameters.
  - Call-by-reference: Passes the reference and not the value of the actual parameters to the formal parameters in a method. Therefore, the changes made to the formal parameters affect the actual parameters.
- You can also pass arguments to the main() method at the run-time of a program.

Passing Arguments to a Method (Contd.)

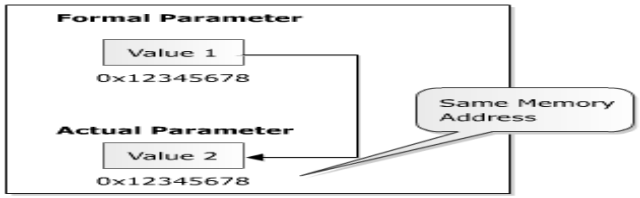
- Passing Arguments by Value:
  - In Java, when arguments of primitive data type, such as int and float are passed to a method then they are passed by value. The following figure shows the concept of passing arguments by value:



  - When arguments are passed by value, a copy of the actual argument is passed to the formal arguments of the called method, which is maintained at a separate memory location. Therefore, when the called method changes the value of the argument, the change is not reflected in the actual argument.

Passing Arguments to a Method (Contd.)

- Passing Arguments by Reference:
  - Passes a reference to an argument to the parameter of a method.
  - In Java, the objects of abstract data type are passed by reference.
  - When arguments are passed to the method by reference then any change made to the formal argument by the called method is also reflected in the actual argument. The following figure shows the concept of passing arguments by reference:



  - The argument passed by reference to the formal parameter has the same memory location as that of the actual parameter.

Passing Arguments to a Method (Contd.)

- Passing Arguments at the Command Line:
  - Command-line arguments are used to provide information that a program needs at startup.
  - The main() method in Java is the first method to be executed, therefore the command-line arguments are passed to the main() method.
  - The main() method stores the command-line arguments in the String array object.
  - In Java, the syntax for command-line argument is:

```
public static void main(String args[])
{    //statements
} //End of main() method
```

    In the preceding syntax, each of the elements in the array named args[] is a reference to the command-line arguments each of which is a String object.

Creating Nested Classes

- Nested Classes:
  - A nested class is a class defined as a member of another class.
  - The scope of nested class is bounded by the scope of its enclosing class.
  - The nested class has access to the members of its enclosing class including private members.
  - The enclosing class does not have any access to the members of the nested class.
  - The nested classes are of two types:
    - Static: A nested class declared as static is called the static nested class.
    - Inner: A non-static nested class is called the inner class.

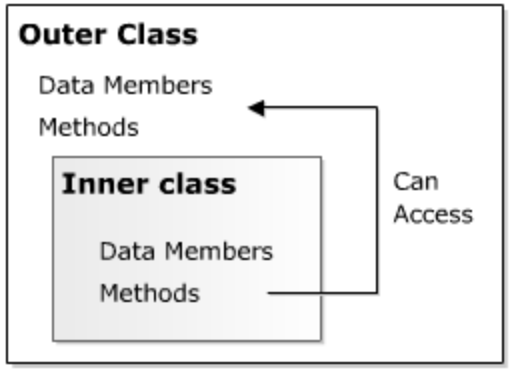
Creating Nested Classes (Contd.)

- Nested Classes (Contd.):
  - Static Nested Class
    - A static nested class cannot access the members of its enclosing class directly because it is declared as static.
    - Therefore, it has to access the members of its enclosing class through an object of enclosing class.
    - The following code snippet shows the declaration of a static nested class:

```
class EnclosingClass{
     //statements
     static AstaticNestedClass
     {
               //statements
     }
}
```

Creating Nested Classes (Contd.)

- Nested Classes (Contd.)
  - Inner Class:
    - An inner class is a non-static nested class, whose instance exists inside the instance of its enclosing class.
    - An inner class can directly access the variables and methods of its enclosing class.
    - The following figure shows a nested class:



Casting and Conversion in Java

- Java supports implicit conversion of one data type to another type. are inbuilt in Java.
- Implicit conversions are the conversions from one data type to another, which occur automatically in a program.
- For example, you can assign a value of int type to a variable of long data type.
- When you assign a value of a particular data type to another variable of a different data type, the two types must be compatible with each other.
- The two data types are compatible to each other if the size of the destination data type variable is larger than or equal to the size of the source data type variable.
- Widening conversion takes place when the destination type is greater than the source type.
- Assigning a wider type to a narrow type is known as narrowing conversion.
- The widening conversion is implicit while the narrowing conversions are explicit.

Casting and Conversion in Java (Contd.)

- Explicit conversion occurs when one data type cannot be assigned to another data type using implicit conversion.
- In an explicit conversion, you must convert the data type to the compatible type.
- Explicit conversion between incompatible data types is known as casting.
- The following syntax shows how to use a cast to perform conversion between two incompatible types:

```
(type) value
```

- Explicit conversion between incompatible data types is known as casting.

Casting and Conversion in Java (Contd.)

- You can use the following code to perform type casting of an int number, 259 and a double number, 350.55 to byte type:

```
class TypeCast
{
  public static void main(String arr[])
  {
          byte b;
          int i = 259;
          double d = 350.55;
          b = (byte) i;
          System.out.println("Value of int to byte conversion " +
b);
```

Casting and Conversion in Java (Contd.)

```
b = (byte) d;
     System.out.println("Value of double to byte conversion " + b);

     i = (int) d;
     System.out.println("Value of double to int conversion " + i);

   }
}
```

Overloading Constructors

- A *constructor* is a method that is automatically invoked every time an instance of a class is created.
- Constructors share the same name as the class name and do not have a return type.
- You can use the following code to overload the Cuboid() constructor to calculate the volume of a rectangle and a square:

```
class Cuboid
 {
        double length;
        double width;
        double height;
```

Overloading Constructors (Contd.)

```
// Constructor declared which accepts three arguments
        Cuboid(double l, double w, double h)
        {
                length = l;
                width = w;
                height = h;
        }
        // Overloaded constructor declared which accepts one argument
        Cuboid(double side)
        {
                length = width = height = side;
        }
        double volume()
        {
                return length*width*height;
        }
}
```

Overloading Constructors (Contd.)

```
class ConstrOverloading
{
        public static void main(String args[])
        {
                Cuboid cub1 = new Cuboid(5, 10, 15);
                Cuboid cub2 = new Cuboid(5);
                double vol;

                vol = cub1.volume();
                System.out.println("Volume of the Cuboid is: "+ vol);
                vol = cub2.volume();
                System.out.println("Volume of the Cube is :" + vol);
        }
}
```

Best Practices

Short-circuiting

- Short-circuiting is the process in which a compiler evaluates and determines the result of an expression without evaluating the complete expression but by partial evaluation of one of the operands.
- Java supports conditional operators, such as AND (&&) and OR (||), which are also called short-circuit operators.
- For example, the expression, op1 && op2 && op3 evaluates true only if all the operands, op1, op2, and op3 are true.

Best Practices

Short-circuiting (Contd.)

- You can use the following code snippet to print a result if both of the conditions specified by the && operator are true:

```
if(3<5 && 3>2)
System.out.println("Result");
else
System.out.println("No result");
```

- You can use the following code snippet to use && operator as a short circuit operator.

```
if(3>5 && 3>2)
System.out.println("Result");
else
System.out.println("No result");
```

Best Practices

Use of switch-case and if-else Construct

- When you have multiple options and one choice, you use the switch-case construct.
- When you have multiple conditions out of which only one condition can be true, you use the nested if-else construct.

Best Practices

Using the for Loop Construct

- When the control variable of the looping statement is of int type and the number of times a loop should execute is known in advance, you use the, for, loop. In the, for, loop, all the three parts, such as initialization, condition, and increment or decrement are defined initially at one place.

Best Practices

Using Brackets with the if Statement

- You should use opening and closing brackets with the `if` statement.
- When the `if` block consists of more than one statement, place the statements in brackets.
- If the if block consists of only one statement, it is not necessary to place single statement in brackets.
- However, it is advisable to place even a single statement in brackets because brackets enhance the code clarity and minimize the chances of errors.

Tips and Tricks

Ternary operator

- The conditional operator (?:) operates on three operands and is therefore called the ternary operator.
- The syntax of the ternary operator is:

```
(boolean_expr) ? val1 : val2
```

- Ternary operator is used to replace the if-else statements.
- You can use the following code snippet to display the grade scored by a student using the ternary operator:

```
Score = 95;
(Score>=90) ? System.out.println("Grade is A."); :
    System.out.println("Grade is B.");
```

FAQs

- *Can you use a switch statement inside another switch statement?*

  Yes, you can use a switch statement within another switch statement. This is called nested switch. Each switch statement creates its own block of case statements. Therefore, no conflict occurs between the case labels of the inner and outer switch statements .

- *How does the program stop running if a loop never ends?*

  A program stops running when it enters an infinite loop by pressing the keys Ctrl and C simultaneously .

FAQs (Contd.)

- *Can you add the numeric value of one string to the value of another if the + operator is used with strings to link up two different strings*?

  Yes, you can use the value of a String variable as an integer only by using a method that converts the value of the string variable into a numeric form. This is known as type casting because it casts one data type, such as a string into another data type, such as int.

- *Can the == operator be used to determine whether two strings have the same value as in name == "John"?*

  Yes, the == operator can be used to determine whether two strings have the same value. In Java, you can also compare two strings using the equals() method.

Challenge

1. Match the following:

a.	++	i. Logical AND
b.	&&	ii. Logical OR
c.	%	iii. Logical NOT
d.	&	iv. Unsigned Shift
e.	||	v. Bit-wise XOR
f.	>>>	vi. Arithmetic Assignment
g.	!	vii. Unary Increment
h.	^	viii. Bit-wise AND
i.	+=	ix. Ternary
j.	?	x. Modulus

2. The if decision construct is used when there are multiple values for the same variable. (True/False).

Challenge (Contd.)

3. Predict the output of the following code snippet

```
int n=5, d=4;
int remainder = n % d;
if (div != 0)
System.out.println(n + " is not completely divisible by " + d);
else
System.out.println(n + " is completely divisible by " + d);
```

a) 5 is not completely divisible by 4
b) 5 is completely divisible by 4

Solutions to Challenge

1. a-vii, b-i, c-x, d-viii, e-ii, f-iv, g-iii, h-v, i-vi, j-ix
2. False
3. a 5 is not completely divisible by 4