GANPAT UNIVERSITY

U. V. PATEL COLLEGE OF ENGINEERING

2CEIT302

OBJECT ORIENTED PROGRAMMING

UNIT 3

THE STRUCTURE OF A JAVA PROGRAM

Prepared by: Prof. Y. J. Prajapati (Asst. Prof in IT Dept. , UVPCE)

Outline

- Structure of a Java Program, Comments
- Expressions and Statements
- Block Statements and Scope

Structure of a Java Program, Comments

- A Java program involves the following sections:
  - Documentation Section
  - Package Statement
  - Import Statements
  - Interface Statement
  - Class Definition
  - Main Method Class
    - Main Method Definition

Structure of a Java Program, Comments

- Documentation Section
  - It includes the comments that improve the readability of the program. A comment is a non-executable statement that helps to read and understand a program especially when your programs get more complex.
  - It is simply a message that exists only for the programmer and is ignored by the compiler. A good program should include comments that describe the purpose of the program, author name, date and time of program creation.
  - This section is optional and comments may appear anywhere in the program.

Comments:

Single line (or end-of line) comment:

- It starts with a double slash symbol (//) and terminates at the end of the current line. The compiler ignores everything from // to the end of the line.

For example:

// Calculate sum of two numbers

Multiline Comment:

- Java programmer can use C/C++ comment style that begins with delimiter /* and ends with */. All the text written between the delimiter is ignored by the compiler. This style of comments can be used on part of a line, a whole line or more commonly to define multi-line comment.

For example

```
/*calculate sum of two numbers

and it is a multiline comment */
```

Documentation comments:

This comment style is new in Java. Such comments begin with delimiter /** and end with */. The compiler also ignores this type of comments just like it ignores comments that use / * and */. The main purpose of this type of comment is to automatically generate program documentation. The java doc tool reads these comments and uses them to prepare your program's documentation in HTML format.

The documentation comment is used to create documentation API. To create documentation API, you need to use **javadoc tool**.

Syntax:

```
/** This is documentation comment */
```

Example:

```
/** The Calculator class provides methods to get addition and subtraction of given 2 numbers.*/
    public class Calculator {
/** The add() method returns addition of given numbers.*
/
   public static int add(int a, int b)
   {
     return a+b;
   }

/** The sub() method returns subtraction of given numbers.
*/
   public static int sub(int a, int b)
   {
    return a-b;
   }
}
```

Compile it by javac tool:
javac Calculator.java
Create Documentation API by javadoc tool:
javadoc Calculator.java

Now, there will be HTML files created for your Calculator class in the current directory. Open the HTML files and see the explanation of Calculator class provided through documentation comment.

Structure of a Java Program

- **Package Statement**
  - You can create a package with any name. A package is a group of classes that are defined by a name. That is, if you want to declare many classes within one element, then you can declare it within a package. It is an optional part of the program, i.e., if you do not want to declare any package, then there will be no problem with it, and you will not get any errors. Here, the package is a keyword that tells the compiler that package has been created.
  - Classes in java could be placed in different directories/packages based on the module they are used in or the functionality it provides. For all classes that belong to a single parent source directory, path from source directory is considered as package declaration.
  - Example : package package_name;

- **Import Statements**
  - Many predefined classes are stored in packages in Java, an import statement is used to refer to the classes stored in other packages. An import statement is always written after the package statement but it has to be before any class declaration.

Example

```
Import java.util.Date; /* imports only the Date class in java.util package */

import java.applet.*; // imports all the classes in java applet
```

- **Interface Statement**
  - In the interface section, we specify the interfaces. An interface is similar to a class but contains only constants and method declarations. Interfaces cannot be instantiated.
  - They can only be implemented by classes or extended by other interfaces.
  - It is an optional section and is used when we wish to implement multiple inheritance feature in the program.

Structure of a Java Program

Class Definition:

- A Java program may contain several class definitions, classes are an essential part of any Java program. It defines the information about the user-defined classes in a program.
- A class is a collection of variables and methods that operate on the fields. Every program in Java will have at least one class with the main method.

```
class Addition
{
    void add(String args[])
    {
        int a=2, b=3, c;
        c=a+b;
        System.out.println(c);
    }
}
```

Main Method Class

- Every Java stand-alone program requires the main method as the starting point of the program. This is an essential part of a Java program.
- There may be many classes in a Java program, and only one class defines the main method. Methods contain data type declaration and executable statements.

```
class HelloJava
{
    public static void main(String args[])
    {
        System.out.println("Hello Java");
    }
}
```

Structure of a Java Program

Expressions and Statements

Expression

- A key component of programming is the means of calculating and testing values. In Java, an **expression** is the line of code that either holds or calculates these values. An expression can be thought of as the setup to the rest of the code. That is, we'll setup a variable to count transactions: This line of code has only one purpose, and that is to declare the variable.
- You can give a value to a variable when you declare it. However, for this lesson, we need to understand the concept of expression versus statement. It is the setup to the rest of the program. The following code example is a great example of a simple Java expression:

```
int userCount;
```

- In this code, all we have done is declare the variable. Now, if we actually want to DO something with that variable, we can. These actions are called statements.

Statement

- Think of a statement as a command to Java - an order to perform a task. The expression above only defined the variable; when we convert that expression into a statement, we will actually do something with it.
- In the expression, we simply declared the variable for our user counter. The next line of code will take action on the variable. In the following example, we'll add a value (in this case another variable) to the user count:

```
int userCount;


userCount = userCount + totalCount;
```

Block Statements and Scope

Block Statement

- A block statement is a sequence of zero or more statements enclosed in braces. A **block statement is generally used to group together several statements**, so they can be used in a situation that requires you to use a single statement. In some situations, you can use only one statement.
- If you want to use more than one statement in those situations, you can create a block statement by placing all your statements inside braces, which would be treated as a single statement. You can think of a block statement as a compound statement that is treated as one statement.

Example :

```
{ //block start
      int var = 20;
      var++;
} //block end
```

Scope

- In Java, as in any programming language, each variable has a scope. This is the segment of the program where a variable can be used and is valid.
- **Class Scope**

Each variable declared inside of a class's brackets ({}) with *private* access modifier but outside of any method, has class scope. As a result, **these variables can be used everywhere in the class, but not outside of it:**

```
public class ClassScopeExample
{
private Integer amount = 0;
public void exampleMethod()
{
 amount++;
}
public void anotherExampleMethod()
{       Integer anotherAmount = amount + 4;
 }
```

Method Scope

When a variable is declared inside a method, it has method scope and **it will only be valid inside the same method**

Exampe

```
public class MethodScopeExample
 {
      public void methodA()
      {
          Integer area = 2;
      }
     public void methodB()
     {
      // compiler error, area cannot be
resolved      to a variable
      area = area + 2;
}}
```

Scope

Loop Scope

- If we declare a variable inside a loop, it will have a loop scope and **will only be available inside the loop**

Example

```
public class LoopScopeExample
{
    List<String> listOfNames = Arrays.asList("Joe", "Susan", "Pattrick");
     public void iterationOfNames()
    {
        String allNames = "";
         for (String name : listOfNames)
        {
                allNames = allNames + " " + name;
        }
            // compiler error, name cannot be resolved to a variable
            String lastNameUsed = name;
     }
}
```

Bracket Scope

- We can define additional scopes anywhere using brackets ({})

Example

```
public class BracketScopeExample
{
    public void mathOperationExample()
    {
            Integer sum = 0;
            {
                Integer number = 2;
                sum = sum + number;
             }
 // compiler error, number cannot be solved as a variable
       number++;
 }
}
```


THANK YOU!
ANY QUESTIONS?