



# **Object Oriented Programming with Java**

## **(Subject Code: BCS-403)**

### **Unit 3**

### **Lecture 25**

# Lecture 25

- Java Module System
- Diamond Syntax with 08 Inner Anonymous Class

# Java 9 Module System

Java Module System is a major change in Java 9 version.

Java added this feature to collect Java packages and code into a single unit called module.

- In earlier versions of Java, there was no concept of module to create modular Java applications, that why size of application increased and difficult to move around.
- Even JDK itself was too heavy in size, in Java 8, rt.jar file size is around 64MB.
- To deal with situation, Java 9 restructured JDK into set of modules so that we can use only required module for our project.

## module system includes various tools and options

- It Includes various options to the Java tools **javac**, **jlink** and **java** where we can specify module paths that locates to the location of module.
- Modular JAR file is introduced. This JAR contains module-info.class file in its root folder.
- JMOD format is introduced, which is a packaging format similar to JAR except it can include native code and configuration files.
- The JDK and JRE both are reconstructed to accommodate modules. It improves performance, security and maintainability.
- Java defines a new URI scheme for naming modules, classes and resources.

# Java 9 Module

Module is a collection of Java programs or software. To describe a module, a Java file `module-info.java` is required.

This file also known as module descriptor and defines the following

- Module name
- What does it export
- What does it require

## Module Name

It is a name of module and should follow the reverse-domain-pattern. Like we name packages, e.g. com.javatpoint.

# How to create Java module

Creating Java module required the following steps.

- Create a directory structure
- Create a module declarator
- Java source code



# Create a Directory Structure

```
src
├── com.javatpoint
│   ├── com
│   │   ├── javatpoint
│   │   │   └── Hello.java
│   └── module-info.java
```

Note: The name of the directory containing a module's sources should be equal to the name of the module, e.g. com.javatpoint.

# Java Source Code

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

Save this file inside **src/com.javatpoint/com/javatpoint/** with **Hello.java** name.

# Compile Java Module

To compile the module use the following command.

```
javac -d mods --module-source-path src/ --module com.javatpoint
```

After compiling, it will create a new directory that contains the following structure.

```
mods/
├── com.javatpoint
│   ├── com
│   │   └── javatpoint
│   │       └── Hello.class
│   └── module-info.class
```

# Run Module

```
java --module-path mods/ --module com.javatpoint/com.javatpoint.Hello
```

Output:

Hello from the Java module

# Diamond Operator

- Diamond operator was introduced in Java 7 as a new feature.
- The main purpose of the diamond operator is to simplify the use of generics when creating an object.
- It avoids unchecked warnings in a program and makes the program more readable.
- The diamond operator could not be used with Anonymous inner classes in JDK 7.
- In JDK 9, it can be used with the anonymous class as well to simplify code and improves readability.

- Before JDK 7, we have to create an object with Generic type on both side of the expression like

```
List<String> ABES= new ArrayList<String>();
```

- When Diamond operator was introduced in Java 7, we can create the object without mentioning generic type on right side of expression like:

```
List<String> ABES = new ArrayList<>();
```

# Problem with Diamond Operator in JDK 7

The problem is it will only work with normal classes. Suppose if we want to use the diamond operator for anonymous inner class then compiler will throw error message.

# Java 9 Anonymous Inner Classes

```
abstract class ABCD<T>{  
    abstract T show(T a, T b);  
}  
  
public class TypeInferExample {  
    public static void main(String[] args) {  
        ABCD<String> a = new ABCD<>()  
        {  
            String show(String a, String b)  
            {  
                return a+b;  
            }  
        };  
        String result = a.show("Java", "9");  
        System.out.println(result);  
    }  
}
```

**Output:**

**Java9**