Session 06:

JAVA Modifiers and Packages

# Describe field and method modifiers

# Explain different types of modifiers

# Explain rules and best practices for using field modifiers

# Describe class variables

# Explain creation of static variables and methods

# Describe package and its advantages

# Explain creation of userdefined package

# Explain creation of .jar files for deployment

# Describe field and method modifiers

### There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

### The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

### There are four types of Java access modifiers:

#### **Access Modifiers** - controls the access level

#### **Non-Access Modifiers** - do not control access level, but provides other functionality

#### **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

#### **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

#### **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

#### **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

# Explain different types of modifiers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifier** | **within class** | **within package** | **outside package by subclass only** | **outside package** |
| **Private** | Y | N | N | N |
| **Default** | Y | Y | N | N |
| **Protected** | Y | Y | Y | N |
| **Public** | Y | Y | Y | Y |

# Explain rules and best practices for using field modifiers

## Access Modifiers

### For **classes**, you can use either public or default:

|  |  |
| --- | --- |
| **Modifier** | **Description** |
| public | The class is accessible by any other class |
| default | The class is only accessible by classes in the same package. This is used when you don't specify a modifier. You will learn more about packages in the Packages chapter |

### For **attributes, methods and constructors**, you can use the one of the following:

|  |  |
| --- | --- |
| **Modifier** | **Description** |
| public | The code is accessible for all classes |
| private | The code is only accessible within the declared class |
| default | The code is only accessible in the same package. This is used when you don't specify a modifier. You will learn more about packages in the Packages chapter |
| protected | The code is accessible in the same package and **subclasses**. You will learn more about subclasses and superclasses in the Inheritance chapter |

## Non-Access Modifiers

### For **classes**, you can use either final or abstract:

|  |  |
| --- | --- |
| **Modifier** | **Description** |
| final | The class cannot be inherited by other classes (You will learn more about inheritance in the Inheritance chapter) |
| abstract | The class cannot be used to create objects (To access an abstract class, it must be inherited from another class. You will learn more about inheritance and abstraction in the Inheritance and Abstraction chapters) |

### For **attributes and methods**, you can use the one of the following:

|  |  |
| --- | --- |
| **Modifier** | **Description** |
| final | Attributes and methods cannot be overridden/modified |
| static | Attributes and methods belongs to the class, rather than an object |
| abstract | Can only be used in an abstract class, and can only be used on methods. The method does not have a body, for example **abstract void run();**. The body is provided by the subclass (inherited from). You will learn more about inheritance and abstraction in the Inheritance and Abstraction chapters |
| transient | Attributes and methods are skipped when serializing the object containing them |
| synchronized | Methods can only be accessed by one thread at a time |
| volatile | The value of an attribute is not cached thread-locally, and is always read from the "main memory" |

# Describe class variables

## Final

### If you don't want the ability to override existing attribute values, declare attributes as final:

### **Example**

public class Main {

**final** int x = 10;

**final** double PI = 3.14;

public static void main(String[] args) {

Main myObj = new Main();

myObj.x = 50; // will generate an error: cannot assign a value to a **final** variable

myObj.PI = 25; // will generate an error: cannot assign a value to a **final** variable

System.out.println(myObj.x);

}

}

## Static

### A static method means that it can be accessed without creating an object of the class, unlike public:

### **Example**

#### An example to demonstrate the differences between static and public methods:

public class Main {

// Static method

static void myStaticMethod() {

System.out.println("Static methods can be called without creating objects");

}

// Public method

public void myPublicMethod() {

System.out.println("Public methods must be called by creating objects");

}

// Main method

public static void main(String[ ] args) {

myStaticMethod(); // Call the static method

// myPublicMethod(); This would output an error

Main myObj = new Main(); // Create an object of Main

myObj.myPublicMethod(); // Call the public method

}

}

## Abstract

### An abstract method belongs to an abstract class, and it does not have a body. The body is provided by the subclass:

### **Example**

// Code from filename: Main.java

// abstract class  
abstract class Main {

public String fname = "John";

public int age = 24;

public **abstract** void study(); // abstract method

}

// Subclass (inherit from Main)

class Student extends Main {

public int graduationYear = 2018;

public void study() { // the body of the abstract method is provided here

System.out.println("Studying all day long");

}

}

// End code from filename: Main.java

// Code from filename: Second.java

class Second {

public static void main(String[] args) {

// create an object of the Student class (which inherits attributes and methods from Main)

Student myObj = new Student();

System.out.println("Name: " + myObj.fname);

System.out.println("Age: " + myObj.age);

System.out.println("Graduation Year: " + myObj.graduationYear);

myObj.study(); // call abstract method  
 }

}

# Explain creation of static variables and methods

## What is Static Method in Java?

### **Static method in Java** is a method which belongs to the class and not to the object. A static method can access only static data. It is a method which belongs to the class and not to the object(instance). A static method can access only static data. It cannot access non-static data (instance variables).

#### A static method can call only other static methods and can not call a non-static method from it.

#### A static method can be accessed directly by the class name and doesn’t need any object

#### A static method cannot refer to “this” or “super” keywords in anyway

**Step 1)** Copy the following code into a editor

public class Demo{

public static void main(String args[]){

Student s1 = new Student();

s1.showData();

Student s2 = new Student();

s2.showData();

//Student.b++;

//s1.showData();

}

}

class Student {

int a; //initialized to zero

static int b; //initialized to zero only when class is loaded not for each object created.

Student(){

//Constructor incrementing static variable b

b++;

}

public void showData(){

System.out.println("Value of a = "+a);

System.out.println("Value of b = "+b);

}

//public static void increment(){

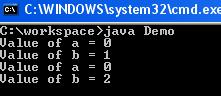
//a++;

//}

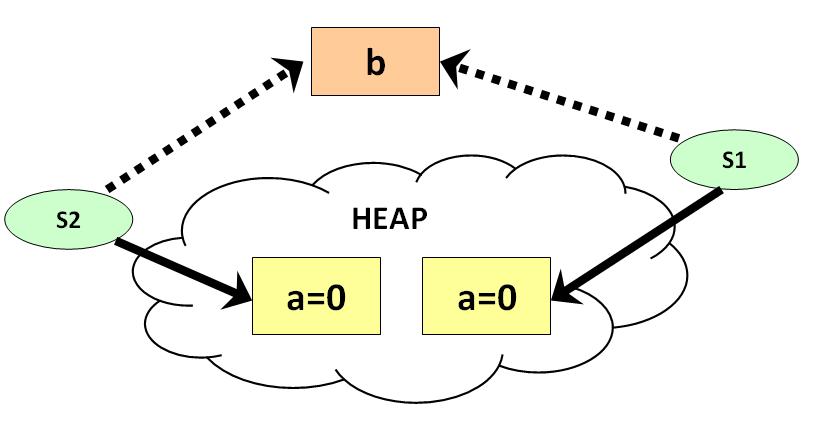
}

**Step 2)** Save & Compile the code. Run the code as, **java Demo**.

**Step 3)** Expected output show below



### Following diagram shows, how reference variables & objects are created and static variables are accessed by the different instances.



**Step 4)** It is possible to access a static variable from outside the class using the syntax **ClassName.Variable\_Name**. Uncomment line # 7 & 8 . Save , Compile & Run . Observe the output.

Value of a = 0

Value of b = 1

Value of a = 0

Value of b = 2

Value of a = 0

Value of b = 3

**Step 5)** Uncomment line 25,26 & 27 . Save , Compile & Run.

error: non-static variable a cannot be referenced from a static context a++;

**Step 6)** Error = ? This is because it is not possible to access instance variable “**a**” from java static class method “**increment**“.

## ****What is Static Block in Java?****

### The **static block** is a block of statement inside a Java class that will be executed when a class is first loaded into the JVM. A **static block helps to initialize the static data members**, just like constructors help to initialize instance members.

class Test{

static {

//Code goes here

}

}

Following program is the example of java static block.

### **Example: How to access static block**

public class Demo {

static int a;

static int b;

static {

a = 10;

b = 20;

}

public static void main(String args[]) {

System.out.println("Value of a = " + a);

System.out.println("Value of b = " + b);

}

}

#### you will get following output of the program.

Value of a = 10

Value of b = 20

# Describe package and its advantages

### The most common **advantages of packages in java** are reusability. During the development, if you feel you are writing some duplicate code that is already written. You can create such things in the form of classes inside a package. So, whenever you need to perform that same task or need the same code then you can use it by import that package.

### **For example:**If a company has two departments one is the HR Department,  and another is the IT Department. Both departments want to perform some calculations. So, we can place the logic of calculation in class that will be part of another package. In JAVA, we will create three classes in different packages.

Graphical user interface, text, application

Description automatically generated

package myCalculator;

public class Calculator {

public int sum(int a, int b)

{

return a+b;

}

public int sub(int a, int b)

{

return a-b;

}

public int multiple(int a, int b)

{

return a\*b;

}

public int divide(int a, int b)

{

return a/b;

}

}

package hR;

import myCalculator.Calculator;

public class HRDepartment {

public static void main(String[] args)

{

Calculator calculator = new Calculator();

System.out.println(calculator.sum(1, 3));

System.out.println(calculator.sub(8, 6));

System.out.println(calculator.multiple(2, 3));

System.out.println(calculator.divide(9, 3));

}

}

package iT;

import myCalculator.Calculator;

public class ITDepartment {

public static void main(String[] args)

{

Calculator calculator = new Calculator();

System.out.println(calculator.sum(1, 3));

System.out.println(calculator.sub(8, 6));

System.out.println(calculator.multiple(2, 3));

System.out.println(calculator.divide(9, 3));

}

}

# Explain creation of userdefined package

## How to Create a package?

### Creating a package is a simple task as follows

#### Choose the name of the package

#### Include the package command as the first line of code in your Java Source File.

#### The Source file contains the classes, interfaces, etc you want to include in the package

#### Compile to create the Java packages

### **Step 1)** Consider the following package program in Java:

package p1;

class c1(){

public void m1(){

System.out.println("m1 of c1");

}

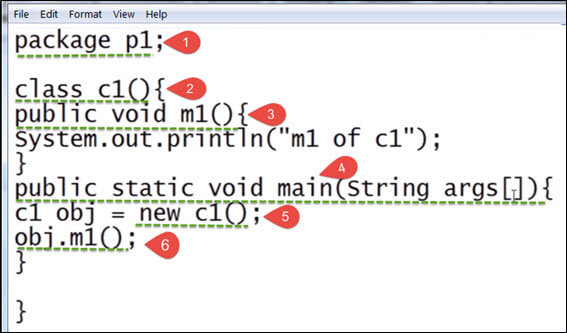
public static void main(string args[]){

c1 obj = new c1();

obj.m1();

}

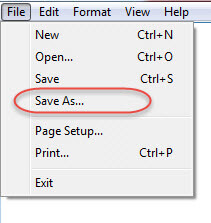
}

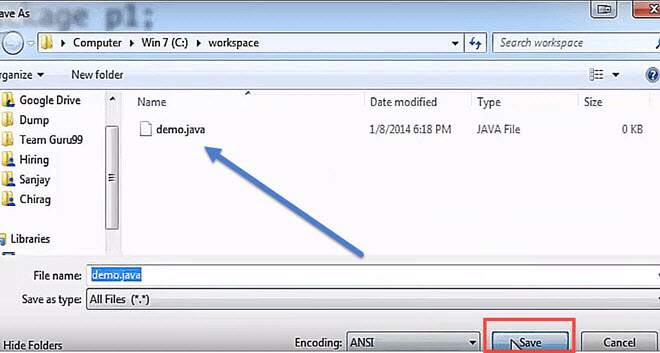


### Here,

1. To put a class into a package, at the first line of code define package p1
2. Create a class c1
3. Defining a method m1 which prints a line.
4. Defining the main method
5. Creating an object of class c1
6. Calling method m1

### **Step 2)** In next step, save this file as demo.java





### **Step 3)** In this step, we compile the file.



### The compilation is completed. A class file c1 is created. However, no package is created? Next step has the solution

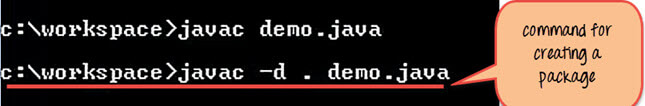


### **Step 4)** Now we have to create a package, use the command

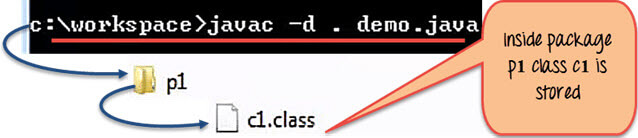
javac –d . demo.java

#### This command forces the compiler to create a package.

#### The **“.”**operator represents the current working directory.



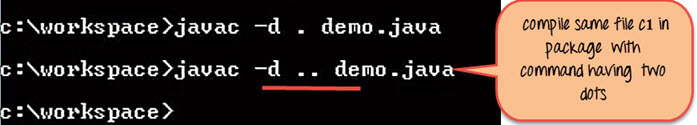
### **Step 5)** When you execute the code, it creates a package p1. When you open the java package p1 inside you will see the c1.class file.



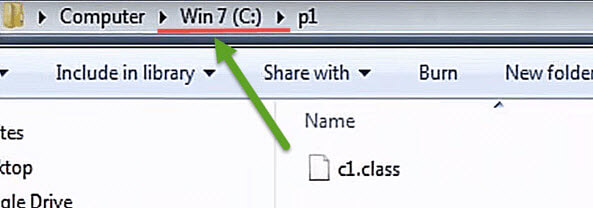
### **Step 6)** Compile the same file using the following code

javac –d .. demo.java

#### Here “..” indicates the parent directory. In our case file will be saved in parent directory which is C Drive



### File saved in parent directory when above code is executed.



### **Step 7)** Now let’s say you want to create a sub package p2 within our existing java package p1. Then we will modify our code as

package p1.p2;

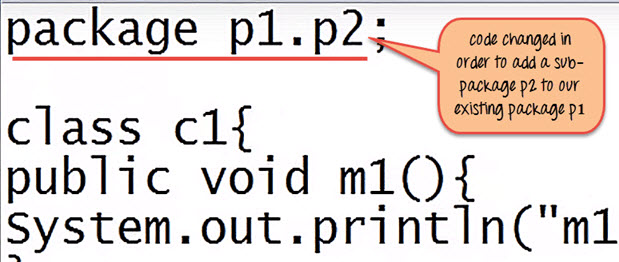
class c1{

public void m1() {

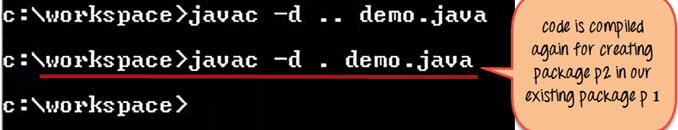
System.out.println("m1 of c1");

}

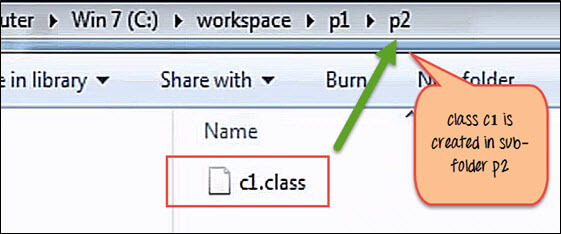
}



### **Step 8)** Compile the file



### As seen in below screenshot, it creates a sub-package p2 having class c1 inside the package.

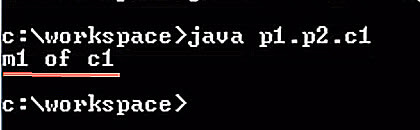


### **Step 9)** To execute the code mention the fully qualified name of the class i.e. the package name followed by the sub-package name followed by the class name –

java p1.p2.c1



### This is how the package is executed and gives the output as “m1 of c1” from the code file.



## How to Import Package

### To create an object of a class (bundled in a package), in your code, you have to use its fully qualified name.

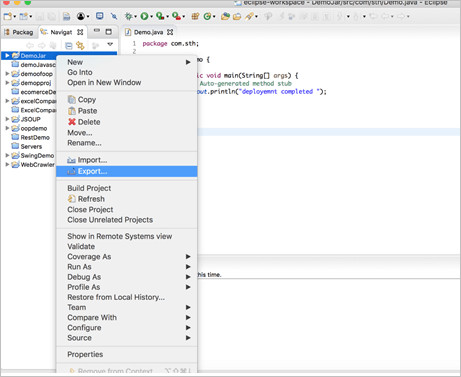
**Syntax**

import packageName;

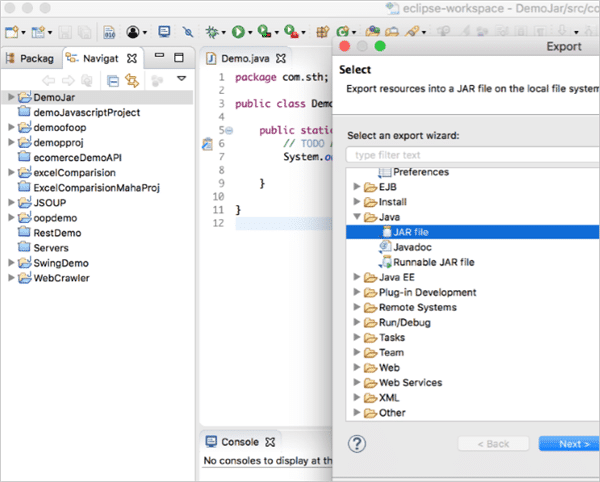
# Explain creation of .jar files for deployment

### To Create a Java JAR file

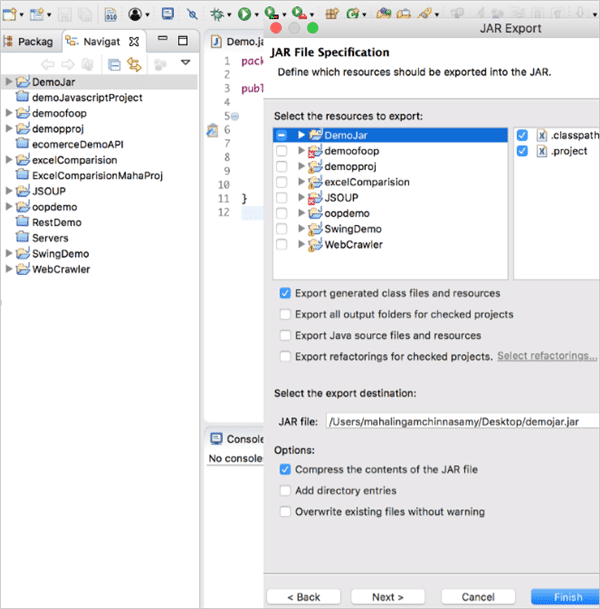
#### Right-click the project folder and select “export “option.



#### Select the java folder and select the jar file inside the java folder.



#### Enter the File name and file location to save.



#### Click on finish.

### Now, you can get the jar file in the specified location.

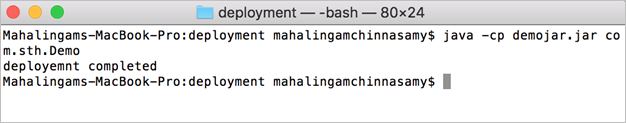
### To Run the JAR file

#### You can run the jar file from the command prompt/ terminal.

#### First, you need to go to the directory in which you have the jar file from the terminal/command prompt.

### Then, type the below command and hit enter,

### java –cp JarName mainclass\_with\_packagename



### Now, the program has been executed.

### Conclusion

#### Java class files can be wrapped as a jar file and that can be executed in any machine.

#### In this tutorial, we learned how to build a jar file and execute it.