UNIT III – INHERITANCE AND POLYMORPHISM

INTRODUCTION TO PACKAGES:

- Defined as a grouping of related types (classes, interfaces, enumerations, and annotations) providing access protection and namespace management.
- Used in Java in order to prevent naming conflicts, control access, make searching/locating and usage of classes, interfaces, enumerations, and annotations easier, etc.
- There are 2 types of packages. They are,
    - ✓ Built-in Packages
    - ✓ User-defined Packages.

USER DEFINED PACKAGES:

- We can define our own packages to bundle groups of classes/interfaces, etc.
- Using packages, it is easier to provide access control and it is also easier to locate the related classes.

DEFINING A PACKAGE:

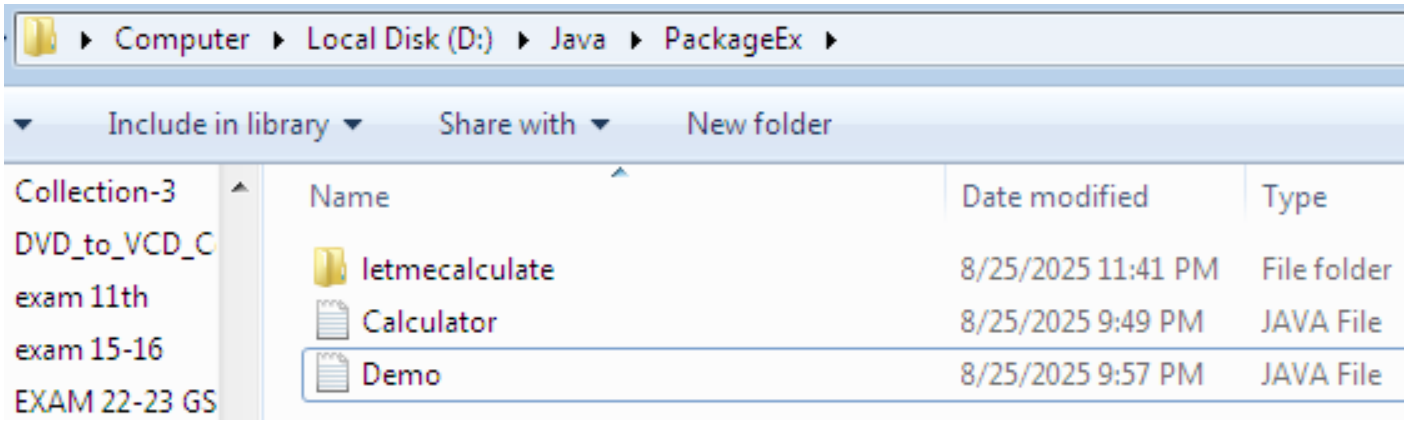
- While creating a package, we should choose a name for the package and include a package statement along with that name at the top of every source file that contains the classes, interfaces, enumerations, etc.

package mypackage;

- The package statement should be the first line in the source file.
- There can be only one package statement in each source file, and it applies to all types in the file.

```
package letmecalculate;
public class Calculator
{
    public int add(int a, int b)
    {
        return a+b;
    }
    public int sub(int a, int b)
    {
        return a-b;
    }
}
```

- While compiling this Calculator.java, automatically letmecalculate folder is created and Calculator.class stored there.
- javac –d . Calculator.java -> generates the class file under the folder letmecalculate.



IMPORTING A PACKAGE:

```
import letmecalculate.Calculator;
public class Demo
{
        public static void main(String args[])
        {
                Calculator obj = new Calculator();
                System.out.println("Addition:" +obj.add(100, 200));
                System.out.println("Subtraction:" +obj.sub(400, 200));
        }
}
```

```
D:\Java\PackageEx>javac -d . Calculator.java

D:\Java\PackageEx>javac Demo.java

D:\Java\PackageEx>java Demo
Addition:300
Subtraction:200

D:\Java\PackageEx>
```

ACCESS CONTROL:

- Java provides four levels of access control (modifiers) that define where members (variables, methods, constructors, classes) can be accessed.

1. public:

- ✓ Accessible everywhere (same class, package, subclass, other packages).
- ✓ Used for APIs, libraries, utility functions.

```
public class Example
{
    public int num = 10;
}                                   // Accessible from anywhere.
```

2. protected:

- ✓ Accessible: Inside the same class, inside the same package and in a subclass (even if in a different package).
- ✓ Not accessible in a non-subclass outside the package.

```
class Parent
{
   protected int value = 100;
}
class Child extends Parent
{
   void show()
{
      System.out.println(value);
   }
}                                          // Accessible in subclass
```

3. default (package-private):

- ✓ When no modifier is specified.
- ✓ Accessible only within the same package.
- ✓ Not visible in subclasses of other packages.

```
class MyClass
{
   int data = 50; // default
}                                   // ✓ Accessible inside same package,
                                    ✗ not accessible outside.
```

4. private:

- ✓ Accessible only inside the same class.
- ✓ Not visible in subclass, not visible in same package.
- ✓ Used for data hiding (encapsulation).

```
class Secret
{
   private int pin = 1234;
   private void displayPin()
   {
      System.out.println(pin);
   }
}                                    // Accessible only within Secret class.
```

Example program:

```
package calculator;
public class Calculator
{
   // Public: accessible everywhere
   public int add(int a, int b)
   {
      return a + b;
   }
  // Protected: accessible in subclasses and same package
   protected int subtract(int a, int b)
   {
      return a - b;
   }
// Default (package-private): accessible only inside this package
   int multiply(int a, int b)
   {
      return a * b;
   }
   // Private: accessible only inside this class
   private int divide(int a, int b)
   {
      if (b == 0)
      {
         throw new ArithmeticException("Division by zero not allowed");
      }
      return a / b;
   }
}
```