Experiment no.2

Title: How to create, access and implement a java package

Aim: Write a Java code to : Create a Java Package Access a Java package Implementation interfaces

Learning Objective

1. Students should able to create, access and implement a java package or a predefined package.
2. It strengthen the ability of the students to identify and apply the suitable Package codes for the given real world problem.
3. It enables them to gain knowledge in practical applications of Java Packages .

Lab Outcomes

1. Write a java program to create, access and implement interfaces of a java package.
2. Being able to effectively use the interfaces in Java Package implementation.

What is java package

Java packages serve as containers for classes, interfaces, enumerations, and other program elements. They play a pivotal role in creating a modular and maintainable codebase, which is essential for building robust and scalable software systems. Through packages, Java enables developers to achieve several vital objectives:

Organization: Packages allow developers to group related classes and components together. This logical grouping enhances code organization, making it more intuitive and comprehensible. By adhering to a systematic package structure, developers can quickly locate, modify, and extend specific parts of their codebase.

Access Control: Packages introduce the concept of access control in Java, providing varying levels of visibility and encapsulation. This means that you can control which classes and members are accessible to other parts of your code and which should remain hidden. This ensures the integrity of your code and helps prevent unintentional misuse of internal components.

Modularity: Java's package system promotes modularity, enabling the development of components that can be reused across different projects. With well-designed packages, you can create libraries and frameworks that are easily shareable and extendable. This promotes code reusability and accelerates development by building on existing solutions.

Collaboration: In team-based software development, packages facilitate collaboration. Developers can work on separate packages or modules independently, reducing the chances of code conflicts and making it easier to integrate contributions from multiple team members.

This introduction to Java packages sets the stage for a deeper exploration of their creation, accessing, and practical implementation. In the following sections, we will delve into the specifics of creating packages, accessing their contents, and implementing packages in real-world scenarios.

Creating a Java Package

How to crate a Java package

A java package is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Here, we will have the detailed learning of creating and using user-defined packages.

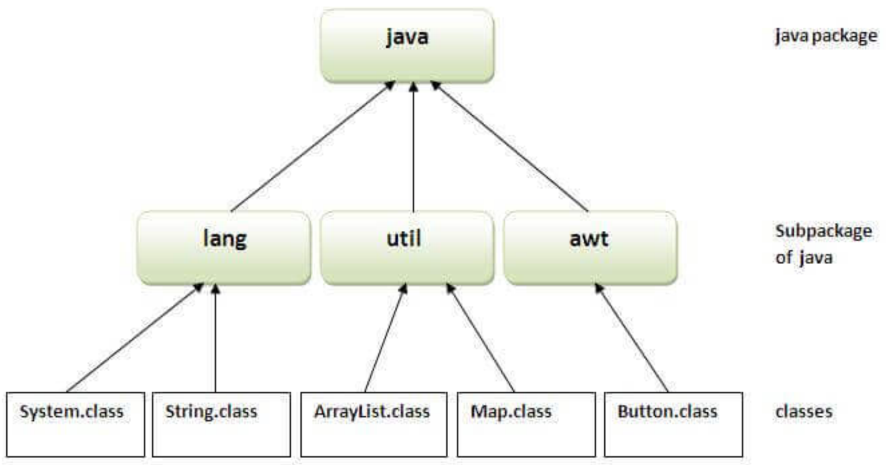

Figure 1: Advantages of Java packages

Advantages

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2) Java package provides access protection.
3) Java package removes naming collision. **??** .

Creating simple package

To create a simple package in Java, follow these steps:

Choose a Package Name : Decide on a meaningful package name, typically in reverse domain format ***(e.g., com.example.mypackage).***

Create a Directory Structure: In your project directory, create a subdirectory matching the package name ***(e.g., com/example/mypackage).***

Place Classes in the Directory: Save your Java classes within the package directory, adding the package statement at the top of each file, specifying the package name.

Compile the Classes: Use the javac compiler to compile your classes. Ensure the class files are stored in the package directory.

Accessing from Other Classes: To use classes from this package in other parts of your project, include an import statement specifying the package and class name.

`myPackage`.

```java
package myPackage;

public class MyClass {
    public void display() {
        System.out.println("Hello from myPackage!");
    }
}
```

Accessing the package

To access a Java package, you need to follow specific conventions to ensure your code can locate and use classes and components within that package. To do so, you should:

Import the Package: At the beginning of your Java source file, use the import statement to specify the package you want to access. For example, if you have a package named com.example.mypackage, include the line import com.example.mypackage.*; or specify individual classes within the package. This statement tells the Java compiler where to find the classes you intend to use.

Ensure Classpath Configuration: If you are working with packages that are external to your project, ensure that the package or JAR file containing the package is on your classpath. This enables the Java runtime to locate the package at runtime.

Use the Classes: Once the package is imported and your classpath is correctly configured, you can use the classes and components from the package within your code. Simply reference them by their names, assuming you have adhered to appropriate access modifiers (e.g., public, protected) within the package to control visibility.

For example, if you imported the com.example.mypackage and it contains a class named MyClass, you can create an instance of MyClass and use its methods within your code.

In summary, accessing a Java package involves importing the package, configuring the classpath if necessary, and then using the package's classes as needed in your code.

```
import myPackage.MyClass;

public class Main {
    public static void main(String[] args) {
        MyClass obj = new MyClass();
        obj.display();
    }
}
```

Implementation interfaces

Create an Interface: Define the interface that specifies a set of methods that classes within the package must implement. For instance, you can create an interface named MyInterface: let's create a Java interface called `MyInterface`.

```
package myPackage;

public interface MyInterface {
    void someMethod();
}
```

Implement the Interface: In the same package or within classes that use the package, create classes that implement the interface. These classes must provide concrete implementations for all methods declared in the interface:

`MyInterface` in a class.

```
package myPackage;

public class MyImplementation implements MyInterface {
    public void someMethod() {
        System.out.println("Interface method implementation");
    }
}
```

Use the Interface: You can now use classes that implement the interface throughout your package or application. When working with these classes, you can treat them as instances of the interface type, providing a common interface for interacting with various implementations:

Lab Outcomes

1. Write a Java code to Create Package
2. Access and implement a Java package

Conclusion

This project demonstrates the creation of a Java package, accessing its classes from another class, and implementing an interface within the package.

```
– Vishal Lalit Dhake SE(B44)
  Vrushabh Chavhan SE (B42)
```