Chapter 14: Packages

14.1 Introduction to Packages

What is a Package?

In Java, a **package** is a namespace that organizes a set of related classes and interfaces. It is used to group related classes, interfaces, and sub-packages. Packages help avoid name conflicts, improve code maintainability, and provide access control.

Packages are essentially a way to organize Java classes into namespaces and allow for easy management of large codebases.

Why are Packages Important?

- **Organize Code:** Packages help in grouping related classes and interfaces, making the codebase cleaner and easier to manage.
- **Avoid Naming Conflicts:** They prevent name conflicts by differentiating classes with the same name from different packages.
- **Access Control:** Packages allow you to control the visibility and accessibility of classes, methods, and variables.
- **Reusability:** Classes in packages can be reused across multiple projects without redefinition.

14.2 Types of Packages

1. **Built-in Packages:**
   Java provides a large number of built-in packages that contain useful classes and interfaces, such as `java.util`, `java.io`, and `java.math`.
   - **Example:**
     The `java.util` package contains utility classes like `ArrayList`, `HashMap`, and `Date`.
2. **User-defined Packages:**
   These are packages that are created by the programmer to organize their own classes

and interfaces.

14.3 Creating a User-Defined Package

To create a package in Java, use the `package` keyword followed by the package name at the beginning of the Java source file.

Syntax:

package package_name;

Example:

Let's say we want to create a package named `com.example.util` for utility classes.

```
package com.example.util;  // Defining a package


public class MathUtils {
    public static int add(int a, int b) {
        return a + b;
    }

    public static int subtract(int a, int b) {
        return a - b;
    }
}
```

Here, the class `MathUtils` is part of the `com.example.util` package.

14.4 Accessing Classes from a Package

To use a class from a package in another class, you need to **import** the class using the `import` keyword.

Syntax:

```
import package_name.ClassName;
```

You can also import all classes from a package using the wildcard (`*`).

Example:

```
import com.example.util.MathUtils;  // Importing the MathUtils class


public class Main {
    public static void main(String[] args) {
        int sum = MathUtils.add(5, 3);
        int difference = MathUtils.subtract(5, 3);


        System.out.println("Sum: " + sum);          // Output: Sum: 8
        System.out.println("Difference: " + difference);  // Output: Difference: 2
    }
}
```

Alternatively, you can import all classes in the `com.example.util` package:

```
import com.example.util.*;  // Importing all classes in the package
```

14.5 Package Naming Conventions

To avoid naming conflicts, Java encourages a specific naming convention for packages:

- Package names are typically written in **lowercase**.
- Use **dot notation** to separate different levels of the package hierarchy (e.g., `com.example.util`).
- Package names should be unique, often based on the domain name of the organization.

Example:

- `com.companyname.projectname`
- `org.apache.commons`

14.6 Access Control in Packages

Java uses **access modifiers** to control the visibility of classes, methods, and variables. The main access levels include:

1. **Public:** The class or member can be accessed from any other class.
2. **Private:** The class or member is accessible only within the class itself.
3. **Protected:** The class or member is accessible within the package and by subclasses.
4. **Default (Package-Private):** The class or member is accessible only within the same package (no modifier specified).

Example:

```
package com.example.util;


public class MathUtils {
    public static int add(int a, int b) {
        return a + b;
```

```
    }

    private static int multiply(int a, int b) {
        return a * b;  // This can only be accessed within the MathUtils class
    }
}
```

In this example:

- The `add()` method is `public` and can be accessed from outside the `MathUtils` class.
- The `multiply()` method is `private` and can only be accessed within the `MathUtils` class.

14.7 Sub-Packages

Java allows the creation of **sub-packages** within a package. Sub-packages are useful for organizing classes into a more structured hierarchy.

Example:

```
package com.example.util.math;  // A sub-package of com.example.util

public class Calculator {
    public static int square(int num) {
        return num * num;
    }
}
```

In this example, `com.example.util.math` is a sub-package of `com.example.util`.

To use the `Calculator` class, you would import it as follows:

```
import com.example.util.math.Calculator;


public class Main {
    public static void main(String[] args) {
        System.out.println("Square of 5: " + Calculator.square(5));  // Output: Square of 5: 25
    }
}
```

14.8 Java's Built-in Packages

Java provides a large set of built-in packages for handling common programming tasks. Some common packages include:

- **java.util:** Contains utility classes such as `ArrayList`, `HashMap`, `Date`, etc.
- **java.io:** Contains classes for input and output operations, such as `File`, `BufferedReader`, `BufferedWriter`, etc.
- **java.lang:** Contains fundamental classes such as `String`, `Math`, `System`, etc. (automatically imported).
- **java.math:** Provides classes for mathematical operations, such as `BigDecimal`, `BigInteger`, etc.
- **java.net:** Contains classes for network programming, such as `URL`, `Socket`, etc.

Example of Using java.util Package:

```
import java.util.ArrayList;
```

```
public class ListExample {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();
        list.add("Apple");
        list.add("Banana");
        list.add("Cherry");

        System.out.println("List: " + list);  // Output: List: [Apple, Banana, Cherry]
    }
}
```

14.9 Using Java's Built-in Packages

Java allows you to use its built-in packages without needing to explicitly define them, as these packages are already available in the JDK.

Example:

```
import java.util.Date;

public class DateExample {
    public static void main(String[] args) {
        Date currentDate = new Date();
        System.out.println("Current date and time: " + currentDate);
    }
}
```

14.10 Conclusion

Summary of Key Points:

- A **package** is a way to organize Java classes into namespaces.
- **Built-in packages** in Java offer pre-defined classes for common tasks, while **user-defined packages** help in organizing your own classes.
- You can **import** classes from packages into your program using the `import` keyword.
- Java provides **access control** mechanisms like `public`, `private`, `protected`, and **default** to control the visibility of classes and members.
- **Sub-packages** allow further organization within a package hierarchy.

Practical Application:

Understanding and using packages is fundamental for writing clean, maintainable, and organized code. Packages also allow you to manage large projects by logically grouping related classes, reducing name conflicts, and improving code reusability.