In Java, packages are a way to organize and structure your code into namespaces, making it easier to manage and maintain large projects. They provide a way to group related classes and interfaces together, preventing naming conflicts and allowing you to create a modular and organized codebase. Here are the key points to understand about Java packages:

1. **Namespace Organization:** Packages are used to create namespaces for classes and interfaces. This means that you can have classes with the same name in different packages without any conflicts. For example, you can have a `Person` class in two different packages, and they won't clash.

2. **Package Declaration:** At the top of every Java source file, you'll typically find a `package` declaration that specifies the package to which the class belongs. For example:

   ```java
   package com.example.myapp;
   ```

   This declaration is followed by the `import` statements to bring in classes from other packages if needed.

3. **Java Standard Library Packages:** Java provides a rich set of standard libraries organized into packages. Commonly used packages include `java.lang` (basic language constructs like `String` and `Object`), `java.util` (utilities like collections and data structures), `java.io` (input and output operations), and many others.

4. **Custom Packages:** In addition to using the standard packages, you can create your own custom packages to organize your code. To define a custom package, create a directory structure that matches the package name and place your Java files within it. For example, if you have a package called `com.example.myapp`, you should create a directory structure like this:

   ```
   com/
   ├── example/
   │   └── myapp/
   │       └── MyClass.java
   ```

   Then, in your `MyClass.java` file, you would declare the package at the top:

   ```java
   package com.example.myapp;
   ```

5. **Access Control:** Packages also provide access control. Classes within the same package can access each other's package-private (default) members (fields, methods, and constructors) without any access modifiers. However, classes outside the package can only access public members.

6. **Import Statements:** To use classes from other packages in your code, you need to import them using import statements. For example:

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

   This allows you to use `ArrayList` and `List` from the `java.util` package in your code without fully qualifying their names.

7. **Classpath:** When you compile and run Java code, the Java Virtual Machine (JVM) uses the classpath to locate classes. The classpath includes directories and JAR files where Java classes are stored. You need to ensure that the directory or JAR containing your custom packages is included in the classpath so that the JVM can find them.

In summary, Java packages are a way to organize, structure, and encapsulate your code, making it more manageable and maintainable, especially in large projects. They help prevent naming conflicts, provide access control, and promote code modularity and reusability.