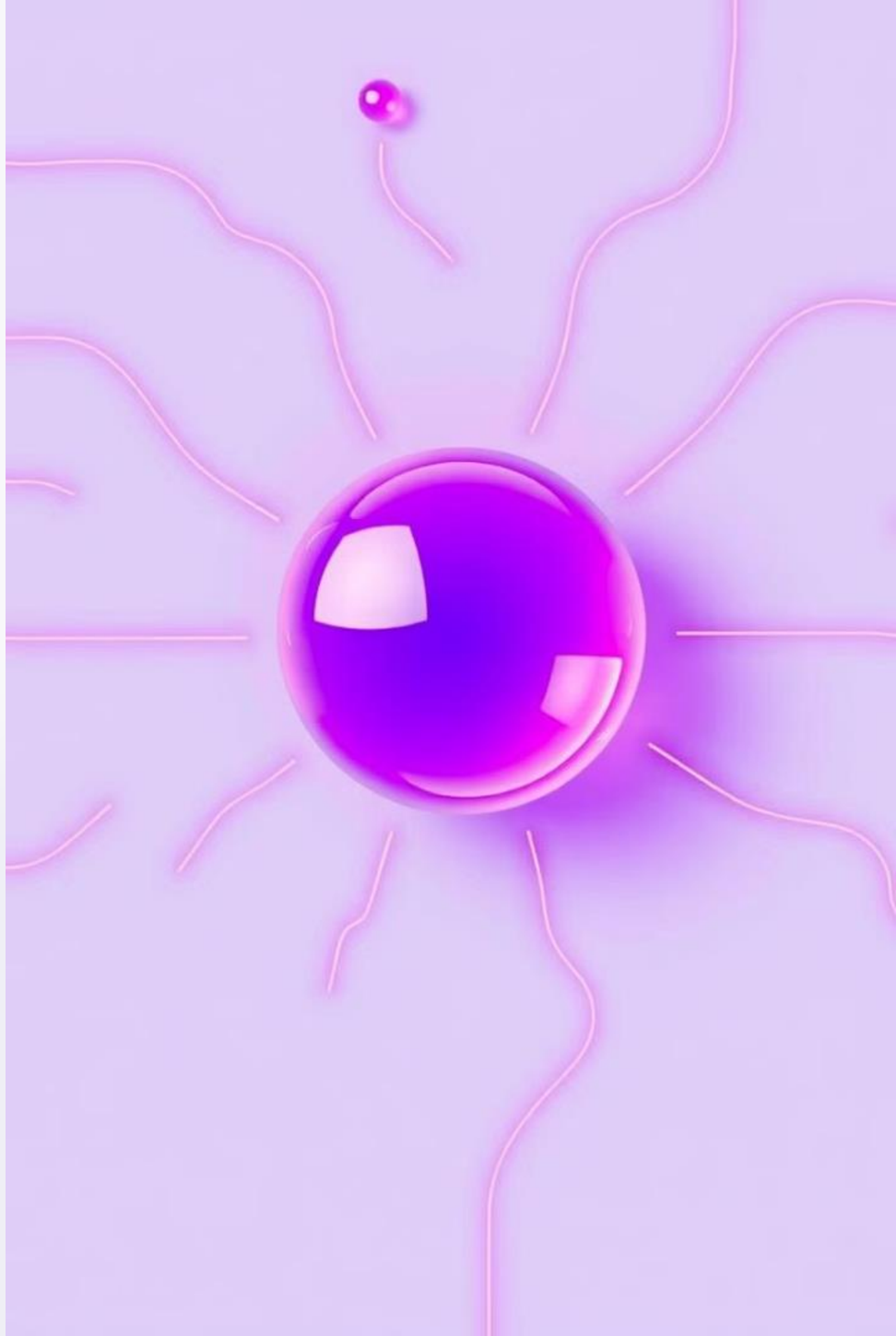
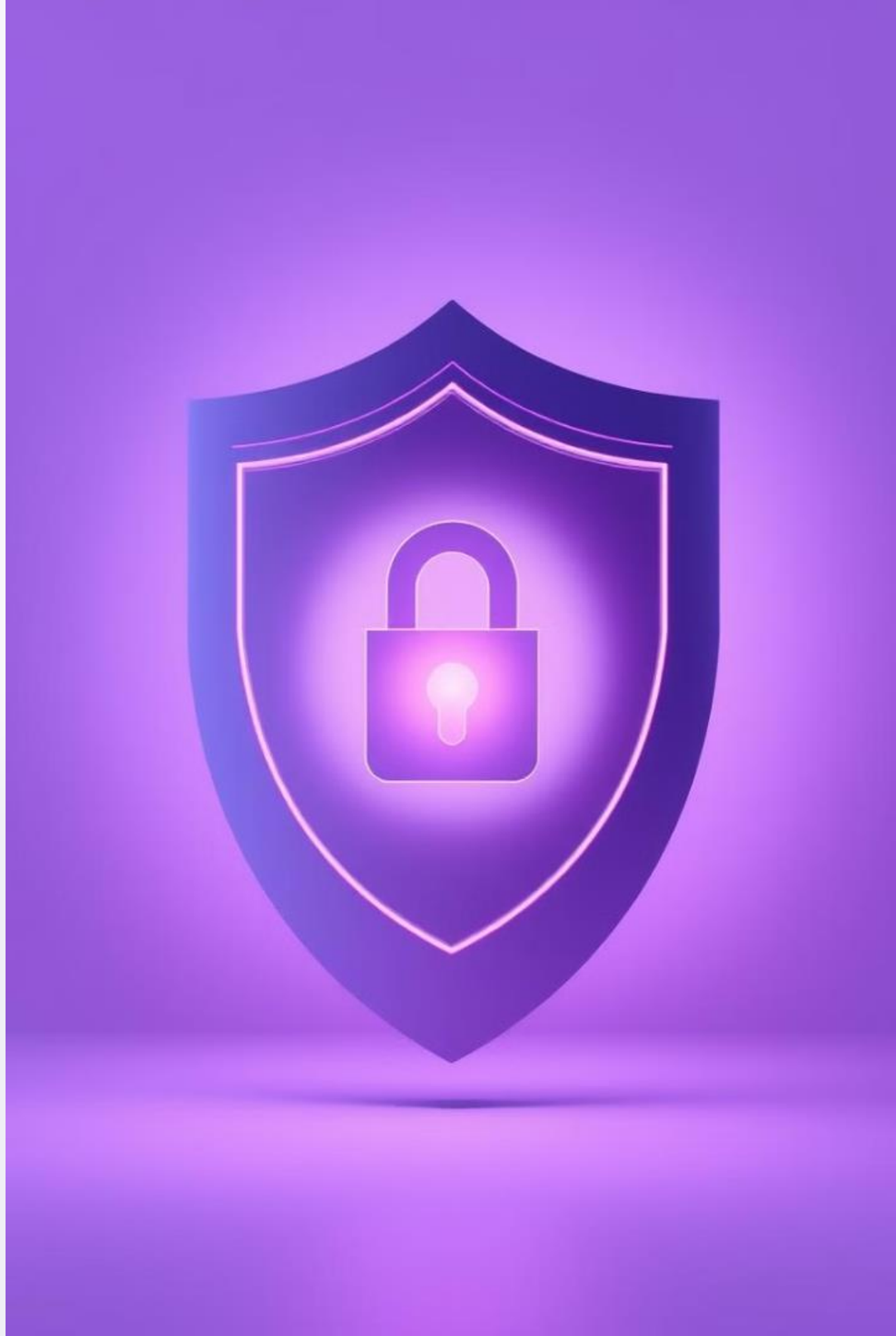
Chapter 2 : Encapsulation

AK by Ali KHALFI



1. Concept of Encapsulation

a. Encapsulation refers to the fact that an object contains its own attributes and methods. The details of an object's implementation are hidden from other objects in the object-oriented system



1.1. Advantages of Encapsulation:

Data Hiding

Encapsulation hides the internal implementation details of an object from the outside world. This means that other objects cannot directly access or modify the object's data. They can only interact with the object through its public methods.

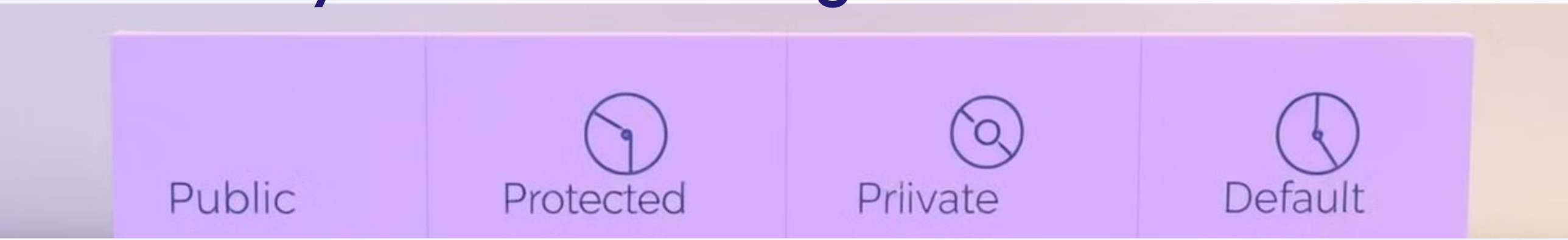
Data Integrity

By controlling access to the data, encapsulation helps to ensure the integrity of the data. The object can enforce rules about how its data can be modified, preventing invalid or inconsistent data from being stored.

Modularity

Encapsulation promotes modularity by creating self-contained objects that can be easily reused in different parts of the program. This makes the code easier to understand, maintain, and debug.

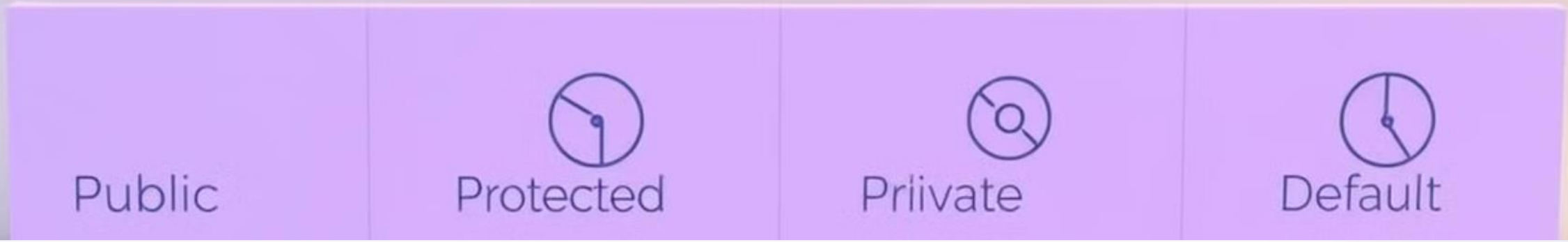
2. Visibility Levels: Controlling Access



a. Access Modifiers for Classes and Interfaces

Modifier	Meaning for a Class or Interface
public	Always accessible
None (no modifier)	Accessible only from classes within the same package

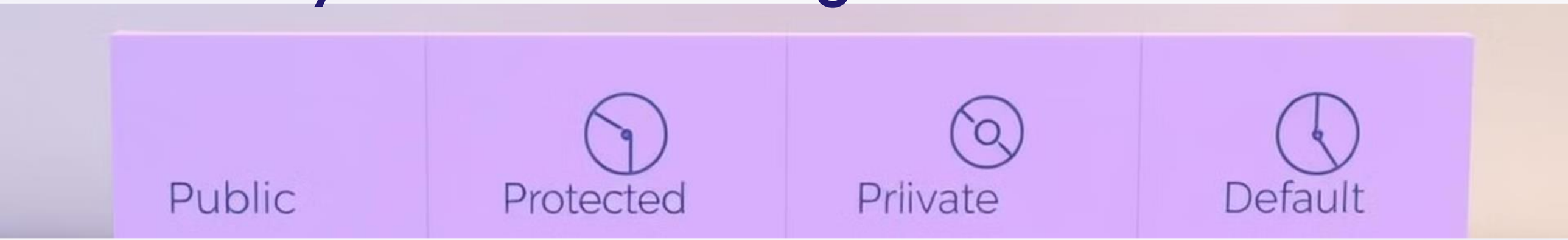
2. Visibility Levels: Controlling Access



b. Access Modifiers for Members and Inner Classes

Modifier	Meaning for Members and Inner Classes
public	Accessible anywhere the class is accessible
None (no modifier)	Accessible from all classes within the same package
protected	Accessible from all classes in the same package or derived classes
private	Access restricted to the class where the member or inner class is declared

2.Visibility Levels: Controlling Access



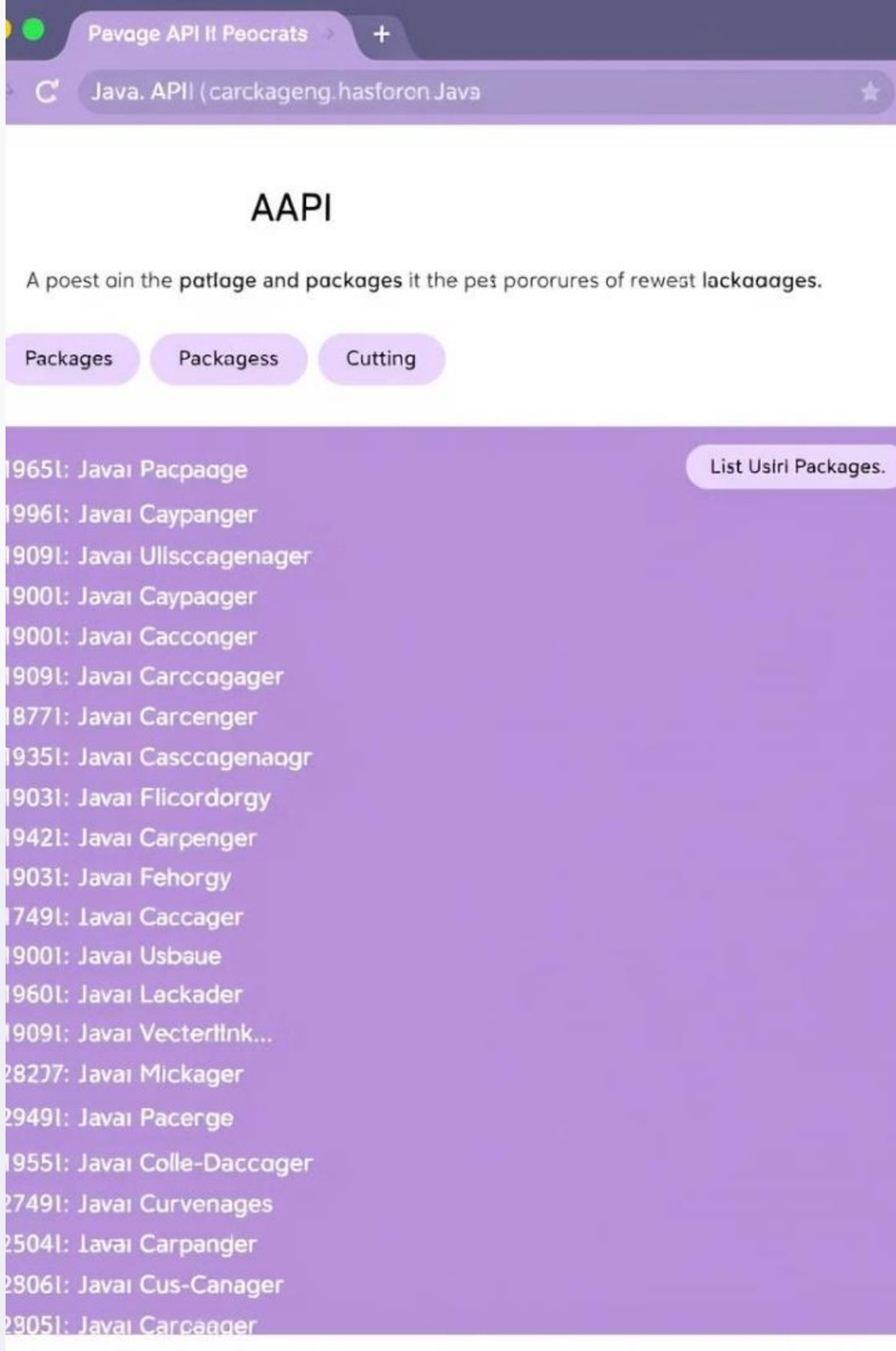
The table below demonstrates the visibility levels **(access rights)** that can be assigned to attributes, methods, and classes:

	public	**protected**	**default**	**private**
in the same class	yes	yes	yes	yes
in a class of the same package	yes	yes	yes	no
in a subclass of another package	yes	yes	no	no
in any class of another package	yes	no	no	no

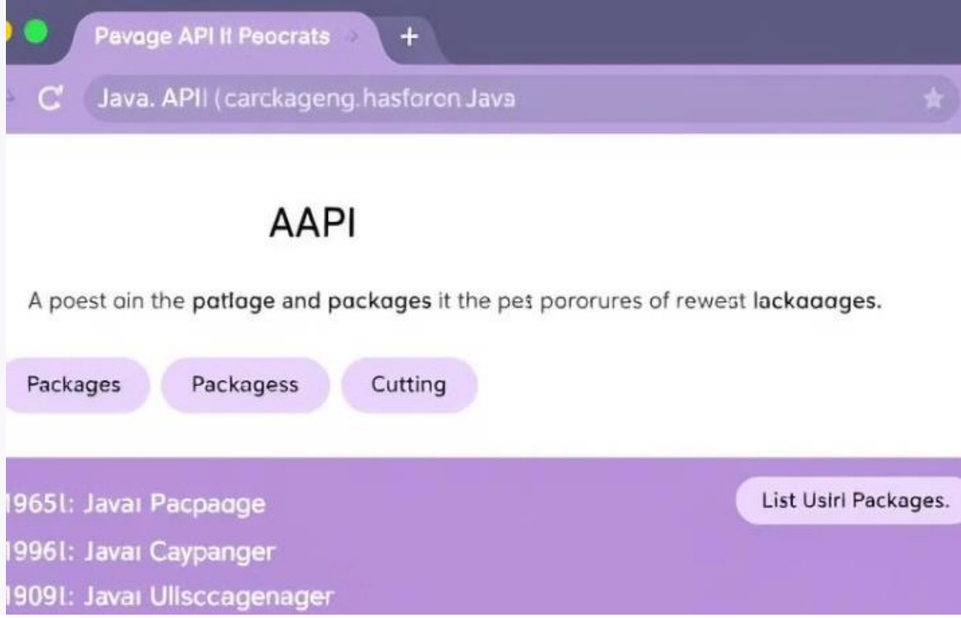
2.1.Packages in Java

A package is a library that groups a large number of classes provided by Java SE. These classes implement generic data and processes that can be used by many applications. These classes constitute the API (Application Programmer Interface) of the Java language. An online documentation for the Java API is available at: http://docs.oracle.com/javase/7/docs/api/

To access a class from a given package, you must first import that class or its package. For example, the **Date** class belonging to the **java.util** package, which implements a set of methods for date manipulation, can be imported in two ways:



2.1.Packages in Java



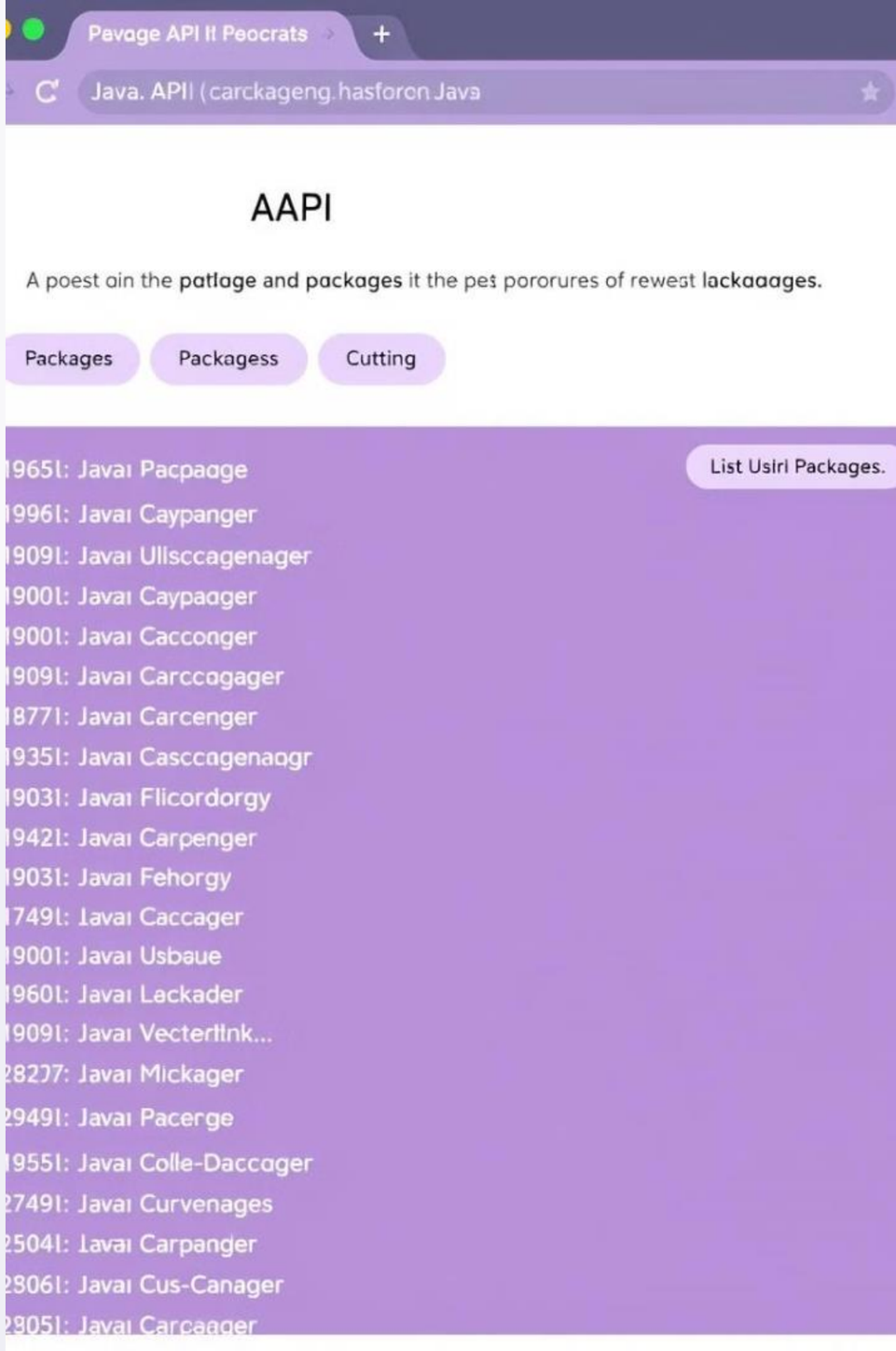
Some of the most commonly used packages include the following:

Package	Description
java.awt	Graphic classes and interface management
java.io	Input/output management
java.lang	Core classes (imported by default)
java.util	Utility classes
javax.swing	Additional graphic classes

2.1.Packages in Java

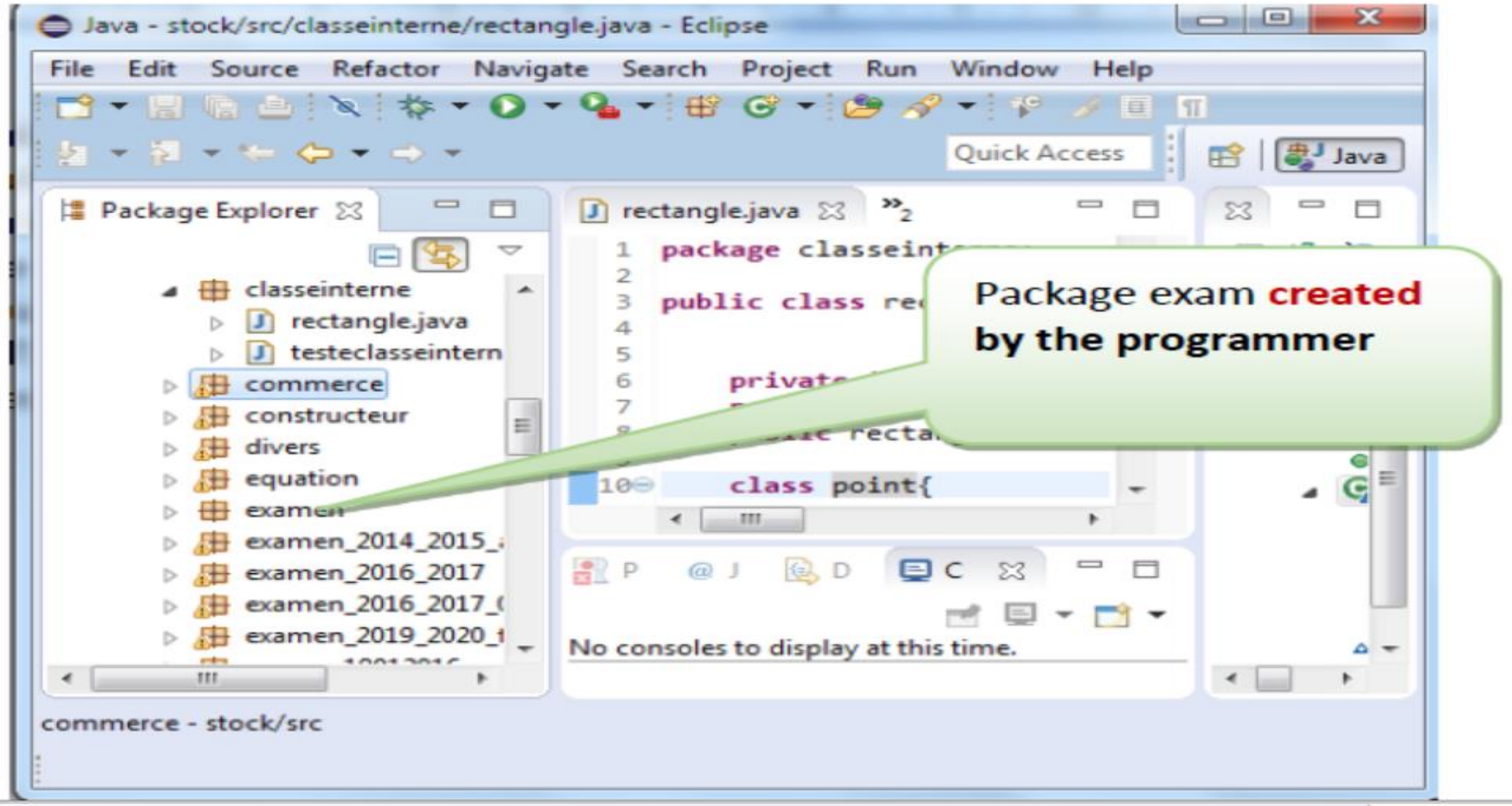
For example, the **Date** class belonging to the **java.util** package, which implements a set of methods for date manipulation, can be imported in two ways:

- A single class from the package is imported: **import java.util.Date;**
- All classes from the package are imported (including unused ones): **import java.util.*;**



2.1.Packages in Java

The following image shows an example of a package created by a programmer.



3. Encapsulation in Java

a. Access Control

Consider the following program, which consists of two packages, p1 and p2. Package p1 contains three classes, C1, C2, and C3, where C2 is a subclass of C1 (**see inheritance in the next chapter**). Package p2 contains two classes, C4 and C5, where C4 is a subclass of C1.

3. Encapsulation in Java

```
package p1 ;
class C1 {
public int a;
protected int b;
int c;
private int d;}
class C2 extends C1 {}
class C3 {}
```

```
package p2 ;
import C1.p1;
class C4 extends C1 {}
class C5 {}
```

Accessibility of variables by class according to the program

	Variable (a)	Variable (b)	Variable (c)	Variable (d)
the accessibility of C2	oui	oui	oui	non
the accessibility of C3	oui	oui	oui	non
the accessibility of C4	oui	oui	non	non
the accessibility of C5	oui	non	non	non

3. Encapsulation in Java

b.Accessors (get and set):

To ensure attribute security in a class, the private modifier is used to restrict direct access. This necessitates the use of getters and setters to read and modify the values safely.

```
public class Rectangle {
private    int largeur;
private    int longueur;
public Rectangle(int largeur, int longueur) {
this.largeur = largeur;this.longueur = longueur;}
public int getLargeur() {return largeur;}
public void setLargeur(int largeur) {this.largeur = largeur;}
public int getLongueur() {return longueur;}
public void setLongueur(int longueur) {this.longueur = longueur;}}
```

3. Encapsulation in Java

Test class

```
public class Test {
public static void main(String[] args) {
// TODO Auto-generated method stub
Rectangle r= new Rectangle (10,10);
System.out.println(r.getLargeur()); //display 10
r.setLargeur(20);
r.setLongueur(20);
System.out.println(r.getLargeur());// display 20}}
```

3. Encapsulation in Java

c.Access to the instance (this) :

You can use **this** as an object or **this** (parameters) as a method

```
public class Rectangle {
private    int largeur;
private    int longueur;
public Rectangle(int largeur, int longueur) {
this.largeur = largeur;this.longueur = longueur;}//use as an object

public Rectangle(int largeur) {this(largeur,10);} //use as a method
```

d.Static Variables and Methods

1

Instance Variables

- Declared outside any method,
- preserve the state of an object, and
- are accessible and shared by all methods in the class.

2

Static Variables and Methods

- Some variables are shared by all instances of a class (static modifier).
- No need to create an object (instances of the class).

3

Static Method

- Provide functionality for the entire class.
- Methods that are not intended to operate on an individual object of the class. Static methods cannot access instance variables
- Examples: Math.random(), Integer.parseInt(String s), main(String args[]).

The reserved word static (called modifier

- This modifier applies to attributes and methods; when this modifier is used, it is said to be a class attribute or method. **It is important to remember that a method or an attribute to which the static modifier is not applied is called an instance method or attribute.**

- A **static** attribute exists as soon as its class is referenced, **independently of any instantiation.** Regardless of the number of class instantiations (0, 1, or more), a class attribute (i.e., static) exists in one and only one instance. Such an attribute is used somewhat like a global variable in a non-object program.

- A method, to be a class method (i.e., **static**), **must not manipulate, either directly or indirectly, non-static attributes of its class**. As a result, a class method cannot directly use any non-static attribute or method of its class; an error would be detected at compilation. In other words, a method that uses (for reading or writing) instance attributes cannot be **static** and must, therefore, be an instance method.

The reserved word static (called modifier

From outside a class or a derived class (see the next chapter), a class attribute or method can be used preceded by the name of the class:

```
class_name.class_data;
class_name.method_name ;
```

Finally, note that a **static** method **cannot be overridden** (see the next chapter), which means that it is automatically **final**.

The reserved word static (called modifier

```
public class MyClass {
static int counter = 0;}
```

Continuation of Example :

```
MyClass m = new MyClass();
int c1 = m.counter;
int c2 = MyClass.counter;
```

c1 and c2 have the same value.

This type of variable **is useful**, for example, to **count the number of instantiations** of the class that have been made.

The chapter end

Questions ....?