JAVA

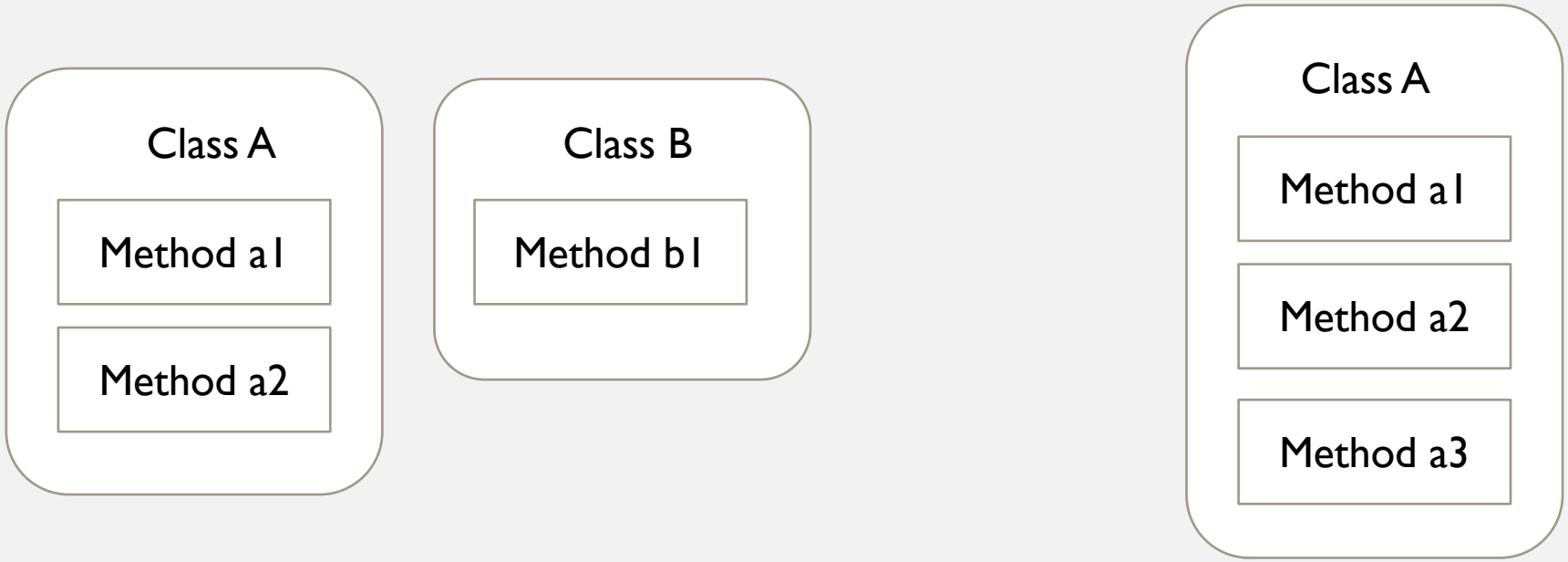
Lecture VII – Packages

PACKAGE FUNDAMENTALS

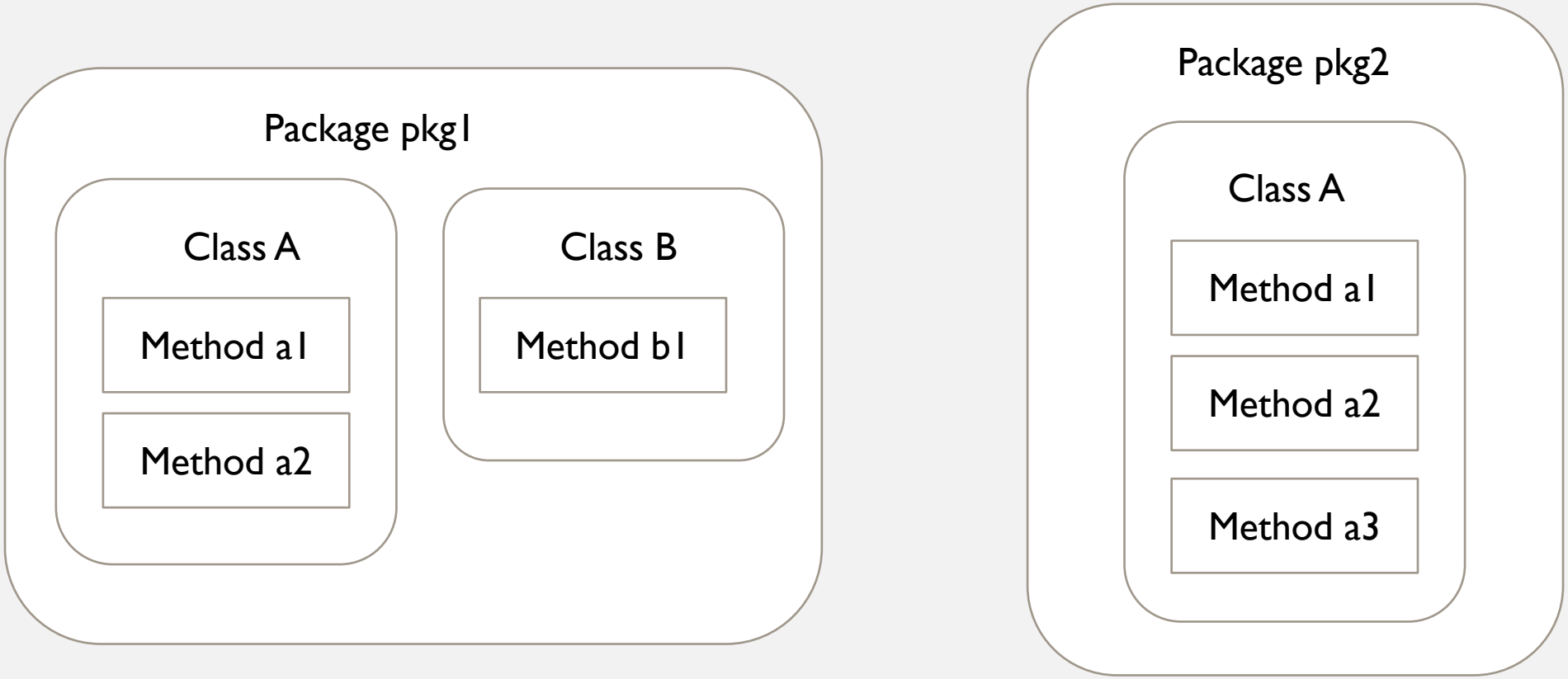
- We have classes with different functionalities and different purposes …
- Packages: a group of related classes and interfaces.
  - Help you organise your code, name a collection of classes
  - Provide another layer of encapsulation, e.g., more access control

- We cannot have two classes of the same name…
- As the programs grow, more and more difficult to find a unique name…
- Solution: package as a namespace.

PACKAGE FUNDAMENTALS



PACKAGE FUNDAMENTALS



DEFINING A PACKAGE

- General from of the package statement:

```
package pkg;        // at the top of a Java file
```

- pkg: package name. By convention, lower case letters are used.
- If no package has been explicitly specified, the default package is used (no name).
- Java uses file system to manage packages, i.e., each package is stored in its own directory.
- Multiple files can be included in the same package.
- A hierarchy of packages:

```
package pack1.pack2.pack3.….packN;
```

- Stored in …/pack1/pack2/pack3/…/packN

EXAMPLE

```
package bookpack;
class Book{
	private String title, author;
	private int pubDate;
	Book(String t, String a, int d){
		title = t;
		author = a;
		pubDate = d;
	}
	void show(){
		System.out.println("title: " + title);
		System.out.println("author: " + author);
		System.out.println("pubDate: " + pubDate);
	}
}
```

EXAMPLE

```
package bookpack;
class BookDemo{
	public static void main(String[] args){
		Book book = new Book(“CS”, “Me”, “2022”);
		book.show();
	}
}
```

CLASSPATH

- How does the Java run-time system know where to look for packages?
  - By default, it uses the current working directory as the starting point.
  - You can specify a directory path by setting the CLASSPATH environment variable.
  - Or you can use the **–classpath** option with **java** and **javac** command to specify the path.
  - Given a package `mypack`, Java can only find this package if:

CLASSPATH

- How does the Java run-time system know where to look for packages?
  - By default, it uses the current working directory as the starting point.
  - You can specify a directory path by setting the CLASSPATH environment variable.
  - Or you can use the **–classpath** option with **java** and **javac** command to specify the path.
  - Given a package `mypack`, Java can only find this package if:
    - mypack is a sub-directory of the current working directory.
    - the classpath is set to include the path to mypack
    - The –classpath option must specify the path to mypack when the program is run

EXAMPLE

```
package bookpack;
class BookDemo{
      public static void main(String[] args){
            Book book = new Book("CS", "Me", "2022");
            book.show();
      }
}
```

Compile the file:

```
javac bookpack/BookDemo.java //make sure you are in the
                                directory above bookpack
```

Execute the file

```
java bookpack.BookDemo          // cannot use java BookDemo
```

COMPILE AND EXECUTE

```
javac bookpack/BookDemo.java
```

It compiles the file `BookDemo.java` in the directory `bookpack`.

`BookDemo.class` will be generated in the same directory.

`Book.class` will be automatically generated as before.

```
javac -d .  BookDemo.java Book.java
```

It creates a new directory (with package name) and stores all class file in the new directory

Execute the file

```
java bookpack.BookDemo          // cannot use java BookDemo
```

ACCESS MODIFIERS

Public: visible to all other classes.

Protected: visible to all subclasses, and all classed in the same package.

Default: visible to all classes within the same package.

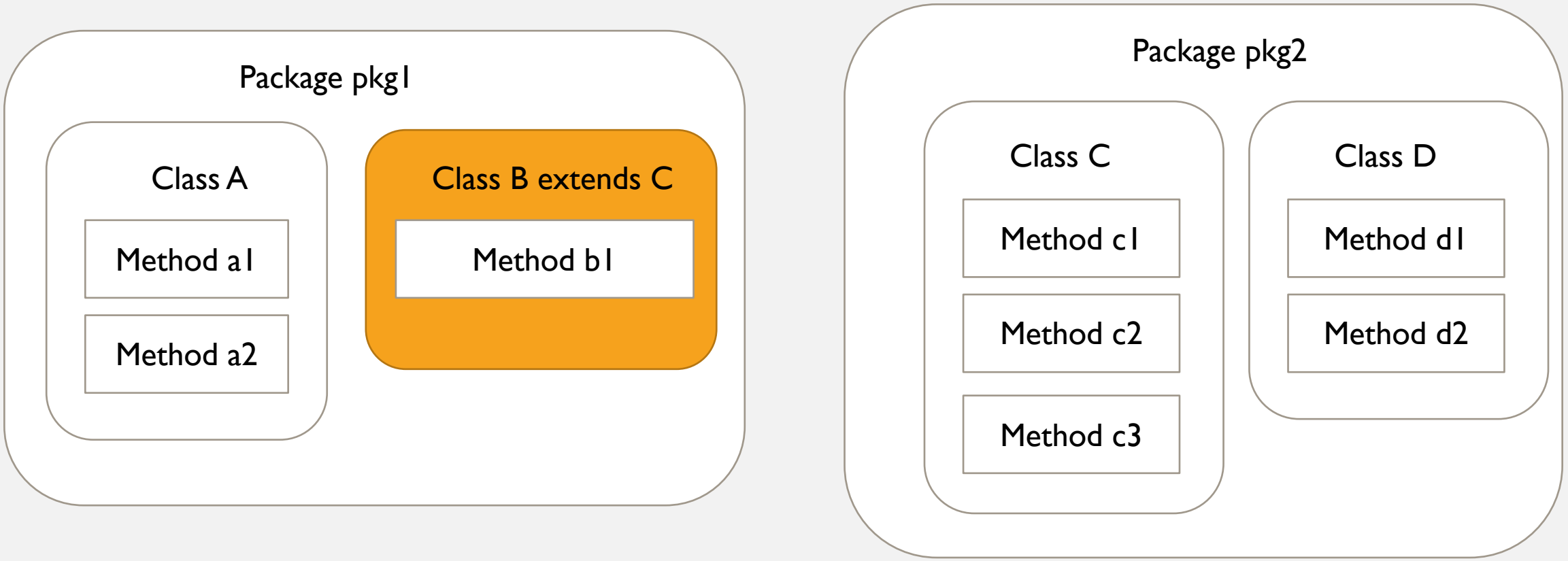
Private: only visible to the current class.

ACCESS MODIFIERS

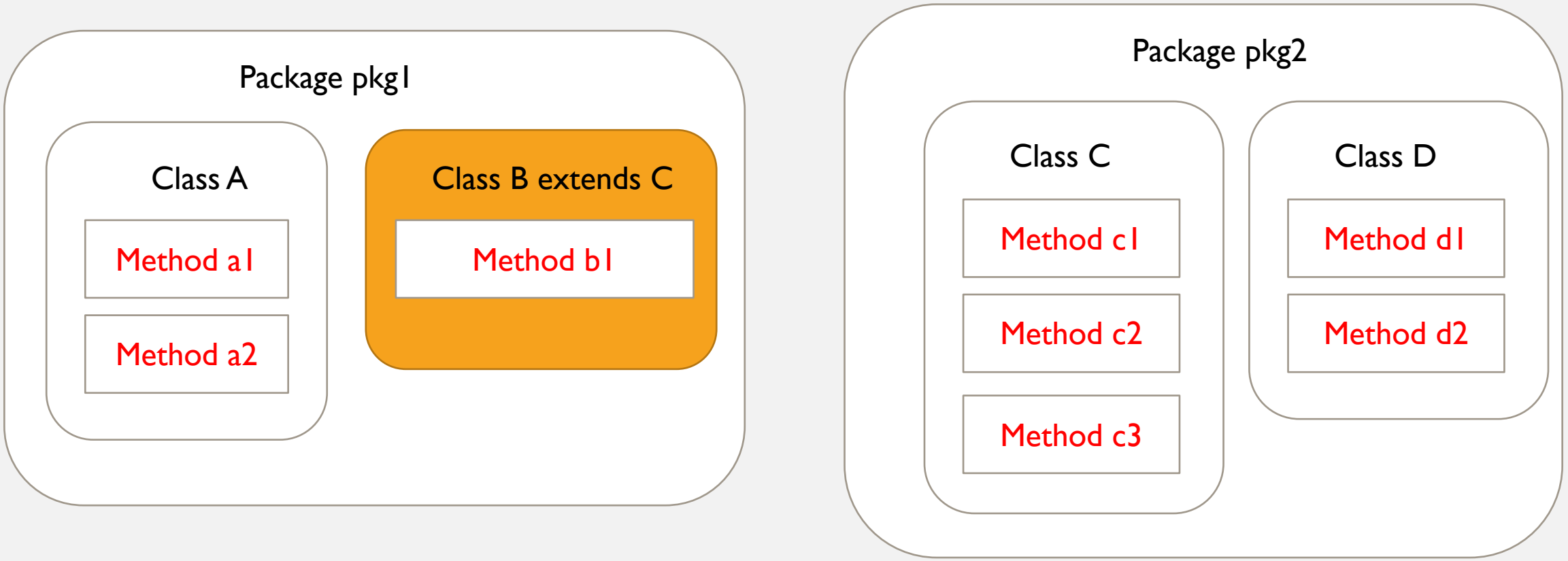
public, private, default, protected

	private	default	protected	public
Within the same class	Yes	Yes	Yes	Yes
Within the same package by subclass	No	Yes	Yes	Yes
Within the same package by non-subclass	No	Yes	Yes	Yes
Different packages by subclass	No	No	Yes	Yes
Different packages by non-subclass	No	No	No	Yes

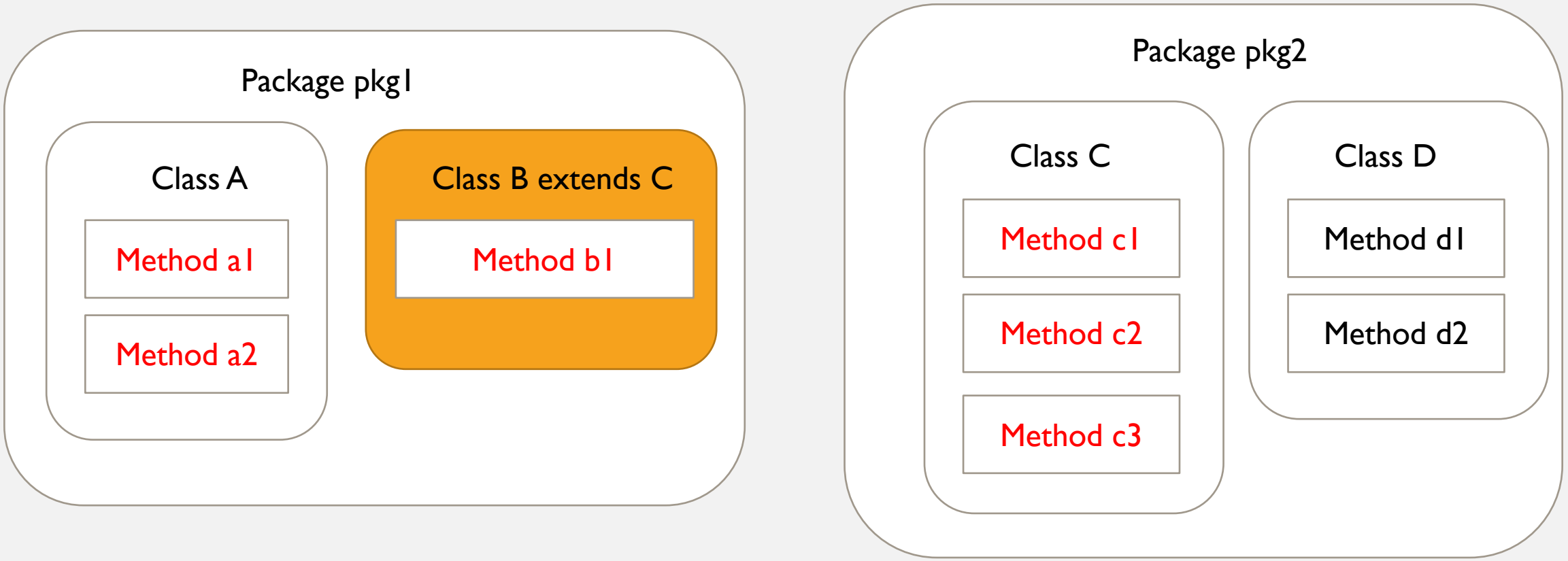
EXAMPLE


Package pkg1
Class A
Method a1
Method a2
Class B extends C
Method b1
Package pkg2
Class C
Method c1
Method c2
Method c3
Class D
Method d1
Method d2

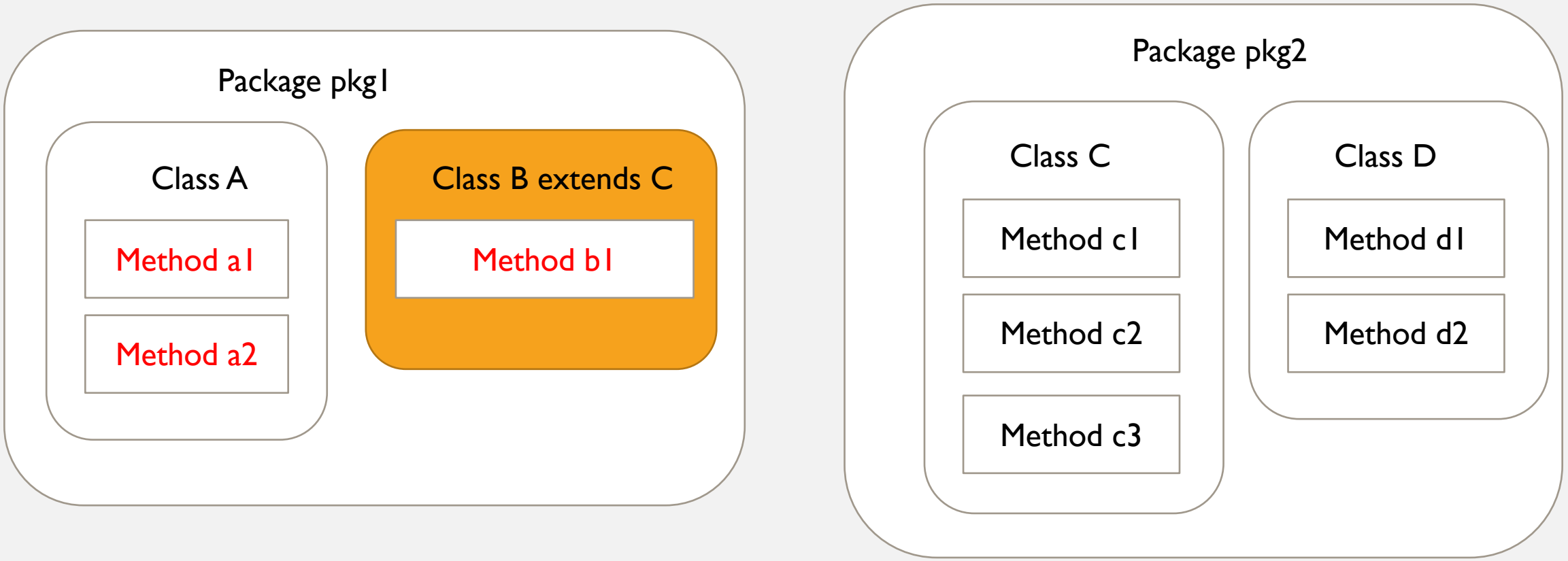
PUBLIC


Package pkg1
Class A
Method a1
Method a2
Class B extends C
Method b1
Package pkg2
Class C
Method c1
Method c2
Method c3
Class D
Method d1
Method d2

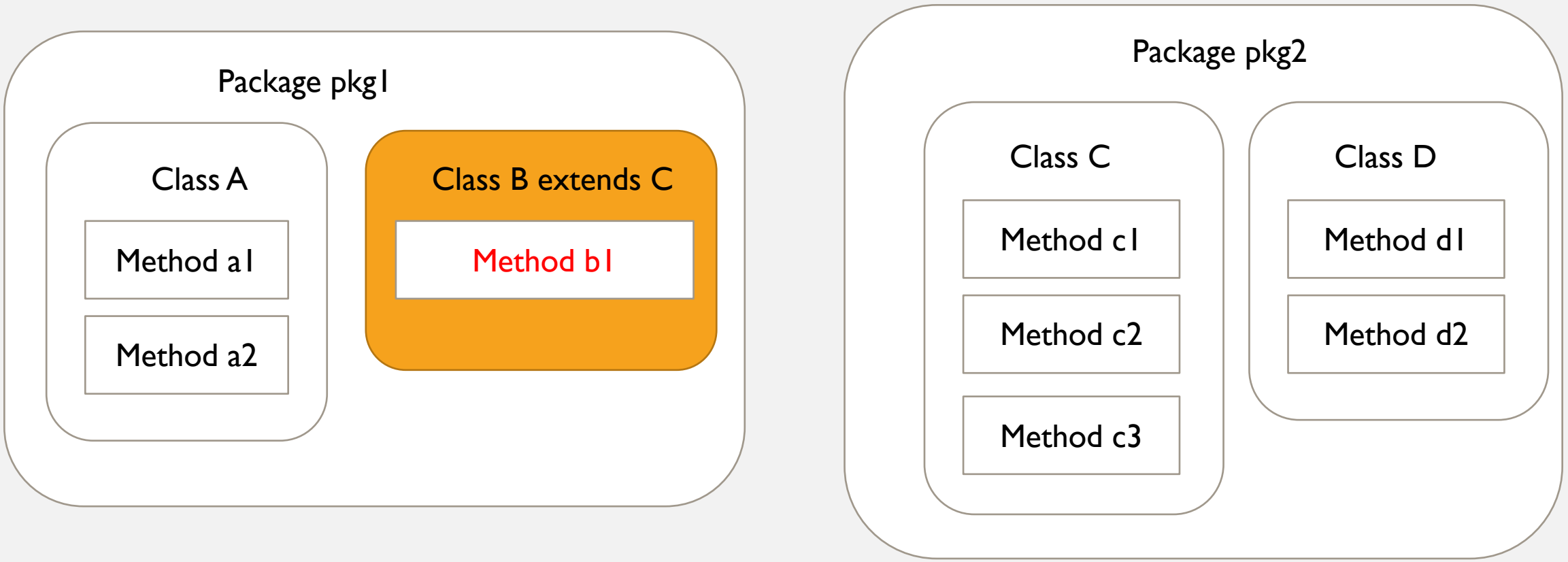
PROTECTED


Package pkg1
Class A
Method a1
Method a2
Class B extends C
Method b1
Package pkg2
Class C
Method c1
Method c2
Method c3
Class D
Method d1
Method d2

DEFAULT


Package pkg1
Class A
Method a1
Method a2
Class B extends C
Method b1
Package pkg2
Class C
Method c1
Method c2
Method c3
Class D
Method d1
Method d2

PRIVATE


Package pkg1
Class A
Method a1
Method a2
Class B extends C
Method b1
Package pkg2
Class C
Method c1
Method c2
Method c3
Class D
Method d1
Method d2

A PACKAGE ACCESS EXAMPLE

- What happens if

```
package mypack;
class BookDemo{
	public static void main(String[] args){
		Book book = new Book("CS", "Me", 2022);
		book.show();
	}
}
```

- What are required ?

A PACKAGE ACCESS EXAMPLE

- What happens if

```
package mypack;
class BookDemo{
      public static void main(String[] args){
            Book book = new Book("CS", "Me", 2022);
            book.show();
      }
}
```

- What are required ?
  - Class Book must be declared as public.
  - Its constructor must be public as well.
  - Method show also needs to be public.

A PACKAGE ACCESS EXAMPLE

- Is that enough?

```
package mypack;
class BookDemo{
	public static void main(String[] args){
		Book book = new Book("CS", "Me", 2022);
		book.show();
	}
}
```

A PACKAGE ACCESS EXAMPLE

- Is that enough?

```
package mypack;
class BookDemo{
      public static void main(String[] args){
            bookpack.Book book =
                  new bookpack.Book("CS", "Me", 2022);
            book.show();
      }
}
```

- Better to use javac –d to create two different directories…
- Remember how Java is finding the packages…

PROTECTED EXAMPLE

```
package bookpack2;
class ExtBook extends bookpack.Book{
	private String condition;
	public ExtBook(String t, String a, int d, String c){
		super(t, a, d);
		condition = c;
	}
	public void show(){
		super.show();
		System.out.println("Condition: " + condition);
	}
	public String getTitle(){
		return title;
	}
}
```

PROTECTED EXAMPLE

```
package bookpack2;
class BookDemo{
	public static void main(String[] args){
		ExtBook book = new ExtBook(“CS”, “Me”, 2022,”T”);
		System.out.println(book.getTitle());
		book.title = “BS”; // will it work?
	}
}
```

IMPORTING PACKAGES

- In previous examples, we need to fully qualify the name of the class in different packages…

```
bookpack.Book
```

- **import** statement: bring together classes in different packages.

```
import pkg.classname;
```

- Import entire package:

```
import pkg.*;
```

- No longer need to qualify the whole package name, i.e., use Book rather than bookpack.Book
- The import statement goes after package statement and before class definition.

IMPORTING PACKAGES

- **import** statement can also be used to import Java standard packages. (Java API)
- Most of them start with **java.**

Sub-package	Description
java.lang	Contains a large number of general-purpose classes
java.io	Contains the I/O classes
java.net	Contains those classes that support networking
java.applet	Contains classes for creating applets
java.awt	Contains classes that support the Abstract Window Toolkit
java.util	Contains various utility classes and the Collections Framework

- The java.lang package is automatically imported.

STATIC IMPORT

- **import** followed by **static** allows you to import static member of a class.
- Instead of calling through their class name, we could call them directly.

```
import static java.lang.System.*;
class A{
      public static void main(String[] args){
            out.println(“Static import”);
      }
}
```