7

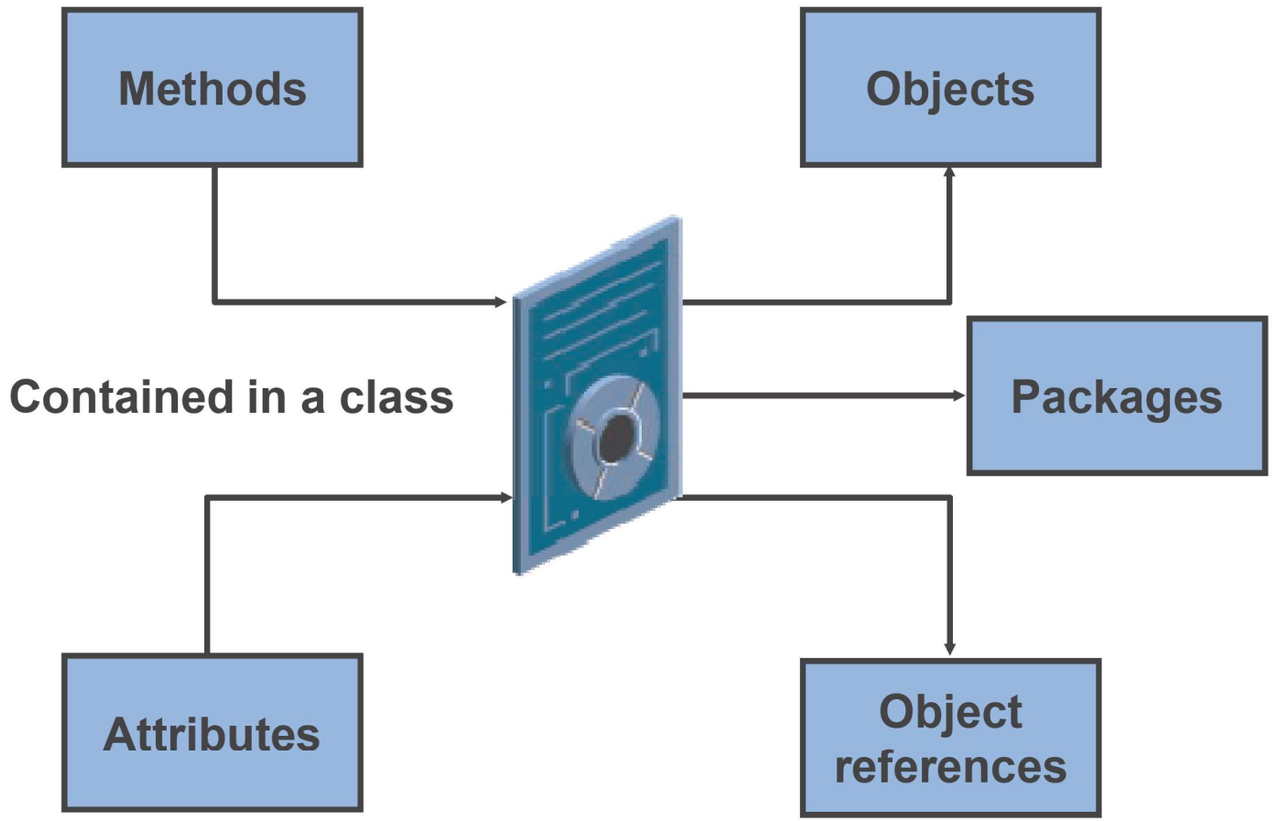
Creating Classes and Objects

Objectives

After completing this lesson, you should be able to do the following:

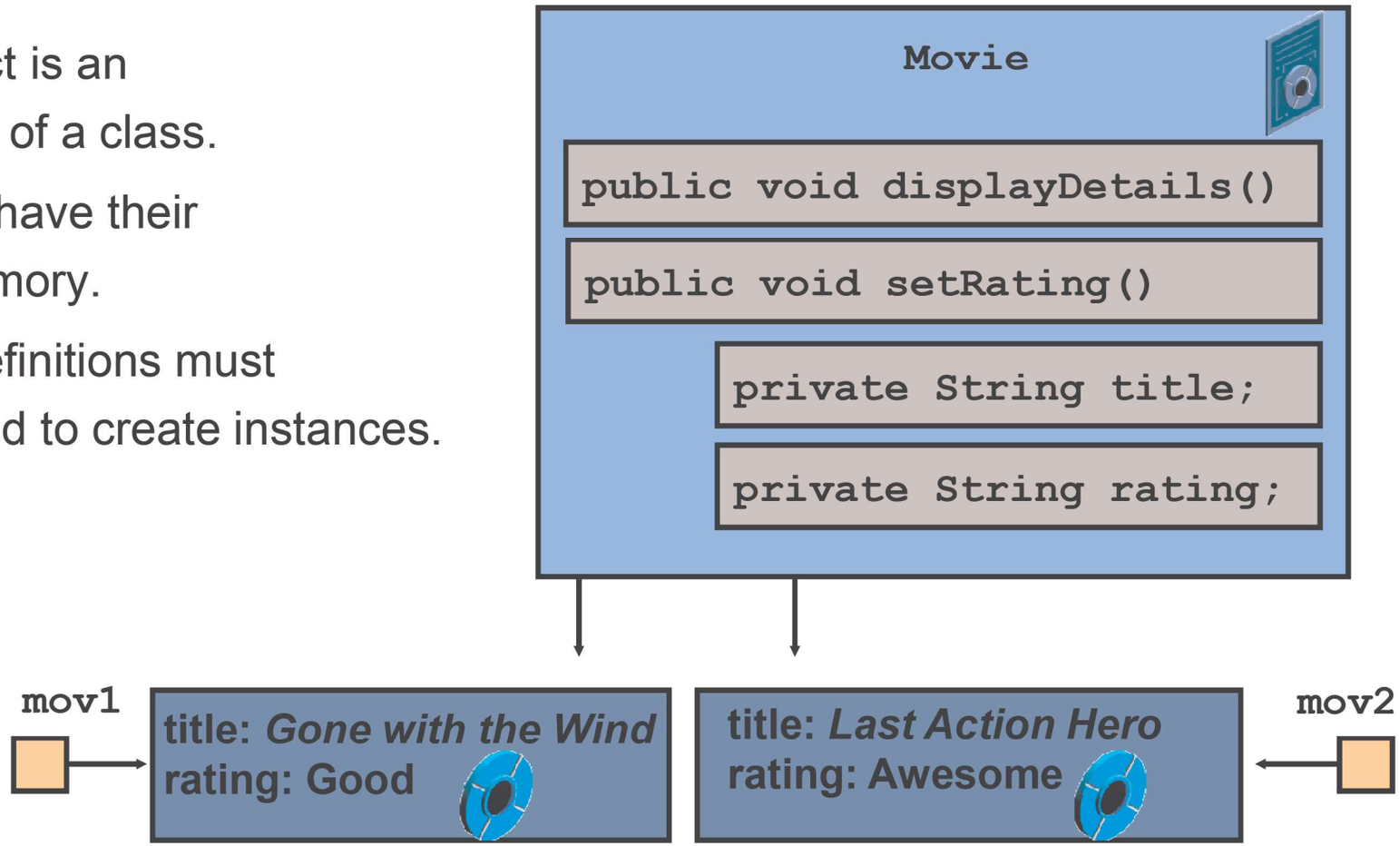
- Describe how object-oriented principles underpin the Java language
- Create objects
- Define instance variables and methods
- Define the `no-arg` (default) constructor method
- Instantiate classes and call instance methods
- Perform encapsulation by using packages to group related classes

Java Classes



Comparing Classes and Objects

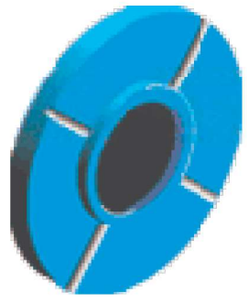
- An object is an instance of a class.
- Objects have their own memory.
- Class definitions must be loaded to create instances.



Creating Objects

- Objects are created by using the `new` operator:

```
ClassName objectRef = new ClassName();
```



- For example, to create two `Movie` objects:

```
Movie mov1 = new Movie("Gone ...");
Movie mov2 = new Movie("Last ...");
```

title: *Gone with the Wind*
rating: Good

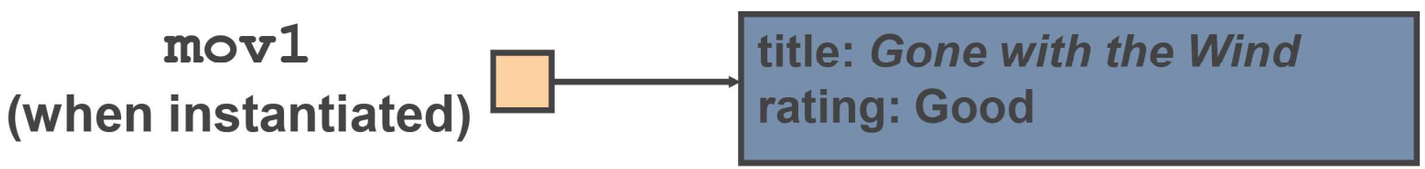
title: *Last Action Hero*
rating: Awesome

new Operator

The new operator performs the following actions:

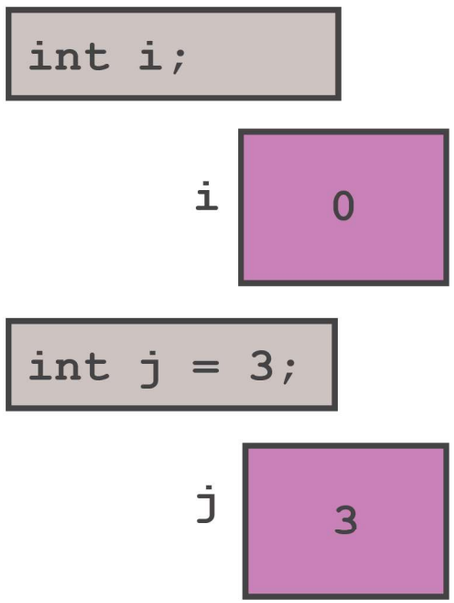
- Allocates and initializes memory for the new object
- Calls a special initialization method in the class (this method is called a *constructor*)
- Returns a reference to the new object

```
Movie mov1 = new Movie("Gone with…");
```

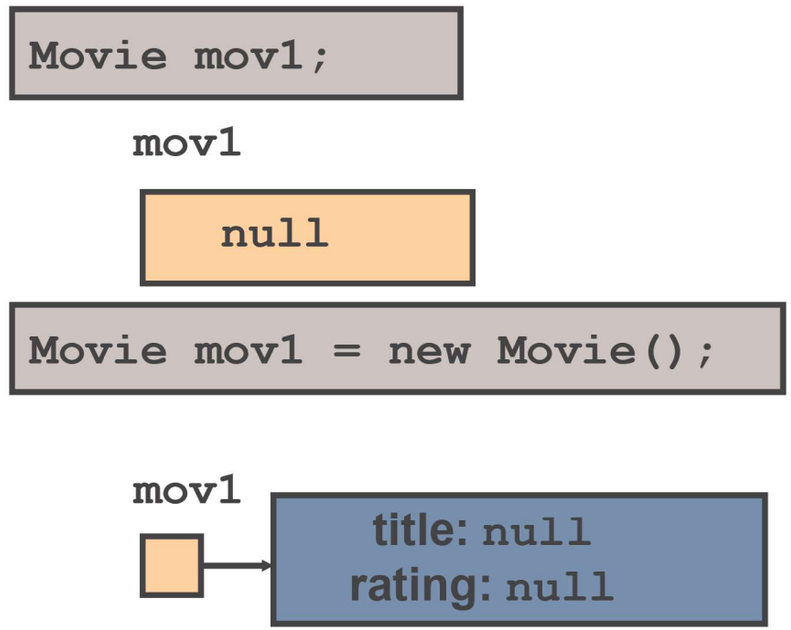


Primitive Variables and Object Variables

Primitive variables hold a value.



Object variables hold references.



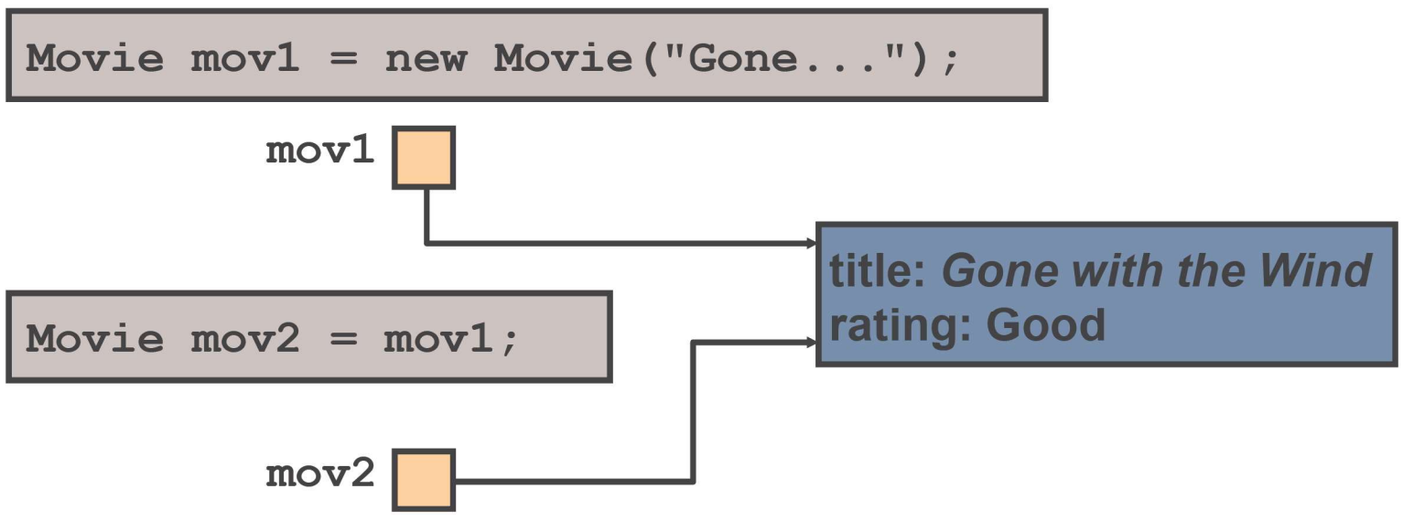
`null` Reference

- A special `null` value can be assigned to an object reference but not to a primitive.
- You can compare object references to `null`.
- You can remove the association to an object by setting the object reference to `null`.

```
Movie mov1;                 //Declare object reference
…
if (mov1 == null)           //Ref not initialized?
 mov1 = new Movie();        //Create a Movie object
…
mov1 = null;                //Forget the Movie object
```

Assigning References

Assigning one reference to another results in two references to the same object:



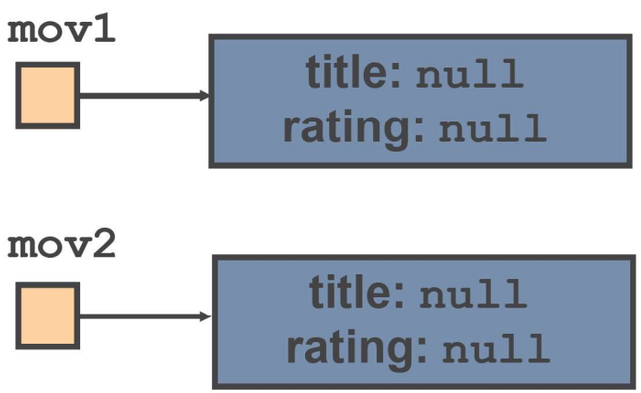
Declaring Instance Variables

Instance variables are declared within the class but outside the methods, instance, or static intializers.

```
public class Movie {
   public String title;
   public String rating;
   public float getPrice(){
       return price;
   }
}
```

Create `movies`:

```
Movie mov1 = new Movie();
Movie mov2 = new Movie();
```



Accessing `public` Instance Variables

`public` instance variables can be accessed by using the `dot` operator:

```
public class Movie {
  public String title;
  public String rating;
  …
}
```

```
Movie mov1 = new Movie();
mov1.title = "Gone ...";
…
if (mov1.title.equals("Gone ... ") )
    mov1.rating = "Good";
```

Defining Methods

A method in Java is equivalent to a function or subroutine in other languages.

```
modifier returnType methodName (argumentList) {
    // method body
    …
}
```

Calling a Method

Objects communicate by using messages:

- All methods are defined within a class and are not defined globally as in traditional languages.
- When you call a method, it is always in the context of a particular object.
  - `myPen.write( )`: Object-oriented programming
  - `Write (myPen)`: Traditional structured programming

Specifying Method Arguments: Examples

- Specify the number and type of arguments in the method definition:

```
public void setRating(String newRating) {
    rating = newRating;
  }
```

- If the method takes no arguments, leave the parentheses empty:

```
public void displayDetails() {
  System.out.println("Title is  " + title);
  System.out.println("Rating is " + rating);
}
```

Returning a Value from a Method

- Use a return statement to exit a method and to return a value from a method:

```
public class Movie {
  private String rating;
  …
  public String getRating () {
    return rating;
  }
}
```

- If the return type is void, no return is needed.
- You can use a return without a value to terminate a method with a void return type.

Calling Instance Methods

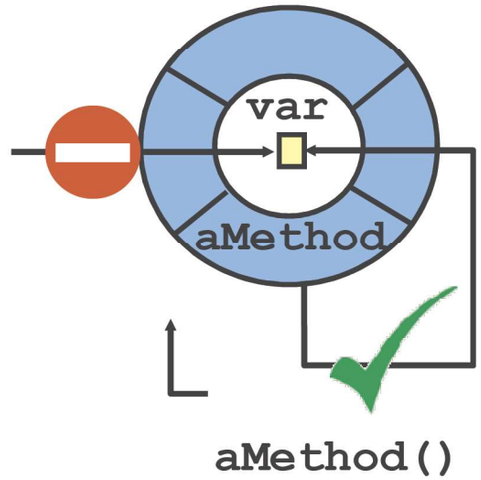
```
public class Movie {
  private String title, rating;
  public  String getRating(){
    return rating;
  }
  public void setRating(String newRating){
    rating = newRating;
  }
}
```

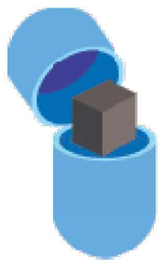
```
Movie mov1 = new Movie();
String r = mov1.getRating();
if (r.equals("Good")) …
```

Use the `dot` operator:

Encapsulation in Java

- Instance variables should be declared as `private`.
- Only instance methods can access `private` instance variables.
- `private` decouples the interface of the class from its internal operation.

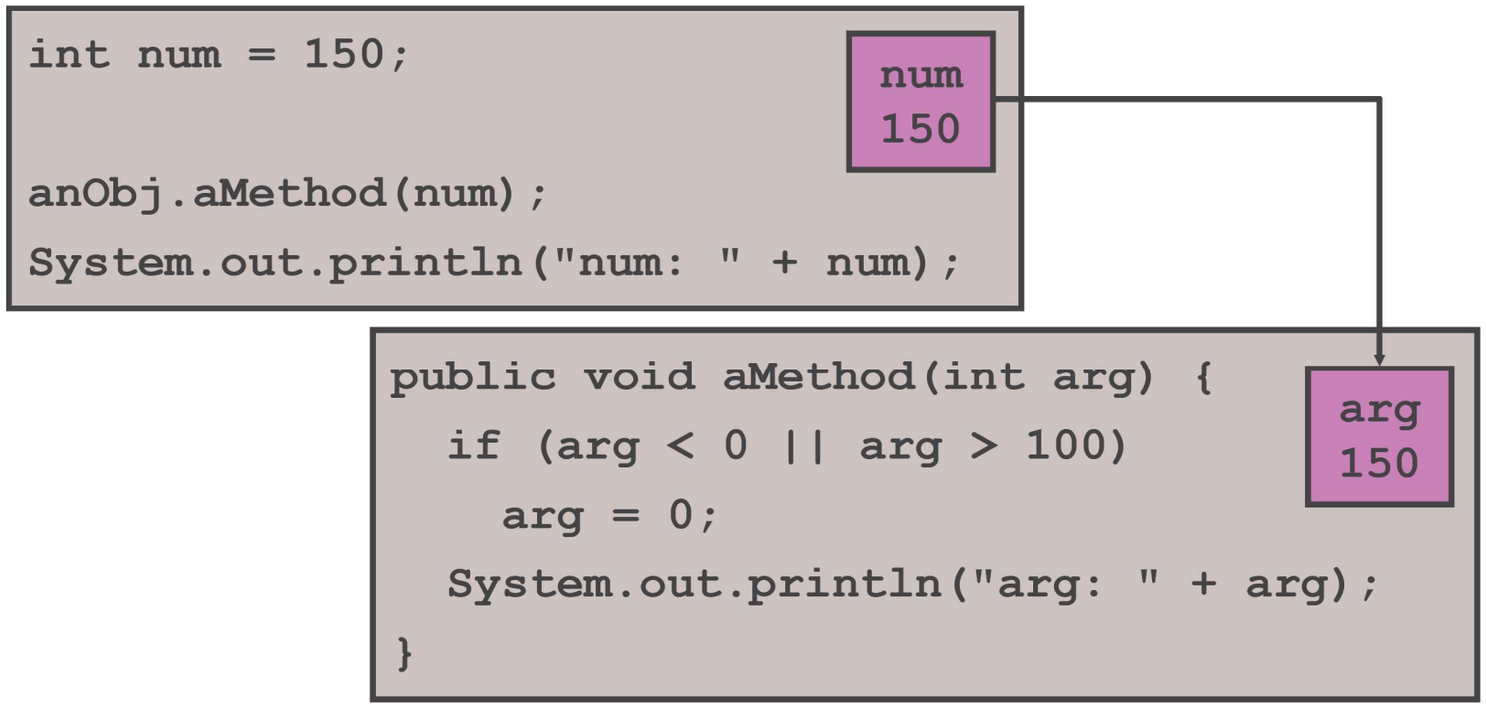




```
Movie mov1 = new Movie();
String rating = mov1.getRating();
String r = mov1.rating;  // error: private
  ...
if (rating.equals("Good"))
```

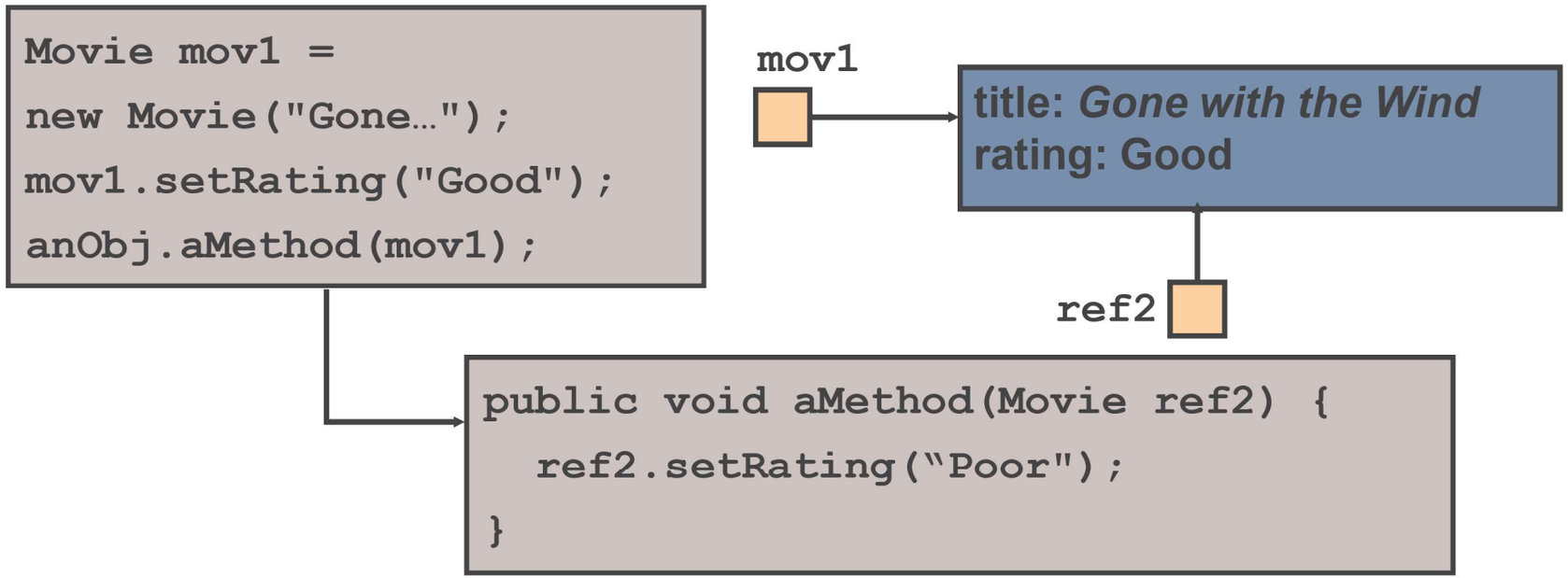
Passing Primitives to Methods

When a primitive or object reference value is passed to a method, a copy of the value is generated:

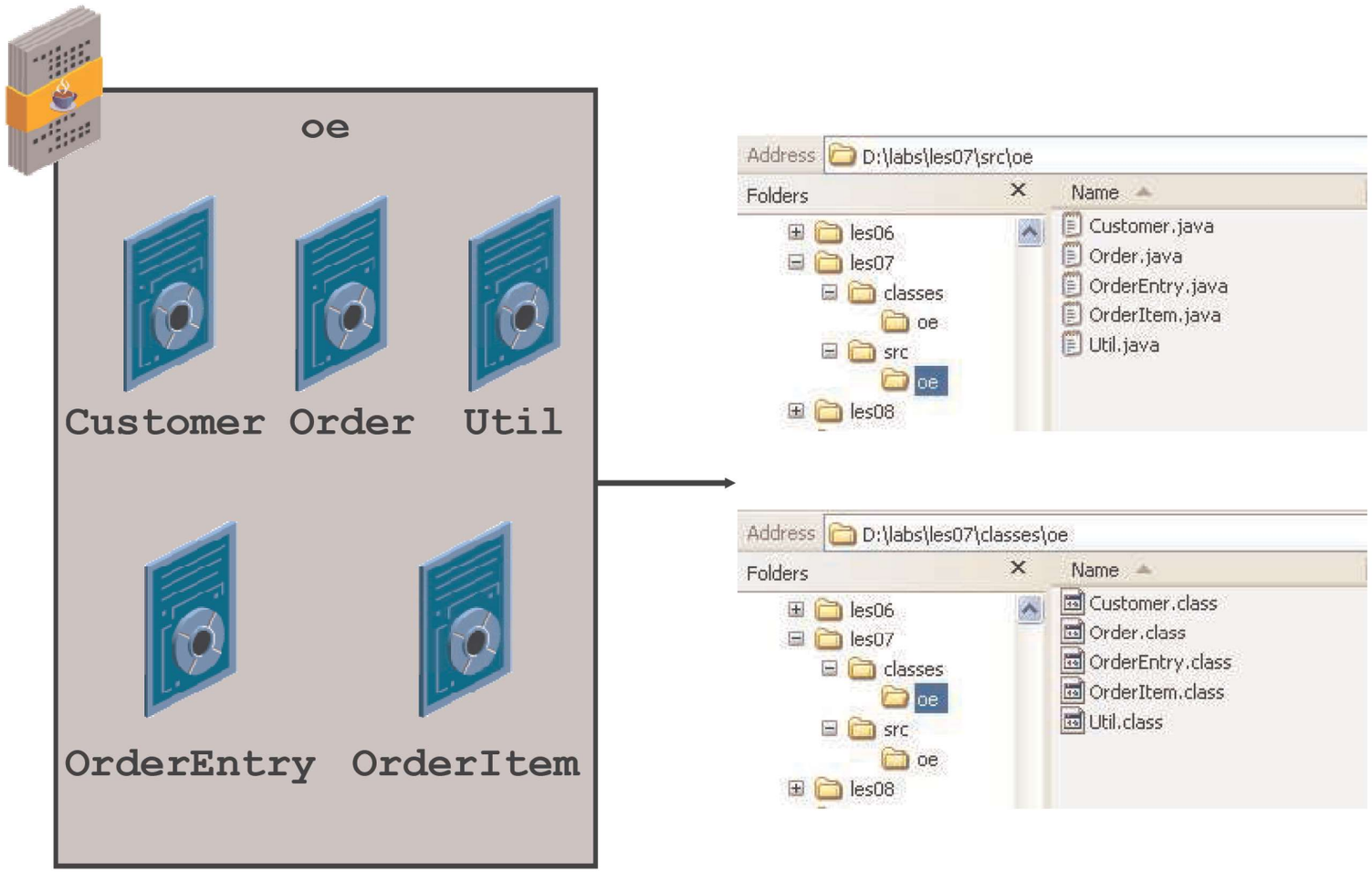


Passing Object References to Methods

When an object reference is passed to a method, the object is not copied but the pointer to the object is copied:



Java Packages



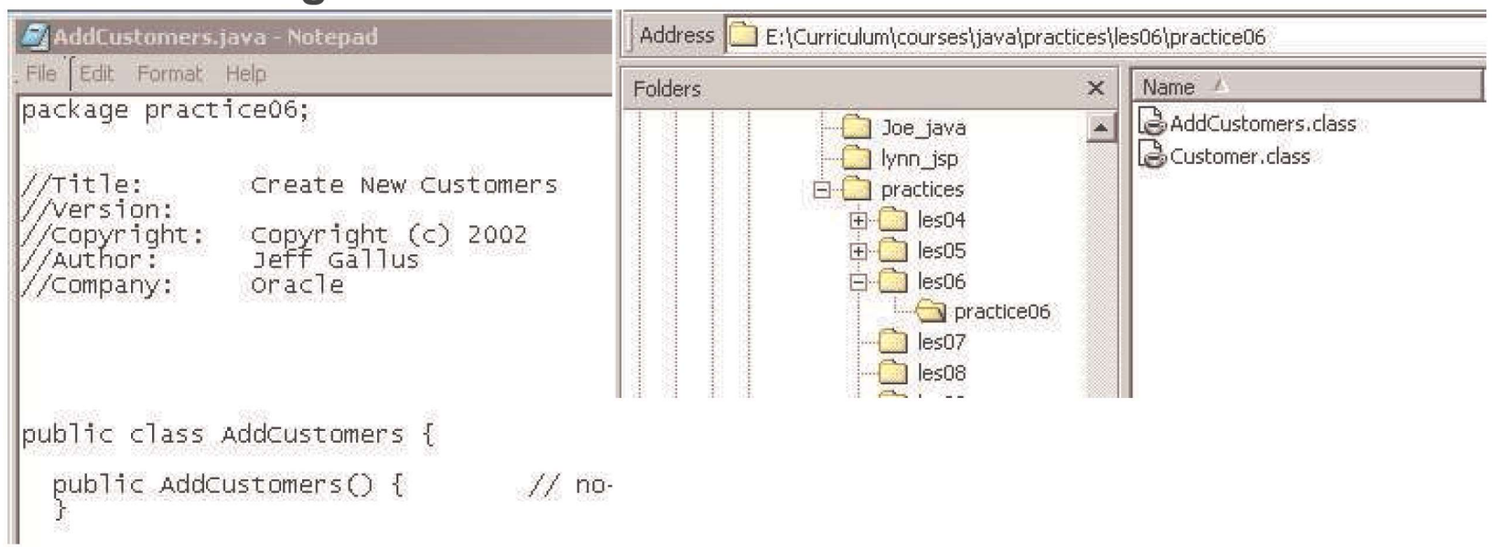
Grouping Classes in a Package

- Include the `package` keyword followed by the package name at the top of the Java source file. Use the dot notation to show the package path.
- If you omit the `package` keyword, the compiler places the class in a default "unnamed" package.
- Use the `-d` flag with the `javac` compiler to create the package tree structure relative to the specified directory.
- Running a `main()` method in a packaged class requires that:
  - The `CLASSPATH` contain the directory having the root name of the package tree
  - The class name be qualified by its package name

Setting the CLASSPATH with Packages

The CLASSPATH includes the directory containing the top level of the package tree:

Package name **.class location**



CLASSPATH

```
C:\>set CLASSPATH=E:\Curriculum\courses\java\practices\les06
```

Access Modifiers

Accessible to:	Public	Protected	Default (absent)	Private
Same class	Y	Y	Y	Y
Any class in same package	Y	Y	Y	N
Subclass in different package	Y	Y	N	N
Non-subclass in different package	Y	N	N	N

Summary

In this lesson, you should have learned how to:

- Describe how object-oriented principles underpin the Java language
- Create objects
- Define instance variables and methods
- Define the `no-arg` (default) constructor method
- Instantiate classes and call instance methods
- Perform encapsulation by using packages to group related classes

Practice : Overview

This practice covers the following topics:

- Defining new classes
- Specifying the classes' instance variables and instance methods
- Creating `Customer` objects in `main()`
- Manipulating `Customer` objects by using public instance methods