13

Structuring Code Using Abstract Classes and Interfaces

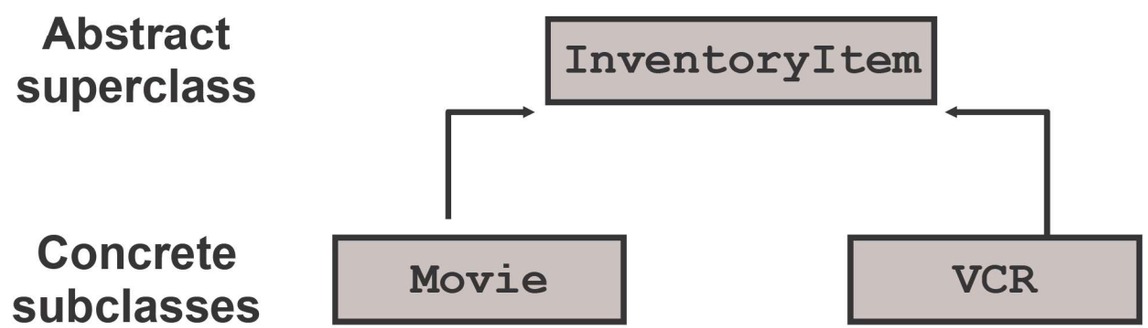
Objectives

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

- Define abstract classes
- Define abstract methods
- Define interfaces
- Implement interfaces

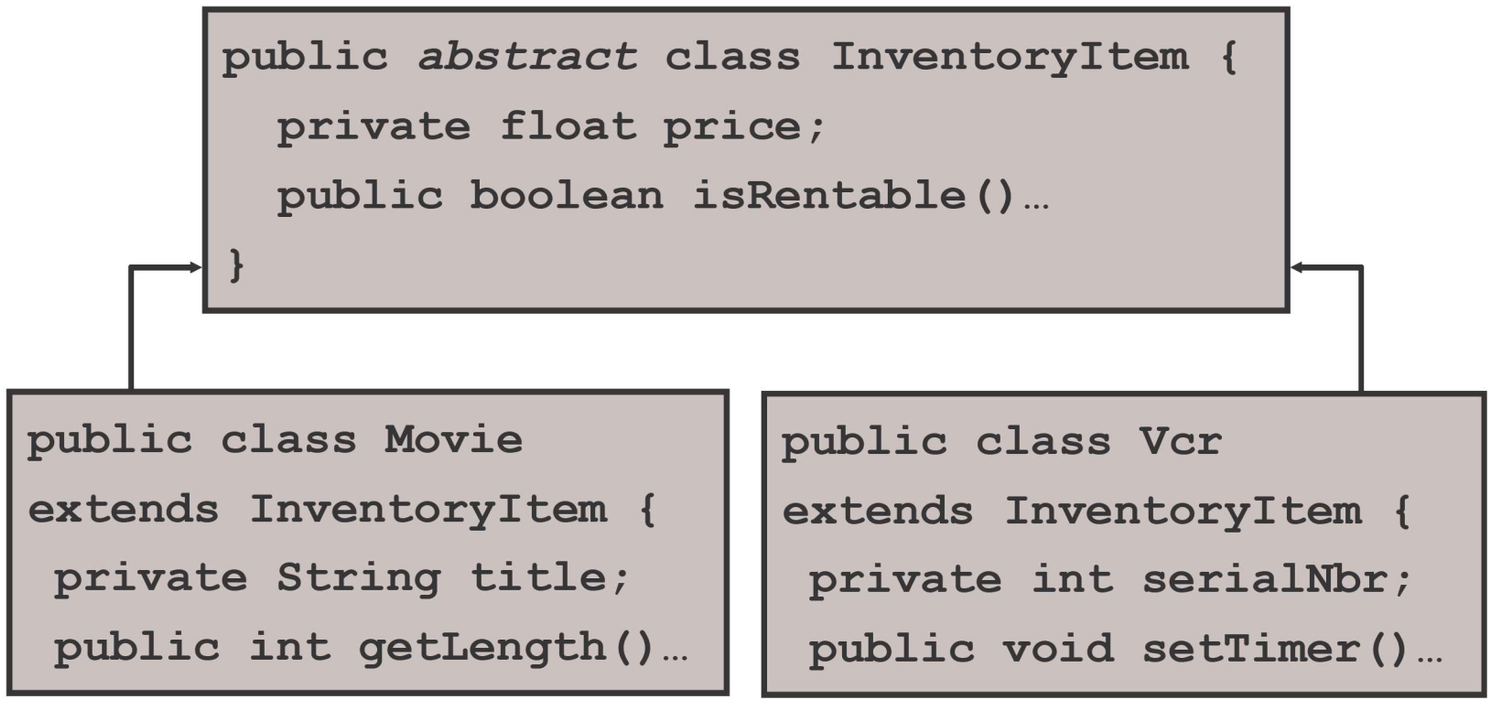
Abstract Classes

- An abstract class cannot be instantiated.
- Abstract methods must be implemented by subclasses.
- Interfaces support multiple inheritance with regard to type.



Creating Abstract Classes

Use the `abstract` keyword to declare a class as abstract.



Abstract Methods

- An abstract method:
  - Is an implementation placeholder
  - Is part of an abstract class
  - Must be overridden by a concrete subclass
- Each concrete subclass can implement the method differently.

Defining Abstract Methods

- Use the `abstract` keyword to declare a method as abstract:
  - Provide the method signature only.
  - The class must also be abstract.
- Why is this useful?
  - You can declare the structure of a given class without providing complete implementation of every method.

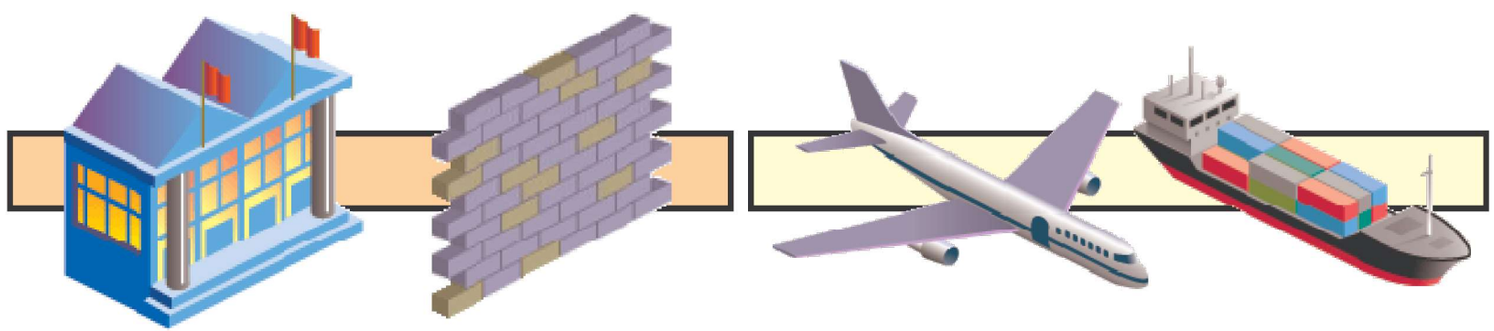
```
public abstract class InventoryItem {
  public abstract boolean isRentable();
  …
```

Defining and Using Interfaces

- An interface is like a fully abstract class:
  - All its methods are abstract.
  - All variables are `public static final`.
- An interface lists a set of method signatures without code details.
- A class that implements the interface must provide code details for all the methods of the interface.
- A class can implement many interfaces but can extend only one class.

Examples of Interfaces

- Interfaces describe an aspect of behavior that different classes require.
- For example, classes that can be steered support the “steerable” interface.
- Classes can be unrelated.



Nonsteerable **Steerable**

Creating Interfaces

- Use the `interface` keyword:

```
public interface Steerable {
  int MAXTURN = 45;
  void turnLeft(int deg);
  void turnRight(int deg);
}
```

- All methods are `public abstract`.
- All variables are `public static final`.

Implementing Interfaces

Use the `implements` keyword:

```
public class Yacht extends Boat
      implements Steerable {
  public void turnLeft(int deg) {…}
  public void turnRight(int deg) {…}
}
```

Interfaces Versus Abstract Classes

	Variables	Constructors	Methods
Abstract Class	No restrictions	Constructors are invoked by subclasses through constructor chaining. An abstract class cannot be instantiated using the `new` operator.	No restrictions
Interface	All variables must be `public static final`.	No constructors. An interface cannot be instantiated using the `new` operator.	All methods must be public abstract methods.

Sort: A Real-World Example

- Is used by several unrelated classes
- Contains a known set of methods
- Is needed to sort any type of object
- Uses comparison rules that are known only to the sortable object
- Supports good code reuse

Overview of the Classes

- Created by the sort expert:

```
public interface
   Sortable
```

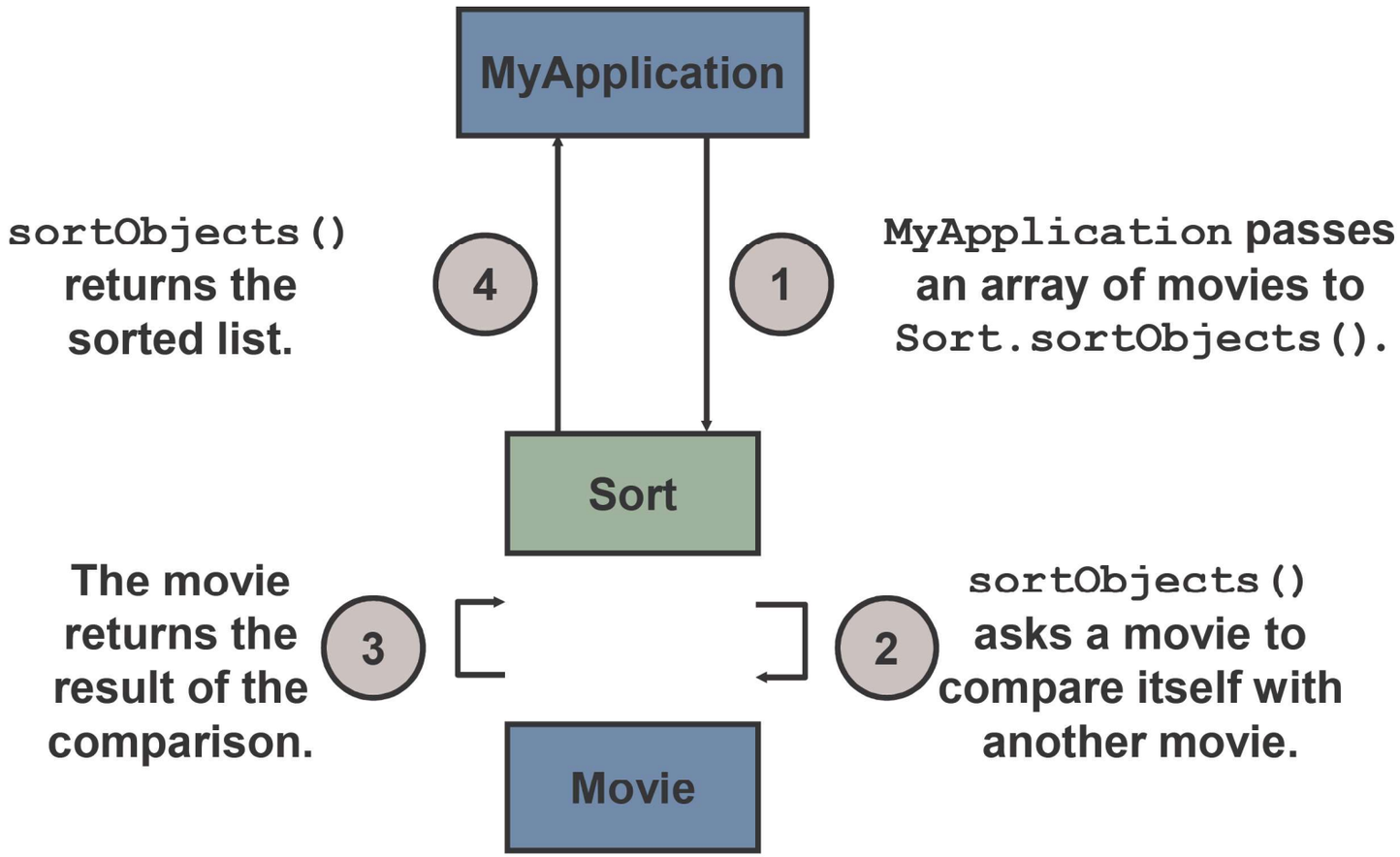
```
public abstract
   class Sort
```

- Created by the movie expert:

```
public class Movie
   implements Sortable
```

```
public class
   MyApplication
```

How the Sort Works



Sortable Interface

Specifies the `compare()` method:

```
public interface Sortable {
  // compare(): Compare this object to another object
  // Returns:
  //    0 if this object is equal to obj2
  //    a value < 0 if this object < obj2
  //    a value > 0 if this object > obj2
  int compare(Object obj2);
}
```

Sort Class

- Holds `sortObjects()`:

```
public abstract class Sort {
  public static void sortObjects(Sortable[] items) {
    // Step through the array comparing and swapping;
    // do this length-1 times
    for (int i = 1; i < items.length; i++) {
      for (int j = 0; j < items.length - 1; j++) {
        if (items[j].compare(items[j+1]) > 0) {
          Sortable tempitem = items[j+1];
          items[j+1] = items[j];
          items[j] = tempitem; } } } } }
```

Movie Class

Implements `Sortable`:

```
public class Movie extends InventoryItem
                   implements Sortable {
  String title;
  public int compare(Object movie2) {
    String title1 = this.title;
    String title2 = ((Movie)movie2).getTitle();
    return(title1.compareTo(title2));
  }
}
```

Using the Sort

- Call `Sort.sortObjects(Sortable [])` with an array of `Movie` as the argument:

```
class myApplication {
  Movie[] movielist;
  …      // build the array of Movie
  Sort.sortObjects(movielist);
}
```

Using `instanceof` with Interfaces

- Use the `instanceof` operator to determine whether an object implements an interface.
- Use downcasting to call methods that are defined in the interface:

```
public void aMethod(Object obj) {
  …
  if (obj instanceof Sortable)
    ((Sortable)obj).compare(obj2);
}
```

Summary

In this lesson, you should have learned the following:

- An abstract class cannot be instantiated.
- An abstract method has a signature but no code.
- An interface is a collection of abstract methods to be implemented elsewhere.
- A class can implement many interfaces.

Practice : Overview

This practice covers the following topics:

- Creating an interface and abstract class
- Testing the abstract and interface classes