CS112

Interfaces
Chapter 13
Lecture 10

الفصل الدراسي الثاني 1443- Spring 2022

College of Computer Science and Engineering

Introduction

- What is an interface?
- Why is an interface useful?
- How do you define an interface?
- How do you use an interface?

What is an interface? Why is an interface useful?

- An interface is a classlike construct that contains only constants and abstract methods
- In many ways, an interface is similar to an abstract class, but the intent of an interface is to specify common behavior for objects
- For example, you can specify that the objects are comparable, edible, cloneable using appropriate interfaces

How do you define an interface?

- To distinguish an interface from a class, Java uses the following syntax to define an interface:

```
public interface InterfaceName {
  constant declarations;
  abstract method signatures;
}
```

- Example:

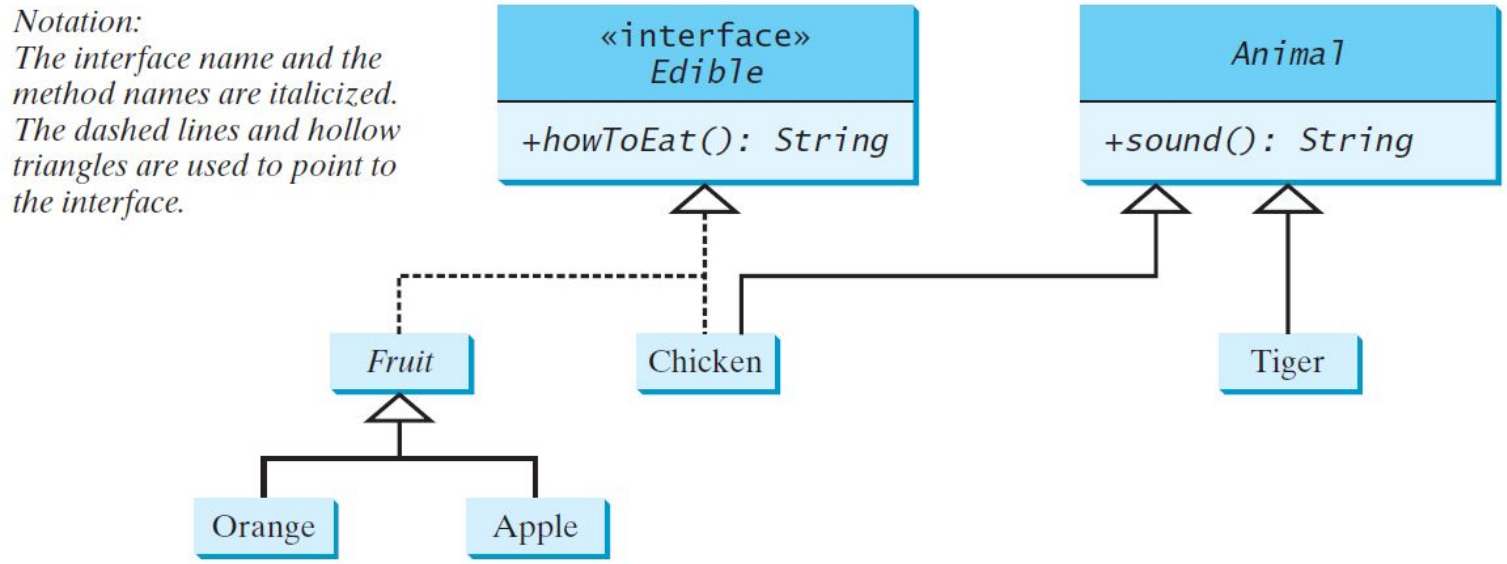
```
public interface Edible {
  /** Describe how to eat */
  public abstract String howToEat();
}
```

How do you use an interface?

- An interface is treated like a special class in Java
- Each interface is compiled into a separate bytecode file, just like a regular class
- Like an abstract class, you cannot create an instance from an interface using the new operator, but in most cases you can use an interface more or less the same way you use an abstract class
- For example, you can use an interface as a data type for a variable, as the result of casting, and so on

Example 1: Edible Class

- You can now use the Edible interface to specify whether an object is edible
- This is accomplished by letting the class for the object implement this interface using the implements keyword
- For example, the classes Chicken and Fruit implement the Edible interface (See Edible.java and TestEdible.java).



Omitting Modifiers in Interfaces

- All data fields are *public final static* and all methods are *public abstract* in an interface. For this reason, these modifiers can be omitted, as shown below:

```
public interface T1 {
  public static final int K = 1;

  public abstract void p();
}
```

Equivalent

```
public interface T1 {
  int K = 1;

  void p();
}
```

- A constant defined in an interface can be accessed using syntax InterfaceName.CONSTANT_NAME (e.g., T1.K).

The Comparable Interface (1)

- Suppose you want to design a generic method to find the larger of two objects of the same type, such as two students, two dates, two circles, two rectangles, or two squares
- In order to accomplish this, the two objects must be comparable, so the common behavior for the objects must be comparable

Java provides the Comparable interface for this purpose. The interface is defined as follows:

```
// This interface is defined in
// java.lang package
package java.lang;

public interface Comparable<E> {
  public int compareTo(E o);
}
```

The compareTo method determines the order of this object with the specified object o and returns a negative integer, zero, or a positive integer if this object is less than, equal to, or greater than o

The Comparable Interface (2)

- The toString, equals, and hashCode Methods:
  - Each wrapper class overrides the toString, equals, and hashCode methods defined in the Object class
  - Since all the numeric wrapper classes and the Character class implement the Comparable interface, the compareTo method is implemented in these classes: See next slide

Integer and BigInteger Classes

```
public class Integer extends Number
    implements Comparable<Integer> {
  // class body omitted

  @Override
  public int compareTo(Integer o) {
    // Implementation omitted
  }
}
```

```
public class BigInteger extends Number
    implements Comparable<BigInteger> {
  // class body omitted

  @Override
  public int compareTo(BigInteger o) {
    // Implementation omitted
  }
}
```

String and Date Classes

```
public class String extends Object
    implements Comparable<String> {
  // class body omitted

  @Override
  public int compareTo(String o) {
    // Implementation omitted
  }
}
```

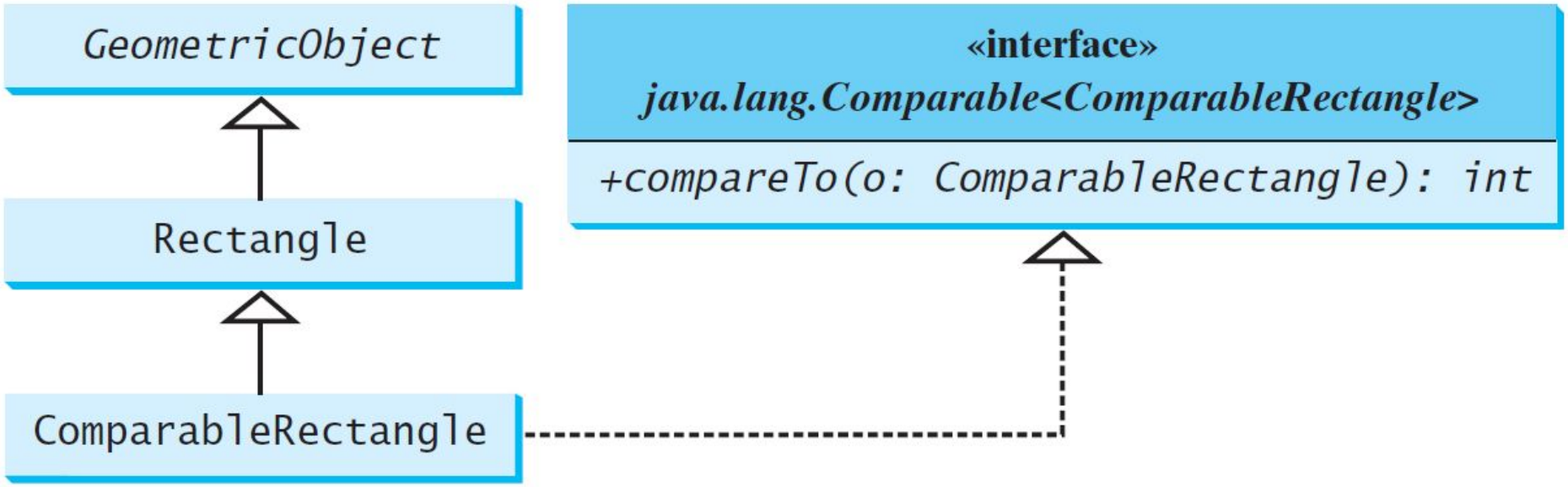
```
public class Date extends Object
    implements Comparable<Date> {
  // class body omitted

  @Override
  public int compareTo(Date o) {
    // Implementation omitted
  }
}
```

Examples

```
System.out.println(new Integer(3).compareTo(new Integer(5)));
System.out.println("ABC".compareTo("ABE"));
java.util.Date date1 = new java.util.Date(2013, 1, 1);
java.util.Date date2 = new java.util.Date(2012, 1, 1);
System.out.println(date1.compareTo(date2));
```

Defining Classes to Implement Comparable



The `Cloneable` Interfaces

- Marker Interface: An empty interface
- A marker interface does not contain constants or methods. It is used to denote that a class possesses certain desirable properties
- A class that implements the Cloneable interface is marked cloneable, and its objects can be cloned using the clone() method defined in the Object class

```
package java.lang;
public interface Cloneable {
}
```

Examples

- Many classes (e.g., Date and Calendar) in the Java library implement Cloneable. Thus, the instances of these classes can be cloned.
- For example, the following code:

```
Calendar calendar = new GregorianCalendar(2003, 2, 1);
Calendar calendarCopy = (Calendar)calendar.clone();
System.out.println("calendar == calendarCopy is " +
  (calendar == calendarCopy));
System.out.println("calendar.equals(calendarCopy) is " +
  calendar.equals(calendarCopy));
```

displays

calendar == calendarCopy is false
calendar.equals(calendarCopy) is true

Implementing Cloneable Interface

- To define a custom class that implements the Cloneable interface, the class must override the clone() method in the Object class
- The following code defines a class named House that implements Cloneable and Comparable:
  - See House.java

Interfaces vs. Abstract Classes (1)

- In an interface, the data must be constants; an abstract class can have all types of data
- Each method in an interface has only a signature without implementation; an abstract class can have concrete methods

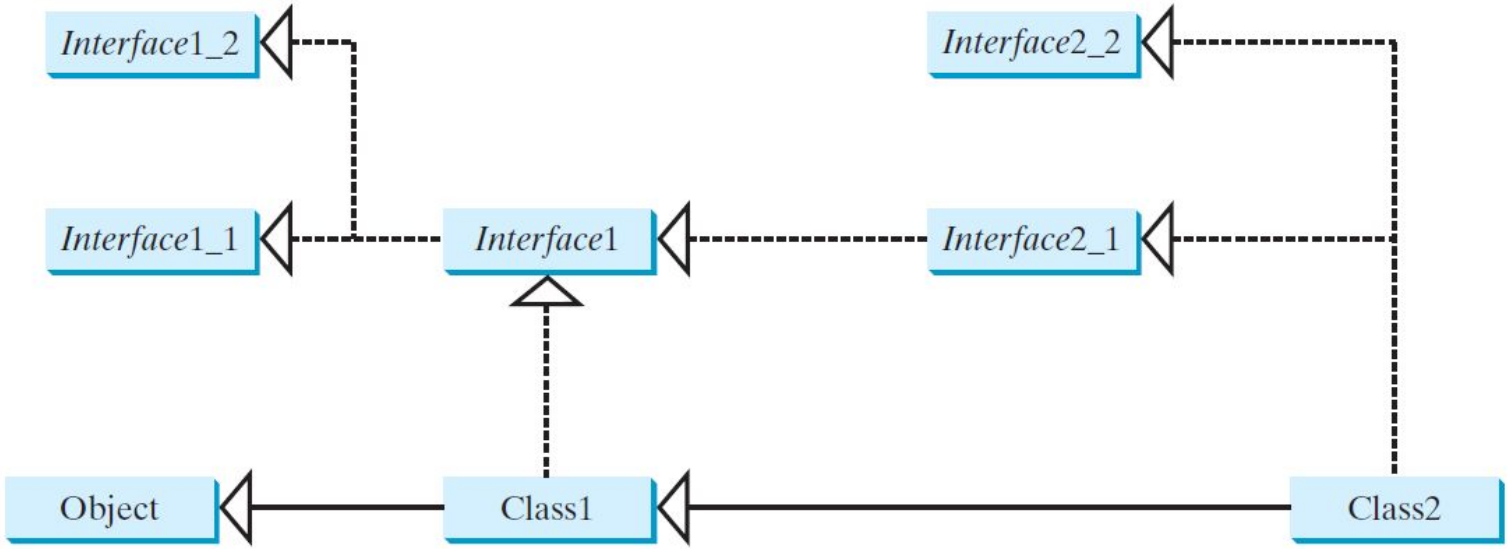
	Variables	*Constructors*	*Methods*
Abstract class	No restrictions.	Constructors are invoked by subclasses through constructor chaining. An abstract class cannot be instantiated using the new operator.	No restrictions.
Interface	All variables must be `public static final`.	No constructors. An interface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

Interfaces vs. Abstract Classes (2)

- All classes share a single root, the Object class, but there is no single root for interfaces
- Like a class, an interface also defines a type
- A variable of an interface type can reference any instance of the class that implements the interface
- If a class extends an interface, this interface plays the same role as a superclass
- You can use an interface as a data type and cast a variable of an interface type to its subclass, and vice versa.

Interfaces vs. Abstract Classes (3)

- Suppose that c is an instance of Class2. c is also an instance of Object, Class1, Interface1, Interface1_1, Interface1_2, Interface2_1, and Interface2_2.



Caution: conflict interfaces

- In rare occasions, a class may implement two interfaces with conflict information (e.g., two same constants with different values or two methods with same signature but different return type). This type of errors will be detected by the compiler.

Whether to use an interface or a class?

- Abstract classes and interfaces can both be used to model common features. How do you decide whether to use an interface or a class?
- In general, a strong is-a relationship that clearly describes a parent-child relationship should be modeled using classes
  - For example, a staff member is a person
- A weak is-a relationship, also known as an is-kind-of relationship, indicates that an object possesses a certain property. A weak is-a relationship can be modeled using interfaces
  - For example, all strings are comparable, so the String class implements the Comparable interface
  - You can also use interfaces to circumvent single inheritance restriction if multiple inheritance is desired
  - In the case of multiple inheritance, you have to design one as a superclass, and others as interface