CISC 3115 TY2

Selected Interfaces in Java API

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Outline

- Discussed
  - Recap
    - Inheritance and polymorphism
    - Abstract method and class
  - Interface
    - Motivation
    - Define interface
    - Extend interface
    - Implement interface
    - Use interface as data type
- Selected interfaces in Java API
  - Comparable, Cloneable, and concept of functional interface
- Static and default methods and constants in interface

Different Classes, Same Behaviors

- Different classes, although vastly different, may exhibit similar behavior
  - Any communication devices can transmit and receive
  - Any vehicles can move
  - Any objects can be compared to each other
  - Any objects may be cloned
  - ……
- Using subclasses (inheritance via subclass) may be too rigid for this kind of flexibility in real life.

Interface in Selected Java API

- Any objects can be compared to each other
  - The Comparable interface
  - The Comparator interface
- Any objects may be cloned
  - The Cloneable interface

The Comparable Interface

- Java defines a Comparable interface
- In the java.lang package, and has a compareTo method

```
package java.lang;


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

The compareTo Method

- Compare two objects, e.g.,
  - lhs.compareTo(rhs)
- Generally, returns an integer
  - negative integer if lhs is less than rhs
  - 0 if lhs is equal to rhs
  - positive integer if lhs is greater than rhs

Implementation of the Interface in Java

- Many Java classes implement the Comparable interface
- Examples
  - Wrapper classes for primitive types
    - Boolean, Byte, Short, Integer, Long, Float, Double, Character
  - Decimal wrapper class
    - BigInteger, BigDecimal

Implementation of the Interface in Java

- See the list in the Java API documentation

Module java.base
Package java.lang

Interface Comparable<T>

Type Parameters:

T - the type of objects that this object may be compared to

All Known Subinterfaces:

AnnotationTypeDoc, AnnotationTypeElementDoc, ArrayType, ByteValue, CharValue,
ConstructorDoc, Delayed, Doc, DoubleValue, ExecutableMemberDoc, Field, FieldDoc
Name, PackageDoc, Path, ProcessHandle, ProgramElementDoc, ReferenceType, RootDo

All Known Implementing Classes:

AbstractChronology, AbstractRegionPainter.PaintContext.CacheMode, AccessMod
AttributeTree.ValueKind, Authenticator.RequestorType, BigDecimal, BigInteger
CertPathValidatorException.BasicReason, Character, Character.UnicodeScript,
Component.BaselineResizeBehavior, CompositeName, CompoundName, ConversionCon
Diagnostic.Kind, Dialog.ModalExclusionType, Dialog.ModalityType, Doclet.Opti
DrbgParameters.Capability, DropMode, Duration, ElementKind, Elements.Origin, E
FormatStyle, Formatter.BigDecimalLayoutForm, FormSubmitEvent.MethodType, Gra
HijrahDate, HijrahEra, HttpClient.Redirect, HttpClient.Version, InquireType, I
JConsoleContext.ConnectionState, JDBCType, JTable.PrintMode, KeyRep.Type, Lam

Examples: Java Implementation of the Comparable Interface

- Examples

```
System.out.println(Integer.valueOf(3).compareTo(Integer.valueOf(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));
```

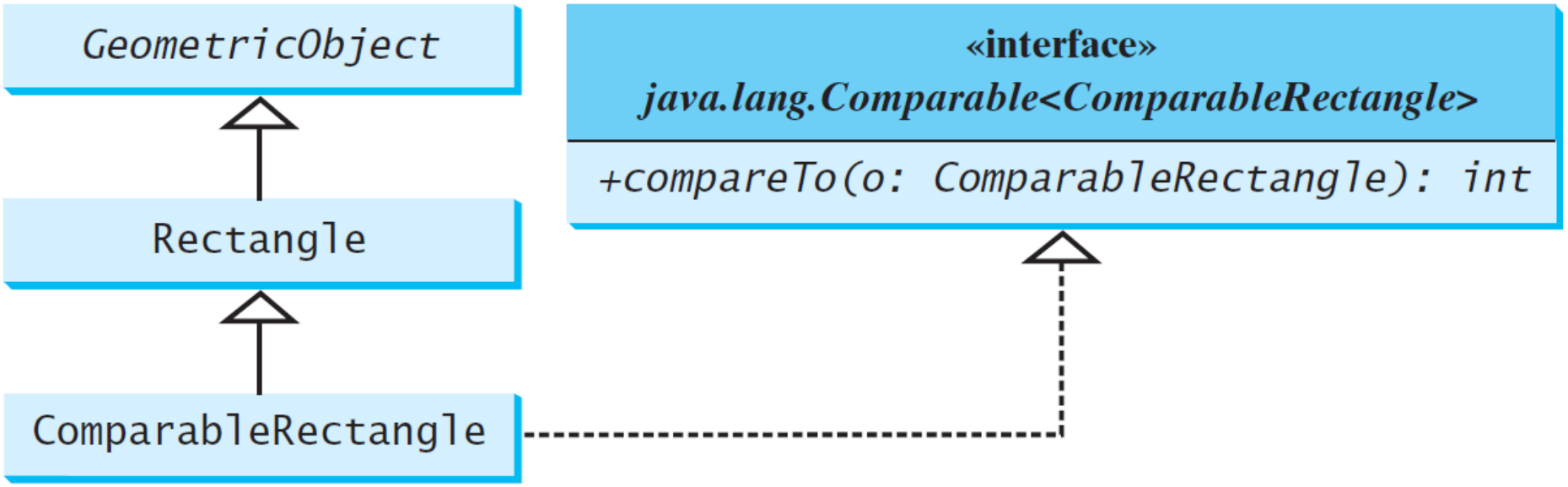
Sorting and Comparable

- Comparable's are often used for sorting objects
  - Sorting arrays
  - Sorting collections (e.g., ArrayList)
- Two scenarios
  - Sorting objects of data types that have already implemented the Comparable interfaces
  - Sorting objects of data types for which you will implement the Comparable interfaces

Sorting: Examples

- Sorting objects of data types that have had the Comparable interfaces implemented
  - e.g., sorting wrapper objects of primitive types
    - Write generic sort method ourselves to sort arrays or collections (only the arrays example given)
    - Sort arrays: use java.util.Arrays::sort method
    - Sort collections (e.g., ArrayList): use java.util.Collections::sort method (your exercise)
- Sorting objects of data types for which you will implement the Comparable interfaces
  - e.g., sorting rectangles according their areas
    - Sort arrays: use java.util.Arrays::sort method
    - Sort collections (e.g., ArrayList): use java.util.Collections::sort method

Sorting: Implementing Comparable: Example



Sorting Order

- Standard or ascending order
  - e.g., 1, 2, 3, 4, 5, …
- Descending order
  - e.g., 5, 4, 3, 2, 1, …
- Can we control the sorting order?
  - Override the compareTo method. But is it always a good approach?
  - Use the Comparator interface and the sort method in java.util.Arrays and java.util.Collections.

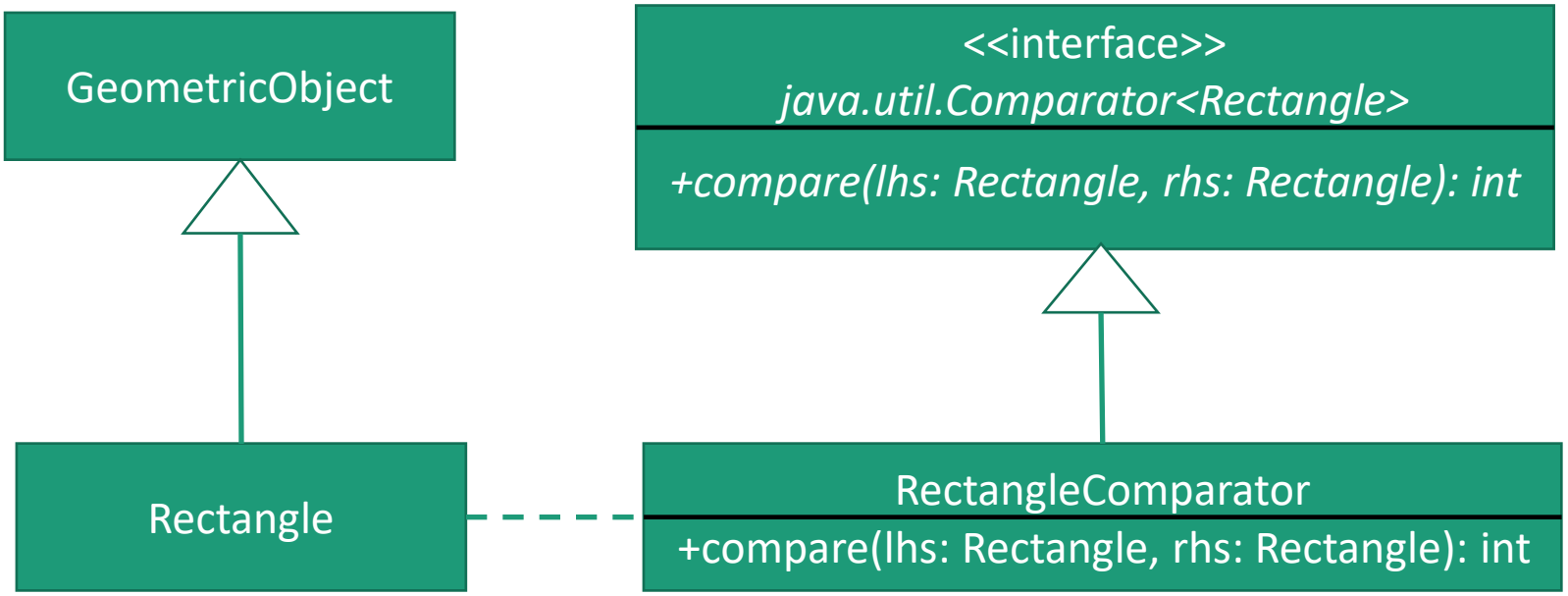
Examples: Control Sorting Order

- Sorting an array of rectangles (Arrays)
- Sorting an ArrayList of rectangles (Collections)

```
class RectangleComparator implements
Comparator<Rectangle> {

  int compare(Rectangle lhs, Rectangle rhs) {

      ….

  }

}
```

Sorting: Implementing Comparator: Example

- Relationship among the classes



Questions?

- The Comparable interface
- The Comparator interface
- Comparable or Comparator, which one to use?
- Comparing objects
- Sorting arrays
- Sorting collections

The Cloneable Interface

- Called a Marker Interface, as it does not contain constants or methods.

```
package java.lang;

public interface Cloneable {

}
```

- Purpose:
  - Marker interface: to denote that a class possesses certain desirable properties.
  - The Cloneable marker interface: to denote that the class's objects can be cloned using the clone() method defined in the Object class
    - The class, in general, should implement the Cloneable interface, and define the mechanism an object is being cloned

Cloneable in Java Library

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

Examples: Using clone and Cloneable

- Example:

```
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));
```

- The result (and why?):
  - calendar == calendarCopy is false
  - calendar.equals(calendarCopy) is true

Implementing Cloneable

- The Cloneable is a marker interface that does not have any method
  - What to implement?
  - The contract is that we ought to implement the clone method in the java.lang.Object class

Example: Implementing Cloneable

- The House class
- The mechanism to implement the clone method
  - Need to duplicate the object state, but …
  - Examine two concepts
    - Shallow copy
    - Deep copy

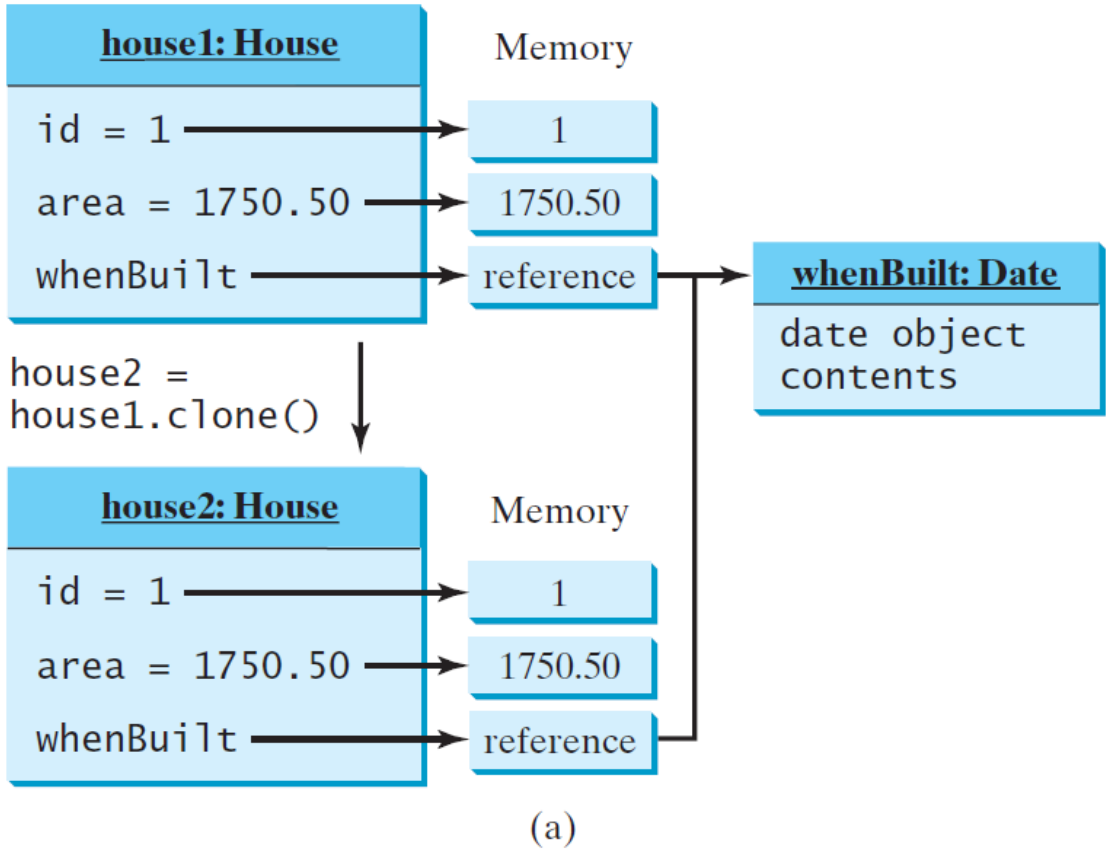
Shallow Copy vs Deep Copy

- Let's examine this code snippet, what does it do?

```
House house1 = new House(1, 1750.50);

House house2 = (House)house1.clone();
```

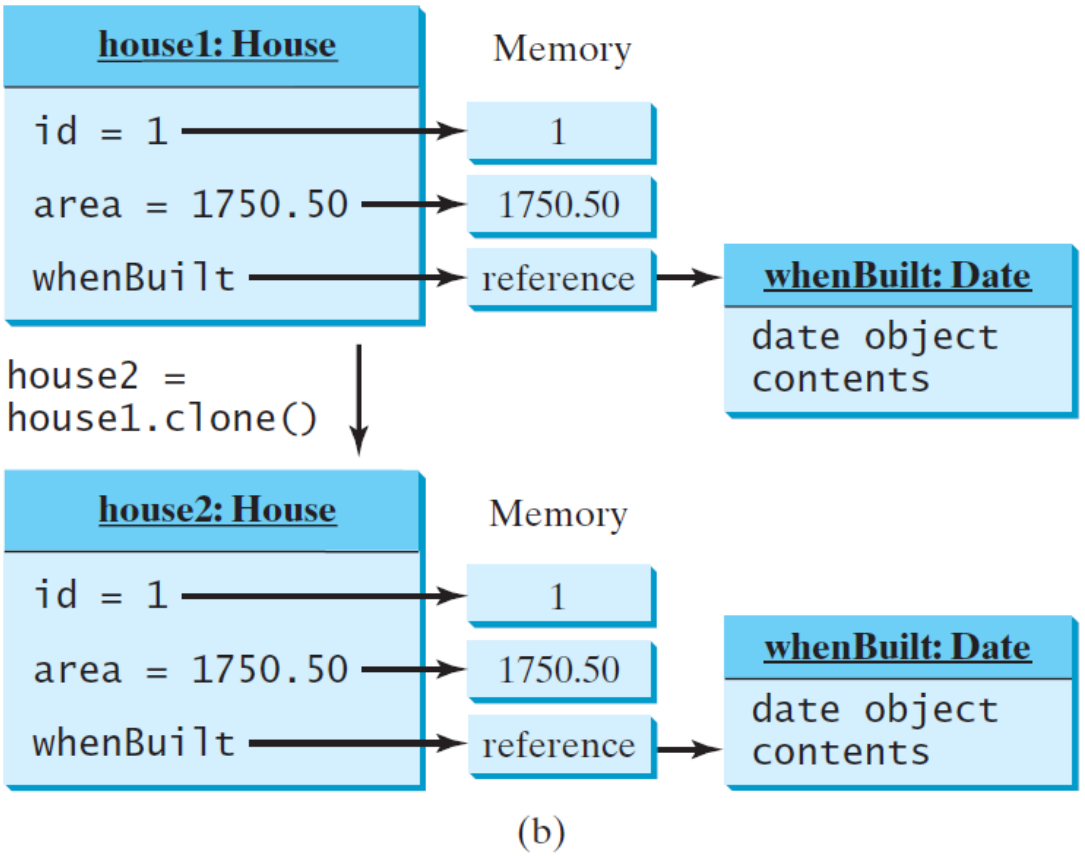
Shallow Copy

- House house2 = (House)house1.clone();



Deep Copy

- House house2 = (House)house1.clone();



Questions?

- Concept of marker interface
- The Cloneable interface
- How to implement the Cloneable interface
- Shallow and deep copy
- But, how deep is deep?

Default and Static Methods and Constants in Interface

- Your tasks: carefully examine the notes in Section 13.5 in the text book
  - Default methods in Java interface
  - Static methods in Java interface
  - Constants in Java interface

Questions?

- Any exercises?