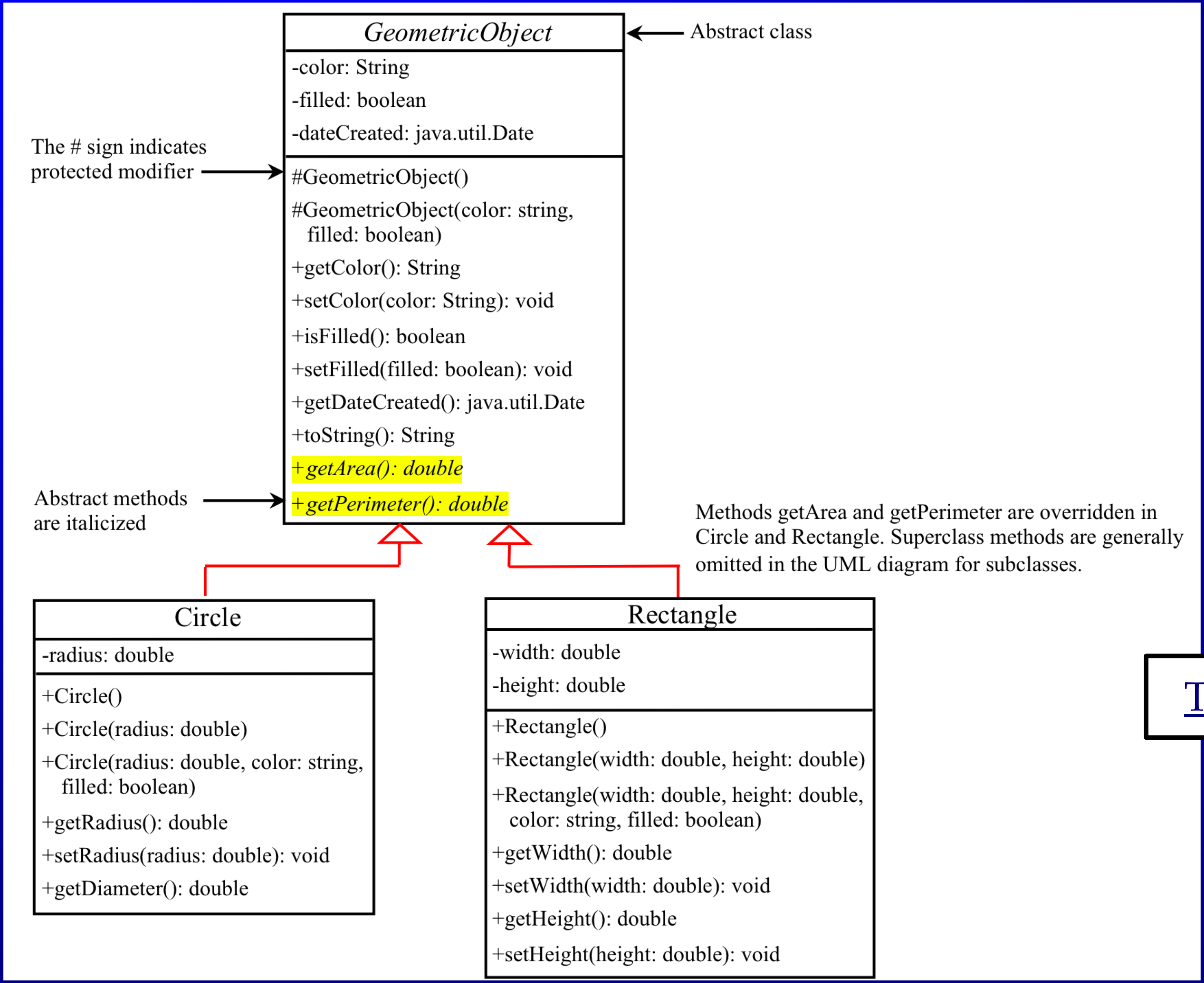
Lecture 7: Abstract Classes and Interfaces (Ch 15)

Adapted by Fangzhen Lin for COMP3021 from Y. Danial Liang's PowerPoints for Introduction to Java Programming, Comprehensive Version, 9/E, Pearson, 2013.

Objectives

- To design and use abstract classes (§15.2).
- To process a calendar using the Calendar and GregorianCalendar classes (§15.3).
- To specify behavior for objects using interfaces (§15.4).
- To define interfaces and declare classes that implement interfaces (§15.4).
- To define a natural order using the Comparable interface (§15.5).
- To enable objects to listen for action events using the ActionListener interface (§15.6).
- To make objects cloneable using the Cloneable interface (§15.7).
- To explore the similarities and differences between an abstract class and an interface (§15.8).
- *To create objects for primitive values using the wrapper classes (Byte, Short, Integer, Long, Float, Double, Character, and Boolean) (§15.9).
- *To create a generic sort method (§15.10).
- *To simplify programming using automatic conversion between primitive types and wrapper class types (§15.11).
- *To use the BigInteger and BigDecimal classes for computing very large numbers with arbitrary precisions (§15.12).

Abstract Classes and Abstract Methods



GeometricObject
Abstract class
-color: String
-filled: boolean
-dateCreated: java.util.Date
The # sign indicates protected modifier
#GeometricObject()
#GeometricObject(color: string, filled: boolean)
+getColor(): String
+setColor(color: String): void
+isFilled(): boolean
+setFilled(filled: boolean): void
+getDateCreated(): java.util.Date
+toString(): String
+getArea(): double
+getPerimeter(): double
Abstract methods are italicized
Methods getArea and getPerimeter are overridden in Circle and Rectangle. Superclass methods are generally omitted in the UML diagram for subclasses.
Circle
-radius: double
+Circle()
+Circle(radius: double)
+Circle(radius: double, color: string, filled: boolean)
+getRadius(): double
+setRadius(radius: double): void
+getDiameter(): double
Rectangle
-width: double
-height: double
+Rectangle()
+Rectangle(width: double, height: double)
+Rectangle(width: double, height: double, color: string, filled: boolean)
+getWidth(): double
+setWidth(width: double): void
+getHeight(): double
+setHeight(height: double): void

GeometricObject

Circle

Rectangle

TestGometricObject

What's an abstract class?

- It is a special kind of class solely used for sharing/reusing data/behavior for subclasses
- Difference from the classes you have seen so far
  - Cannot be "newed".
  - Can own "abstract methods".
- Analogy:
  - Special molds used to make other molds not to make cakes

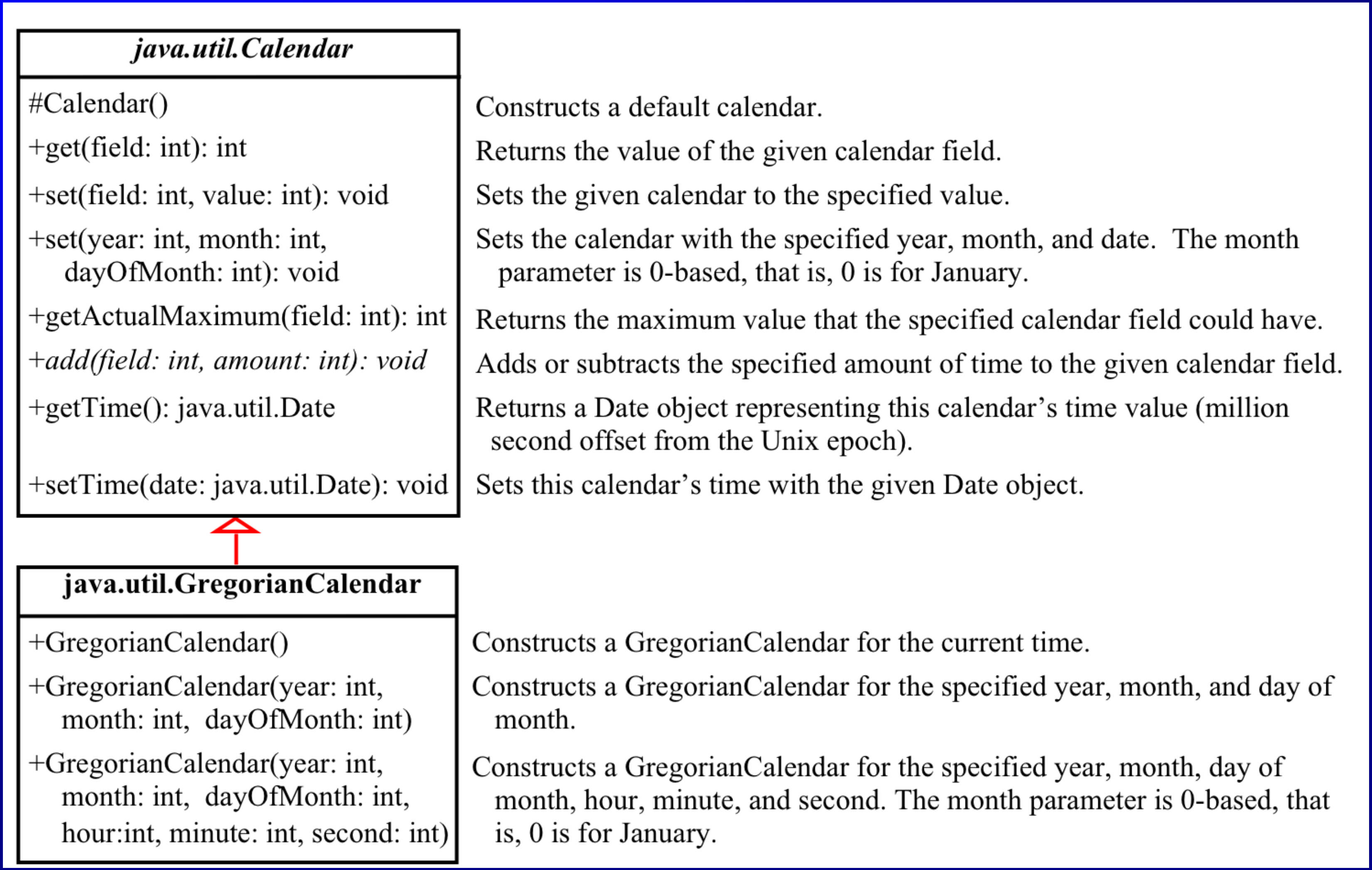
What’s an abstract method?

- We need “abstract classes” for reusing data/behavior for subclasses.
- Often we only say what a method looks like and don’t provide any definition
  - No need → subclass will and must override
  - Impossible → depends on what subclass does
- Such methods are “abstract” methods with the `abstract` modifier.

Interesting facts

- Only abstract class can have abstract methods, regular class cannot.
- Abstract classes don't have to have abstract methods.
- Subclasses must provide definitions for all abstract methods in parents unless themselves are abstract
- Subclass can be abstract and its parent can be concrete.
- Subclass can override a concrete method as abstract

The Abstract Calendar Class and Its GregorianCalendar subclass



java.util.Calendar	
#Calendar()	Constructs a default calendar.
+get(field: int): int	Returns the value of the given calendar field.
+set(field: int, value: int): void	Sets the given calendar to the specified value.
+set(year: int, month: int, dayOfMonth: int): void	Sets the calendar with the specified year, month, and date. The month parameter is 0-based, that is, 0 is for January.
+getActualMaximum(field: int): int	Returns the maximum value that the specified calendar field could have.
+add(field: int, amount: int): void	Adds or subtracts the specified amount of time to the given calendar field.
+getTime(): java.util.Date	Returns a Date object representing this calendar's time value (million second offset from the Unix epoch).
+setTime(date: java.util.Date): void	Sets this calendar's time with the given Date object.

java.util.GregorianCalendar	
+GregorianCalendar()	Constructs a GregorianCalendar for the current time.
+GregorianCalendar(year: int, month: int, dayOfMonth: int)	Constructs a GregorianCalendar for the specified year, month, and day of month.
+GregorianCalendar(year: int, month: int, dayOfMonth: int, hour:int, minute: int, second: int)	Constructs a GregorianCalendar for the specified year, month, day of month, hour, minute, and second. The month parameter is 0-based, that is, 0 is for January.

The Abstract Calendar Class and Its GregorianCalendar subclass

- An instance of java.util.Date represents a specific instant in time with millisecond precision.
- java.util.Calendar is an abstract base class for extracting detailed information such as year, month, date, hour, minute and second from a Date object.
- Subclasses of Calendar can implement specific calendar systems such as Gregorian calendar, Lunar Calendar and Jewish calendar.
- Currently, java.util.GregorianCalendar for the Gregorian calendar is supported in the Java API.

The GregorianCalendar Class

You can use new GregorianCalendar() to construct a default GregorianCalendar with the current time and use new GregorianCalendar(year, month, date) to construct a GregorianCalendar with the specified year, month, and date. The month parameter is 0-based, i.e., 0 is for January.

TestCalendar

Interfaces

What is an interface?

Why is an interface useful?

How do you define an interface?

How do you use an interface?

What’s an interface?

- One of the most important concepts in the Java language:
  - To satisfy the need of multiple inheritance.
  - Key for building massive scale Java libraries
- Similar to abstract classes:
  - Methods defined in a class can be used only by objects of this class and its subclasses.
  - Methods defined in an interface can be used by objects belong to different classes.

Define an Interface

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

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

Example:

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

Conceptual Example

- Chicken
  - An animal, belong to the class of Bird.
  - A type of poultry
  - Our favorite dish in LG1/7
- Multiple roles of Chicken class
  - Latin name/Gene sequence/Origin
  - Origin (farm)/stock number/Weight
  - Price etc.

Code Example

```
abstract class Bird {};

interface Poultry {
   public int getStockNumber();
}

interface  Dish {
   public void placeOrder();
}

class Chicken extends Bird implements Poultry, Dish{};
```

Chicken mychicken = new Chicken();
mychicken instanceof Bird ?
mychicken instanceof Dish;
mychicken instanceof Poultry;

Example

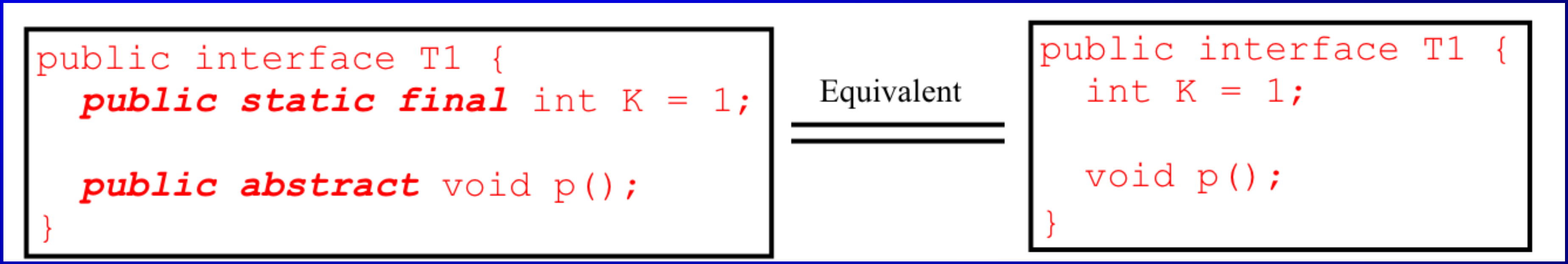
You can now use the Edible interface to specify whether an object is edible. This is accomplished by letting the class for the object implement this interface using the implements keyword.

Edible

TestEdible

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();
}
```

```
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).

Example: The Comparable Interface

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

public interface Comparable {
  public int compareTo(Object o);
}
```

String and Date Classes

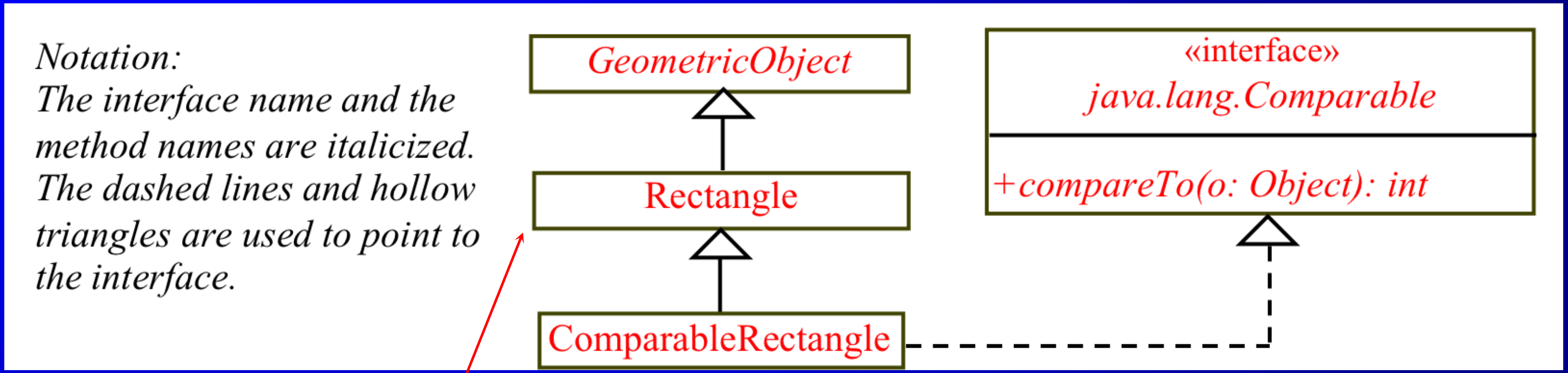
Many classes (e.g., String and Date) in the Java library implement Comparable to define a natural order for the objects. If you examine the source code of these classes, you will see the keyword implements used in the classes, as shown below:

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

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

```
new String() instanceof String
new String() instanceof Comparable
new java.util.Date() instanceof java.util.Date
new java.util.Date() instanceof Comparable
```

Declaring Classes to Implement Comparable


Notation:
The interface name and the method names are italicized. The dashed lines and hollow triangles are used to point to the interface.
GeometricObject
Rectangle
ComparableRectangle
«interface»
java.lang.Comparable
+compareTo(o: Object): int

ComparableRectangle

You cannot use the max method to find the larger of two instances of Rectangle, because Rectangle does not implement Comparable. However, you can declare a new rectangle class that implements Comparable. The instances of this new class are comparable. Let this new class be named ComparableRectangle.

```
ComparableRectangle rectangle1 = new ComparableRectangle(4, 5);
ComparableRectangle rectangle2 = new ComparableRectangle(3, 6);
System.out.println(Max.max(rectangle1, rectangle2));
```

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

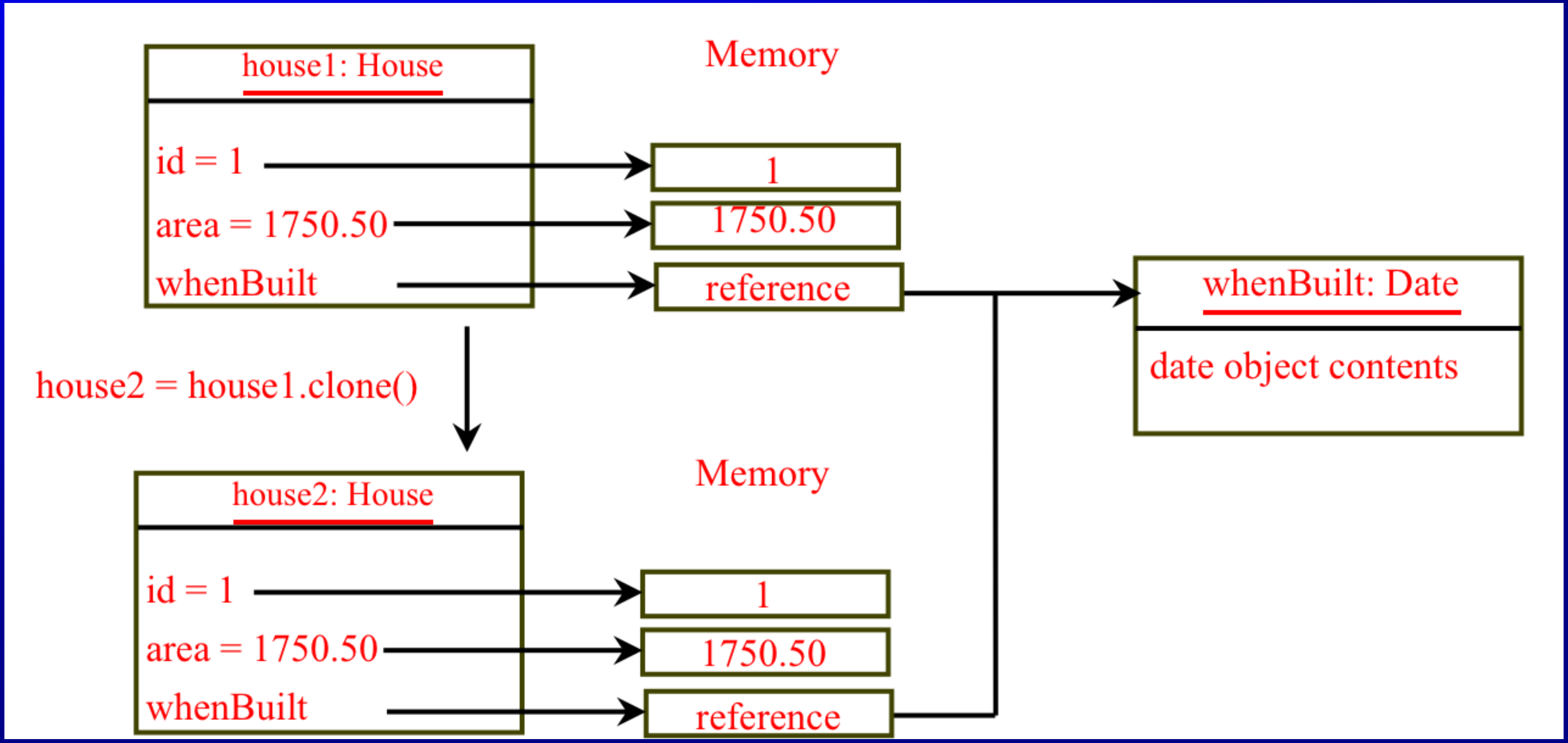
To declare a custom class that implements the Cloneable interface, the class must override the clone() method in the Object class. The following code declares a class named House that implements Cloneable and Comparable.

House

Shallow vs. Deep Copy

House house1 = new House(1, 1750.50);

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



Interfaces vs. Abstract Classes

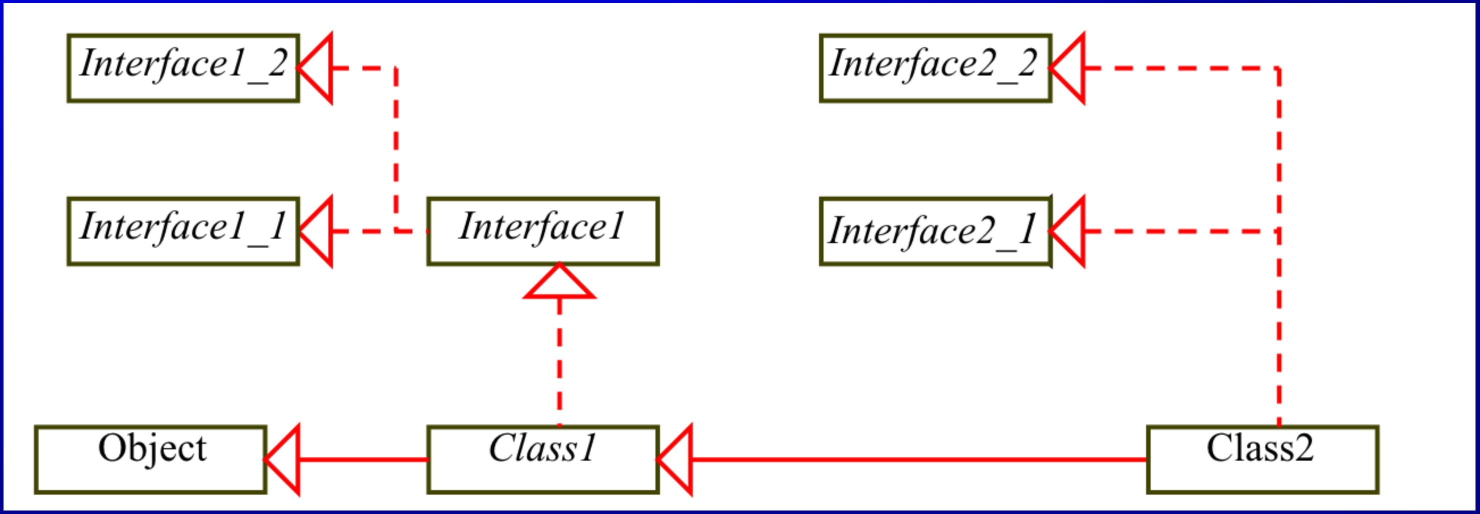
In an interface, the data must be constants; an abstract class can have all types of data.

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, cont.

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.



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.