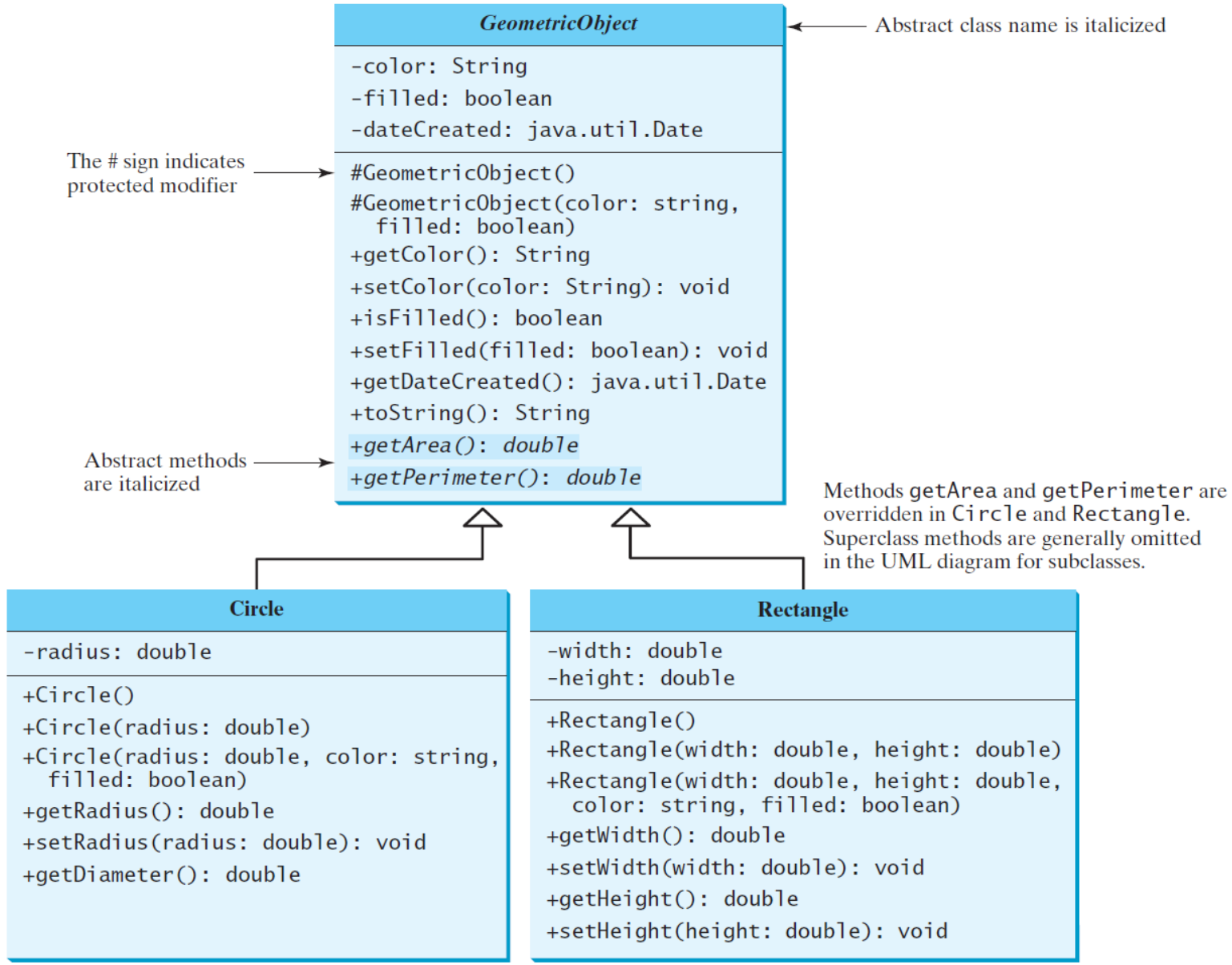
Chương 6
Hướng đối tượng
(Lớp trừu tượng – Interface)

Lớp trừu tượng và phương thức trừu tượng



Phương thức trừu tượng trong lớp trừu tượng

Một phương thức trừu tượng không thể được chứa trong một lớp non-abstract.

Nếu một lớp con của một lớp cha trừu tượng *không cài đặt (implement)* tất cả các phương thức trừu tượng thì lớp con đó cần phải định nghĩa là trừu tượng.

Một lớp con non-abstract kế thừa từ một lớp trừu tượng thì cần cài đặt tất cả các phương thức trừu tượng *(kể cả các phương thức không sử dụng trong lớp con)*.

Không thể khởi tạo đối tượng từ lớp trừu tượng

Một lớp trừu tượng không thể khởi tạo đối tượng bằng toán tử **new**, nhưng vẫn có thể định nghĩa hàm khởi tạo *(được gọi thực thi trong lớp con)*.

Ví dụ: các phương thức khởi tạo của lớp **GeometricObject** được gọi thực thi trong lớp **Circle** và **Rectangle**.

Lớp trừu tượng không chứa phương thức trừu tượng

Một lớp chứa các phương thức trừu tượng ➔ cần được khai báo là trừu tượng.

Tuy nhiên, có thể định nghĩa một lớp trừu tượng không chứa bất kỳ phương thức trừu tượng nào. Trong trường hợp này, không thể tạo các thể hiện của lớp bằng toán tử **new**. Lớp này được sử dụng như lớp cơ sở để định nghĩa một lớp con mới.

Lớp cha của lớp con có thể là lớp cụ thể

Một lớp con có thể là trừu tượng mặc dù lớp cha của nó là lớp cụ thể.

Ví dụ, lớp Object là lớp cụ thể, nhưng các lớp con của nó *(chẳng hạn: GeometricObject)* có thể là trừu tượng.

Phương thức cụ thể bị ghi đè thành trừu tượng

Một lớp con có thể ghi đè *(override)* một phương thức từ lớp cha của nó thành phương thức trừu tượng.

Điều này hiếm, nhưng hữu ích khi việc cài đặt phương thức trong lớp cha trở nên không hợp lý trong lớp con.

Trường hợp này, lớp con phải là trừu tượng.

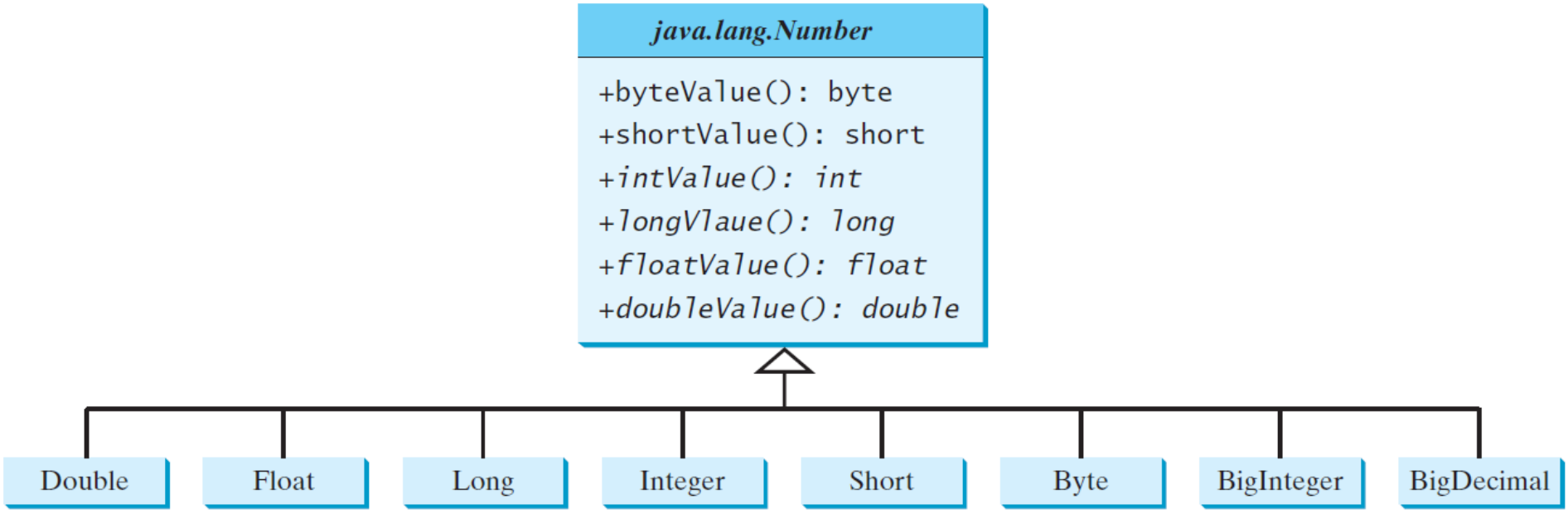
Lớp trừu tượng là một kiểu *dữ liệu*

Không thể tạo một thể hiện từ một lớp trừu tượng bằng toán tử **new**, nhưng một lớp trừu tượng có thể được sử dụng như một kiểu dữ liệu.

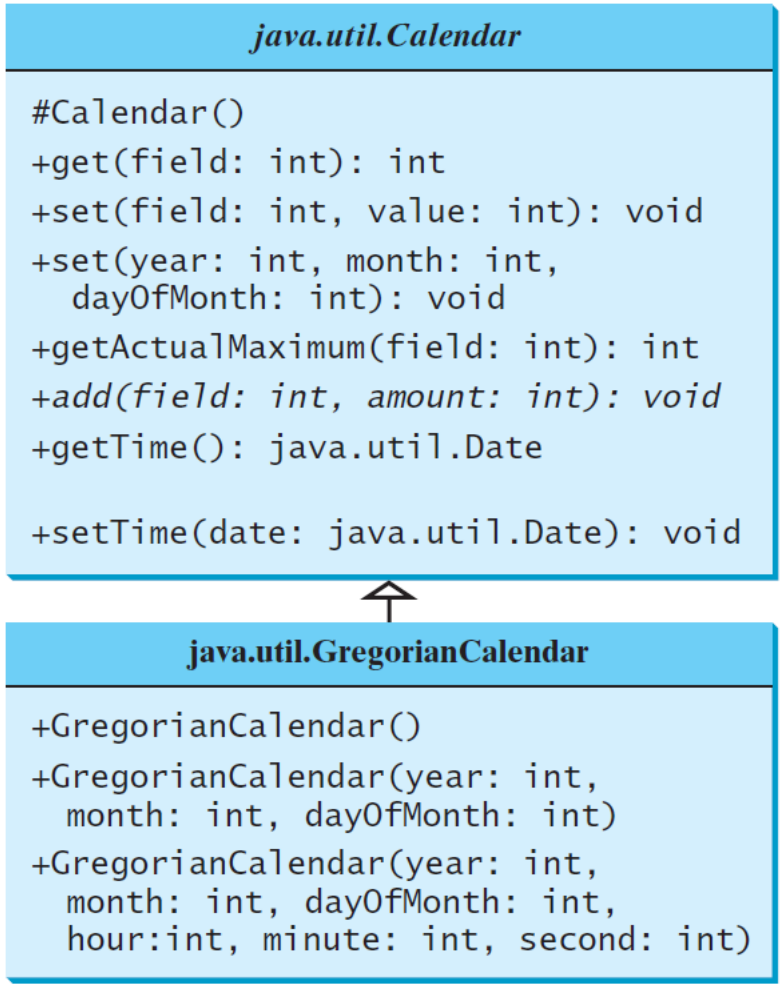
Ví dụ:

GeometricObject[] geo = new GeometricObject[10];

Lớp trừu tượng **Number**



Lớp trừu tượng **Calendar** và lớp con **GregorianCalendar**



- Khởi tạo một lịch (calendar) mặc định
- Trả về giá trị của một trường dữ liệu trong lịch
- Gán giá trị cho một trường dữ liệu trong lịch
- Gán lịch với năm, tháng, và ngày xác định (giá trị 0 là tháng 1)

- Trả về giá trị lớn nhất của một trường dữ liệu trong lịch
- Cộng/trừ một giá thời gian xác định cho một trường dữ liệu trong lịch
- Trả về một đối tượng Date biểu diễn cho giá trị thời gian của lịch hiện tại (tính bằng mili giây từ thời điểm giờ UNIX)
- Gán thời gian của lịch hiện tại với một đối tượng Date cho trước

- Khởi tạo một đối tượng lịch **GregorianCalendar** cho thời gian hiện tại
- Khởi tạo một đối tượng lịch **GregorianCalendar** với năm, tháng và ngày xác định
- Khởi tạo một đối tượng lịch **GregorianCalendar** với năm, tháng, ngày, giờ, phút và giây xác định (giá trị 0 là tháng 1)

Phương thức *get* trong lớp **Calendar**

Phương thức *get(int field)* được định nghĩa trong lớp Calendar giúp trích xuất các thông tin ngày giờ từ một đối tượng Calendar.

Các trường dữ liệu được định nghĩa là các hằng số sau:

Constant	*Description*
YEAR	The year of the calendar.
MONTH	The month of the calendar, with 0 for January.
DATE	The day of the calendar.
HOUR	The hour of the calendar (12-hour notation).
HOUR_OF_DAY	The hour of the calendar (24-hour notation).
MINUTE	The minute of the calendar.
SECOND	The second of the calendar.
DAY_OF_WEEK	The day number within the week, with 1 for Sunday.
DAY_OF_MONTH	Same as DATE.
DAY_OF_YEAR	The day number in the year, with 1 for the first day of the year.
WEEK_OF_MONTH	The week number within the month, with 1 for the first week.
WEEK_OF_YEAR	The week number within the year, with 1 for the first week.
AM_PM	Indicator for AM or PM (0 for AM and 1 for PM).

Interface

Interface là gì?

Tại sao interface lại có ích?

Làm thế nào để định nghĩa một interface?

Sử dụng interface như thế nào?

Interface là gì?
Tại sao interface lại có ích?

Một interface *giống như một lớp* chỉ chứa các hằng số và các phương thức trừu tượng.

Hay nói cách khác, một interface giống như một lớp trừu, nhưng mục đích của một interface thì xác định các hành vi phổ biên cho các đối tượng.

Ví dụ: các đối tượng có thể so sánh, chỉnh sửa, sao chép bằng cách sử dụng các interface thích hợp.

Định nghĩa một interface

Để phân biệt một interface với một lớp, Java sử dụng cú pháp sau để định nghĩa một interface:

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

Ví dụ:

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

Interface là một lớp đặc biệt

Một interface được xem như là một lớp đặc biệt trong Java. Mỗi interface được biên dịch thành một tập tin **bytecode** riêng biệt, tương tự như một lớp bình thương.

Giống như một lớp trừu tượng, không thể tạo một thể hiện từ một interface bằng toán tử **new**, nhưng trong hầu hết các trường hợp, có thể sử dụng một interface như một lớp trừu tượng.

Ví dụ: *có thể sử dụng một interface như một kiểu dữ liệu cho 1 biến số, kết quả của việc ép kiểu…*

Việc lượt bỏ các chỉ định truy cập trong Interfaces

Trong một interface:

- Tất cả các trường dữ liệu là: *public final static*
- Tất cả các phương thức là: *public abstract*

==> các chỉ định đó có thể bị lượt bỏ:

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

  public abstract void p();
}
```

Equivalent

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

  void p();
}
```

Một hằng số được định nghĩa trong một interface có thể được truy cập bằng cú pháp: **InterfaceName.CONSTANT_NAME** *(ví dụ: T1.K).*

Ví dụ: Interface Comparable

```
// interface này được định nghĩa trong
// java.lang package
package java.lang;

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

Phương thức toString, equals, và hashCode

Mỗi lớp wrapper ghi đè phương thức toString, equals, và hashCode được định nghĩa trong lớp **Object**.

==> Tất cả các **lớp wrapper số học** và lớp **Character** cài đặt *interface Comparable*, phương thức compareTo được cài đặt trong các lớp này.

Lớp Integer và BigInteger

```
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
  }
}
```

Lớp String và Date

```
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
  }
}
```

Ví dụ

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

Phương thức `sort` tổng quát

Cho **n** là một đối tượng **Integer**, **s** là một đối tượng **String**, và **d** là một đối tượng **Date**.

Tất cả các biểu thức sau đều có giá trị **true**:

```
n instanceof Integer
n instanceof Object
n instanceof Comparable
```

```
s instanceof String
s instanceof Object
s instanceof Comparable
```

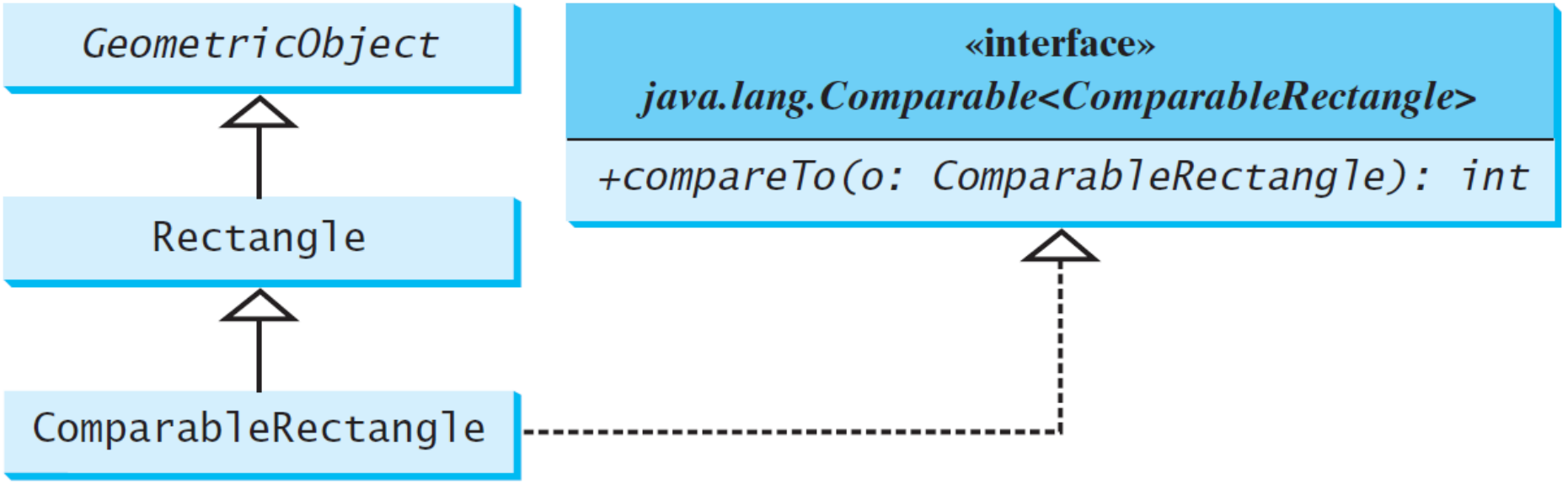
```
d instanceof java.util.Date
d instanceof Object
d instanceof Comparable
```

Phương thức **java.util.Arrays.sort(array)** yêu cầu các phần tử trong mảng phải là các thể hiện của **Comparable<E>**.

SortComparableObjects

Run

Định nghĩa các lớp để cài đặt Comparable



Interface `Cloneable`

Marker Interface: một interface rỗng.

Một marker interface không chứa bất kỳ hằng số hay phương thức nào.

Một lớp cài đặt interface Cloneable thì được đánh dấu có thể sao chép, và các đối tượng của nó có thể sao chép bằng phương thức clone() được định nghĩa trong lớp Object.

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

Ví dụ

Nhiều lớp (e.g., Date và Calendar) trong thư viện Java cài đặt **Cloneable**. Do đó, các thể hiện của các lớp này có thể được sao chép.

```
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));
```

Kết quả:

calendar == calendarCopy là **false**

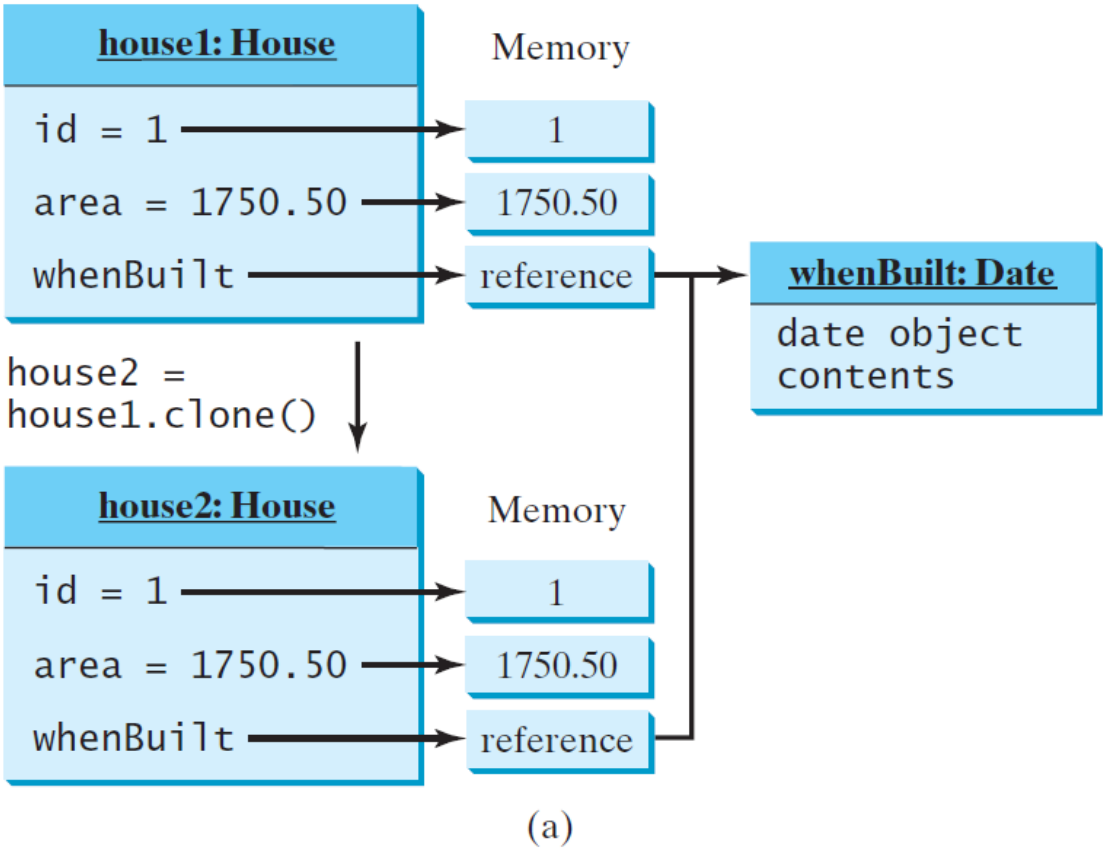
calendar.equals(calendarCopy) là **true**

Shallow vs. Deep Copy

House house1 = new House(1, 1750.50);

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

Shallow Copy

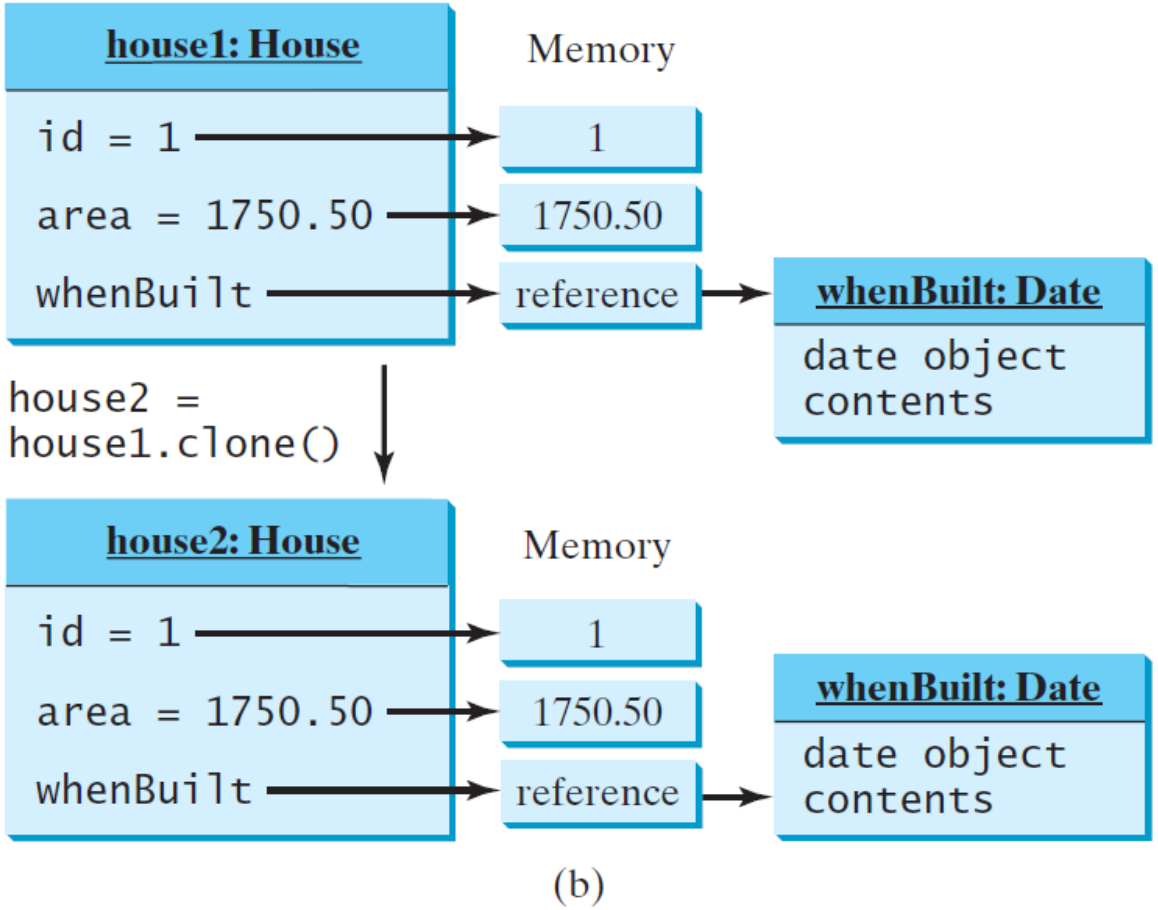


Shallow vs. Deep Copy

House house1 = new House(1, 1750.50);

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

Deep Copy



Interface vs. Lớp trừu tượng

Trong một interface, dữ liệu phải là các hằng số; một lớp trừu tượng có thể có là biến hoặc hằng số.

Mỗi phương thức trong một interface chỉ có 1 chữ ký và không có cài đặt; một lớp trừu tượng có thể có các phương thức cụ thể.

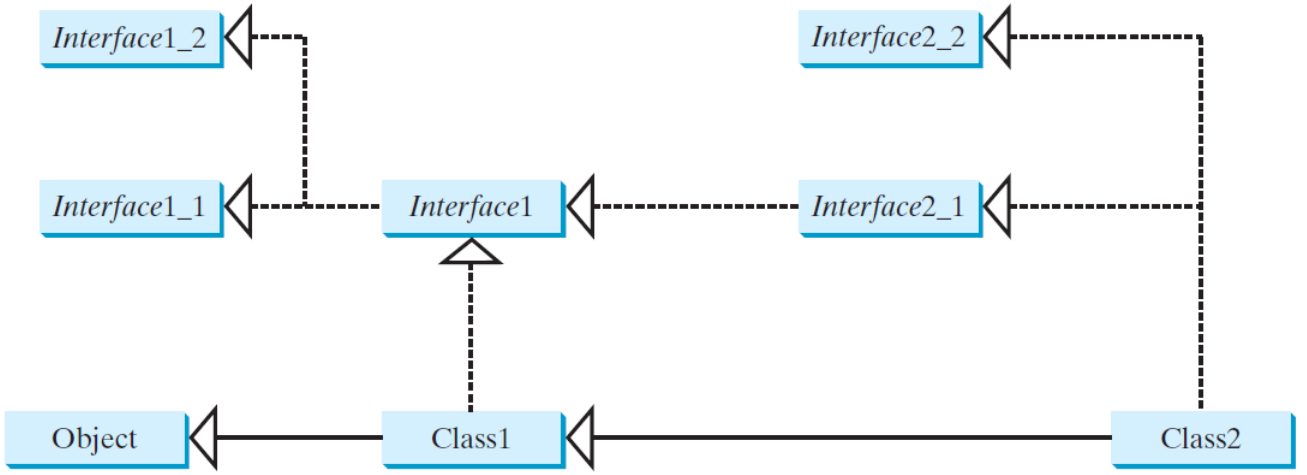
	Dữ liệu	Phương thức khởi tạo	Phương thức
Lớp trừu tượng	Không ràng buộc	- Được gọi thực thi bởi lớp con trong chuỗi khởi tạo. - Không thể khởi tạo đối tượng bằng toán tử new	Không ràng buộc
Interface	Tất cả phải là public static final	- Không có phương thức khởi tạo - Không thể khởi tạo đối tượng bằng toán tử new	Tất cả phải là phương thức trừu tượng public

Interface vs. Lớp trừu tượng

Tất cả các lớp đều có chung một gốc là lớp Object, nhưng không có gốc chung cho các interface.

Giống như một lớp, một interface cũng định nghĩa một kiểu dữ liệu. Một biến của một kiểu interface có thể tham chiếu đến bất kỳ thể hiện nào của lớp cài đặt interface đó.

Nếu một lớp kế thừa một interface thì interface này đóng vai trò như một lớp cha. Chúng ta có thể sử dụng một interface như một kiểu dữ liệu và ép kiểu một biết của một kiểu interface thành lớp con của nó.



Giả sử *c* là một thể hiện của **Class2**. *c* cũng là một thể hiện của **Object**, **Class1**, **Interface1**, **Interface1_1**, **Interface1_2**, **Interface2_1** và **Interface2_2**.

Lưu ý: đụng độ các interface

Trong một số ít các trường hợp, một lớp có thể cài đặt 2 interface có các thông tin đụng độ nhau *(vd: 2 hằng số giống nhau nhưng có giá trị khác nhau hoặc 2 phương thức có cùng chữ ký nhưng khác nhau kiểu dữ liệu trả về)*. Loại lỗi này sẽ được phát hiện bởi trình biên dịch.

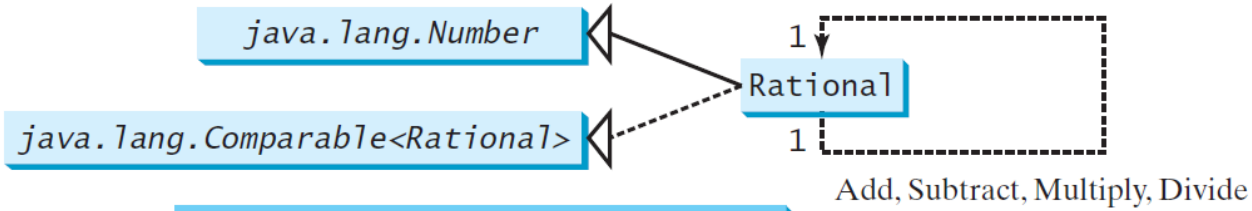
Sử dụng interface hay lớp?

- Một mối quan hệ **is-a** mô tả một cách rõ ràng mối quan hệ **parent-child** ➔ có thể được mô hình hóa bằng **lớp**. *Ví dụ: một* ***nhân viên*** *là một* ***con người.***

- Một mối quan hệ **is-a** yếu còn được gọi là mối quan hệ **is-kind-of** ➔ có thể được mô hình hóa bằng **interface**. *Ví dụ: tất cả chuỗi đều có thể so sánh được, nên lớp* ***String*** *cài đặt interface* ***Comparable***.

- Cũng có thể sử dụng để phá vỡ ràng buộc đơn kế thừa nếu đa kế thừa được yêu cầu. ➔ Trong trường hợp đa kế thừa, phải có 1 lớp cha và tất cả còn lại là các interface.

Lớp `Rational`



Rational

```
-numerator: long
-denominator: long
+Rational()
+Rational(numerator: long,
  denominator: long)
+getNumerator(): long
+getDenominator(): long
+add(secondRational: Rational):
  Rational
+subtract(secondRational:
  Rational): Rational
+multiply(secondRational:
  Rational): Rational
+divide(secondRational:
  Rational): Rational
+toString(): String
-gcd(n: long, d: long): long
```

- Tử số
- Mẫu số
- Tạo một phân số với tử số = 0 và mẫu số = 1
- Tạo một phân số với tử số và mẫu số xác định

- Lấy giá trị tử số của phân số
- Lấy giá trị mẫu số của phân số
- Cộng phân số hiện hành với một phân số khác
- Trừ phân số hiện hành với một phân số khác

- Nhân phân số hiện hành với một phân số khác
- Chia phân số hiện hành với một phân số khác

- Trả về chuỗi theo định dạng **"tử số / mẫu số"**. (Chỉ trả về tử số nếu mẫu số = 1)
- Trả về ước số chung lớn nhất của 2 số n và d