Chapter 13 Abstract Classes and Interfaces

Exercise 07 编程练习题(12.17 提交)

题一：13.15

*13.15 (在 Rational 类中使用 BigInteger) 使用 BigInteger 表示分子和分母，重新设计和实现 13.13 节中的 Rational 类。

题2：13.17

*13.17 (数学：Complex 类) 一个复数是一个形式为 $a+bi$ 的数，这里的 a 和 b 都是实数，i 是 $\sqrt{-1}$ 的平方根。数字 a 和 b 分别称为复数的实部和虚部。可以使用下面的公式完成复数的加、减、乘、除：

$$a + bi + c + di = (a + c) + (b + d)i$$

$$a + bi - (c + di) = (a - c) + (b - d)i$$

$$(a + bi)*(c + di) = (ac - bd) + (bc + ad)i$$

$$(a + bi)/(c + di) = (ac + bd)/(c^2 + d^2) + (bc - ad)i/(c^2 + d^2)$$

还可以使用下面的公式得到复数的绝对值：

$$|a+bi| = \sqrt{a^2+b^2}$$

(复数可以解释为一个平面上的点，将 (a, b) 值作为该点的坐标。复数的绝对值是该点到原点的距离，如图 13-10b 所示。)

设计一个名为 Complex 的复数来表示复数以及完成复数运算的 add、substract、multiply、divide 和 abs 方法，并且覆盖 toString 方法以返回一个表示复数的字符串。方法 toString 返回字符串 a+bi。如果 b 是 0，那么它只返回 a。Complex 类应该也实现 Cloneable 接口。

提供三个构造方法 Complex(a,b)、Complex(a) 和 Complex()。Complex() 创建数字 0 的 Complex 对象，而 Complex(a) 创建一个 b 为 0 的 Complex 对象。还提供 getRealPart() 和 getImaginaryPart() 方法以分别返回复数的实部和虚部。

编写一个测试程序，提示用户输入两个复数，然后显示它们做加、减、乘、除之后的结果。下面是一个运行示例：

```
Enter the first complex number: 3.5 5.5 ↵Enter
Enter the second complex number: -3.5 1 ↵Enter
(3.5 + 5.5i) + (-3.5 + 1.0i) = 0.0 + 6.5i
(3.5 + 5.5i) - (-3.5 + 1.0i) = 7.0 + 4.5i
(3.5 + 5.5i) * (-3.5 + 1.0i) = -17.75 + -13.75i
(3.5 + 5.5i) / (-3.5 + 1.0i) = -0.5094 + -1.7i
|(3.5 + 5.5i)| = 6.519202405202649
```

Exercise 07 编程练习题(12.17 提交）

题三：13.19

13.19 （将十进制数转化为分数）编写一个程序，提示用户输入一个十进制数，然后以分数的形式显示该数字。提示：将十进制数以字符串的形式读入，从字符串中抽取其整数部分和小数部分，然后运用编程练习题 13.15 中使用 BigInteger 实现的 Rational 类，来获得该十进制数的有理数。这里是一些运行示例：

```
Enter a decimal number: 3.25 ↵Enter
The fraction number is 13/4
```

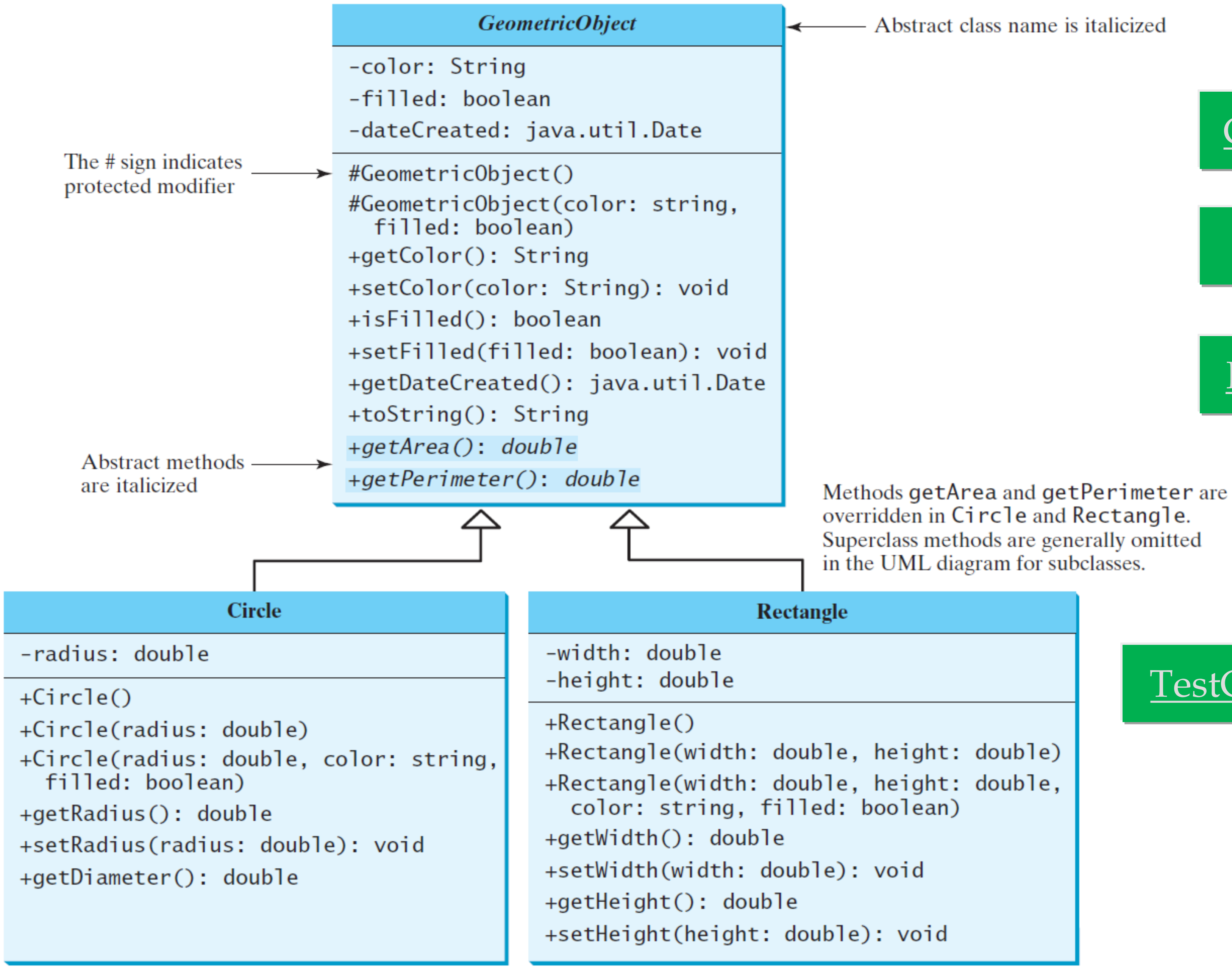
```
Enter a decimal number: -0.45452 ↵Enter
The fraction number is -11363/25000
```

题四：（搜索web）修改程序清单ch12/WebCrawler.java，从网址 http://cs.Armstrong.edu/liang 开始搜索某个单词（例如Computer Programming）。你的程序提示输入单词以及起始URL，并且一旦搜索到该单词则终止程序。显示包含了单词的页面的URL地址。

Objectives

- To design and use abstract classes (§ 13.2).
- To generalize numeric wrapper classes, **BigInteger**, and **BigDecimal** using the abstract **Number** class (§ 13.3).
- To process a calendar using the **Calendar** and **GregorianCalendar** classes (§ 13.4).
- To specify common behavior for objects using interfaces (§ 13.5).
- To define interfaces and define classes that implement interfaces (§ 13.5).
- To define a natural order using the **Comparable** interface (§ 13.6).
- To make objects cloneable using the **Cloneable** interface (§ 13.7).
- To explore the similarities and differences among concrete classes, abstract classes, and interfaces (§ 13.8).
- To design the **Rational** class for processing rational numbers (§ 13.9).
- To design classes that follow the class-design guidelines (§ 13.10).

Abstract Classes and Abstract Methods



GeometricObject

Circle

Rectangle

TestGeometricObject

abstract method in abstract class

An abstract method cannot be contained in a nonabstract class. If a subclass of an abstract superclass does not implement all the abstract methods, the subclass must be defined abstract. In other words, in a nonabstract subclass extended from an abstract class, all the abstract methods must be implemented, even if they are not used in the subclass.

object cannot be created from abstract class

An abstract class cannot be instantiated using the new operator, but you can still define its constructors, which are invoked in the constructors of its subclasses. For instance, the constructors of GeometricObject are invoked in the Circle class and the Rectangle class.

abstract class without abstract method

A class that contains abstract methods must be abstract. However, it is possible to define an abstract class that contains no abstract methods. In this case, you cannot create instances of the class using the new operator. This class is used as a base class for defining a new subclass.

superclass of abstract class may be concrete

A subclass can be abstract even if its superclass is concrete. For example, the Object class is concrete, but its subclasses, such as GeometricObject, may be abstract.

concrete method overridden to be abstract

A subclass can override a method from its superclass to define it abstract. This is rare, but useful when the implementation of the method in the superclass becomes invalid in the subclass. In this case, the subclass must be defined abstract.

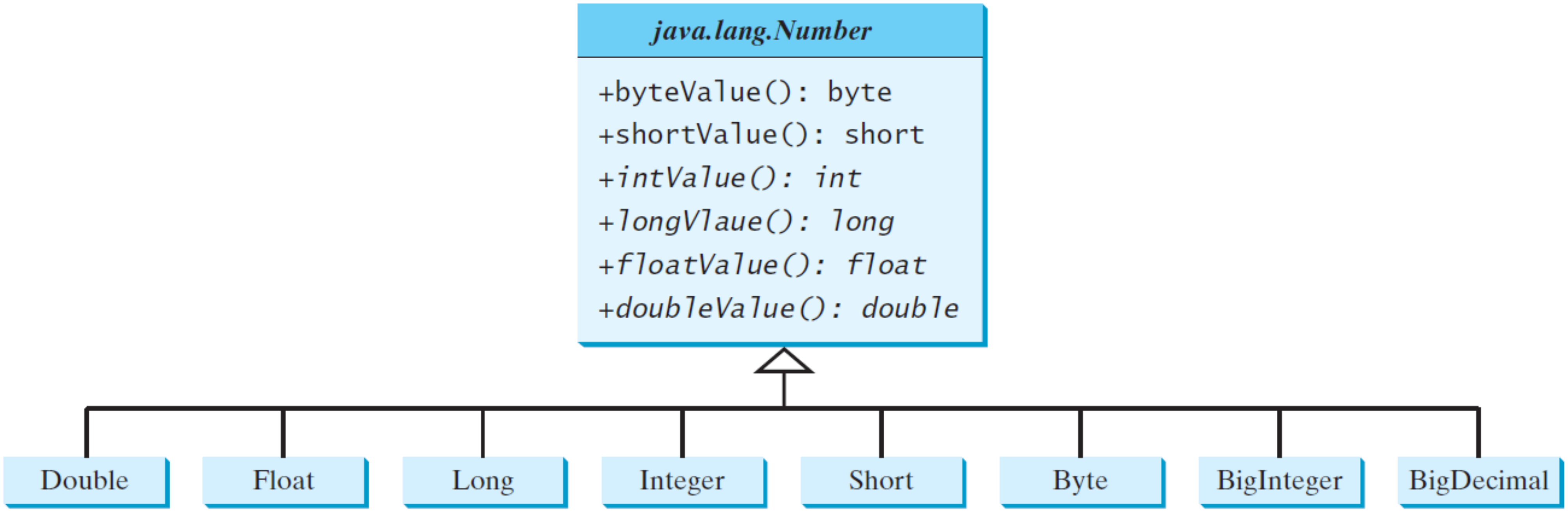
abstract class as type

You cannot create an instance from an abstract class using the new operator, but an abstract class can be used as a data type. Therefore, the following statement, which creates an array whose elements are of GeometricObject type, is correct.

```
GeometricObject[] geo = new    GeometricObject[10];

GeometricObject geo = new    Circle(1);
```

Case Study: the Abstract Number Class



LargestNumbers

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 date.
`+GregorianCalendar(year: int, month: int, dayOfMonth: int, hour:int, minute: int, second: int)`	Constructs a `GregorianCalendar` for the specified year, month, date, 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.

The get Method in Calendar Class

The get(int field) method defined in the Calendar class is useful to extract the date and time information from a Calendar object. The fields are defined as constants, as shown in the following.

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).

Getting Date/Time Information from Calendar

TestCalendar

Interfaces

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.

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();
}
```

Interface is a Special Class

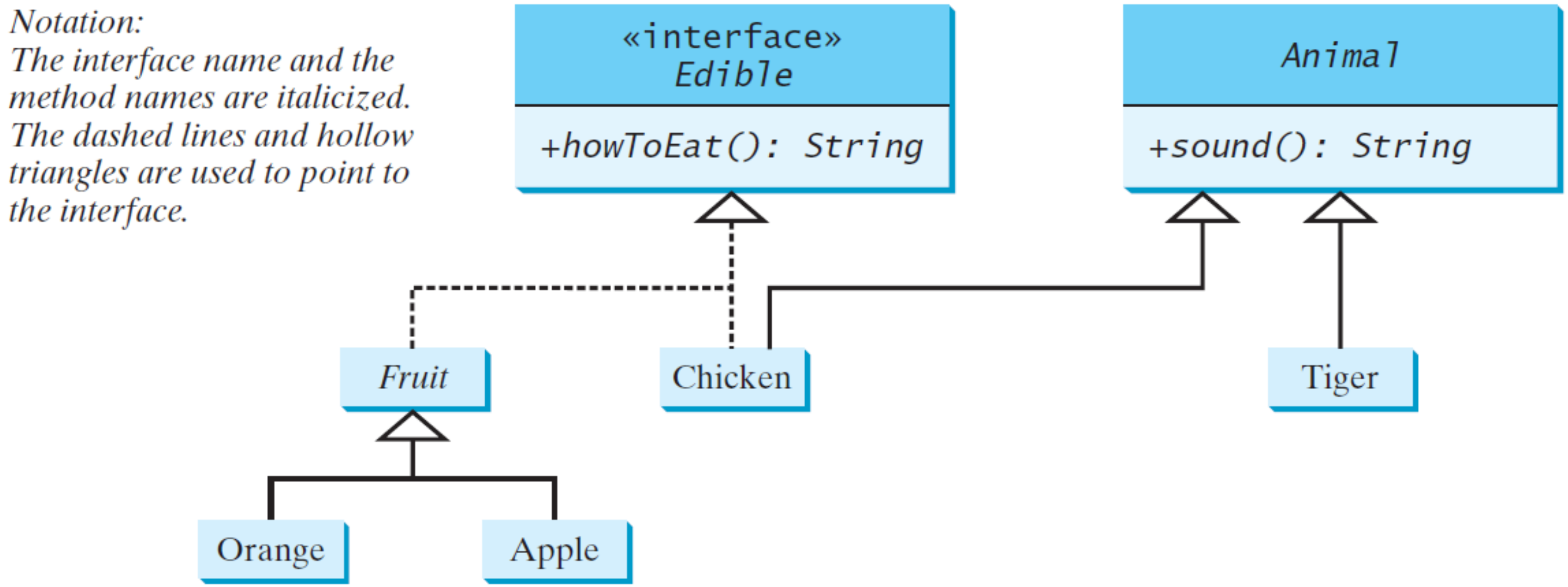
An interface is treated like a special class in Java. Each interface is compiled into a separate bytecode file, just like a regular class. Like an abstract class, you cannot create an instance from an interface using the new operator, but in most cases you can use an interface more or less the same way 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

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();
}
```

Equivalent

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

  void p();
}
```

A constant defined in an interface can be accessed using syntax InterfaceName.CONSTANT_NAME

Example: The Comparable Interface

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

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

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.

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
  }
}
```

Example

```
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));
```

Generic `sort` Method

Let **n** be an **Integer** object, **s** be a **String** object, and **d** be a **Date** object. All the following expressions are **true**.

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

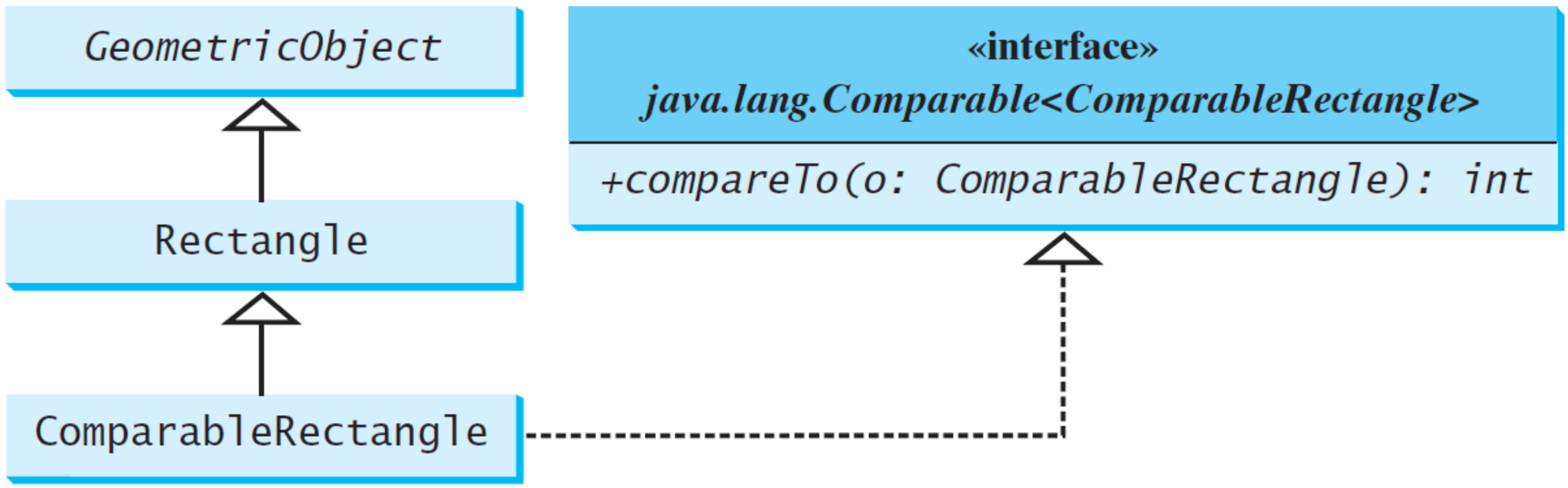
```
s instanceof String
s instanceof Object
s instanceof Comparable
```

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

The java.util.Arrays.sort(array) method requires that the elements in an array are instances of Comparable<E>.

SortComparableObjects

Defining Classes to Implement Comparable



ComparableRectangle

SortRectangles

Comparator Interface

Call Back

TimerTest

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.

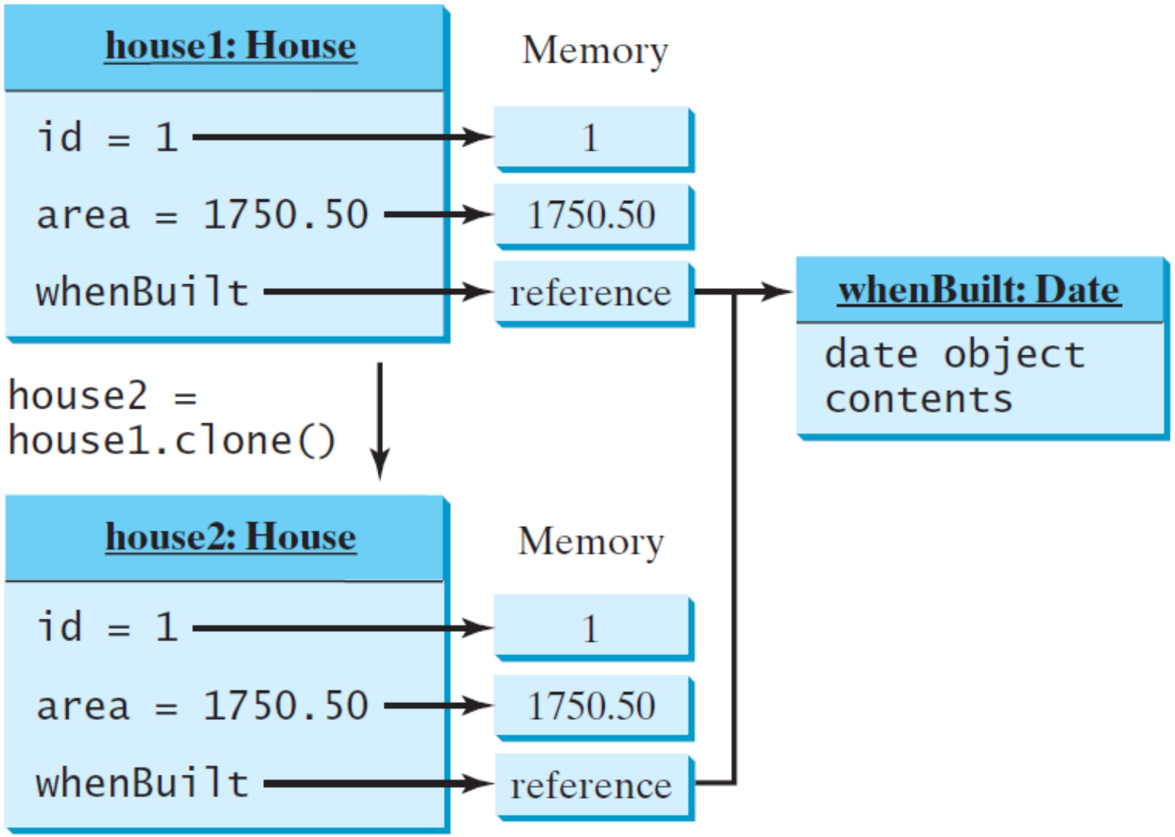
House

Shallow vs. Deep Copy

House house1 = new House(1, 1750.50);

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

Shallow Copy



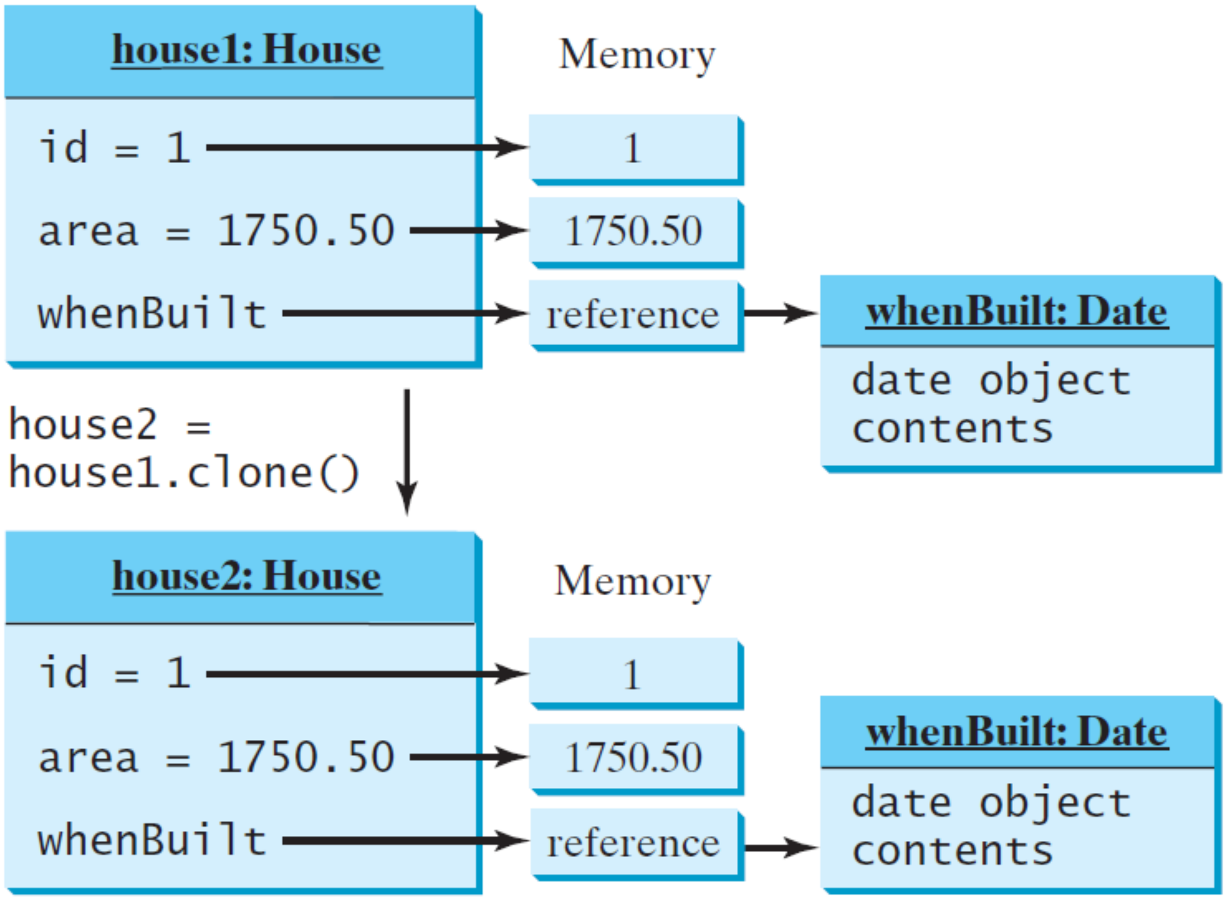
(a)

Shallow vs. Deep Copy

House house1 = new House(1, 1750.50);

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

Deep Copy



(b)

Interfaces vs. Abstract Classes

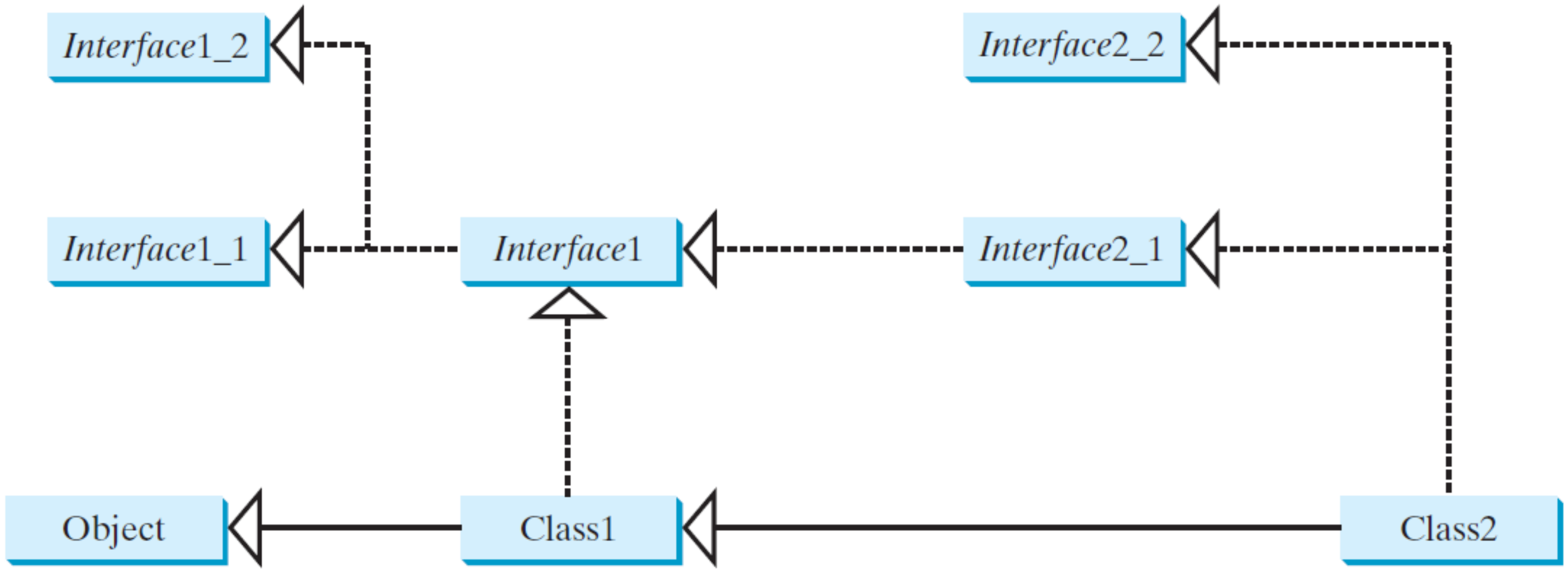
In an interface, the data must be constants; an abstract class can have all types of data.

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 implements 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.

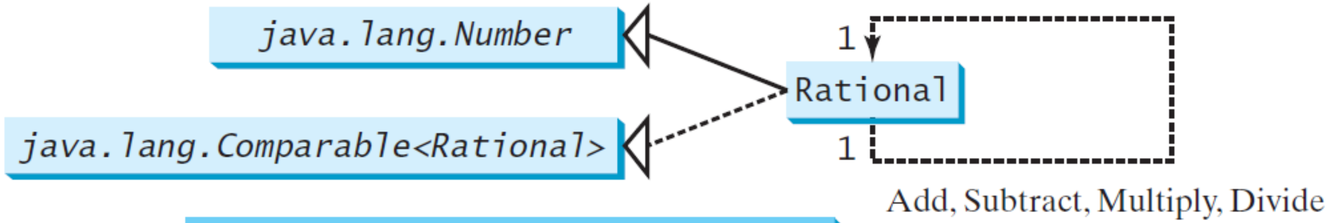
Caution: conflict interfaces

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.

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.

The `Rational` Class



Rational	
-numerator: long	The numerator of this rational number.
-denominator: long	The denominator of this rational number.
+Rational()	Creates a rational number with numerator 0 and denominator 1.
+Rational(numerator: long, denominator: long)	Creates a rational number with a specified numerator and denominator.
+getNumerator(): long	Returns the numerator of this rational number.
+getDenominator(): long	Returns the denominator of this rational number.
+add(secondRational: Rational): Rational	Returns the addition of this rational number with another.
+subtract(secondRational: Rational): Rational	Returns the subtraction of this rational number with another.
+multiply(secondRational: Rational): Rational	Returns the multiplication of this rational number with another.
+divide(secondRational: Rational): Rational	Returns the division of this rational number with another.
+toString(): String	Returns a string in the form “numerator/denominator.” Returns the numerator if denominator is 1.
-gcd(n: long, d: long): long	Returns the greatest common divisor of n and d.

Rational

TestRationalClass

Exercise 07 编程练习题(12.17 提交）

题一：13.15

*13.15 （在 Rational 类中使用 BigInteger）使用 BigInteger 表示分子和分母，重新设计和实现 13.13 节中的 Rational 类。

题2：13.17

*13.17 （数学：Complex 类）一个复数是一个形式为 $a+bi$ 的数，这里的 a 和 b 都是实数，i 是 $\sqrt{-1}$ 的平方根。数字 a 和 b 分别称为复数的实部和虚部。可以使用下面的公式完成复数的加、减、乘、除：

$$a+bi+c+di=(a+c)+(b+d)i$$

$$a+bi-(c+di)=(a-c)+(b-d)i$$

$$(a+bi)*(c+di)=(ac-bd)+(bc+ad)i$$

$$(a+bi)/(c+di)=(ac+bd)/(c^2+d^2)+(bc-ad)i/(c^2+d^2)$$

还可以使用下面的公式得到复数的绝对值：

$$|a+bi|=\sqrt{a^2+b^2}$$

（复数可以解释为一个平面上的点，将（a, b）值作为该点的坐标。复数的绝对值是该点到原点的距离，如图 13-10b 所示。）

设计一个名为 Complex 的复数来表示复数以及完成复数运算的 add、substract、multiply、divide 和 abs 方法，并且覆盖 toString 方法以返回一个表示复数的字符串。方法 toString 返回字符串 a+bi。如果 b 是 0，那么它只返回 a。Complex 类应该也实现 Cloneable 接口。

提供三个构造方法 Complex(a,b)、Complex(a) 和 Complex()。Complex() 创建数字 0 的 Complex 对象，而 Complex(a) 创建一个 b 为 0 的 Complex 对象。还提供 getRealPart() 和 getImaginaryPart() 方法以分别返回复数的实部和虚部。

编写一个测试程序，提示用户输入两个复数，然后显示它们做加、减、乘、除之后的结果。下面是一个运行示例：

```
Enter the first complex number: 3.5 5.5 ↵Enter
Enter the second complex number: -3.5 1 ↵Enter
(3.5 + 5.5i) + (-3.5 + 1.0i) = 0.0 + 6.5i
(3.5 + 5.5i) - (-3.5 + 1.0i) = 7.0 + 4.5i
(3.5 + 5.5i) * (-3.5 + 1.0i) = -17.75 + -13.75i
(3.5 + 5.5i) / (-3.5 + 1.0i) = -0.5094 + -1.7i
|(3.5 + 5.5i)| = 6.519202405202649
```

Exercise 07 编程练习题(12.17 提交）

题三：13.19

13.19 （将十进制数转化为分数）编写一个程序，提示用户输入一个十进制数，然后以分数的形式显示该数字。提示：将十进制数以字符串的形式读入，从字符串中抽取其整数部分和小数部分，然后运用编程练习题 13.15 中使用 BigInteger 实现的 Rational 类，来获得该十进制数的有理数。这里是一些运行示例：

```
Enter a decimal number: 3.25
The fraction number is 13/4
```

```
Enter a decimal number: -0.45452
The fraction number is -11363/25000
```

题四：（搜索web）修改程序清单ch12/WebCrawler.java，从网址http://cs.Armstrong.edu/liang 开始搜索某个单词（例如Computer Programming）。你的程序提示输入单词以及起始URL，并且一旦搜索到该单词则终止程序。显示包含了单词的页面的URL地址。