13.2.2 The `getArea()` and `getPerimeter()` methods may be removed from the `GeometricObject` class. What are the benefits of defining `getArea()` and `getPerimeter()` as abstract methods in the `GeometricObject` class?

13.2.3 True or false?

a. An abstract class can be used just like a nonabstract class except that you cannot use the `new` operator to create an instance from the abstract class.

b. An abstract class can be extended.

c. A subclass of a nonabstract superclass cannot be abstract.

d. A subclass cannot override a concrete method in a superclass to define it as abstract.

e. An abstract method must be nonstatic.

13.3 Case Study: The Abstract `Number` Class

`Number` is an abstract superclass for numeric wrapper classes `BigInteger` and `BigDecimal`.

Section 10.7 introduced numeric wrapper classes and Section 10.9 introduced the `BigInteger` and `BigDecimal` classes. These classes have common methods `byteValue()`, `shortValue()`, `intValue()`, `longValue()`, `floatValue()`, and `doubleValue()` for returning a `byte`, `short`, `int`, `long`, `float`, and `double` value from an object of these classes. These common methods are actually defined in the `Number` class, which is a superclass for the numeric wrapper classes `BigInteger` and `BigDecimal`, as shown in Figure 13.2.

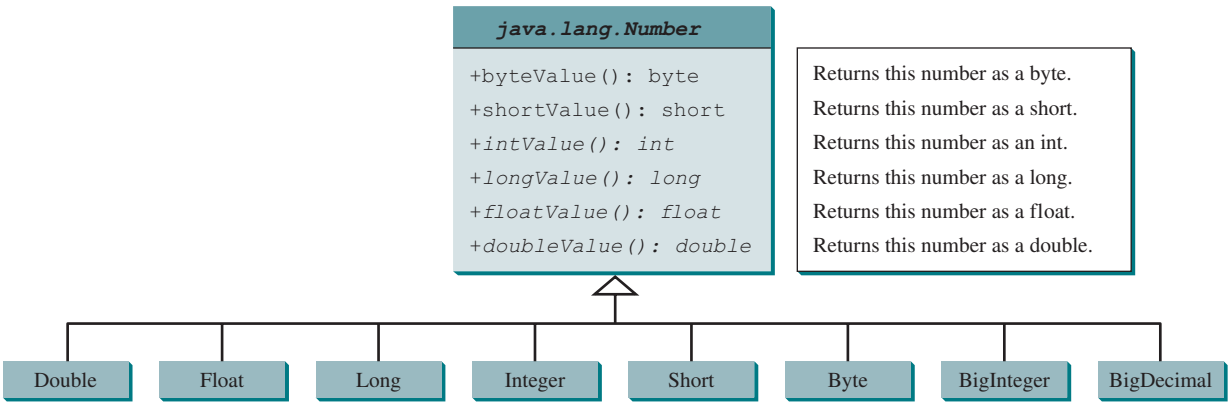

FIGURE 13.2 The `Number` class is an abstract superclass for `Double`, `Float`, `Long`, `Integer`, `Short`, `Byte`, `BigInteger`, and `BigDecimal`.

Since the `intValue()`, `longValue()`, `floatValue()`, and `doubleValue()` methods cannot be implemented in the `Number` class, they are defined as abstract methods in the `Number` class. The `Number` class is therefore an abstract class. The `byteValue()` and `shortValue()` methods are implemented from the `intValue()` method as follows:

```
public byte byteValue() {
  return (byte)intValue();
}

public short shortValue() {
  return (short)intValue();
}
```

With `Number` defined as the superclass for the numeric classes, we can define methods to perform common operations for numbers. Listing 13.5 gives a program that finds the largest number in a list of `Number` objects.

LISTING 13.5 LargestNumber.java

```
1  import java.util.ArrayList;
2  import java.math.*;
3
4  public class LargestNumber {
5    public static void main(String[] args) {
6      ArrayList<Number> list = new ArrayList<>();
7      list.add(45); // Add an integer
8      list.add(3445.53); // Add a double
9      // Add a BigInteger
10     list.add(new BigInteger("3432323234344343101"));
11     // Add a BigDecimal
12     list.add(new BigDecimal("2.0909090989091343433344343"));
13
14     System.out.println("The largest number is " +
15       getLargestNumber(list));
16   }
17
18   public static Number getLargestNumber(ArrayList<Number> list) {
19     if (list == null || list.size() == 0)
20       return null;
21
22     Number number = list.get(0);
23     for (int i = 1; i < list.size(); i++)
24       if (number.doubleValue() < list.get(i).doubleValue())
25         number = list.get(i);
26
27     return number;
28   }
29 }
```

create an array list
add number to list
invoke getLargestNumber
doubleValue

```
The largest number is 3432323234344343101
```

The program creates an `ArrayList` of `Number` objects (line 6). It adds an `Integer` object, a `Double` object, a `BigInteger` object, and a `BigDecimal` object to the list (lines 7–12). Note `45` is automatically converted into an `Integer` object and added to the list in line 7, and `3445.53` is automatically converted into a `Double` object and added to the list in line 8 using autoboxing.

Invoking the `getLargestNumber` method returns the largest number in the list (line 15). The `getLargestNumber` method returns `null` if the list is `null` or the list size is `0` (lines 19 and 20). To find the largest number in the list, the numbers are compared by invoking their `doubleValue()` method (line 24). The `doubleValue()` method is defined in the `Number` class and implemented in the concrete subclass of `Number`. If a number is an `Integer` object, the `Integer`'s `doubleValue()` is invoked. If a number is a `BigDecimal` object, the `BigDecimal`'s `doubleValue()` is invoked.

If the `doubleValue()` method was not defined in the `Number` class, you will not be able to find the largest number among different types of numbers using the `Number` class.

13.3.1 Why do the following two lines of code compile but cause a runtime error?

```
Number numberRef = Integer.valueOf(0);
Double doubleRef = (Double)numberRef;
```

13.3.2 Why do the following two lines of code compile but cause a runtime error?

```
Number[] numberArray = Integer[2];
numberArray[0] = Double.valueOf(1.5);
```

13.3.3 Show the output of the following code:

```
public class Test {
  public static void main(String[] args) {
    Number x = 3;
    System.out.println(x.intValue());
    System.out.println(x.doubleValue());
  }
}
```

13.3.4 What is wrong in the following code? (Note the `compareTo` method for the `Integer` and `Double` classes was introduced in Section 10.7.)

```
public class Test {
  public static void main(String[] args) {
    Number x = Integer.valueOf(3);
    System.out.println(x.intValue());
    System.out.println(x.compareTo(4));
  }
}
```

13.3.5 What is wrong in the following code?

```
public class Test {
  public static void main(String[] args) {
    Number x = Integer.valueOf(3);
    System.out.println(x.intValue());
    System.out.println((Integer)x.compareTo(4));
  }
}
```

13.4 Case Study: `Calendar` and `GregorianCalendar`

`GregorianCalendar` is a concrete subclass of the abstract class `Calendar`.

VideoNote
Calendar and GregorianCalendar classes

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 calendar information, such as the year, month, date, hour, minute, and second. Subclasses of `Calendar` can implement specific calendar systems, such as the Gregorian calendar, the lunar calendar, and the Jewish calendar. Currently, `java.util.GregorianCalendar` for the Gregorian calendar is supported in Java, as shown in Figure 13.3. The `add` method is abstract in the `Calendar` class because its implementation is dependent on a concrete calendar system.

abstract `add` method

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

construct calendar

The `get(int field)` method defined in the `Calendar` class is useful for extracting the date and time information from a `Calendar` object. The fields are defined as constants, as shown in Table 13.1.

`get(field)`

Listing 13.6 gives an example that displays the date and time information for the current time.