9.5.1 Is an array an object or a primitive-type value? Can an array contain elements of an object type? Describe the default value for the elements of an array.

9.5.2 Which operator is used to access a data field or invoke a method from an object?

9.5.3 What is an anonymous object?

9.5.4 What is `NullPointerException`?

9.5.5 What is wrong with each of the following programs?

```java
public class ShowErrors {
  public static void main(String[] args) {
    ShowErrors t = new ShowErrors();
    t.x();
  }
}
```

(a)

```java
public class ShowErrors {
  public void method1() {
    Circle c;
    System.out.println("What is radius "
      + c.getRadius());
    c = new Circle();
  }
}
```

(b)

9.5.6 What is the output of the following code?

```java
public class A {
  boolean x;

  public static void main(String[] args) {
    A a = new A();
    System.out.println(a.x);
  }
}
```

9.6 Using Classes from the Java Library

The Java API contains a rich set of classes for developing Java programs.

Listing 9.1 defined the `Circle` class and created objects from the class. You will frequently use the classes in the Java library to develop programs. This section gives some examples of the classes in the Java library.

9.6.1 The `Date` Class

In Listing 2.7, ShowCurrentTime.java, you learned how to obtain the current time using `System.currentTimeMillis()`. You used the division and remainder operators to extract the current second, minute, and hour. Java provides a system-independent encapsulation of date and time in the `java.util.Date` class, as shown in Figure 9.10.

java.util.Date class

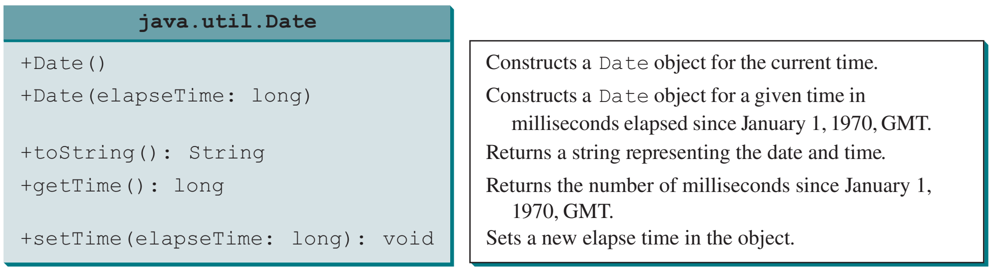

java.util.Date
+Date()
+Date(elapseTime: long)
+toString(): String
+getTime(): long
+setTime(elapseTime: long): void
Constructs a Date object for the current time.
Constructs a Date object for a given time in milliseconds elapsed since January 1, 1970, GMT.
Returns a string representing the date and time.
Returns the number of milliseconds since January 1, 1970, GMT.
Sets a new elapse time in the object.

FIGURE 9.10 A `Date` object represents a specific date and time.

You can use the no-arg constructor in the **Date** class to create an instance for the current date and time, the **getTime()** method to return the elapsed time in milliseconds since January 1, 1970, GMT, and the **toString()** method to return the date and time as a string, For example, the following code:

```
java.util.Date date = new java.util.Date();
System.out.println("The elapsed time since Jan 1, 1970 is " +
  date.getTime() + " milliseconds");
System.out.println(date.toString());
```

create object

get elapsed time

invoke toString

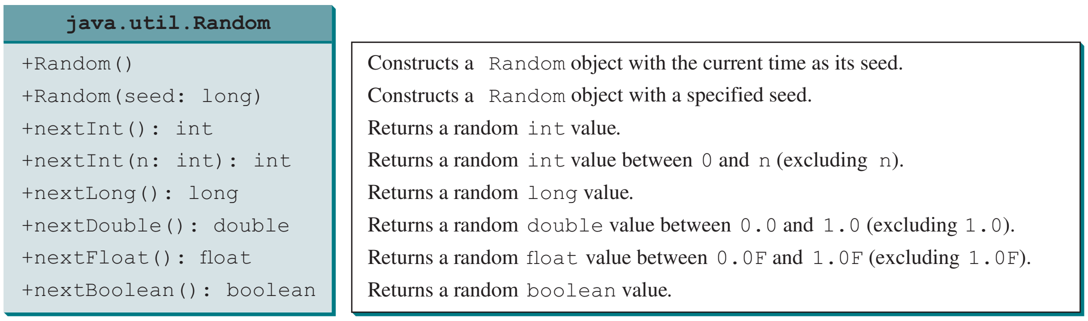
displays the output as follows:

```
The elapsed time since Jan 1, 1970 is 1324903419651 milliseconds
Mon Dec 26 07:43:39 EST 2011
```

The **Date** class has another constructor, **Date(long elapseTime)**, which can be used to construct a **Date** object for a given time in milliseconds elapsed since January 1, 1970, GMT.

9.6.2 The Random Class

You have used **Math.random()** to obtain a random **double** value between **0.0** and **1.0** (excluding **1.0**). Another way to generate random numbers is to use the **java.util.Random** class, as shown in Figure 9.11, which can generate a random **int**, **long**, **double**, **float**, and **boolean** value.



java.util.Random	
+Random()	Constructs a Random object with the current time as its seed.
+Random(seed: long)	Constructs a Random object with a specified seed.
+nextInt(): int	Returns a random int value.
+nextInt(n: int): int	Returns a random int value between 0 and n (excluding n).
+nextLong(): long	Returns a random long value.
+nextDouble(): double	Returns a random double value between 0.0 and 1.0 (excluding 1.0).
+nextFloat(): float	Returns a random float value between 0.0F and 1.0F (excluding 1.0F).
+nextBoolean(): boolean	Returns a random boolean value.

FIGURE 9.11 A **Random** object can be used to generate random values.

When you create a **Random** object, you have to specify a seed or use the default seed. A seed is a number used to initialize a random number generator. The no-arg constructor creates a **Random** object using the current elapsed time as its seed. If two **Random** objects have the same seed, they will generate identical sequences of numbers. For example, the following code creates two **Random** objects with the same seed, **3**:

```
Random generator1 = new Random(3);
System.out.print("From generator1: ");
for (int i = 0; i < 10; i++)
  System.out.print(generator1.nextInt(1000) + " ");

Random generator2 = new Random(3);
System.out.print("\nFrom generator2: ");
for (int i = 0; i < 10; i++)
  System.out.print(generator2.nextInt(1000) + " ");
```

The code generates the same sequence of random **int** values:

```
From generator1: 734 660 210 581 128 202 549 564 459 961
From generator2: 734 660 210 581 128 202 549 564 459 961
```