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();                    create object
System.out.println("The elapsed time since Jan 1, 1970 is " +
  date.getTime() + " milliseconds");                           get elapsed time
System.out.println(date.toString());                           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
```

same sequence

Note

The ability to generate the same sequence of random values is useful in software testing and many other applications. In software testing, often you need to reproduce the test cases from a fixed sequence of random numbers.

SecureRandom

Note

You can generate random numbers using the `java.security.SecureRandom` class rather than the `Random` class. The random numbers generated from the `Random` are deterministic and they can be predicted by hackers. The random numbers generated from the `SecureRandom` class are nondeterministic and are secure.

9.6.3 The `Point2D` Class

Java API has a convenient `Point2D` class in the `javafx.geometry` package for representing a point in a two-dimensional plane. The UML diagram for the class is shown in Figure 9.12.

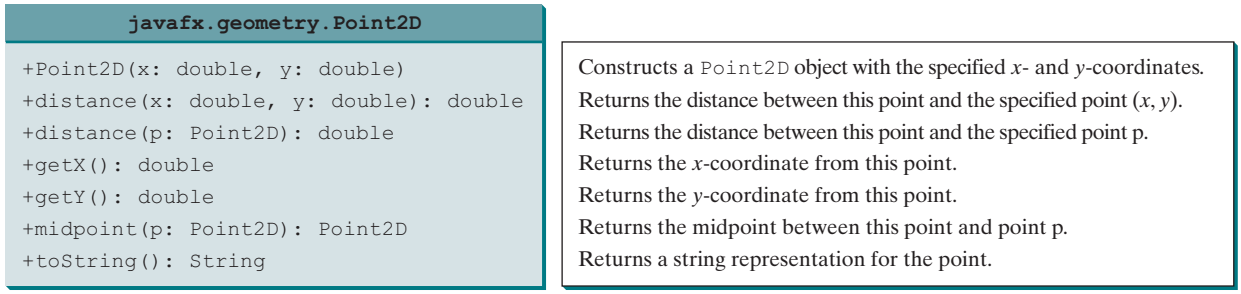

FIGURE 9.12 A `Point2D` object represents a point with *x*- and *y*-coordinates.

You can create a `Point2D` object for a point with the specified *x*- and *y*-coordinates, use the `distance` method to compute the distance from this point to another point, and use the `toString()` method to return a string representation of the point. Listing 9.5 gives an example of using this class.

LISTING 9.5 TestPoint2D.java

```
1  import java.util.Scanner;
2  import javafx.geometry.Point2D;
3
4  public class TestPoint2D {
5    public static void main(String[] args) {
6      Scanner input = new Scanner(System.in);
7
8      System.out.print("Enter point1's x-, y-coordinates: ");
9      double x1 = input.nextDouble();
10     double y1 = input.nextDouble();
11     System.out.print("Enter point2's x-, y-coordinates: ");
12     double x2 = input.nextDouble();
13     double y2 = input.nextDouble();
14
15     Point2D p1 = new Point2D(x1, y1);
16     Point2D p2 = new Point2D(x2, y2);
17     System.out.println("p1 is " + p1.toString());
18     System.out.println("p2 is " + p2.toString());
19     System.out.println("The distance between p1 and p2 is " +
20       p1.distance(p2));
21     System.out.println("The midpoint between p1 and p2 is " +
22       p1.midpoint(p2).toString());
23   }
24 }
```

create an object

invoke `toString()`

get distance

get midpoint