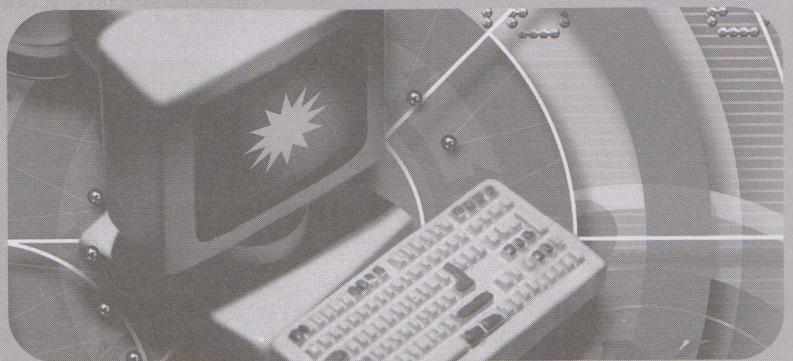


CAPÍTULO 11



Cálculos

En este capítulo conoceremos cómo:

- Utilizar las funciones de la biblioteca matemática.
- Aplicar formato a los números para mostrarlos de una manera conveniente.
- Llevar a cabo cálculos comerciales y científicos.

● Introducción

Ya vimos en el capítulo 4 cómo llevar a cabo cálculos simples. Este capítulo trata sobre cálculos más serios. Mejora la explicación anterior y reúne toda la información necesaria para escribir programas que lleven a cabo cálculos. Si no le interesan los programas que realizan cálculos numéricos, puede pasar al siguiente capítulo.

Los cálculos surgen en muchos programas, no sólo en los que realizan cálculos matemáticos, científicos o de ingeniería, sino también en los sistemas de información, la contabilidad y los pronósticos. En los gráficos, los cálculos son necesarios para escalar y desplazar imágenes en la pantalla.

En el capítulo 4 explicamos varias ideas importantes sobre los números y los cálculos. Tal vez quiera repasar ese capítulo antes de continuar. Estas ideas son:

- Declaración de variables como `int` o `double`.
- Entrada y salida de datos mediante campos de texto.
- Conversión entre las representaciones de cadena de los números y sus representaciones internas.
- Reglas de precedencia en las expresiones.
- Conversiones en expresiones que mezclan datos `int` y `double`.

● Funciones y constantes de la biblioteca matemática

Es común en los programas matemáticos, científicos o de ingeniería utilizar funciones como seno, coseno y logaritmo. En Java estas funciones se incluyen en una de las bibliotecas: la biblioteca **Math**. Para usar una de las funciones podemos escribir algo como lo siguiente:

```
x = Math.sqrt(y);
```

A continuación le mostraremos algunas de las funciones más utilizadas en la biblioteca **Math**. Cuando el parámetro es un ángulo, se debe expresar en radianes.

<code>cos(x)</code>	coseno del ángulo x , en donde x se expresa en radianes
<code>sin(x)</code>	seno del ángulo x , expresado en radianes
<code>tan(x)</code>	la tangente del ángulo x , expresada en radianes
<code>abs(x)</code>	el valor absoluto de x , que algunas veces se escribe como $ x $ en matemáticas.
<code>min(x, y)</code>	el menor de x y y
<code>max(x, y)</code>	el mayor de x y y
<code>log(x)</code>	logaritmo natural de x (en base e)
<code>random()</code>	provee un número seudoaleatorio en el rango de 0.0 a 0.999...
<code>sqrt(x)</code>	la raíz cuadrada positiva de x
<code>pow(x, y)</code>	x elevado a la potencia de y , o x^y
<code>exp(x)</code>	e^x
<code>round(x)</code>	el número entero más cercano a x

Al utilizar estos métodos, algunas veces debemos tener cuidado en cuanto al tipo de variables o literales que utilizamos como parámetros. Por ejemplo, el método `abs` puede recibir cualquier valor numérico, pero el método `cos` sólo puede recibir un número `double`.

Las constantes matemáticas π y e también están disponibles como constantes dentro de la biblioteca **Math**, así que podemos escribir lo siguiente:

```
double x, y;  
x = Math.PI;  
y = Math.E;
```

● Cómo aplicar formato a los números

Aplicar formato significa mostrar los números de una forma conveniente. Por ejemplo, no siempre requerimos el detalle de las posiciones decimales innecesarias. Si el valor `33.124765` representa el área de una habitación en metros cuadrados, entonces tal vez no sean necesarias todas las posiciones decimales, ya que por lo general sólo habría que mostrar el número como `33.12`.

Java cuenta con una variedad de herramientas para aplicar formato a los valores, pero aquí nos limitaremos a los casos más comunes en los que se aplicará formato a valores `int` y `double`. Los pasos son:

1. Crear una instancia de la clase de biblioteca `DecimalFormat` y suministrarle un patrón como parámetro.
2. Llamar al método `format` y suministrarle como parámetro el número al que se va a aplicar formato, para que devuelva el valor con formato como una cadena de texto.

Empezaremos con valores `int`. Suponga que tenemos el siguiente valor entero:

```
int i = 123;
```

Ya hemos utilizado antes el método `toString` en varias ocasiones para convertir un número. Por ejemplo:

```
campoTexto.setText(Integer.toString(i));
```

con esto obtenemos la siguiente cadena de texto:

```
123
```

Pero ahora utilizaremos el método `format` para aplicar formato y obtener el mismo resultado:

```
int i = 123;
DecimalFormat formato = new DecimalFormat("###");
campoTexto.setText(formato.format(i));
```

Con esto el campo de texto recibe el mismo resultado:

```
123
```

El carácter # dentro de un patrón (tal como el anterior) significa insertar un dígito. En este caso, si hay menos de tres dígitos, no se crea ningún carácter.

Cuando sabemos que un entero puede ser hasta de cinco caracteres, por ejemplo, y queremos alinear los números, podemos utilizar lo siguiente:

```
int i = 123;
DecimalFormat formato = new DecimalFormat("00000");
campoTexto.setText(formato.format(i));
```

con lo cual obtenemos:

```
00123
```

Esto aplica formato al número utilizando exactamente cinco dígitos alineados a la derecha y se rellena con ceros a la izquierda según sea necesario.

Para números más grandes, las comas pueden mejorar la legibilidad del número. Esto se ilustra en el siguiente código:

```
int i = 123456;
DecimalFormat formato = new DecimalFormat("###,###");
campoTexto.setText(formato.format(i));
```

Aquí, el carácter , indica que el formato debe colocar una coma en esta posición. Con esto obtenemos la siguiente cadena:

```
123,456
```

Este formato tiende a ser más útil cuando se van a mostrar valores **double**. En este ejemplo:

```
double d = 12.34;
DecimalFormat formato = new DecimalFormat("##.##");
campoTexto.setText(formato.format(d));
```

el campo de texto recibe el siguiente valor:

12.34

Los dos caracteres **#** especifican dos dígitos después del punto decimal. A la izquierda del punto decimal se muestran los dígitos que sean necesarios para presentar todo el número. Pero por lo menos se muestra un dígito. A la derecha del punto decimal, el número se redondea si es necesario para ajustarlo a los dos dígitos especificados.

También se provee una herramienta para mostrar números **double** en notación científica, para lo cual utilizamos la letra **E** dentro de la información de formato:

```
double numero = 12300000;
DecimalFormat formato = new DecimalFormat("0.###E0");
campoTexto.setText(formato.format(numero));
```

mediante este código obtenemos lo siguiente:

1.23E7

Finalmente, podemos mostrar los números en formato de moneda utilizando el carácter **\$** dentro del patrón. Por ejemplo, si usamos el siguiente código:

```
double dinero = 12.34;
DecimalFormat formato = new DecimalFormat("$###.##");
campoTexto.setText(formato.format(dinero));
```

obtendremos lo siguiente:

\$12.34

en donde el símbolo de moneda (**\$** en este caso) se determina de acuerdo con la especificación local. El número siempre se muestra con dos dígitos después del punto decimal (se redondea si es necesario).

La siguiente tabla muestra un resumen de algunos patrones comunes.

Tipo de datos	Valor de ejemplo	Patrón	Cadena con formato
entero	123	###	123
entero con ceros a la izquierda	123	00000	00123
punto flotante	12.34	##.##	12.34
notación científica	12300000	0.####E0	1.23E7
moneda	12.34	\$###.##	\$12.34

Y la siguiente tabla sintetiza los caracteres de formato que se pueden utilizar para crear patrones:

Carácter	Significado
#	inserta un dígito si hay uno
0	siempre inserta un dígito
,	inserta una coma
.	inserta un punto decimal
E	inserta una E seguida de la potencia de 10
\$	inserta un signo de moneda

La clase `DecimalFormat` está dentro del paquete `java.text.DecimalFormat`, por lo que se necesita la siguiente instrucción `import` en el encabezado de un programa que la utilice:

```
import java.text.DecimalFormat;
```

PRÁCTICA DE AUTOEVALUACIÓN

- 11.1** Sabemos que el resultado de un cálculo será un número de punto flotante en el rango de 0 a 99, y deben aparecer dos dígitos después del punto decimal. Seleccione un formato apropiado.

Ejemplo práctico: dinero

Ahora vamos a rastrear el desarrollo de un programa para realizar cálculos con números. En la mayoría de los países el dinero consta de dos partes: dólares y centavos, euros y centavos, libras y peniques. Tenemos una opción: podemos representar un monto de dinero ya sea como cantidad `double` (como 20.25 dólares) o como `int` (2025 centavos). Si utilizamos centavos, tenemos que convertir las cantidades en dólares y centavos, y viceversa. En este caso utilizaremos variables `double` para representar valores.

Vamos a construir un programa que calcula el interés compuesto. Se invierte un monto a una tasa de interés anual específica y se acumula su valor. El usuario introduce la cantidad inicial (como número entero) y una tasa de interés (un número que puede tener un punto decimal) en campos de texto. Después el usuario hace clic en un botón para ver el monto acumulado por año, como se muestra en la figura 11.1.

Al hacer clic en el botón para pasar al siguiente año, el programa debe realizar el siguiente cálculo:

```
montoNuevo = montoAnterior + (montoAnterior * tasa / 100);
```

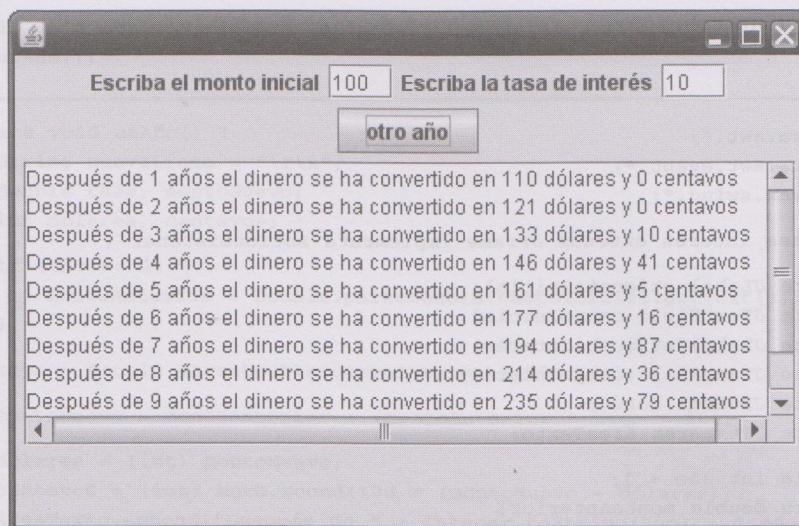


Figura 11.1 Cálculo de intereses.

Al mostrar un monto de dinero, necesitamos que aparezca un número entero de dólares y un número entero de centavos; por ejemplo, si el valor es 127.2341 dólares, necesitamos mostrarlo como 127 dólares y 23 centavos.

Primero veamos la parte de los dólares. Si utilizamos el operador de conversión (`int`) podemos convertir el número `double` en un `int`, truncando la parte fraccionaria:

```
dólares = (int) montoNuevo;
```

Ahora la parte de los centavos. Necesitamos deshacernos de la parte del número correspondiente a los dólares. Podemos restar el número entero de dólares, de manera que un número como 127.2341 se convertiría en 0.2341. Luego multiplicamos ese número por 100.0 para convertirlo en centavos, de manera que 0.2341 se convertiría en 23.41. A continuación usamos `Math.round` para convertirlo al número entero más cercano (23.0). Por último convertimos el valor `double` en un valor `int` mediante (`int`):

```
centavos = (int) Math.round(100 * (montoNuevo - dólares));
```

Ahora podemos mostrar los valores que se convirtieron en forma apropiada. Finalmente tenemos que:

```
montoAnterior = montoNuevo;
```

y esto es de lo que trata la inversión.

A nivel de clase, las declaraciones de instancias son:

```
private int año = 1;
private double montoAnterior;
```

He aquí el programa completo:

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Interés extends JFrame implements ActionListener {

    private JLabel etiquetaInicial;
    private JTextField campoInicial;
    private JLabel etiquetaInterés;
    private JTextField campoInterés;
    private JButton botón;
    private JTextArea áreaTexto;

    private int año = 1;
    private double montoAnterior;

    public static void main(String [] args) {
        Interés marco = new Interés();
        marco.setSize(460,300);
        marco.creargUI();
        marco.setVisible(true);
    }

    private void creargUI() {
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        Container ventana = getContentPane();
        ventana.setLayout(new FlowLayout());

        etiquetaInicial = new JLabel("Escriba el monto inicial");
        ventana.add(etiquetaInicial);

        campoInicial = new JTextField(3);
        ventana.add(campoInicial);

        etiquetaInterés = new JLabel("Escriba la tasa de interés");
        ventana.add(etiquetaInterés);

        campoInterés = new JTextField(3);
        ventana.add(campoInterés);

        botón = new JButton("otro año");
        ventana.add(botón);
        botón.addActionListener(this);

        áreaTexto = new JTextArea(10, 40);
        ventana.add(áreaTexto);

        JScrollPane panelDesplazable = new JScrollPane(áreaTexto);
        ventana.add(panelDesplazable);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == botón) {
            calcular();
        }
    }

    private void calcular() {
        double monto = Double.parseDouble(campoInicial.getText());
        double tasa = Double.parseDouble(campoInterés.getText());
        double resultado = monto * (tasa / 100) * año;
        String resultadoString = "El resultado es: " + resultado;
        áreaTexto.setText(resultadoString);
        año++;
    }
}

```

```

public void actionPerformed(ActionEvent event) {
    unAño();
}

private void unAño() {
    String nuevaLínea = "\r\n";
    double tasa, montoNuevo;
    int dólares, centavos;

    if (año == 1) {
        montoAnterior = Double.parseDouble(campoInicial.getText());
    }

    tasa = Double.parseDouble(campoInterés.getText());

    montoNuevo = montoAnterior + (montoAnterior * tasa / 100);

    dólares = (int) montoNuevo;
    centavos = (int) Math.round(100 * (montoNuevo - dólares));
    áreaTexto.append("Después de " + Integer.toString(año) + " años "
        + "el dinero se ha convertido en "
        + Integer.toString(dólares) + " dólares y "
        + Integer.toString(centavos) + " centavos" + nuevaLínea);

    montoAnterior = montoNuevo;
    año++;
}
}

```

Cabe mencionar que agregamos un panel desplazable (**JScrollPane**), para que el área de texto tenga una barra de desplazamiento.

Ejemplo práctico – iteración

En la programación numérica es muy común escribir iteraciones —ciclos que continúan buscando una solución a una ecuación hasta encontrarla con una precisión adecuada.

Como ejemplo del uso de iteraciones, veamos una fórmula para el seno de un ángulo:

$$\operatorname{sen}(x) = x - x^3/3! + x^5/5! - x^7/7! + \dots$$

Tenga en cuenta que si necesitamos el seno de un ángulo en un programa no tenemos que usar esta fórmula, ya que está disponible como función de biblioteca.

Podemos ver que cada término se deriva del término anterior con base en la siguiente multiplicación:

$$-x^2/((n+1) \times (n+2))$$

por lo tanto, podemos construir un ciclo que itere hasta que el nuevo término sea menor que cierta cifra aceptable, por decir 0.0001:

```

private double seno(double x) {
    double término, resultado;
    resultado = 0.0;
    término = x;
    for (int n = 1; Math.abs(término) >= 0.0001; n = n + 2) {
        resultado = resultado + término;
        término = - término * x * x / ((n + 1) * (n + 2));
    }
    return resultado;
}

```

en donde el método de biblioteca `abs` calcula el valor absoluto de su parámetro.

● Gráficos

Es común presentar la información matemática, de ingeniería y financiera en forma gráfica. Ahora veremos un programa para trazar funciones matemáticas. Suponga que queremos dibujar la siguiente función:

$$y = ax^3 + bx^2 + cx + d$$

en donde los valores para a , b , c y d se introducen mediante controles deslizables, como en la figura 11.2.

Debemos resolver varias cuestiones de diseño. En primer lugar, queremos ver el gráfico de manera que los valores positivos de la coordenada y aumenten hacia arriba en la pantalla, mientras que las coordenadas de los píxeles y se miden hacia abajo. Tendremos que distinguir entre x y su coordenada de píxel equivalente `pixelX`, y entre y y `pixelY`.

Ahora tenemos que asegurarnos de que el gráfico se ajuste de manera conveniente al panel; es decir, que no sea demasiado pequeño ni demasiado grande. Al proceso de resolver este problema se le conoce como escalamiento. Vamos a suponer que el área disponible en un panel es de 200 píxeles en la dirección x y de 200 píxeles en la dirección y . Diseñaremos el programa de manera que se muestren los valores x y y en el rango de -5.0 a +5.0. Por lo tanto, 1 unidad de x (o de y) es de 20 píxeles.

Por último, y ya que utilizaremos el método `drawLine` para dibujar el gráfico, tendremos que dibujar una forma curvada como una gran cantidad de líneas pequeñas. Nos desplazaremos a lo largo de la dirección x , un píxel a la vez, dibujando una línea desde la coordenada y equivalente hasta la siguiente. Por cada píxel x , el programa:

1. Debe calcular el valor de x a partir del valor del píxel x .
2. Debe calcular el valor de y , el valor de la función.
3. Debe calcular el valor del píxel y a partir del valor de y .

Para lo cual utilizamos las siguientes instrucciones:

```

x = escalarX(pixelX);
y = laFunción(x);
pixelY = escalarY(y);

```

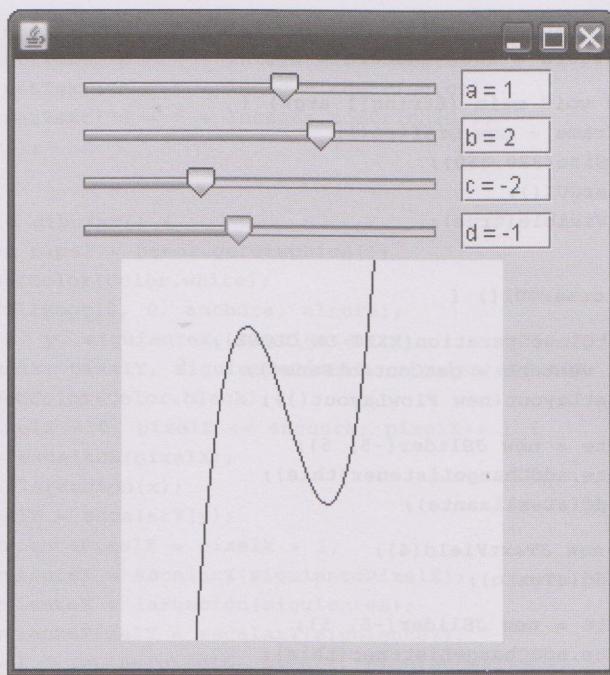


Figura 11.2 Programa para dibujar gráficos.

Después el programa avanza al siguiente píxel x :

```
siguientePixelX = pixelX + 1;
```

Por último se traza la pequeña sección de la curva:

```
papel.drawLine(pixelX, pixelY, siguientePixelX, siguientePixelY);
```

Cabe mencionar que el programa utiliza varios métodos privados para ayudarnos a simplificar la lógica. Además, sólo se utiliza un método para manejar los eventos de los cuatro controles deslizables. He aquí el código completo de este programa para dibujar gráficos.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class Gráfico extends JFrame implements ChangeListener {
    int a, b, c, d;

    private JSlider aDeslizante, bDeslizante, cDeslizante, dDeslizante;
    private JTextField aTexto, bTexto, cTexto, dTexto;
```

```
private JPanel panel;
private int altura = 200, anchura = 200;

public static void main (String[] args) {
    Gráfico frame = new Gráfico();
    frame.setSize(320,350);
    frame.crearGUI();
    frame.setVisible(true);
}

private void crearGUI() {

    setDefaultCloseOperation(EXIT_ON_CLOSE);
    Container ventana = getContentPane();
    ventana.setLayout(new FlowLayout());

    aDeslizante = new JSlider(-5, 5);
    aDeslizante.addChangeListener(this);
    ventana.add(aDeslizante);

    aTexto = new JTextField(4);
    ventana.add(aTexto);

    bDeslizante = new JSlider(-5, 5);
    bDeslizante.addChangeListener(this);
    ventana.add(bDeslizante);

    bTexto = new JTextField(4);
    ventana.add(bTexto);

    cDeslizante = new JSlider(-5, 5);
    cDeslizante.addChangeListener(this);
    ventana.add(cDeslizante);

    cTexto = new JTextField(4);
    ventana.add(cTexto);

    dDeslizante = new JSlider(-5, 5);
    dDeslizante.addChangeListener(this);
    ventana.add(dDeslizante);

    dTexto = new JTextField(4);
    ventana.add(dTexto);

    panel = new JPanel();
    panel.setPreferredSize(new Dimension(anchura, altura));
    panel.setBackground(Color.white);
    ventana.add(panel);
}

public void stateChanged(ChangeEvent e) {
    a = aDeslizante.getValue();
    b = bDeslizante.getValue();
    c = cDeslizante.getValue();
    d = dDeslizante.getValue();
}
```

```

aTexto.setText("a = " + Integer.toString(a));
bTexto.setText("b = " + Integer.toString(b));
cTexto.setText("c = " + Integer.toString(c));
dTexto.setText("d = " + Integer.toString(d));
dibujar();
}

private void dibujar() {
    Graphics papel = panel.getGraphics();
    papel.setColor(Color.white);
    papel.fillRect(0, 0, anchura, altura);
    double x, y, siguienteX, siguienteY;
    int pixelX, pixelY, siguientePixelX, siguientePixelY;
    papel.setColor(Color.black);
    for (pixelX = 0; pixelX <= anchura; pixelX++) {
        x = escalarX(pixelX);
        y = laFunción(x);
        pixelY = escalarY(y);
        siguientePixelX = pixelX + 1;
        siguienteX = escalarX(siguientePixelX);
        siguienteY = laFunción(siguienteX);
        siguientePixelY = escalarY(siguienteY);
        papel.drawLine(pixelX, pixelY, siguientePixelX, siguientePixelY);
    }
}

private double laFunción(double x) {
    return a * x * x * x * x + b * x * x * x + c * x + d;
}

private double escalarX(int pixelX) {
    double xInicial = -5, xFinal = 5;
    double xEscala = anchura / (xFinal - xInicial);
    return (pixelX - (anchura / 2)) / xEscala;
}

private int escalarY(double y) {
    double yInicial = -5, yFinal = 5;
    int coordPixel;
    double yEscala = altura / (yFinal - yInicial);
    coordPixel = (int) (-y * yEscala) +
        (int) (altura / 2);
    return coordPixel;
}
}

```

Si ejecuta este programa, puede alterar los valores de los controles deslizables para ver el efecto de modificar los parámetros. También podemos dibujar ecuaciones cuadráticas (si hacemos el valor del coeficiente a igual a cero) y líneas rectas.

● Excepciones

Si lee este capítulo por primera vez, probablemente sea mejor que omita esta sección ya que trata sobre cosas que no ocurren con mucha frecuencia.

Al escribir un programa que realiza cálculos tenemos que cuidarnos de no exceder el tamaño de los números permitidos. No es como realizar un cálculo en una hoja de papel, en donde los números pueden ser del tamaño que queramos; es algo más parecido a utilizar una calculadora, la cual tiene un límite superior finito en cuanto al tamaño de los números que puede guardar.

Por ejemplo, si declara un `int` de la siguiente manera:

```
int número;
```

debe tener en cuenta que el número más grande que se puede guardar en una variable `int` es extenso, pero se limita a 2147483647. Por lo tanto, si escribe lo siguiente:

```
número = 2147483647;  
número = número + 2147483647;
```

el resultado de la suma no se podrá guardar como valor `int`. El programa terminará y aparecerá un mensaje de error. A esto se le conoce como desbordamiento y es una de varias posibles excepciones que pueden surgir al momento en que se ejecuta un programa.

El desbordamiento puede ocurrir con más sutileza que en el ejemplo anterior, especialmente cuando un usuario introduce datos en un cuadro de texto y, por ende, su tamaño es impredecible. Por ejemplo, he aquí un programa simple para calcular el área de una habitación en donde podría ocurrir un desbordamiento:

```
int longitud, área;  
longitud = Integer.parseInt(campoTextoEntrada.getText());  
área = longitud * longitud;
```

Algunas de las situaciones que podrían provocar un desbordamiento son:

- Sumar dos números grandes.
- Restar un número positivo grande a un número negativo grande.
- Dividir entre un número muy pequeño.
- Multiplicar dos números grandes.

De todo esto podemos concluir que aun cuando un simple cálculo se vea inofensivo, necesitamos vigilarlo. Hay varias formas de lidiar con una excepción:

1. Ignorarla esperando que no vaya a ocurrir, y estar preparados para que el programa falle y/o produzca resultados extraños cuando surja la excepción. Esto está bien para los programadores principiantes, pero no es nada conveniente para los programas reales diseñados para ser robustos.
2. Permitir que surja la excepción pero manejárla escribiendo un manejador de excepciones, como veremos más adelante en el capítulo 16.
3. Evitarla escribiendo comprobaciones para asegurarnos de evitar dicha situación. Por ejemplo, en un programa para calcular el área de una habitación, podemos evitar el desbordamiento si comprobamos el tamaño de los datos de la siguiente manera:

```
if (longitud > 10000) {  
    campoTextoRespuesta.setText("valor demasiado grande");  
}
```

Ta vimos cómo puede ocurrir un desbordamiento cuando un programa utiliza valores `int`. Podríamos esperar que ocurriera lo mismo si los valores `double` se vuelven demasiado grandes, pero no es así. Si un valor se vuelve demasiado grande, el programa sigue funcionando y el valor toma uno de los valores especiales: `PositiveInfinity` (infinito positivo) o `NegativeInfinity` (infinito negativo) según sea apropiado.

Fundamentos de programación

- Muchos programas científicos, de ingeniería, matemáticos y estadísticos emplean muchos cálculos. Pero incluso los programas pequeños que tal vez no requieren cálculos utilizan a menudo algo de aritmética.
- El primer paso (y el más importante) es decidir qué tipos de variables utilizar para representar los datos. La principal elección es entre `int` y `double`.
- La biblioteca de funciones matemáticas es invaluable en los programas de este tipo.
- Se puede dar formato a los números para mostrarlos en pantalla.
- Es común utilizar las iteraciones en los cálculos numéricos en los que la solución converge hacia la respuesta. Para ello se requiere un ciclo.
- Las situaciones excepcionales, como el desbordamiento, pueden ocurrir durante los cálculos, y debemos anticiparnos a ellas si queremos que nuestro programa sea robusto en toda circunstancia.

Errores comunes de programación

- Las situaciones excepcionales, como tratar de dividir entre cero, pueden producir resultados extraños, o el programa puede terminar su ejecución. Cerciórese de que sus programas sean robustos.

Resumen

- Las principales formas de representar números son mediante `int` o `double`. Estos tipos de datos producen distintas precisiones y rangos.
- Las funciones de biblioteca proveen las funciones matemáticas comunes; por ejemplo, el seno de un ángulo.
- El método `format` de la clase de biblioteca `DecimalFormat` se puede usar para dar formato a los números y mostrarlos en pantalla.
- El programador debe tener en cuenta las excepciones que podrían surgir durante los cálculos.

Ejercicios

- 11.1 Costo de una llamada telefónica** Una llamada telefónica cuesta 10 centavos por minuto. Escriba un programa que reciba mediante campos de texto la duración de una llamada telefónica, expresada en horas, minutos y segundos, y que muestre el costo de esta llamada en centavos, con una precisión hasta el centavo más cercano.
- 11.2 Conversión de mediciones** Escriba un programa que reciba como entrada una medición utilizando dos campos de texto, expresada en pies y pulgadas. Al hacer clic en un botón hay que convertir la medición a centímetros y mostrarla en un campo de texto, con dos lugares decimales. Hay 12 pulgadas en un pie; una pulgada equivale a 2.54 centímetros.
- 11.3 Clic del ratón** Escriba un programa que muestre un panel. Cuando el usuario haga clic dentro del panel deberá aparecer un panel de opción que muestre la distancia del ratón desde la esquina superior izquierda del panel. Consulte el apéndice A para ver cómo obtener las coordenadas del clic del ratón.
- 11.4 Caja registradora** Escriba un programa que represente una caja registradora. Los montos de dinero se pueden introducir en un campo de texto y se deben sumar al total acumulado cuando se haga clic en un botón. El total acumulado se debe mostrar en otro campo de texto. Debe haber otro botón que permita borrar la suma (hacerla cero).
- 11.5 Suma de enteros** La suma de los enteros de 1 a n se obtiene mediante la siguiente fórmula

$$\text{suma} = n(n + 1)/2$$

Escriba un programa que reciba como entrada un valor para n mediante un campo de texto y que calcule la suma de dos formas: primero utilizando la fórmula y después sumando los números mediante un ciclo.

- 11.6 Cálculo de intereses** El programa que se muestra en el texto calcula los intereses año por año. Una alternativa sería utilizar una fórmula para calcular el interés. Si se invierte un monto p a una tasa de interés r durante n años, el valor acumulado v sería:

$$v = p(1 + r)^n$$

Escriba un programa que acepte valores para p (en dólares), r (como porcentaje) y n mediante campos de texto y que muestre el valor acumulado en otro campo de texto.

- 11.7 Números aleatorios** Los números aleatorios se utilizan con frecuencia en los programas computacionales y de simulación, conocidos como métodos de Monte Carlo. Ya vimos cómo utilizar la clase de biblioteca `Random` que nos permite crear un generador de números aleatorios, como se muestra a continuación:

```
Random aleatorio = new Random();
```

Después llamamos a `nextInt` para obtener un número aleatorio entero en el rango de 0 a 1, como se muestra a continuación:

```
int número = aleatorio.nextInt(2);
```

Escriba un programa para verificar el método generador de números aleatorios. Provea un botón que cree un conjunto de 100 números aleatorios que tengan el valor 0 o 1. Cuente el número de valores iguales a 0 y el número de valores iguales a 1 (deben ser aproximadamente iguales).

- 11.8 Series para e** El valor de e^x se puede calcular mediante la suma de la siguiente serie:

$$e^x = 1 + x + x^2/2! + x^3/3! + \dots$$

Escriba un programa que reciba como entrada un valor de x mediante un campo de texto y que calcule e^x hasta cierto grado de precisión deseado. Compruebe el valor con el valor que se obtiene al usar el método `exp` de la biblioteca `Math`.

- 11.9 Cálculo de impuestos** Escriba un programa que lleve a cabo el cálculo de impuestos. Los impuestos son cero para los primeros \$10,000, pero son del 33% para cualquier monto mayor a ese. Escriba el programa para recibir como entrada un salario en dólares mediante un campo de texto y calcular los impuestos a pagar. Vigile los errores al realizar el cálculo; ¡la respuesta necesita ser precisa hasta el centavo más cercano!

- 11.10 Área de un triángulo** El área de un triángulo cuyos lados son de una longitud a , b y c es:

$$\text{área} = \sqrt{s(s - a)(s - b)(s - c)}$$

en donde:

$$s = (a + b + c)/2$$

Escriba un programa que reciba como entrada los tres valores de los lados de un triángulo mediante campos de texto y utilice esta fórmula para calcular el área. Su programa debe comprobar primero que las tres longitudes especificadas en realidad formen un triángulo. Por ejemplo, $a + b$ debe ser mayor que c .

- 11.11 Raíz cuadrada** La raíz cuadrada de un número se puede calcular mediante iteraciones, como se muestra a continuación. Escriba un programa para hacer esto con un número que se introduzca mediante un campo de texto.

La primera aproximación a la raíz cuadrada de x es $x/2$.

Después las aproximaciones sucesivas se obtienen mediante la siguiente fórmula:

$$\text{siguienteAproximación} = (\text{últimaAproximación}^2 - x)/2 + \text{últimaAproximación}.$$

Compare el valor con el que se obtiene al utilizar el método de biblioteca `sqrt`.

- 11.12 Calculadora matemática** Escriba un programa que actúe como una calculadora matemática. Debe tener botones con los cuales se puedan introducir números, los que se deben mostrar como en la pantalla de una calculadora de escritorio. También debe haber botones para realizar cálculos matemáticos estándar, como seno, coseno, logaritmo natural y raíz cuadrada.

- 11.13 Calculadora de interés** Modifique la parte del cálculo del programa anterior sobre el cálculo de intereses, de manera que se pueda utilizar un número `int` (en vez de `double`) para representar un monto de dinero (expresado en centavos).

- 11.14 Trazador de gráficos** Mejore el programa para dibujar gráficos que vimos en este capítulo de manera que:

- Dibuje los ejes x y y .
- Reciba como entrada los coeficientes mediante campos de texto, en vez de controles deslizables (para tener precisión).
- Reciba como entrada un factor de escala horizontal y vertical (acercamiento) mediante controles deslizables.
- Dibuje un segundo gráfico de la misma función, pero con coeficientes distintos.
- Dibuje los gráficos de algunas otras funciones. Una manera de hacer esto sería volver a escribir el método `laFunción`.

- 11.15 Integración numérica** Escriba un programa que calcule la integral de una función y utilizando la “regla del trapecio”. El área bajo el gráfico de la función se divide en n bandas iguales de longitud d . Despues el área bajo la curva (la integral) es aproximadamente igual a la suma de todos los (pequeños) trapecios:

$$\text{área} \cong \frac{1}{2}d(y_0 + 2y_1 + 2y_2 + \dots + 2y_{n-1} + y_n)$$

O:

$$\text{área} = (\text{la mitad de la anchura de la banda}) \times (\text{primera} + \text{última} + \text{dos veces la suma de las demás}).$$

Utilice una función para la cual conozca la respuesta y experimente utilizando valores cada vez más pequeños de d .

- 11.16 Conjunto de Mandelbrot** El conjunto de Mandelbrot (figura 11.3) es una famosa e impactante imagen que se produce al evaluar en forma repetida una fórmula en cada punto, en un espacio de dos dimensiones. Tome un punto con las coordenadas x_{inicial} y y_{inicial} . Despues calcule en forma repetida nuevos valores de x y de y a partir de los valores anteriores, utilizando las siguientes fórmulas:

$$x_{\text{nueva}} = x_{\text{anterior}}^2 - y_{\text{anterior}}^2 - x_{\text{inicial}}$$

$$y_{\text{nueva}} = 2x_{\text{anterior}}y_{\text{anterior}} - y_{\text{inicial}}$$

Los primeros valores de x_{anterior} y y_{anterior} son x_{inicial} y y_{inicial} . Para cada iteración, calcule $r = \sqrt{x_{\text{nueva}}^2 + y_{\text{nueva}}^2}$. Repita hasta que $r > 10000$ o hasta llegar a 100 iteraciones, lo que ocurra primero. Si r es mayor que 10000, coloree de blanco el píxel correspondiente a esta coordenada; en caso contrario coloréelo de negro.

Repita el proceso para todos los puntos con un valor de x entre -1.0 y $+2.0$, y con un valor de y en el rango de -2.0 a $+2.0$.

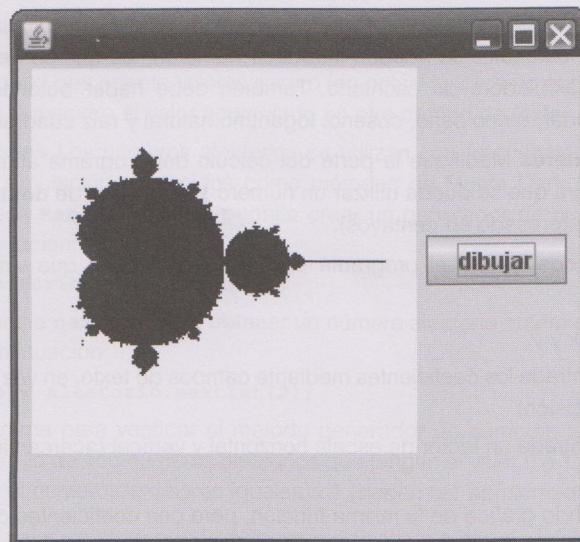


Figura 11.3 El conjunto de Mandelbrot.

A medida que avanza la iteración, partiendo de valores específicos de x_{inicial} y y_{inicial} , el valor de r algunas veces permanece razonablemente pequeño (cerca de 1.0). Para otros valores de x_{inicial} y y_{inicial} el valor de r se vuelve rápidamente muy grande y tiende a dispararse hacia el infinito.

A propósito, algunas veces podemos ver imágenes del conjunto de Mandelbrot que son la imagen espejo de la que se muestra en la figura 11.3. Esas imágenes se generan mediante las siguientes fórmulas, que son ligeramente distintas:

$$x_{\text{nueva}} = x_{\text{anterior}}^2 - y_{\text{anterior}}^2 + x_{\text{inicial}}$$

$$y_{\text{nueva}} = 2x_{\text{anterior}}y_{\text{anterior}} + y_{\text{inicial}}$$

Respuesta a la práctica de autoevaluación

```
11.1    double respuesta;
        DecimalFormat formato = new DecimalFormat("##.###");
        campoTexto.setText(formato.format(respuesta));
```