```
        ? epsilon/result    // return epsilon/result if (...) is true
        : epsilon;          // else return epsilon
}
```

The constant `defaultNumericalPrecision` is equal to `Math.sqrt(Double.MIN_VALUE)`.

Problem 11.4 The `Integral` class

Add a static counter to keep track of the number of times the test function is evaluated in the `IntegralCalcApp` class. Use this counter to compare the efficiencies of the various integration algorithms.

(a) How many function calls are required for each numerical method if the tolerance is set to 10^{-3} or 10^{-12}?

(b) How does the execution time depend on the numerical method? You can compute the time using the statement `System.currentTimeMillis()`, which is the time between the current time and January 1, 1970 measured in milliseconds. Note that the return value is `long` and not `int`.

(c) The method `Integral.ode` determines when it has reached the input tolerance differently than the other integration algorithms in the `Integral` class. Compare the results from these other integrators. Is the accuracy of each method always within the tolerance set by the user? ■

Another way of evaluating one-dimensional integrals is to recast them as differential equations. Consider an indefinite integral of the form

$$F(x) = \int_a^x f(t)\,dt, \tag{11.9}$$

where $F(a) = 0$. If we differentiate $F(x)$ with respect to x, we obtain the first-order differential equation:

$$\frac{dF(x)}{dx} = f(x). \tag{11.10}$$

with the boundary condition

$$F(a) = 0. \tag{11.11}$$

Because the function $f(x)$ is known, (11.10) can be solved for $F(x)$ using the numerical algorithms that we introduced earlier for obtaining the numerical solutions of first-order differential equations.

In general, the methods for solving ordinary differential equations and evaluating numerical integrals are not equivalent. For example, we cannot use Simpson's rule to obtain the numerical solution of an differential equation of the form $dy/dx = f(x, y)$. Why?

Simpson's rule fails (or is slowly convergent) if the function has regions of rapid change. In this case an adaptive step-size ODE algorithm is usually effective. The `RK45MultiStep` solver adapts the integration step to the behavior of the integrand and allows the user to set the tolerance. In Problem 11.5 we use it to examine an approximation to the Dirac delta function.

Problem 11.5 Delta function approximation

The Dirac delta function can be approximated by the function

$$\delta(x) = \frac{1}{\pi} \lim_{\epsilon \to 0} \frac{\epsilon}{x^2 + \epsilon^2}. \tag{11.12}$$

Test this approximation by integrating (11.12) over a suitable range of x using various values of ϵ. Try adaptive ODE solvers with a tolerance of 10^{-8}. How small a value of ϵ produces an error of 10^{-4}? 10^{-5}? ■

As computers become more powerful and software to perform mathematical operations and numerical analysis becomes more prevalent, there is a strong temptation to use libraries written by others. This use is appropriate because usually these libraries are well written, and there is no reason why a user should re-invent them. However, the downside is that users do not always know or care what the libraries are doing and sometimes can obtain puzzling or even incorrect results. In Problem 11.6 we explore this issue.

Problem 11.6 Understanding errors in integration routines

(a) Use the `ode`, `simpson`, `trapezoid`, and `rhomberg` methods in the `Integral` class of the Open Source Physics library to estimate the integral of $\sin^2(2\pi x)$ between $x = 0$ and 1 with a tolerance of 0.01. The exact answer is 0.5. Do all four methods return results within the tolerance? Are some results much more accurate than the tolerance? Change the tolerance to 0.1. How do the results change? Notice that for some of the methods, the results are much better than might be expected because the positive and negative errors cancel. Why does the trapezoid method always give the exact answer?

(b) Integrate the same function from $x = 0.2$ to 1.0. How accurate are the results now? Explore how the results change with the input tolerance. Does the behavior of the `ode` integrator differ from the others. Why? How do you think the tolerance parameter is used for each of the methods?

(c) Integrate $f(x) = x^n$ for various values of n. How do the different integrators compare? Why does the trapezoid integrator do worse than the others for $n = 2$? ■

11.2 ■ SIMPLE MONTE CARLO EVALUATION OF INTEGRALS

We now explore a much different method of estimating integrals. Consider the following example. Suppose an irregularly shaped pond is in a field of known area A. The area of the pond can be estimated by throwing stones so that they land at random within the boundary of the field and counting the number of splashes that occur when a stone lands in a pond. The area of the pond is approximately the area of the field times the fraction of stones that make a splash. This simple procedure is an example of a *Monte Carlo* method.

To be more specific, imagine a rectangle of height h, width $b - a$, and area $A = h(b - a)$ such that the function $f(x)$ is within the boundaries of the rectangle (see Figure 11.3). Compute n pairs of random numbers x_i and y_i with $a \le x_i \le b$ and $0 \le y_i \le h$. The fraction of points x_i, y_i that satisfy the condition $y_i \le f(x_i)$ is an estimate of the ratio of the integral