

ECE 491 Object-Oriented Programming in Java

Spring'04

Lab 3

Purpose: (1) Understanding the Java Applet, Graphics and Exceptions classes. (2) Making a matrix package (3) Experience in code reuse

Part 1

Play with the Snowman applet. It is not a serious, but a time-consuming exercise. Applets are revisited when we discuss event-driven programming.

1. Copy the Snowman applet.
2. Confirm to which class each method belongs and what it does.
3. Increase the width and height of the applet's display area by 2 times and 1.5 times respectively.
4. Define a method `drawSnowman(mid, top, g)`, where `g` is a Graphics object. This method draws a snowman at the location `(mid, top)`.
5. Define a method `drawTree(mid, top, g)`, where `g` is a Graphics object. This method draws a triangle on top of a triangle so that they look a tree. Fill the shape with colors you want.
6. Define a method `drawSnow(x, y, g)`, where `g` is a Graphics object. This method draws a small white solid circle at the location `(x, y)`.
7. Define a new paint method so that you can have a picture with as many as snowmen, trees and falling snow.

Part 2

We are going to develop a class `SmallMatrix` whose elements are objects of the class `Fraction`. You should utilize the textbook's `Fraction` and `Matrix` classes defined in chapters two and six. While you are working on this exercise, pay attention to how exceptions are used. Further, consider how we can achieve the goal with the minimal modification of the give classes. Although we have not dealt with inheritance in detail, try to find out how inheritance can be applied to this exercise.

- 1 Add the `ZeroDenominatorException` to the class `Fraction`.
- 2 Add another constructor `Fraction(int)`. This method generates two random integers n and d using the class `java.util.Random`, and instantiates a fraction representing n/d .
- 3 Add a public method `getDoubleValue()` to the `Fraction` class, which returns the value of the fraction in type `double`. Throw the `TooSmallException`, if the resulting magnitude is less than 0.001, and `TooLargeException`, if the magnitude is greater than 999. These two exceptions are subclasses of the `TooException`, which would be defined as a subclass of the class `Exception`.

- 4 Modify the original definition of the class Matrix to have a class SmallMatrix. This is a matrix whose elements are of class Fraction.
 - 4.1 Substitute SmallMatrix for Matrix lexically
 - 4.2 Change the type of the private variable mat to Fraction[][].

- 5 Modify constructors to reflect the change in representation of the matrix.
 - 5.1 Modify SmallMatrix(n, m) so that it generates a $n \times m$ matrix whose elements are randomly generated fractions. Note that Fraction objects may throw TooException only when the method getDoubleValue is invoked.
 - 5.2 Add another constructor SmallMatrix(n, m, f). This generates an $n \times m$ SmallMatrix object and initializes all of its elements to fraction f . This constructor should also be able to throw InvalidDimException.
 - 5.3 All of them throw exceptions InvalidDimException when it is requested to instantiate a matrix whose row or column is greater than five or it is zero. A messages should tell if the nature of troubled dimension. InvalidDimException is a subclass of MatrixException.

- 6 Make a TestSmallMatrix class for testing...
 - 6.1 The *main* method loops until it is told to stop.
 - 6.2 The *main* method should iterate endlessly, providing a message asking to continue or not after each iteration. Any response other than the end of file continues experiment. You may program following your favorite style. EOF is not important for this exercise.
 - 6.3 The main method generates three random integers l , m and n less than 10. These numbers represent two matrices $A_{l \times m}$ and $B_{m \times n}$. Generate random fractions to populate $A_{l \times m}$ and $B_{m \times n}$ and multiply them to get a resulting matrix $C_{l \times n}$.
 - 6.4 Display $A_{l \times m}$ and $B_{m \times n}$ in fractional form “n/d” and $C_{l \times n}$ in decimal + fractional form.