this class to be initialized when it's declared. The constructor should store the fraction in reduced form. The fraction

```
2/4
```

is equivalent to `1/2` and would be stored in the object as 1 in the `numerator` and 2 in the `denominator`. Provide a no-argument constructor with default values in case no initializers are provided. Provide `public` methods that perform each of the following operations:

a) Add two `Rational` numbers: The result of the addition should be stored in reduced form. Implement this as a `static` method.
b) Subtract two `Rational` numbers: The result of the subtraction should be stored in reduced form. Implement this as a `static` method.
c) Multiply two `Rational` numbers: The result of the multiplication should be stored in reduced form. Implement this as a `static` method.
d) Divide two `Rational` numbers: The result of the division should be stored in reduced form. Implement this as a `static` method.
e) Return a `String` representation of a `Rational` number in the form `a/b`, where `a` is the `numerator` and `b` is the `denominator`.
f) Return a `String` representation of a `Rational` number in floating-point format. (Consider providing formatting capabilities that enable the user of the class to specify the number of digits of precision to the right of the decimal point.)

8.16 *(Huge Integer Class)* Create a class `HugeInteger` which uses a 40-element array of digits to store integers as large as 40 digits each. Provide methods `parse`, `toString`, `add` and `subtract`. Method `parse` should receive a `String`, extract each digit using method `charAt` and place the integer equivalent of each digit into the integer array. For comparing `HugeInteger` objects, provide the following methods: `isEqualTo`, `isNotEqualTo`, `isGreaterThan`, `isLessThan`, `isGreaterThanOrEqualTo` and `isLessThanOrEqualTo`. Each of these is a predicate method that returns `true` if the relationship holds between the two `HugeInteger` objects and returns `false` if the relationship does not hold. Provide a predicate method `isZero`. If you feel ambitious, also provide methods `multiply`, `divide` and `remainder`. [*Note:* Primitive `boolean` values can be output as the word "true" or the word "false" with format specifier `%b`.]

8.17 *(Tic-Tac-Toe)* Create a class `TicTacToe` that will enable you to write a program to play Tic-Tac-Toe. The class contains a private 3-by-3 two-dimensional array. Use an `enum` type to represent the value in each cell of the array. The `enum`'s constants should be named `X`, `O` and `EMPTY` (for a position that does not contain an `X` or an `O`). The constructor should initialize the board elements to `EMPTY`. Allow two human players. Wherever the first player moves, place an `X` in the specified square, and place an `O` wherever the second player moves. Each move must be to an empty square. After each move, determine whether the game has been won and whether it's a draw. If you feel ambitious, modify your program so that the computer makes the moves for one of the players. Also, allow the player to specify whether he or she wants to go first or second. If you feel exceptionally ambitious, develop a program that will play three-dimensional Tic-Tac-Toe on a 4-by-4-by-4 board [*Note:* This is an extremely challenging project!].

8.18 *(`Account` Class with `BigDecimal balance`)* Rewrite the `Account` class of Section 3.5 to store the `balance` as a `BigDecimal` object and to perform all calculations using `BigDecimals`.

Making a Difference

8.19 *(Border Protection)* Many border protection agencies such as the Australian Customs and Border Protection Service and the Canada Border Services Agency have a great responsibility in protecting their national borders. Not only are they protecting their borders from the flow of illegal immigration, but they are also protecting their mainland from the flow of illegal substances such as narcotics or harmful