Programming Assignment 8 (100 Points)

Due: 11:59pm Wednesday, November 26

Start early … Start often!

README (10 points)

You are required to provide a text file named **README,** NOT Readme.txt, README.pdf, or README.doc, with your assignment in your pa8 directory. There should be no file extension after the file name "**README**". Your README should include the following sections:

Program Description **(4 points) :** Explain how to work your program. What sort of inputs or events does it take and what are the expected outputs or results? How did you test your program? What are unit tests? How well do you think your program was tested?

Write your READMEs as if it was intended for a 5 year old or your grandmother. Do not assume your reader is a computer science major. **The more detailed the explanation, the more points you will receive.**

Short Response Questions**(6 points):**

First, you will need to copy a file from the public directory to answer the following questions.

```
$ cp ~/../public/PA8/readFile .
```

Unix Questions:

Using the "cat" command and the readFile file,

1. How would you display the contents of readfile with each of its lines enumerated (line numbers)?
2. How would you display the file's contents showing the spaces and tabs present within this file? What symbol indicates the end of the line? A tab? How can you tell if there is trailing whitespace on a line?
3. What Linux command will write a file to standard output adding line numbers to a file (cmd readFile - will display readFile with line numbers - what Linux cmd is cmd)? Pipe this to cat with the above flag to answer the following: 1) What divides the beginning of the line and the line numberm and 2) what divides the line number from the first character of the line?

Vim Questions:

Using vim and the readFile file,

4. In vim, how would you display the contents of readFile with each of its lines enumerated (line numbers)?
5. In vim, how would you display the file's contents showing the spaces and tabs present within this file? What symbol indicates the end of the line? A tab? How can you tell if there is trailing whitespace on a line?
6. In vim, what command can you use to find and replace every instance of the word "fun" with "amazing"? How would you replace only the first instance of "fun" with "amazing" in each line? (*Hint*: go through the Vim tutorials provided in the "Useful links" page of the course site.)

STYLE (20 points)

Please see previous programming assignment. Below is an additional style point to consider.

- **Use accessor/mutator methods to access numerator and denominator private fields (getters and setters)**

You will be specifically graded on commenting, file headers, class and method headers, meaningful variable names, sufficient use of blank lines, not using more than 80 characters on a line, perfect indentation, no magic numbers/hard-coded numbers other than zero, and **use of accessor/mutator methods to access numerator and denominator private fields (getters and setters)**

CORRECTNESS (70 points)

Program 1: CS11Turtle_Threaded.java

For this assignment, you will modify and extend your PA1 CS11Turtle programming assignment. You will create multiple turtles (in separate threads) to draw the letters and numbers in parallel.

A primer on Java threading can be found here: http://www.javaworld.com/javaworld/jw-06-2012/120626-modern-threading.html

Make a new pa8 directory.

Copy your pa1 single-threaded turtle graphics program to your new pa8 directory.

```
% cp ~/pa1/CS11Turtle.java ~/pa8/CS11Turtle_Threaded.java
% cp ~/../public/turtleClasses.jar ~/pa8
```

First, you will change the class CS11Turtle to CS11Turtle_Threaded. You will have to change the filename (you just did this with the cp above), class name, constructor name, and all the other places CS11Turtle shows up in your program.

You will add (implement) the Runnable interface to your class.

You will need to modify the constructor to take the character to be drawn, the x and y coordinates where the character is to be drawn, and a delay for create the animation.

```
public CS11Turtle_Threaded(World w, char ch, int x, int y, int delay)
```

Remember, the call to the superclass (Turtle) constructor needs to be the first statement in the constructor:

```
public CS11Turtle_Threaded(World w, char ch, int x, int y, int delay) {
      super(w, delay);
      //…
}
```

The delay is to help see the animation. Set the delay to half a second.

```
private final static int DELAY = 500; // Half sec delay for the animation
```

You will set the pen width and color in the constructor also. And most importantly add

```
new Thread(this).start();
```

as the **last line** of your CS11Turtle_Threaded constructor. The Thread start() method will call the run() method for this Thread. **Do not call run() directly**!

Every time you create a new CS11Turtle_Threaded object, you will be creating a new turtle in a new Thread and starting this Thread (turtle graphics object) to draw a specific character at a specific location. If you want to learn more about Java's Thread class and Runnable interface, go to : http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html and http://docs.oracle.com/javase/7/docs/api/java/lang/Runnable.html .

You will need to add a run() method to your CS11Turtle_Threaded class. It should look something like this:

```
public void run() {
  switch(ch) {
    case 'C': this.drawC(x,y); break;
    case 'S': this.drawS(x,y); break;

    // … and so on.
    // You may need to tweak the draw methods depending
    // on how you did it in the first assignment.
  }
}
```

Again, you do not call run() directly - it is automatically called from the Thread start() method after the Thread begins.

In main() you will want to create a new CS11Turtle_Threaded object (a new turtle drawing a character in a separate thread) for each letter or number that needs to be drawn, so they can all execute simultaneously in parallel.

```
new CS11Turtle_Threaded(w, 'C', x, y, DELAY); // This turtle draws a C char
new CS11Turtle_Threaded(w, 'S', x, y, DELAY); // This turtle draws a S char
// … and so on.
```

When you run your program, you will see all the turtles drawing all the characters at the same time in parallel. Here are some screenshots (yours should spell out your cs11f course-specific account name):

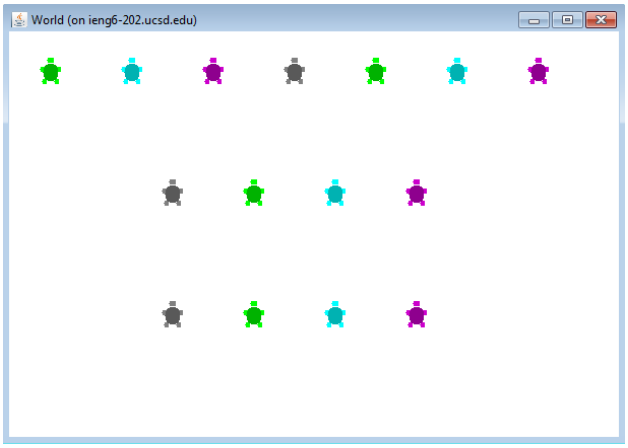
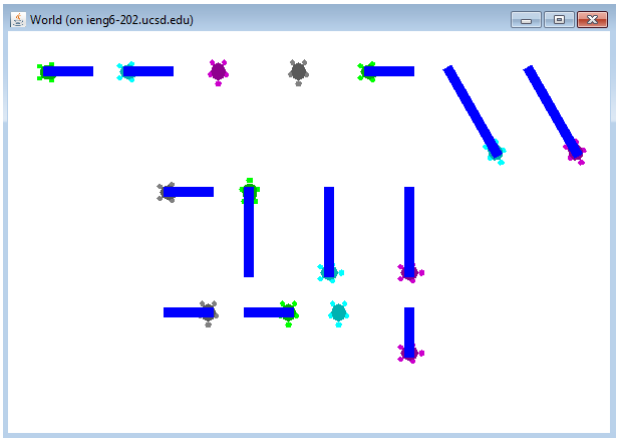
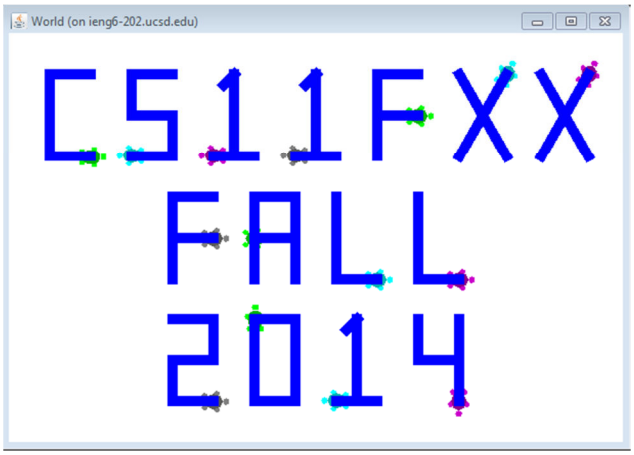


Image of what the applet starts with



Nearly done drawing…



Completed.

***IMPORTANT*:** the feature for centering the text horizontally and vertically is required. If you did not do this for PA1 as extra credit, you must complete this feature for PA8.

Running:

Multi-Threaded Turtle Graphics (application):

You should already know how to run this program/application. (Hint: you must include turtleClasses.jar in the classpath)

Program 2: Rational.java

Recall from mathematics that a rational number is any number that can be expressed as the quotient or fraction n/d of two integers, n and d, with the denominator d not equal to zero. The objective of this PA is to create a library class for Rational numbers, stored as both a numerator and a denominator. Within this class, you are to have constructors to initialize instances of these Rational objects as well as methods for performing arithmetic operations on them (addition, subtraction, etc, as detailed below), while keeping them as rational numbers (i.e., 2 integers as numerator and denominator) instead of decimal approximations like floats.

Copy over the test drivers from public directory:

```
% cp ~/../public/PA8/TestRational.java ~/pa8/
% cp ~/../public/PA8/EC_TestRational.java ~/pa8/
```

In **Rational.java**, create:

```
public class Rational
```

Use two **private** instance variables of type int to hold the value of the numerator and the value of the denominator. Define the following constructors. All should call setNumerator()/setDenominator() appropriately.

`public Rational()`
- creates a default rational number of 0/1.

`public Rational( int num )`
- creates a rational number of num/1.

`public Rational( int num, int denom )`
- creates a rational number of num/denom stored **in reduced form**

`public Rational( Rational r )`
- copy ctor initializing this new Rational object with the numerator and denominator of the parameter r (should already be in reduced form).

Define the following **private** methods:

```
private int getNumerator()
private int getDenominator()
private void setNumerator( int num )
private void setDenominator( int denom )
```
- if denom is 0, throws a `new IllegalArgumentException()`.

By making the mutator methods **private**, this makes the Rational objects immutable (like Strings). These methods can only be used from within class Rational. There is some question as to the utility of making the accessor methods public, but let's just keep them private for now. **Be sure to use these private accessor and mutator methods in your class Rational constructors and methods - do not directly access the instance variables!**

`private int gcd( int x, int y )`
- calculates the greatest common divisor of x and y. One recursive algorithm to do this is:
  if y is equal to 0, then gcd(x, y) is x [return x];
  else gcd(x, y) is gcd(y, x % y) [return gcd(y, x % y)].

`private void reduce()`
- reduces the fraction in this Rational object by finding the greatest common divisor (gcd) and then dividing both the numerator and denominator by gcd. The gcd() defined above only guaranteed to work for positive numbers, so you will want to pass the absolute value of the numerator and denominator to the gcd() method [Math.abs()].

- as part of reduce() we want to store the sign correctly. For example, if the fraction is -1/-2, then store as 1/2. And if the fraction is 1/-2, then store as -1/2.

`public Rational add( Rational r )`

```
public Rational subtract( Rational r )
public Rational multiply( Rational r )
public Rational divide( Rational r )
```

- add, subtract, multiply, and divide **this** Rational object with the parameter **r**. They all return a **new** Rational object representing the result of this operation. Note: Do not change the state of the existing objects this or r.

Tip: Make sure your method names are exactly the same as the write-up. If they are not written exactly, you will be docked half a grade.

The following are the algorithms for the above operations for those who don't remember how to add, subtract, multiply, and divide fractions:

`"right"` is the right-hand operand, ie. the parameter reference passed in to each of the methods.
`"left"` is the left-hand operand, ie. the object on which the method is being invoked (in reality **this**).
`"cd"` stands for common denominator.

Addition

```
cd = left.denominator * right.denominator;  // common denominator
num = left.numerator * right.denominator +
      right.numerator * left.denominator;   // cross multiply
```

return a **new** Rational object with

```
numerator = num
denominator = cd
```

Subtraction

```
cd = left.denominator * right.denominator;  // common denominator
num = left.numerator * right.denominator -
      right.numerator * left.denominator;  //cross multiply
```

return a **new** Rational object with

```
numerator = num
denominator = cd
```

Multiplication

return a **new** Rational object with

```
numerator = left.numerator * right.numerator
denominator = left.denominator * right.denominator
      // straight multiply
```

Division

return a **new** Rational object with

```
numerator = left.numerator * right.denominator
denominator = left.denominator * right.numerator
      // flip right num & denom (invert) and multiply
```

Remember your Rational ctor should reduce the resulting values. And remember to use the accessor methods - do not access the numerator and denominator directly.

`public String toString()`

- returns the String representation of this Rational object. For example, "-1/2" where -1 is the numerator and 2 is the denominator. If the denominator is 1, represent as a whole number instead of a fraction (without the denominator of 1).

Tip: The toString() method will be used in all the test cases. Make sure this works!

`public float toFloat()`

- returns the floating point representation of this Rational object. For example, for the rational number -1/2, the floating point representation returned should be -0.5 as a floating point value.

`public boolean equals( Object obj )`

- returns true if the numerators and denominators of **this** Rational object and that of **obj** are the same. Returns false otherwise. Also returns false if obj is null or if obj is not the same type as this. (See notes on class Object and class Class.)

`public int hashCode()`

- computes a hash code for this object. If toString() is written in a way such that all objects that have the same numerator and denominator have the same String representation (which they should in this assignment - remember to store all numerators and denominators in reduced form), then a good hash code would be to just **return this.toString().hashCode();** to use class String's good hashCode() method. More on this later in class.

Here is the **expected output** of your Rational program:

```
[cs11fzz]:~:420$ java TestRational
Running TestRational...


===== Testing No-Arg  Constructor =====
  Test 1:  PASS

===== Testing One-Arg Constructor =====
  Test 2:  PASS
  Test 3:  PASS
  Test 4:  PASS

===== Testing Two-Arg Constructor =====
  Test 5:  PASS
  Test 6:  PASS
  Test 7:  PASS
  Test 8:  PASS
  Test 9:  PASS
  Test 10: PASS
  Test 11: PASS
  Test 12: PASS

===== Testing   Copy  Constructor =====
  Test 13: PASS

===== Testing     add()    method =====
  Test 14: PASS
  Test 15: PASS
  Test 16: PASS
  Test 17: PASS

===== Testing  subtract()  method =====
  Test 18: PASS
  Test 19: PASS
  Test 20: PASS
```

```
  Test 21: PASS

===== Testing  multiply()  method =====
  Test 22: PASS
  Test 23: PASS
  Test 24: PASS
  Test 25: PASS
  Test 26: PASS

===== Testing   divide()   method =====
  Test 27: PASS
  Test 28: PASS
  Test 29: PASS
  Test 30: PASS
  Test 31: PASS
  Test 32: PASS

===== Testing   toFloat()  method =====
  Test 33: PASS
  Test 34: PASS
  Test 35: PASS
  Test 36: PASS

===== Testing   equals()   method =====
  Test 37: PASS
  Test 38: PASS
  Test 39: PASS

===== Testing  hashCode()  method =====
  Test 40: PASS


...Finished.

RESULT: Total # of Tests Passed: 40/40 -- Yay! :)
```

If your output differs from this, you must fix your class Rational code to pass all the tests in TestRational. Do not change TestRational.java.

EXTRA CREDIT (5 points)

`public Rational reciprocal()` (1 point)

- Reciprocal returns the multiplicative inverse of this Rational object. For example, if the Rational is x/y, then its reciprocal is y/x. Make sure to return a **new** Rational object.

Check if this numerator is 0 at the very beginning and throw an `UnsupportedOperationException`.

For example, new Rational(0).reciprocal() must throw an `UnsupportedOperationException`.

`public Rational floor()` (1 point)

- Without using the Math class in the Java library, implement a method that changes a rational into a whole number rounding down. Return a new Rational object.

`public Rational ceiling()` (1 point)

- Again, without using the Math class in the Java library, implement a method that changes a rational into a whole number rounding up. Return a new Rational object.

Extra Credit Unit Tests: (2 points)

In the blank template EC_TestRational.java provided for you, fill in the **TODO** sections with your own tests for reciprocal(), floor(), and ceiling(). These extra credit points will be given to you based on your test cases for testReciprocal(), testFloor(), and testCeiling() written in EC_TestRational.java. We will be looking for how well you write them and if you thought about all the edge/corner cases. Test for exceptions, negatives, big numbers, etc. Check for special cases, like denominator equal to 0, numerators equal to 0, etc. We will not be testing values or results that do not fit in an int (overflow) for the numerator or denominator.

Turnin

To turnin your code, navigate to your home directory and run the following command:

```
> turnin pa8
```

You may turn in your programming assignment as many times as you like - each turnin over-writes the previous turnin. The last submission you turn in before the deadline is the one that we will collect and grade.

Verify

To verify a previously turned in assignment,

```
> verify pa8
```

If you are unsure your program has been turned in, use the verify command. **We will not take any late files you forgot to turn in.** Verify will help you check which files you have successfully submitted. It is your responsibility to make sure you properly turned in your assignment.

Files to be collected:

- README
- Rational.java
- TestRational.java
- EC_TestRational.java
- CS11Turtle_Threaded.java
- turtleClasses.jar

NO LATE ASSIGNMENTS ACCEPTED.

DO NOT EMAIL US YOUR ASSIGNMENT!

And above all … Have Fun!