**Program Thirteen Part One:**

//summary: This program uses the GeometricObject class and traingle class from a few assignments ago and has us make

//the geormetricObject class an abstract class. We then use the classes to take in the side lengths, color, and filled

//property from the user and display each attribute along with the area and perimeter

//name: Jenna Wolf

//class: Fundamentals of Programming, CS155 - 01

//instructor: Dr. Art Kazmierczak

//date: 11/12/2023

import java.util.Scanner; //lets inputs be made

public class Main

{

public static void main(String[] args)

{

Scanner input = new Scanner(System.in); //names the input object

double triSide1, triSide2, triSide3; //holds the data for these objects

String triColor; //holds the triColor data

boolean triFilled; //holds the triFilled data

//takes in the sides from the user

System.out.print("Enter sides of your triangle: ");

triSide1 = input.nextDouble();

triSide2 = input.nextDouble();

triSide3 = input.nextDouble();

//takes in the color from the user

System.out.print("Enter the color of your triangle: ");

triColor = input.next();

//takes in the filled property from the user

System.out.print("indicate weather your triangle is filled of empty (true or false): ");

triFilled = input.nextBoolean();

triangle tri = new triangle(triSide1, triSide2, triSide3); //creates a new triangle object

tri.setColor(triColor); //sets the color of the triangle

tri.setFilled(triFilled); //sets the filled property of the triangle

//calls the toString method and outputs information

System.out.println(tri);

}

}

**Geometric Object**

import java.util.Date; //lets the Date class be used

//abstract verision of the GeometricObject class

public abstract class GeometricObject{

private String color = "white"; //holds the color data and defaults to white

private boolean filled; //holds the filled data

private java.util.Date dateCreated; //holds the dateCreated data

//no args constructor

public GeometricObject(){

dateCreated = new java.util.Date();

}

//constructor that takes in a string and a boolean

public GeometricObject(String temp1, boolean temp2){

color = temp1;

filled = temp2;

dateCreated = new java.util.Date();

}

//returns the color data

public String getColor()

{

return color;

}

//returns the filled data

public boolean getFilled()

{

return filled;

}

//returns the dateCreated data

public Date getDateCreated()

{

return dateCreated;

}

//sets color to the data sent over

public void setColor(String temp)

{

color = temp;

}

//sets filled to the data sent over

public void setFilled(boolean temp)

{

filled = temp;

}

//returns a string with the GeometricObject data

@Override

public String toString()

{

return "\nCreated on: " + dateCreated + "\nColor: " + color + "\nFilled: " + filled;

}

public abstract double getArea();

public abstract double getPerimeter();

}

**Triangle**

public class triangle extends GeometricObject{

private double side1 = 1; //holds the side1 data and defaults to 1

private double side2 = 1; //holds the side2 data and defaults to 1

private double side3 = 1; //holds the side3 data and defaults to 1

//no args constructor

public triangle(){

}

//constructor that takes in 3 doubles

public triangle(double temp1, double temp2, double temp3){

side1 = temp1;

side2 = temp2;

side3 = temp3;

}

//returns the info in side1

public double getSide1()

{

return side1;

}

//returns the info in side2

public double getSide2()

{

return side2;

}

//returns the info in side3

public double getSide3()

{

return side3;

}

//sets side1 to the double sent over

public void setSide1(double temp)

{

side1 = temp;

}

//sets side2 to the double sent over

public void setSide2(double temp)

{

side2 = temp;

}

//sets side3 to the double sent over

public void setSide3(double temp)

{

side3 = temp;

}

//gets the area of a triangle. Overides the function in GeometricObject

@Override

public double getArea()

{

double s = (side1 + side2 + side3) / 2; //finds s (needed for the area equation)

return Math.sqrt(s \* (s - side1) \* (s - side2) \* (s - side3)); //caculates and returns the area

}

//gets the perimeter of a triangle. Overides the function in GeometricObject

@Override

public double getPerimeter()

{

return (side1 + side2 + side3); //caculates and returns the perimeter

}

//displays the info about a triangle. Overides the function in GeometricObject

@Override

public String toString()

{

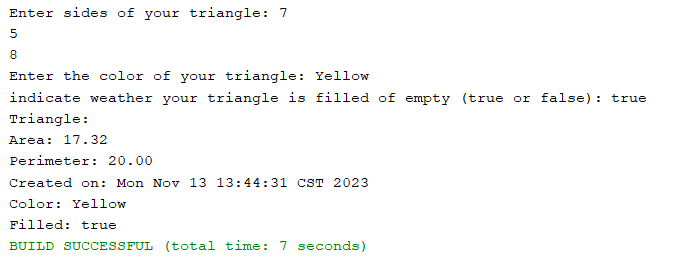
return "Triangle:\nArea: " + String.format("%.2f", getArea()) + "\nPerimeter: " +

String.format("%.2f", getPerimeter()) + super.toString();

}

}

**Output**



**Program Thirteen Part Two:**

//summary: This program takes the rational class made in 13.13 and changes the internal representation of the fraction.

// it then replaces the methods in the class to work with the new representation.

//name: Jenna Wolf

//class: Fundamentals of Programming, CS155 - 01

//instructor: Dr. Art Kazmierczak

//date: 11/12/2023

public class Main

{

public static void main(String[] args)

{

Rational fractionOne = new Rational(4, 5); //holds the fractionOne data

Rational fractionTwo = new Rational(6, 8); //holds the fractionTwo data

//outputs adding, subtracting, multiplying, and dividing the fractions

System.out.println(fractionOne + " + " + fractionTwo + " = " + fractionOne.add(fractionTwo));

System.out.println(fractionOne + " - " + fractionTwo + " = " + fractionOne.subtract(fractionTwo));

System.out.println(fractionOne + " \* " + fractionTwo + " = " + fractionOne.multiply(fractionTwo));

System.out.println(fractionOne + " / " + fractionTwo + " = " + fractionOne.divide(fractionTwo));

//tells the user which fraction is larger

if(fractionOne.compareTo(fractionTwo) == 1)

System.out.println("Fraction one is larger");

else if (fractionOne.compareTo(fractionTwo) == -1)

System.out.println("Fraction two is larger");

else

System.out.println("The two fractions are equal");

}

}

**Rational**

public class Rational extends Number implements Comparable<Rational> {

private long[] r = new long[2]; //holds the r data which is set to a size of 2

//no args constructor

public Rational() {

this(0, 1);

}

//constructor that takes in a numerator and denominator

public Rational(long num, long den) {

long gcd = gcd(num, den);

r[0] = (den > 0 ? 1 : -1) \* num / gcd;

r[1] = Math.abs(den) / gcd;

}

//finds the greatest common denominator of a function

private static long gcd(long n, long d)

{

long n1 = Math.abs(n);

long n2 = Math.abs(d);

int gcd = 1;

for (int k = 1; k <= n1 && k <= n2; k++)

{

if (n1 % k == 0 && n2 % k == 0)

gcd = k;

}

return gcd;

}

//returns the data stored in the numerator

public long getNumerator()

{

return r[0];

}

//returns the data stored in the denominator

public long getDenominator()

{

return r[1];

}

//adds two fractions and returns the value

public Rational add(Rational r2)

{

long n = r[0] \* r2.getDenominator() + r[1] \* r2.getNumerator();

long d = r[1] \* r2.getDenominator();

return new Rational(n, d);

}

//subtracts two fractions and returns the value

public Rational subtract(Rational r2)

{

long n = r[0] \* r2.getDenominator() - r[1] \* r2.getNumerator();

long d = r[1] \* r2.getDenominator();

return new Rational(n, d);

}

//multiplies two fractions and returns the value

public Rational multiply(Rational r2)

{

long n = r[0] \* r2.getNumerator();

long d = r[1] \* r2.getDenominator();

return new Rational(n, d);

}

//divides two fractions and returns the value

public Rational divide(Rational r2)

{

long n = r[0] \* r2.getDenominator();

long d = r[1] \* r2.getNumerator();

return new Rational(n, d);

}

//makes a toString method for a fraction. overrides the Number class

@Override

public String toString()

{

if (r[1] == 1)

return r[0] + "";

else

return r[0] + "/" + r[1];

}

//finds if two fractions are equal to eachother. overrides the Number class

@Override

public boolean equals(Object other)

{

if ((this.subtract((Rational)(other))).getNumerator() == 0)

return true;

else

return false;

}

//returns an int value. overrides the Number class

@Override

public int intValue()

{

return (int)doubleValue();

}

//returns an float value. overrides the Number class

@Override

public float floatValue()

{

return (float)doubleValue();

}

//returns a double value. overrides the Number class

@Override

public double doubleValue()

{

return r[0] / r[1];

}

//returns a long value. overrides the Number class

@Override

public long longValue()

{

return (long)doubleValue();

}

//compares two fractions to each other and returns a value based on which is greater. overrides the Number class

@Override

public int compareTo(Rational o)

{

if (this.subtract(o).getNumerator() > 0)

return 1;

else if (this.subtract(o).getNumerator() < 0)

return -1;

else

return 0;

}

}

**Output**

