```java
/**
   Source Code: BetterRectangle.java
   Author: Alp Karavil
   Student ID: 5827197
   Assignment: Program #4 BetterRectangle

   Course: 3337-Programming II
   Section: U09
   Instructor: William Feild
   Due Date: 10/18/2018 - Beginning of class

   I hereby certify that this collective work is my own
   and none of it is the work of any other person or entity

   Signature:


   Language: Java
   Compile/Run:
   javac BetterRectangle.java

   ------------------------------------------------------------------

   Description:
   This class extends the java.awt.Rectangle class to implement more
   sophisticated methods. The accessor methods include area, perimeter, slope,
   and mid-point calculators. This class contains only one mutator method,
   scaleBy (int multiplier), which allows the user to scale the current
   BetterRectangle object by a fixed positive integer. Combined with these
   accessor and mutator methods are utility methods which allow the user to
   check if two BetterRectangles are congruent, equivalent, similar, or
   concentric with regards to each other.

   This class overrides the super's (Rectangle) toString() and equals()
   method to make sure String output is formatted to the user's needs, and
   equal checks are correct between BetterRectangle objects.

   This class does not add any new instance variables, and does not directly
   access the super class's (Rectangle) instance variables, but instead
   accesses them through super's provided methods.

   Input:
   Use one of 4 constructors to create a new BetterRectangle object. These
   constructors allow for inputs such as no input, just x and y values, x and
   y values combined with the width and height, or a current BetterRectangle
   object that the values (x,y,width,height) will be copied from.

   If no input is provided, the BetterRectangle will be created at the origin
   (0,0) and the length and width will be initialized at 1.

   Output:
   This program has no output other than when the toString() method is called.
   If a user wants to request specific values about the rectangle, accessor
   methods should be called.

   Process:
   1. Create a BetterRectangle object with the use of one of the 4 constructors.
   2. Use mutator, accessor or utility methods, or print BetterRectangle
   information with toString() if needed.

   No particular algorithms are used.

   Known Bugs: None
**/

//Import the Rectangle (super) class which this class extends
import java.awt.Rectangle;
//Import the Point class to represent points on a 2D plane.
import java.awt.Point;
//Import the BigDecimal class for precise decimal calculations/rounding.
import java.math.BigDecimal;
//Import RoundingMode, which will be used to round and round UP decimals.
import java.math.RoundingMode;
//Import the NaN (not a number) value from the Float class.
import static java.lang.Float.NaN;

public class BetterRectangle extends Rectangle
{
   //Default (unit rectangle) width and height values.
   public final static int UNIT_RECTANGLE_WIDTH = 1;
   public final static int UNIT_RECTANGLE_HEIGHT = 1;
   //Origin (2D) x and y values.
   public final static int ORIGIN_X_POSITION = 0;
   public final static int ORIGIN_Y_POSITION = 0;

   /**
    * Constructs a new BetterRectangle with its upper-left corner at (0,0) in
    * the 2D coordinate space with a width and length of 1.
    */
   public BetterRectangle()
   {
      super();
      this.setSize(UNIT_RECTANGLE_WIDTH, UNIT_RECTANGLE_HEIGHT);
      this.setLocation(ORIGIN_X_POSITION, ORIGIN_Y_POSITION);
   }

   /**
    * Constructs a new BetterRectangle with its upper-left corner at (0,0) in
    * the 2D coordinate space, with a width of rectangleWidth and height of
    * rectangleHeight inputs.
    * @param rectangleWidth the width of the BetterRectangle
    * @param rectangleHeight the height of the BetterRectangle
    */
   public BetterRectangle(int rectangleWidth, int rectangleHeight)
   {
      super();
      this.setSize(rectangleWidth, rectangleHeight);
      this.setLocation(ORIGIN_X_POSITION, ORIGIN_Y_POSITION);
   }

   /**
    * Constructs a new BetterRectangle with its upper-left corner at the
    * coordinates (xPosition, yPosition) in the 2D coordinate space, with a
    * width of rectangleWidth and height of rectangleHeight inputs.
    * @param xPosition the x position of the BetterRectangle
    * @param yPosition the y position of the BetterRectangle
    * @param rectangleWidth the width of the of the BetterRectangle
    * @param rectangleHeight the height of the BetterRectangle
    */
   public BetterRectangle(int xPosition, int yPosition, int rectangleWidth,
                              int rectangleHeight)
   {
      super();
      this.setLocation(xPosition, yPosition);
      this.setSize(rectangleWidth, rectangleHeight);
   }

   /**
    * Constructs a new BetterRectangle, initialized to match to characteristics
    * (x,y position, width and height) of the specified BetterRectangle.
    * @param rectangleInput the BetterRectangle from which to copy the values
    *                         of to this new BetterRectangle object.
    */
   public BetterRectangle(BetterRectangle rectangleInput)
   {
      super();

      //Store the values of the inputted BetterRectangle.
      int xPosition = (int) rectangleInput.getX();
      int yPosition = (int) rectangleInput.getY();
      int width = (int) rectangleInput.getWidth();
      int height = (int) rectangleInput.getHeight();

      //Update the values of the new BetterRectangle to match the input.
      this.setLocation(xPosition, yPosition);
      this.setSize(width, height);
   }

   /**
    * Checks whether two BetterRectangle objects are equal.
    * @param rectangleInput The Object to compare this BetterRectangle
    * @return true if the objects are equal, false otherwise
    */
   @Override
   public boolean equals(Object rectangleInput)
   {
      //If there is no object reference, return false.
      if (rectangleInput == null)
      {
         return false;
      }

      //Cast the Object to a BetterRectangle object for value checks.
      BetterRectangle inputRectangle = (BetterRectangle) rectangleInput;
      //Store the rectangle characteristics.
      int inputWidth = (int) inputRectangle.getWidth();
      int inputHeight = (int) inputRectangle.getHeight();
      int inputX = (int) inputRectangle.getX();
      int inputY = (int) inputRectangle.getY();

      //Return false if any characteristics are not equal
      if (inputWidth != (int) this.getWidth())
      {
         return false;
      }
      else if (inputHeight != (int) this.getHeight())
      {
         return false;
      }
      else if (inputX != (int) this.getX())
      {
         return false;
      }
      else if (inputY != (int) this.getY())
      {
         return false;
      }

      //If no values are different (they are equal), return true
      return true;
   }

   /**
    * Returns a string representation of this BetterRectangle, which includes
    * its 2D location, width, height, area, perimeter, slope, and mid-point
    * location.
    * @return A String representation of this BetterRectangle and its values
    */
   @Override
   public String toString()
   {
      String output = super.toString() + "[area=" + this.getArea()
              + ",perimeter=" + this.getPerimeter() + ",slope="
              + this.getSlope() + ",mid-point=" + this.getMidPoint() + "]";

      return output;
   }

   /**
    * Returns the area of this BetterRectangle in integer form.
    * @return the area of this BetterRectangle
    */
   public int getArea()
   {
      int area = (int) (this.getWidth() * this.getHeight());
      return area;
   }

   /**
    * Returns the perimeter of this BetterRectangle in integer form.
    * @return the perimeter of this BetterRectangle
    */
   public int getPerimeter()
   {
      //Perimeter = width + width + height + height = 2 * (width + height)
      int perimeter = (int) (2 * (this.getWidth() + this.getHeight()));
      return perimeter;
   }

   /**
    * Returns the slope of this BetterRectangle, which can be calculated by
    * dividing the height by the width (height / width), in (float) form.
    *
    * (All output is currently rounded to 2 decimal places.)
    *
    * If the width is 0, the method will return the a NaN value as a slope
    * cannot be calculated.
    * @return the slope of this BetterRectangle rounded to 2 decimal places, NaN
    * if width is 0 and slope cannot be calculated.
    */
   public float getSlope()
   {
      //If width is 0, slope cannot be found (either undefined/infinity)
      if (this.getWidth() == 0) {
         return NaN;
      }
      //The decimal accuracy which the slope value will be rounded to.
      final int SLOPE_DECIMAL_ACCURACY = 2;
      //Calculate slope value
      float slope = (float) (this.getHeight() / this.getWidth());
      //Round slope value to 2 decimal places.
      slope = roundFloat(slope, SLOPE_DECIMAL_ACCURACY);

      return slope;
   }

   /**
    * Returns a Point object representing mid-point of the line between the
    * enter of this BetterRectangle and its "anchor" point (upper-left
    * corner). The mid-point is found through the formula (x1+x2) / 2 for the
    * x value, and (y1+y2) / 2 for the y value.
    *
    * The midpoint coordinates are rounded UP to the closest integer.
    * @return a Point object representing the midpoint of two points, the
    * center of the BetterRectangle and its "anchor" point (upper-left corner).
    * The coordinates are rounded UP to the nearest integer.
    */
   public Point getMidPoint()
   {
      //Center X and Y values of this BetterRectangle
      double centerX = this.getCenterX();
      double centerY = this.getCenterY();

      //Anchor (upper-left corner) X and Y values of this BetterRectangle
      double anchorX = this.getX();
      double anchorY = this.getY();

      //Decimal accuracy is 0, as an integer has no decimal places.
      final int INT_DECIMAL_ACCURACY = 0;

      //Use midpoint formula to calculate midpoint between center and anchor.
      int midPointX = (int) roundUpDouble((centerX + anchorX) / 2.0,
              INT_DECIMAL_ACCURACY);
      int midPointY = (int) roundUpDouble((centerY + anchorY) / 2.0,
              INT_DECIMAL_ACCURACY);
      Point midPoint = new Point(midPointX, midPointY);

      return midPoint;
   }

   /**
    * Checks whether two BetterRectangle objects are congruent with each other.
    * Congruency is true if the (width + height) of both objects are equal.
    * @param rectangleInput BetterRectangle object that will be compared
    * @return true if the rectangles are congruent, false otherwise
    */
   public boolean congruent(BetterRectangle rectangleInput)
   {
      //Store the width and height of the input and their total (width + height)
      int inputWidth = (int) rectangleInput.getWidth();
```

```java
      int inputHeight = (int) rectangleInput.getHeight();
      int inputTotal = inputWidth + inputHeight;

      //Store this BetterRectangle's width and height, and total.
      int thisWidth = (int) this.getWidth();
      int thisHeight = (int) this.getHeight();
      int thisTotal = thisWidth + thisHeight;

      //If the totals (width + height) are equal, they are congruent rectangles.
      if (inputTotal == thisTotal)
      {
         return true;
      }

      return false;
   }

   /**
    * Checks whether two BetterRectangle objects are equivalent. Equivalency
    * is true if perimeters of both objects are equal.
    * @param rectangleInput BetterRectangle object that will be compared
    * @return true if the rectangles are equivalent, false otherwise
    */
   public boolean equivalent(BetterRectangle rectangleInput)
   {
      int inputPerimeter = rectangleInput.getPerimeter();
      //If perimeters are equal, they are equivalent rectangles.
      if (inputPerimeter == this.getPerimeter())
      {
         return true;
      }
      return false;
   }

   /**
    * Checks whether two BetterRectangle objects are similar. Similarity is
    * true if area for both objects are equal.
    * @param rectangleInput BetterRectangle object that will be compared
    * @return true if the rectangles are similar, false otherwise
    */
   public boolean similar(BetterRectangle rectangleInput)
   {
      //Store areas of both rectangles.
      int inputArea = rectangleInput.getArea();
      int thisArea = this.getArea();

      //If areas are equal, they are similar rectangles.
      if (inputArea == thisArea)
      {
         return true;
      }
      return false;

   }

   /**
    * Checks whether two BetterRectangle objects are concentric. These
    * BetterRectangles are concentric if the mid-point is the same for both.
    * @param rectangleInput BetterRectangle object that will be compared
    * @return true if the rectangles are concentric, false otherwise
```

```java
     */
    public boolean concentric(BetterRectangle rectangleInput)
    {
       //Store both mid-points in Point objects.
       Point inputMidPoint = rectangleInput.getMidPoint();
       Point thisMidPoint = this.getMidPoint();

       //Return true if the mid-points have equal x and y values, false otherwise
       return inputMidPoint.equals(thisMidPoint);

    }

    /**
     * Multiplies the height & width by an inputted positive
     * integer; returns true upon success, returns false upon failure (such as
     * entering a non-positive number as the scale multiplier)
     * @param multiplier the scale multiplier for the height and width
     * @return true upon success, false upon failure (ex: non-positive scale)
     */
    public boolean scaleBy(int multiplier)
    {
       //If multiplier is non-positive (negative or 0) operation fails.
       if (multiplier < 1)
       {
          return false;
       }

       //Store updated (scaled) width and height values.
       int updatedWidth = (int) this.getWidth() * multiplier;
       int updatedHeight = (int) this.getHeight() * multiplier;
       //Update the size of current BetterRectangle with updated values.
       this.setSize(updatedWidth, updatedHeight);

       return true;
    }

    /**
     * Private method that rounds float inputs to specified decimal accuracy
     * using BigDecimal objects and a rounding mode of HALF_UP (0.5 goes to 1).
     * @param input float object that will be rounded
     * @param decimalAccuracy decimal place that the float will be rounded to
     * @return rounded float value to specified decimal places
     */
    private float roundFloat(float input, int decimalAccuracy)
    {
       BigDecimal roundedValue = new BigDecimal(String.valueOf(input));
       //Round the BigDecimal object of input to parameter decimal accuracy.
       roundedValue = roundedValue.setScale(decimalAccuracy,
               RoundingMode.HALF_UP);
       return roundedValue.floatValue();
    }

    /**
     * Private method that rounds double values UP to the specified decimal
     * accuracy. This method uses BigDecimal objects and a rounding mode of UP
     * (0.25 goes to 1).
     * @param input double value that will be rounded
     * @param decimalAccuracy decimal place that the double will be rounded to
     * @return rounded UP double value to specified decimal places
     */
```

```java
   private double roundUpDouble(double input, int decimalAccuracy)
   {
      BigDecimal roundedValue = new BigDecimal(String.valueOf(input));
      //Round the BigDecimal object UP at the specified decimal accuracy.
      roundedValue = roundedValue.setScale(decimalAccuracy,
              RoundingMode.UP);
      return roundedValue.doubleValue();
   }
}
```