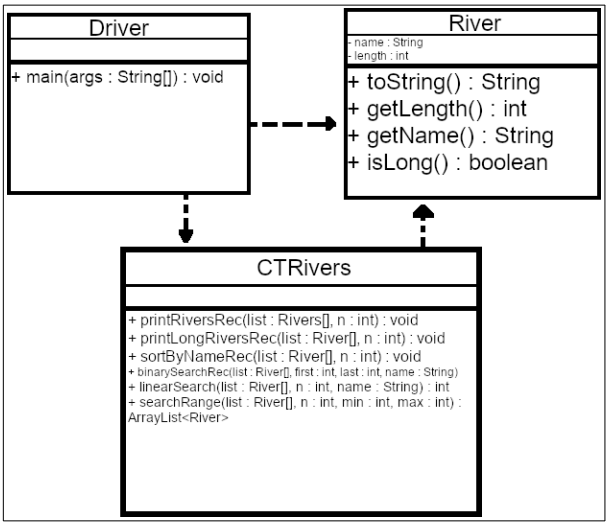
UML



Output

```
Options
The rivers in Connecticut:
River "Naugatuck" is 40 miles long
River "Pawcatuck" is 34 miles long
River "Quinebaug" is 69 miles long
River "Shepaug" is 26 miles long
River "Connecticut" is 407 miles long
River "Still" is 25 miles long
River "Quinnipiac" is 46 miles long
River "Housatoic" is 139 miles long

The long rivers (> 30 miles):
River "Naugatuck" is 40 miles long
River "Pawcatuck" is 34 miles long
River "Quinebaug" is 69 miles long
River "Connecticut" is 407 miles long
River "Quinnipiac" is 46 miles long
River "Housatoic" is 139 miles long

Sorted rivers:
River "Connecticut" is 407 miles long
River "Housatoic" is 139 miles long
River "Quinebaug" is 69 miles long
River "Quinnipiac" is 46 miles long
River "Naugatuck" is 40 miles long
River "Still" is 25 miles long
River "Shepaug" is 26 miles long
River "Pawcatuck" is 34 miles long

Using binary search to find "Still":
River "Still" is 25 miles long

Linear river search for "Pawcatuck":
River "Pawcatuck" is 34 miles long

Rivers with length between 30 and 50 miles:
River "Quinnipiac" is 46 miles long
River "Naugatuck" is 40 miles long
River "Pawcatuck" is 34 miles long
Can only enter input while your programming is running
BlueJ: Terminal Window - Rivers
```

Driver.java

```
import java.util.Scanner;
import java.io.File;

/**
 * @author (Kyle Guarco)
 * @version (July 17, 2020)
 */
public class Driver
{
    public static void main(String[] args)
    {
        River[] rivers = new River[10];
        int count = 0;

        try {
            Scanner scan = new Scanner(new File("inData.txt"));

            while (scan.hasNextLine())
            {
                String[] line = scan.nextLine().split(" ");

                rivers[count] = new River(line[0], Integer.parseInt(line[1]));
                count++;
            }

            scan.close();
        } catch (Exception e) {
            System.err.println("Error reading inData.txt");
            System.err.println(e.getMessage());

            System.exit(-1);
        }

        CTRivers ctr = new CTRivers();

        System.out.println("The rivers in Connecticut: ");
        ctr.printRiversRec(rivers, count);

        System.out.println("\nThe long rivers (> 30 miles):");
        ctr.printLongRiversRec(rivers, count);

        System.out.println("\nSorted rivers: ");
        ctr.sortByNameRec(rivers, count);
        ctr.printRiversRec(rivers, count);
```

```
        System.out.println("\nUsing binary search to find \"Still\":");
        System.out.println(rivers[ctr.binarySearchRec(rivers, 0, count - 1, "Still")]);

        System.out.println("\nLinear river search for \"Pawcatuck\":");
        System.out.println(rivers[ctr.linearSearch(rivers, count, "Pawcatuck")]);

        System.out.println("\nRivers with length between 30 and 50 miles:");
        ctr.searchRange(rivers, count, 30, 50).forEach(System.out::println);
    }
}
```

CTRivers.java

```
import java.util.ArrayList;

/**
* @author (Kyle Guarco)
* @version (July 17, 2020)
*/
public class CTRivers
{
    // A function I added which just prints all the rivers.
    public void printRiversRec(River[] list, int n)
    {
        if (n < 1)
            return;

        printRiversRec(list, n - 1);

        System.out.println(list[n - 1]);
    }

    public void printLongRiversRec(River[] list, int n)
    {
        if (n < 1)
            return;

        printLongRiversRec(list, n - 1);

        if (list[n - 1].isLong())
            System.out.println(list[n - 1]);
    }

    public void sortByNameRec(River[] list, int n)
    {
        if (n < 1) return;

        sortByNameRec(list, n - 1);
```

```
    River river = list[n - 1];

    int index = 0;
    River compare = list[index];

    while (index < n - 1)
    {
        if (compare.getName().compareTo(river.getName()) > 0)
            break;
        index++;
        compare = list[index];
    }

    list[n - 1] = compare;
    list[index] = river;
}

// sortByNameRec() must be called first.
public int binarySearchRec(River[] list, int first, int last, String name)
{
    if (first > last)
        return -1;

    int index = (first + last) / 2;
    String nameIdx = list[index].getName();

    if (nameIdx.equals(name))
        return index;

    String nameFirst = list[first].getName();
    String nameLast = list[last].getName();

    if (nameIdx.compareTo(name) < 0)
    {
        first = index;
    } else if (nameIdx.compareTo(name) > 0) {
        last = index;
    }

    return binarySearchRec(list, first, last, name);
}

public int linearSearch(River[] list, int n, String name)
{
    for (int i = 0; i < n; i++)
    {
```

```
            if (list[i].getName().equals(name))
                return i;
        }

        return -1;
    }

    // Returns rivers with length min->max (inclusive)
    public ArrayList<River> searchRange(River[] list, int n, int min, int max)
    {
        ArrayList<River> rivers = new ArrayList<River>();

        for (int i = 0; i < n; i++)
        {
            int length = list[i].getLength();

            if (length >= min && length <= max)
                rivers.add(list[i]);
        }

        return rivers;
    }
}
```

River.java

```
/**
 * @author (Kyle Guarco)
 * @version (July 17, 2020
 */
public class River
{
    private String name;
    // 'length' in miles
    private int length;

    public River(String name, int length)
    {
        this.name = name;
        this.length = length;
    }

    @Override
    public String toString()
    {
        return String.format("River \"%s\" is %d miles long", name, length);
    }
```

```
    public String getName()
    {
        return name;
    }

    public int getLength()
    {
        return length;
    }

    public boolean isLong()
    {
        return length > 30;
    }
}
```