In this task, you will create a simple spatial database for handling inserting, deleting, and performing queries on a collection of rectangles. The data structure used to store the collection will be a Binary Search Tree (BST).

Overview:

Some of the classes in this task are:

1. Define class `Node`.
2. Define class `BST`.
3. Define class `RectangleDB`.
4. Define class `CommandProcessor`.
5. Define class `World`.

Note that this task is similar to task 1 for skiplist, and some of the classes defined there are reusable in this task.

Invocation and Input/Output

The program will be invoked from the command-line as:

```
%> java RectangleDB {commandFile}
```

where: `RectangleDB` is then name of the program. The file where you have your `main()` method must be called `RectangleDB.java` . `commandFile` is the name of the command file to read. It is a text file.

Your program will read from the text file `{command-file}` a series of commands, with one command per line. The program should terminate after reading the end of the file. The commands are free-format in that any number of spaces may come before, between, or after the command name and its parameters. All commands should generate the required output message. **All output should be written to the standard output console.**

The command file may contain any mix of the following additional commands. In the following description, terms in `{ }` are parameters to the command. The curly-brace characters '{' and '}' seen below will not actually be present. However, the output of several commands will include parentheses.

INSERT

Insert a rectangle named `name` with upper left corner `(x, y)`, width `w` and height `h`. It is permissible for two or more rectangles to have the same name, and it is permissible for two or more rectangles to have the same spatial dimensions and position. The name must begin with a letter, and may contain letters, digits, and underscore characters. Names are case sensitive. A rectangle should be rejected for insertion if its height or width are not greater than 0. All rectangles must fit within the "world box" that is 1024 by 1024 units in size and has upper left corner at (0, 0). If a rectangle is all or partly out of this world box, it should be rejected for insertion.

Command Format	`insert {name} {x} {y} {w} {h}`
Command Example	`insert  hello 2  5  1  5`
Output on Success	`Rectangle accepted: (hello, 2, 5, 1, 5)`

Example	
Output on Failure Example	`Rectangle rejected: (hello, 1020, 1020, 1, 7)`

REMOVE-BY-NAME

Remove the rectangle with name `name`. If two or more rectangle have the same name, then any one such rectangle may be removed. If no rectangle exists with this name, it should be so reported.

Command Format	`remove {name}`
Command Example	`remove someRect`
Output on Success Example	`Display Nothing`
Output on Failure Example	`Rectangle rejected: (rectangleB)`

REMOVE-BY-COORDS

Remove the rectangle with the specified dimensions. If two or more rectangles have the same dimensions, then any one such rectangle may be removed. If no rectangle exists with these dimensions, it should be so reported.

Command Format	`remove {x} {y} {w} {h}`
Command Example	`remove  2  5  1  5`

Output on Success Example	`Display Nothing`
Output on Failure Example	`Rectangle rejected: (2, 5, 1, 5)`
Output on Failure Example2	`Rectangle rejected: (-15, 5, 1, 5)`

Region Search

Report all rectangles currently in the database that intersect the query rectangle specified by the `regionsearch` parameters. For each such rectangle, list out its name and coordinates. A `regionsearch` command should be rejected if the height or width are not greater than 0. However, it is (syntactically) acceptable for the regionsearch rectangle to be all or partly outside of the 1024 by 1024 world box.

Command Format	`regionsearch {x} {y} {w} {h}`
Command Example	`regionsearch  -900 5 5000 20`

Output on Success Example	`Rectangles intersecting region (-900, 5, 5000, 20):`
	`(a, 45, 0, 10, 10)`
	`(b, 400, 0, 100, 310)`

Output on Failure Example	`Display Nothing`

Intersections

Report all pairs of rectangles within the database that intersect.

Command Format	`intersections`
Command Example	`intersections`

Output on Success Example	`Intersections pairs:`
	`(goodRect, 5, 3, 56, 56 | goodRect3, 25, 3, 6, 6)`
	`(goodRect3, 25, 3, 6, 6 | goodRect, 5, 3, 56, 56)`

Output on Failure Example	`Intersection pairs:`

SEARCH

Return the information about the rectangles(s), if any, that have name `{name}`.

Command Format	`search {name}`
Command Example	`search mainRect`
Output on Success Example	`Rectangles found: (mainRect, 25, 3, 6, 6)`
Output on Success Example2 (multiple matches)	`Rectangles found: (mainRect, 25, 3, 6, 6)` `Rectangles found: (mainRect, 111, 23, 16, 16)`
Output on Failure	`Rectangle not found: mainRect`

Example	

DUMP

Return a "dump" of the BST. The BST dump should print out each BST node (use the ***in-order traversal***). For each BST node, print that node's depth and value (rectangle info). At the end, please print out the size of the BST.

Command Format	dump
Command Example	dump
Output on Success Example	BST dump: Node has depth 0, Value (name1, 4, 6, 2, 2) Node has depth 1, Value (name2, 1, 0, 2, 4) Node has depth 2, Value (name3, 1, 2, 1023, 4) BST size is: 3
Output on Success Example2	BST dump: Node has depth 0, Value (null) BST size is: 0

Rules

1. All data fields should be declared private or protected

2. You may add your own helper methods or data fields, but they should be private or protected. You may add additional constructors which are public.

3. When commenting code, use JavaDoc-style comments. Include @param and/or @return tags to explain what is passed in or returned wherever it isn't obvious. Any other JavaDoc tags are optional.

Implementation

The rectangles will be maintained in a BST, sorted by the name. Use `compareTo()` to determine the relative ordering of two names, and to determine if two names are identical. You are using the BST to maintain your list of rectangles, but the BST is a general container class. Therefore, it should not be implemented to know anything about rectangles.

Be aware that for this task, the BST is being asked to do two things. First, the BST will handle searches on rectangle name, which acts as the record's key value. The BST can do this efficiently, as it will organize its records using the name as the search key. But you also need to do several things that the BST cannot handle well, including removing by rectangle shape, doing a region search, and computing rectangle

intersections. So you will need to add functions to the BST to handle these actions. Make sure you handle these actions in a general way that does not require the BST to understand its data type.

The biggest implementation difficulty that you are likely to encounter relates to traversing the BST during the intersections command. The problem is that you need to make a complete traversal of the BST for each rectangle in the BST (comparing it to all of the other 2 rectangles). This leads to the question of how do you remember where you are in the “outer loop" of the operation during the processing of the “inner loop" of the operation. One design choice is to augment the BST with an iterator class. An iterator object tracks a current position within the BST, and has a method that permits the position of the iterator object within the BST to move forward. In this way, one iterator object can be tracking the current rectangle in the “outer loop" of the process, while a second iterator can be used to track the current rectangle for the “inner loop".

For the `regionsearch` and `intersections` commands, you need to determine intersections between two rectangles. Rectangles whose edges abut one another, but which do not overlap, are not considered to intersect. For example, (10, 10, 5, 5) and (15, 10, 5, 5) do NOT overlap, while (10, 10, 5, 5) and (14, 10, 5 5) do overlap. Note that rectangles (10, 10, 5, 5) and (11, 11, 1, 1) also overlap.

Programming Standards

- Source files should be under 600 lines.
- There should be a single class in each source file. You can make an exception for small inner classes (less than 100 lines including comments) if the total file length is less than 600 lines.
- You are required to submit your own test cases with your program, and part of your grade will be determined by how well your test cases test your program, as defined by evaluation of code coverage

Allowed Java Classes

You are not permitted to use Java classes that implement complex data structures, or that handle the merging of multiple files. This includes, for example, HashMap, Vector, or any other classes that implement lists, hash tables, heaps, trees, graphs or the like. You may use the standard array operators. You may use typical classes for string processing, byte array manipulation, parsing, etc.

The following classes/interfaces are allowed for this project: **LinkedList(util), Itererator(util), Rectangle(awt).**