CS3114 (Summer II 2017)
PROGRAMMING ASSIGNMENT#2
Due August 1 at 11:59 PM

Background

This project continues the theme of a Geographic Information System that was begun in Project 1. This time, the focus is organizing the city records into a database for fast search. For each city you will store the name and its location (X and Y coordinates). Searches can be by either name or location. Searches by name are handled by the hash table from Project 1. You will implement a spatial data structure called a Bintree to support searches by position. To see how a Bintree works, see Module 26.4 in the complete version of the OpenDSA textbook. You should pay particular attention to the discussion regarding parameter passing of coordinates in recursive functions, and to the discussion regarding flyweights (see Module 11.1 in the complete version of the OpenDSA textbook). Both of these discussions are relevant to your implementation. You should use a flyweight design for your empty leaf nodes. A composite design is recommended to make your code simple, but is not required.

Invocation and I/O Files

Your program must be named Binprog, and should be invoked as:
java Binprog <initial-pool-size> <hash-size> <command-file>
where <command-file> is a command line argument for the name of the command file. Your program should write to standard output (System.out). The program should terminate when it reads the end of file mark from the input file. The input for this project will consist of a series of commands (some with associated parameters, separated by spaces), one command for each line. Commands are free format in that an arbitrary number of additional spaces may be interspersed between parameters. The input file may also contain blank lines, which your program should ignore. You do not need to check for syntax errors in the command lines (although you do need to check for logical errors such as duplicate insertions or removals of non-existent cities).
Each input command should result in meaningful feedback in terms of an output message. Each input command should be echo'ed to the output. In addition, some indication of success or error should be reported. Some of the command specifications below indicate particular additional information that is to be output. Commands and their syntax are as follows. Note that a name is defined to be a string containing only upper and lower case letters and the underscore character.

insert x y name

A city at coordinate (x, y) with name "name" is entered into the database. That means you will store the city record in the memory manager, with its handle then stored once in hash table, and once in the Bintree. Spatially, the database should be viewed as a square whose origin is in the upper left (NorthWest) corner at position (0, 0). The world is 1024 by 1024 units wide, and so x and y are integers in the range 0 to 1023. The x-coordinate increases to the right, the y-coordinate increases going down. Note that it is an error to insert two cities with identical coordinates. In addition, it is also incorrect to have duplicate names for cities as we are using a hash table.

remove x y
The city with coordinate (x, y) is removed from the database (if it exists). That means it must be removed from both the Bintree and the hash table. Be sure to print the name and coordinates of the city that is removed.

remove name
A city with name "name" is removed from the database (if any exists). That means it must be removed from both the Bintree and the hash table. Be sure to print the name and coordinates of the city that is removed.

find name
Print the name and coordinates from the city record with name " name". You would search the hash table for this information.

find x y
Print the name and coordinates from the city record coordinates x and y. You would search the Bintree for this information.

regionsearch x y w h
Report all cities currently in the database that intersect the query rectangle specified by the regionsearch parameters. For each such city, list out its name and coordinates. In addition, you should report the number of Bintree nodes visited. You should validate the regionsearch parameters to make sure that the query rectangle doesn't fall entirely outside of the world square.

print
Print a listing of the Bintree nodes in preorder traversal order, one node per line, with each line indented by 2 spaces for each level in the tree (the root will indent 0 spaces since it is considered to be at level 0).
• For an internal node, print “I”.
• For an empty leaf node (the flyweight), print “E”.
• For a leaf node containing a data point, print NAME, X, Y where X is the x-coordinate, Y is the y-coordinate, and NAME is the city name.

hashtable
Print listing of the hash table's contents, one record per line. If a given slot in the table has no record, then print [EMPTY]. Print the value of the position handle along with the record.

freelist
Print a listing of the free blocks, in order of their occurrence in the freeblock list.

Example:
Note: in this example, statements enclosed in {} are comments to help you understand the example; comments do NOT appear in the data file!

```
insert 900 500 Blacksburg
insert 500 140 Roanoke
insert 910 510 New_York
print { print coords, and name for 3 cities }
remove 500 140 { it's there to remove }
find 901 501 {print info for one city }
```

Implementation:
All operations that traverse or descend the Bintree MUST be implemented recursively. You must use inheritance or an interface to implement Bintree nodes. You should declare either abstract base class or an interface, and define separate subclasses for the internal nodes and the leaf nodes. A Bintree internal node should store references to its children. Leaf nodes should store a (reference to a) city record object. Empty leaf nodes should be implemented by a flyweight. The empty leaf node can either be its own separate subclass, or an instance of the leaf node subclass.

Java Code:
For this project, you may only use standard Java classes, Java code that you have written yourself, and Java code supplied by the CS3114 instructor (see the class website for the distribution). You may not use other third-party Java code.

Programming Standards:
You must conform to good programming/documentation standards. Web-CAT will provide feedback on its evaluation of your coding style, and be used for style grading. Beyond meeting Web-CAT's checkstyle requirements, here are some additional requirements regarding programming standards.

- You should include a comment explaining the purpose of every variable or named constant you use in your program.
- You should use meaningful identifier names that suggest the meaning or purpose of the constant, variable, function, etc. Use a consistent convention for how identifier names appear, such as "camel casing".
- Always use named constants or enumerated types instead of literal constants in the code.
- Source files should not be too long. This ensures that you code follows the modularity principle.
- There should be a single class in each source file. You can make an exception for small inner classes.

We can't help you with your code unless we can understand it. Therefore, you should no bring your code to the GTAs or the instructors for debugging help unless it is properly documented and exhibits good programming style. Be sure to begin your internal documentation right from the start. You may only use code you have written, either specifically for this project or for earlier programs, or code provided by the instructor. Note that the OpenDSA code is not designed for the specific purpose of this assignment, and is therefore likely to require modification. It may, however, provide a useful starting point.

Deliverables:
You will implement your project using Eclipse, and you will submit your project using the Eclipse plugin to Web-CAT. Links to the Web-CAT client are posted at the class website. If you make multiple submissions, only your last submission will be evaluated. There is no limit to the number of submissions that you may make.
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 Web-CAT's evaluation of code coverage. Of course, your program must pass your own test cases. Part of your grade will also be determined by test cases that are provided by the graders. Web-CAT will report to you which test files have passed correctly, and which have not. Note that you will not be given a copy of these test files, only a brief description of what each accomplished in order to guide your own testing process in case you did not pass one of our tests.
When structuring the source files of your project, use a flat directory structure; that is, your source files will all be contained in the project "src" directory. Any subdirectories in the project will be ignored. You are permitted to work with a partner on this project. When you work with a partner, then only one member of the pair will make a submission. Be sure both names are included in the documentation. Whatever is the final submission from either of the pair members is what we will grade unless you arrange otherwise with the GTA.

Pledge:
Your project submission must include a statement, pledging your conformance to the Honor Code requirements for this course. Specifically, you must include the following pledge statement near the beginning of the file containing the function main() in your program.
The text of the pledge will also be posted online.

```
// On my honor:
//
// - I have not used source code obtained from another student,
// or any other unauthorized source, either modified or
// unmodified.
//
// - All source code and documentation used in my program is
// either my original work, or was derived by me from the
// source code published in the textbook for this course.
//
// - I have not discussed coding details about this project with
// anyone other than my partner (in the case of a joint
// submission), instructor, ACM/UPE tutors or the TAs assigned
// to this course. I understand that I may discuss the concepts
// of this program with other students, and that another student
// may help me debug my program so long as neither of us writes
// anything during the discussion or modifies any computer file
// during the discussion. I have violated neither the spirit nor
// letter of this restriction.
```

<u>Programs that do not contain this pledge will not be graded.</u>