

## CS3114 (Fall 2025)

### PROGRAMMING ASSIGNMENT #4

Due Wednesday, December 10 @ 11:00 PM for 100 points

Due Tuesday, December 9 @ 11:00 PM for 10 point bonus

Note: This project also has three intermediate milestones. See the Piazza forum for details.

#### Assignment:

For this project, you will implement a small piece in an Air Traffic Control (ATC) system. Tracking the locations of many objects and determining if any of the objects are at risk of colliding is a critical part of any such system.

#### Design Considerations:

Your project will implement two data structures: A (3 dimensional) Bintree (see OpenDSA Module 15.6) and a Skip List (see Module 15.1). Both of these data structures will store various objects that extends a class with the following methods.

```
public class AirObject {
    public int getXorig() {}
    public int getYorig() {}
    public int getZorig() {}
    public int getXwidth() {}
    public int getYwidth() {}
    public int getZwidth() {}
    public String getName() {}
}
```

The classes that extend `AirObject` and their data fields are as follows:

```
AirPlane {name} {x} {y} {z} {xwid} {ywid} {zwid} {carrier} {flight#} {#engines}
Balloon {name} {x} {y} {z} {xwid} {ywid} {zwid} {type} {ascent_rate}
Bird {name} {x} {y} {z} {xwid} {ywid} {zwid} {type} {number}
Drone {name} {x} {y} {z} {xwid} {ywid} {zwid} {brand} {#engines}
Rocket {name} {x} {y} {z} {xwid} {ywid} {zwid} {ascent_rate} {trajectory}
```

All of these numbers are integers, except for Rocket trajectory, which is a floating point number. See the sample test file in the starter kit for details. The (x, y, z) coordinates must be integers in the range 0 to 1023, and the widths must be integers in the range 1 to 1024.

The Skip List should accept any Comparable class object, with the classes that extend `AirObject` being comparable on their `Name` field. Names are non-empty, non-null strings. The SkipList should not directly know about the `AirObject` class. The Bintree is permitted to be implemented explicitly for three dimensions, and explicitly to handle objects that extend/implement `AirObject`.

The Bintree represents a three-dimensional world that is 1024 units in each dimension. Since your Bintree is operating in three dimensions, it must rotate through the x, then, y, then z coordinates for making node split decisions. The objects being stored in the Bintree represent 3D boxes (not point objects, as shown in the OpenDSA examples). Each object is stored in every leaf node of the Bintree that it intersects. The splitting criteria for a leaf node is this: A leaf node splits if it contains more than three boxes, unless all of the boxes have a non-empty intersection box. Note

that if two boxes are adjacent (share a face), but do not overlap, then they are **not** considered to intersect.

Your Bintree implementation must meet the following design requirements. You must use a class hierarchy for the nodes, with a base node interface and classes that implement this interface for the internal and leaf nodes. You might choose to have separate classes for full and empty leaf nodes, or you might use the same class for both. You must use a FlyWeight design (Module 6.1.1.1) to implement the empty leaf nodes. Your Bintree must be implemented using the Composite design pattern (Module 6.1.1.3). You may not use a parent pointer in any Bintree node. You may not store the location/size information for a node in that node (such information should be passed as parameters in the appropriate method calls).

## Interfaces

The name of the program's main class is `AirControl`. Your program will implement an interface called `ATC`, with this interface implemented by a class named `WorldDB`. Graded reference tests for the project will work by making calls to the `ATC` interface. Note that this means none of the reference tests will read from or write to `System.out`, so reference tests will not conflict with whatever debug print statements you happen to implement. You may add other test files, but all tests that you add to the file `AirControlTest.java` should not call any internal methods of your project that were not specified here. One service that Web-CAT will provide for this project is to run any tests that you place in `AirControlTest.java` against the project reference implementation. If any of your tests fail against the reference implementation, that will indicate some mistake that you have made in your understanding of the project requirements.

The interface, example tests, and other starter code will be made available in the Starter Code available from Eclipse under **Project -> Download Assignment**.

The requirements for the `ATC` methods will be found in the example test file to be posted with the project starter kit.

## 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 each be under 600 lines. Test files may be longer if needed.
- 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.

We can't help you with your code unless we can understand it. Therefore, you should not 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 might, however, provide a useful starting point.

### Java Standard Data Structures Classes:

You are not permitted to use Java classes that implement complex data structures. This includes `ArrayList`, `HashMap`, `Vector`, or any other classes that implement lists, hash tables, or extensible arrays. (You may of course use the standard array operators.) You may use typical classes for string processing, byte array manipulation, parsing, etc.

If in doubt about which classes are permitted and which are not, you should ask. There will be penalties for using classes that are considered off limits.

### Deliverables:

You will implement your project using Eclipse (be sure that you have updated to a 2025 distribution) that has been set to use the Java 11 compiler (see the instructions in Module 2.4 of the OpenDSA course textbook) and you will submit your project using the Eclipse plugin to Web-CAT. Links to Web-CAT client are posted at the class website. If you make multiple submissions, only your last submission will be evaluated unless you arrange otherwise with the GTA. 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 mutation testing coverage. Of course, your program must pass your own test cases. Part of your grade will also be determined by reference test cases that are provided executed by Web-CAT. 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.

There are two additional deliverables with this project, in the form of two Canvas assignments. One of these is a "quiz" where you provide information about your use of an LLM (if any) for the project. Note that this **replaces** the requirement from previous assignments to provide a transcript file. It is considered an honor code violation to use an LLM for this project without disclosing it through the quiz. You must also submit a short video presentation about the project, to help the grader insure your conceptual understanding of the project implementation. Detailed instructions for this video requirement are linked at the Programming Assignments Resources page.

You are permitted to work with a partner on this project. While the partner need not be the same as who you worked with on any other projects this semester, you may only work with a single partner during the course of one project unless you get special permission from the course instructor. When you work with a partner, then **only one member of the pair** need make a Web-CAT submission. Be sure both names are included in the javadoc documentation and on the Web-CAT submission. Whatever is the final submission from either of the pair members is what

we will grade unless you arrange otherwise with the GTA. Note that if you work with a partner, each individual is required to submit their own video.

**Pledge and Honor Code:**

Your project submission must include a statement pledging your conformance to the Honor Code requirements for this course. Specifically, you must include the pledge statement that is provided with the project starter code. The text of the pledge will also be posted online. Programs that do not contain the pledge will not be graded. Use of an LLM without submitting the transcript as part of the associated Canvas assignment as required is considered an honor code violation.