CS 350 Project Specifications, Version 0.2

Description

This living document describes the team elements of the project:

- Part 1 is to build the parser for the command-line interface that allows a user to create, connect, and manipulate a subset of the aspects of the project.
- Part 2 is to do low-level unit testing on parts of the project.
- Part 3, which will done concurrently with Part 2, is to demonstrate through formal testing how well the complete, provided solution works.

This document will be revised with more details as the project progresses. The architecture will be covered in detail in lecture. Due to occasional refactoring, the code elements specified here may inadvertently be out of date. If you encounter any reference that appears inconsistent with the code, let me know immediately. Always make sure that you are using the most current version of this document and the code.

This document serves both as design instructions and usage instructions. As such, you must determine which role anything applies to. A lot of error checking is already handled by the architecture. When the instructions state that something must or must not happen, it may or may not be necessary for you as the programmer to enforce this. It is, however, incumbent upon the user (and tester) to know and follow these usage instructions. Eventually this document would be separated by role, but since you are playing all roles, the combined form is more appropriate for now.

Definitions

The commands reference the following grammatical fields, the type of each being a Java primitive or a class in `w16cs350.datatype`.

Field	Description	Datatype
`angle`	`number`	`Angle`
`coordinates_delta`	`number ':' number`	`CoordinatesDelta`
`coordinates_world`	`latitude '/' longitude`	`CoordinatesWorld`
`id`	*standard java variable name, underscore included*	`String`
`integer`	`[-|+]` *integer value*	`int`
`latitude`	`integer '*' integer ''' number '"'`	`Latitude`
`string`	*ordinary java string delimited by single quotes, no escape characters*	`String`
`longitude`	`integer '*' integer ''' number '"'`	`Longitude`
`number`	`( integer | real )`	`double`
`real`	`[-|+]` *real value, leading zero required*	`double`

Commands

Carets, parentheses, and square brackets are not part of commands. Vertical bar indicates logical *or*; asterisk indicates zero or more instances of the preceding term or parenthetical group; plus indicates one or more. Question mark or square brackets indicate an optional group.

White space (spaces and tabs), except in literals, does not matter. All text except identifiers is case insensitive.

All identifiers must be unique within the entire system.

Commands may appear on the same line if they are separated by a semicolon.

All failure modes not already handled must be accounted for appropriately. The messages and delivery mechanism need not be elaborate or particularly user-friendly.

Architecture

The architecture is provided in a JAR file. Configure your code to reference it. For example, in Eclipse, right-click on your project in the Package Explorer pane, select Build Path → Configure Build Path, and click Add External JARs.

For now build your parser such that it takes a string containing a command (or multiple, if delimited by a semicolon) and creates the appropriate command object. All `Command`*`X`* classes in the grammar are found below `w16cs350.controller.command`.

Implement the commands in blue in the spreadsheet. (Teams of two omit the ones with an asterisk.) Recognize the rule from its definition, extract its variable elements, use these to create the specified command object, and submit it to the action processor for execution.

Rule 1 is the entry point.

Call your parser `CommandParser` in package `w16cs350.controller.cli.parser` with constructor `CommandParser(A_ParserHelper parserHelper, String commandText)`. The command text is the string representation of any input as specified. The parser helper is a combination of the supplied `A_ParserHelper` and whatever you might need stored in a subclass of it. You are not required to use this class yourself, but it must be passed into your parser because it contains the `ActionProcessor` to which you submit your command through `schedule(A_Command command)`.