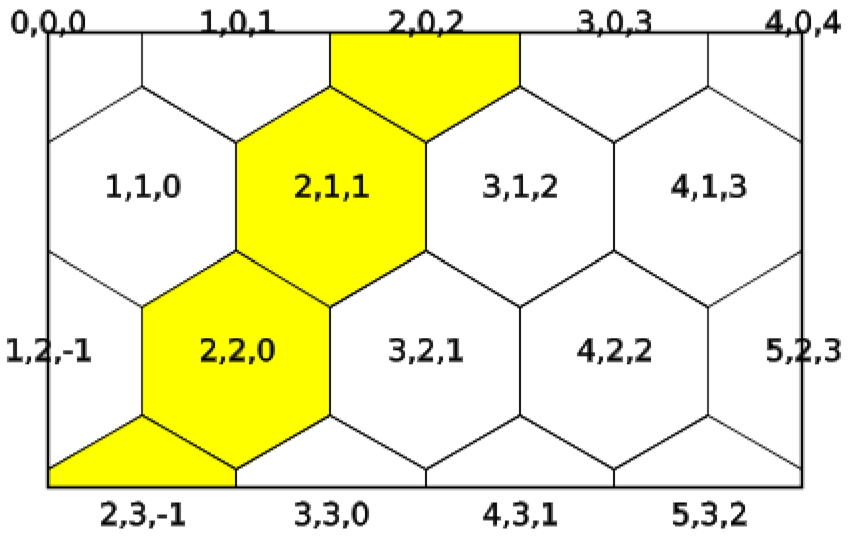
Homework # 1 (v 1.1)
due Monday, January 28, 10:00 PM

In this assignment, you will implement an ADT for a hexagonal coordinate system for creating games that operate on hexagon-tile boards. You will also implement a simple "enumerated class" for various kinds of terrains.

1 ADT: HexCoordinate

For this homework, you are creating an immutable data type to represent hexagon coordinates. Each coordinate represents the *center* of a hexagon in a tiling of the plane:



Each hexagon has three coordinates: the first increases as you go to the right (and down), the second as you go down, the third as you go right (and up). In the electronic version of this document, you can see that a band of hexagons with first coordinate equal to two have been highlighted in yellow. If you have a paper copy of this document, we recommend that you highlight in three contrasting colors bands of hexagons with the same coordinate equal to two. (There are a number of different hexagonal grids used in practice; I found several on the web. You may use code or ideas from the web, but you must give credit, including a URL, in your homework, and should be careful that the code will actually work.)

The third coordinate can always be computed from the other two[1] (how?) and so we often use just the first two numbers to create a coordinate. Two hexagonal coordinates are equal exactly when they have the same numbers in the same order.

The "distance" from one hex coordinate is the minimum number of moves to adjacent hexagons needed to get from one to another. Thus the distance from $\langle 3,0,3\rangle$ to $\langle 5,2,3\rangle$ is

[1]Unsurprisingly, given that this is a two dimensional grid.

two, and the distance from $\langle 2,2,0\rangle$ to $\langle 3,0,3\rangle$ is three. The distance from any hex coordinate to itself is zero. As it happens, the shortest path only requires travel along two of the three "dimensions", and thus you can compute the minimum distance by testing each possible pair of dimensions and choosing the shortest. For example, assuming we use "a", "b", and "c" for our three coordinates, then to compute the shortest path from $\langle 2,2,0\rangle$ to $\langle 3,0,3\rangle$, we can try all three pairs of dimensions to travel on:

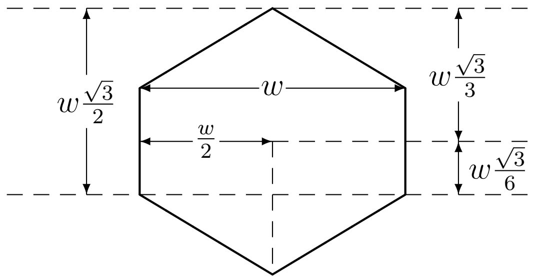
`ac` Travel first along the "a" dimension, keeping the "c" dimension constant: $\langle 2,2,0\rangle$ to $\langle 3,3,0\rangle$ and then along the "c" dimension while keeping the "a" dimension constant: $\langle 3,3,0\rangle$ to $\langle 3,2,1\rangle$ to $\langle 3,1,2\rangle$ to $\langle 3,0,3\rangle$. Total travel: $1+3=4$.

`bc` Travel first along the "b" dimension, keeping the "c" dimension constant: $\langle 2,2,0\rangle$ to $\langle 1,1,0\rangle$ to $\langle 0,0,0\rangle$, and then along the "c" dimension while keeping the "b" dimension constant: $\langle 0,0,0\rangle$ to $\langle 1,0,1\rangle$ to $\langle 2,0,2\rangle$ to $\langle 3,0,3\rangle$. Total travel: $2+3=5$.

`ab` Travel first along the "a" dimension, keeping the "b" dimension constant: $\langle 2,2,0\rangle$ to $\langle 3,2,1\rangle$ and then along the "b" dimension while keeping the "a" dimension constant: $\langle 3,2,1\rangle$ to $\langle 3,1,2\rangle$ to $\langle 3,0,3\rangle$. Total travel: $1+2=3$.

The shortest distance is the sum of the two smaller distances in each of the three dimensions (a, b, c). An easy way to compute the shortest distance is to add the three distances and then subtract the maximum distance: $1+2+3-3=3$; think about why this works. (Of course, you must use the *absolute value* of the difference: e.g.: $|2-3|=|-1|=1$).

In order to render a hexagon in a Java graphics context, we have to decide how big the hexagons are going to be. This might change and so we use a "width" parameter. For those of you who don't recall your hexagonal geometry, here's a helpful diagram (not quite to scale) of a hexagon with "width" $= w$:



Thus if $w=30$ and the hexagon is centered at $(0,0)$, then $(15,5\sqrt{3})$ is one of the corners.

Once you figure out how to create the `Polygon` for the hexagon at the hex coordinate $\langle 0,0,0\rangle$, you need to see how much the (x,y) points should shift for other coordinate. The first coordinate is easy; it shifts the hexagon right (increasing x) by w. The second is trickier since it shifts the hexagon down (increasing y) by $w\frac{\sqrt{3}}{2}$ and back to the left (decreasing x) by $\frac{w}{2}$.

In all, you must implement the following methods:

`HexCoordinate(int,int)` Main constructor.

`HexCoordinate(int,int,int)` Alternate constructor.

`equals(Object)` Return true if the argument is another hex coordinate with the same coordinate values.

`hashCode()` Return a combination of the coordinate values that does not simply add up two or more.

`toString()` Return a string of the form `<3,2,1>`.

`fromPoint(Point,int)` Return the coordinate of the hexagon that encloses the given (x, y) point using the given width. (We implement this method for you.)

The following methods do not have stubs in the skeleton; you will need to write them from scratch:

`toPoint(int)` Return the (x, y) center of the hexagon at this coordinate using the given width.

`toPolygon(int)` Return a AWT Polygon object for the hexagon at this coordinate with specified width; the first point should be at the top of the hexagon, and the rest going clockwise from there. There should be six points (it is a hexagon).

`distance(HexCoordinate)` Return the numbers of moves required to get from this hex coordinate to the parameter.

Note: AWT Point and Polygon objects use integers, not floats. The supplied test files require that calculated values be clamped to the *nearest* integer so that the value `3.4` becomes `3`, and the value `-2.6` becomes `-3`. If many of your test cases initially fail with many values off by one, this may be the reason.

2 Enum: Terrain

Java has "enumeated types." Please consult the Oracle documentation for information on how to declare an enumerated type.

Implement a terrain enumerated type where each terrain is associated with a color:

`INACCESSIBLE` Black

`WATER` Cyan

`LAND` White

`FOREST` Green

`MOUNTAIN` Light gray

`CITY` Orange

`DESERT` Yellow

Write a `getColor` accessor to get the associated color for a terrain.

3 Required Coding Conventions

In the code that you write for CS 351, you must follow some guidelines:

1. Classes and public methods must have appropriate "Java-doc" comments.
2. Fields should be private.
3. Methods should start with lowercase characters and use "camelCase" to handle multiple words.
4. Methods that override those in a super class (notably `toString()` and `clone()`) should be annotated `@Override`.
5. Code should be appropriately indented with tabs and/or spaces.

4 Test Programs

We provide two JUnit test suites `TestHexCoordinate.java` and `TestTerrain` as well as a sample driver program `Main.java`. **Do not modify them.** Ensure that your project passes all tests and that the driver program displays something reasonable before submission.

5 Files

We use "git" to access and turn in programs. Follow the instructions given in the lab to clone your homework repository onto your own or lab computer. Make sure you always push changes back (commit is insufficient) before switching computers and before the deadline.

Your task is to write the following files:

```
src/edu/uwm/cs351/HexCoordinate.java

src/edu/uwm/cs351/Terrain.java
```

These files must exist in your homework1.git repository before the deadline (10:00 PM, Monday). Do not forget to write documentation for all classes, constructors and non-standard public methods.