Jacob's Module Tests

Point Class:

- **Overview**
  - A Point is the basic unit of a Map object, it has an X position, a Y position, elevation, and a biome.
- **Methods:**
  - **getX ()**
    - **Purpose:** Syntactic sugar to make writing code more clear, returns the X position of the point.
    - **Equivalence Classes:**
      - X has value
      - X is undefined
    - **Test Casses:**
      - **Test 1:** X is defined
        - Expected output, value of X
      - **Test 2:** X is undefined
        - Expected output, undefined
  - **getY ()**
    - **Purpose:** Syntactic sugar to make writing code more clear, returns the Y position of the point.
    - **Equivalence Classes:**
      - Y has value
      - Y is undefined
    - **Test Casses:**
      - **Test 1:** Y is defined
        - Expected output, value of Y
      - **Test 2:** Y is undefined
        - Expected output, undefined
  - **getElevation ()**
    - **Purpose:** Syntactic sugar to make writing code more clear, returns the Elevation position of the point.
    - **Equivalence Classes:**
      - Elevation has value
      - Elevation is undefined
    - **Test Casses:**
      - **Test 1:** Elevation is defined
        - Expected output, value of Elevation
      - **Test 2:** Elevation is undefined
        - Expected output, undefined
  - **getBiome ()**
    - **Purpose:** Syntactic sugar to make writing code more clear, returns the Biome position of the point.
    - **Equivalence Classes:**

            - Biome has value
            - Biome is undefined
        - **Test Casses:**
            - **Test 1:** Biome is defined
                - Expected output, value of Biome
            - **Test 2:** Biome is undefined
                - Expected output, undefined
- **setElevation(e)**
    - **Purpose:** Syntactic sugar to set the value of Elevation. Does not have a return value, expected result, the elevation of this point is altered
    - **Equivalence Class:**
        - Input e has any value
    - **Test Case:**
        - Run with input
            - Expected result: point.elevation = input
- **setBiome(b)**
    - **Purpose:** Syntactic sugar to set the value of Biome. Does not have a return value, expected result, the elevation of this point is altered
    - **Equivalence Class:**
        - Input b has any value
    - **Test Case:**
        - Run with input
            - Expected result: point.elevation = input
- **dist (p)**
    - **Purpose:** Calculate the euclidean distance between this point and the input point.
    - **Equivalence Classes:**
        - P is a well defined point
        - P is not a point
        - The x point or y point of either point is not defined
    - **Test Cases:**
        - **Test 1:** P is a point
            - Expected result: Returns the euclidean distance between this point and p
        - **Test 2:** P is not a point or x or y of either is undefined
            - Expected result: Error
- **dir (p)**
    - **Purpose:** Calculate the direction of p in relation to this point
    - **Equivalence Classes:**
        - P is a well defined point
        - P is not a point
        - The x point or y point of either point is not defined
    - **Test Cases:**

- **Test 1:** P is a point
  - Expected result: Returns a string indicating the direction 'north', 'northwest', 'west', etc.
- **Test 2:** P is not a point or x or y of either is undefined
  - Expected result: Error

Map Class:

- **Overview**
  - The data structure used to hold the map. It contains a width, height, and an array of points to represent pixels in the final resulting image.
- **Methods:**
  - **Constructor(width, height)**
    - **Purpose:** Create the point array
    - **Equivalence classes:**
      - Width or Height are less than 1
      - Width and Height are greater than or equal to 1
    - **Test Cases:**
      - **Test 1:** Width or Height are less than 1
        - Expected result: Error
      - **Test 2:** Width and Height are greater than or equal 1
        - Expected result: Forms an array of points of size Width * Height
  - **point (x, y)**
    - **Purpose:** Retrieve the point on the map at position x, y
    - **Equivalence classes:**
      - X and Y are not numbers
      - There exists a point at X, Y
      - There does not exist a point at X, Y
    - **Test Cases:**
      - **Test 1:** X or Y is not a number
        - Expected result: Error
      - **Test 2:** There exists a point at X, Y
        - Expected result: Return a reference to the point object at X, Y
      - **Test 3:** There does not exist a point at X, Y
        - Expected result: return null
  - **getNeighbors (point, onlyOrthogonal = false)**
    - **Purpose:** retrieve all points neighboring the point passed in.
    - **Equivalence Classes:**
      - Point is in the middle of the map
      - Point is on the edge of the map
      - Only orthogonal is true
      - Point does not exist

        - **Test Cases:**
            - **Test 1:** Point is in the middle of the map
                - Expected result: return list of eight neighboring points
            - **Test 2:** Point is on the edge of the map
                - Expected result: return a list of the points that exist on the map
            - **Test 3:** onlyOthogonal is true
                - Expected result: Only return points to the north south east and west that exist
            - **Test 4:** Point does not exist
                - Expected result: Error
    - **getNeighbor (point, dir)**
        - **Purpose:** retrieve neighboring point in a specified direction
        - **Equivalence Classes:**
            - Point exists and dir is a valid string
            - Point does not exist
            - Dir is not a valid string
        - **Test Cases:**
            - **Test 1:** Point exists and dir is valid
                - Expected result: return value of neighbor or null if that neighbor does not exist
            - **Test 2:** Point does not exist
                - Expected result: Error
            - **Test 3:** Dir is not a valid string
                - Expected result: return null
    - **getRandomNeighbor(point, onlyOrthogonal = false)**
        - **Purpose:** Pick a random neighbor of point
        - **Equivalence Classes:**
            - Point Exists
            - Point does not exist
            - onlyOrthogonal = true
        - **Test cases**
            - **Test 1:** Point exists
                - Expected result: Returns a random member of the list produced by getNeighbors()
            - **Test 2:** Point does not exist
                - Expected result: Error
            - **Test 3:** onlyOrhtogonal = true
                - Expected result: Returns a random member of the list produced by getNeighbors() where only orthogonal was set to true
    - **getRandomNeighborOfType(point, biome, onlyOrthogonal = false)**

        - **Purpose:** return a random neighbor with the same biome as the string passed in
        - **Equivalence Classes:**
            - Point exists
            - There are neighbors of type biome
            - There are not neighbors of type biome
            - Point does not exist
            - onlyOrthogonal = true
        - **Test Cases**
            - **Test 1:** point exists and there are neighbors of type biome
                - Expected result: return random appropriate neighbor
            - **Test 2:** point exists and there are no neighbors of type biome
                - Expected result: return null
            - **Test 3:** point does not exist
                - Expected result: error
            - **Test 4:** onlyOrthogonal is true
                - Same as above but will exclude diagonal neighbors
    - **hasNeighbors (point, onlyOrthogonal = false)**
        - **Purpose:** returns true if neighbors exist
        - **Equivalence classes**
            - Point exists
            - Point does not exist
            - Only orthogonal is true
        - **Test Cases**
            - **Test 1:** Point exists
                - Expected result: returns true if neighbors exist
            - **Test 2:** Point does not exist
                - Expected result: Error
            - **Test 3:** Only orthogonal is true
                - Expected result: return true if orthogonal neighbors exist
    - **hasNeighborsOfType (point, biome, onlyOrthogonal = false)**
        - **Purpose:** returns true if neighbors of type biome exist
        - **Equivalence classes**
            - Point exists
            - Point does not exist
            - Only orthogonal is true
        - **Test Cases**
            - **Test 1:** Point exists
                - Expected result: returns true if neighbors of type biome exist
            - **Test 2:** Point does not exist
                - Expected result: Error
            - **Test 3:** Only orthogonal is true

        - Expected result: return true if orthogonal neighbors of type biome exist
- **getNeighborsOfType (point, biome, onlyOrthogonal = false)**
    - **Purpose:** retrieve all points of type biome neighboring the point passed in.
    - **Equivalence Classes:**
        - Point exists
        - Only orthogonal is true
        - Point does not exist
    - **Test Cases:**
        - **Test 1:** Point is in the middle of the map
            - Expected result: return list of neighbors of type biome
        - **Test 2:** onlyOthogonal is true
            - Expected result: Only return points to the north south east and west that exist of type biome
        - **Test 3:** Point does not exist
            - Expected result: Error
- **getPointsOfType (biome)**
    - **Purpose:** get all points on the map of type biome
    - **Equivalence classes**
        - There exist points of type biome
        - There do not exist points of type biome
    - **Test Cases:**
        - **Test 1:** There exist points of type biome
            - Expected result: return list of all points of the type biome
        - **Test 2:** There exist no points of type biome
            - Expected result: return the empty list
- **getRandomPointOfTypeBiome(biome)**
    - **Purpose:** get a random point of the appointed type
    - **Equivalence classes**
        - There are points of type biome
        - There are no points of type biome
    - **Test cases**
        - **Test 1:** There are points of type biome
            - Expected result: return a point of the specified time
        - **Test 2:** There are no points of type biome
            - Expected result: return null
- **randomDirection ()**
    - **Purpose:** return a random valid direction
    - **Equivalence class**
        - Function is called
    - **Test cases**
        - **Test 1:** call function

        - Expected result: Return proper direction string

River Agent Class:

- **Overview**
    - A RiverAgent takes in a number of rivers to create. It then operates on a map drawing a river from a shore point, to a mountain point. It takes maxRivers as a constructor parameter
- **Methods**
    - **generate (map)**
        - **Purpose:** called when the map is made to attempt to create maxRivers rivers
        - **Equivalence Classes:**
            - Map exists
            - Map does not exist
        - **Test Cases**
            - **Test 1:** Map exists
                - Expected result: rivers created on map. List of altered points returned
            - **Test 2:** Map does not exist
                - Expected result: Error
    - **generateRiver (map)**
        - **Purpose:** Attempt to generate a single river.
        - **Equivalence Classes:**
            - Map exists
            - Map does not exist
            - Mountain Points exist
            - Shore points exist
        - **Test Cases:**
            - **Test 1:** Map, mountain, and shore points exist
                - Expected result: A river will be created on the map, the points altered will be returned
            - **Test 2:** Map does not exist
                - Expected result: Error
            - **Test 3:** Mountain or shore points do not exist
                - Expected result: River will not be drawn, the function will return an empty list without error