1.0 Class Design

1.1 Grid

1.1.1 Data Structures

- Using two 2D arrays (one double and one char) to store the potential values and the location of the objects in the grid
  - This is not memory efficient, but it makes the code more readable and concise
  - This is intuitively how I thought of the problem, by representing the map directly as a 2D array where the positions in the array directly correlate to the x and y positions on the map.
  - I also considered using a 1D array, but this would require extra logic since the API want to use X and Y locations to modify and read the map.
  - Could probably have figured out a way to encode the potential and object information into a single array but again this adds complexity so save memory where memory is not a major concern.

1.1.2 Public Methods

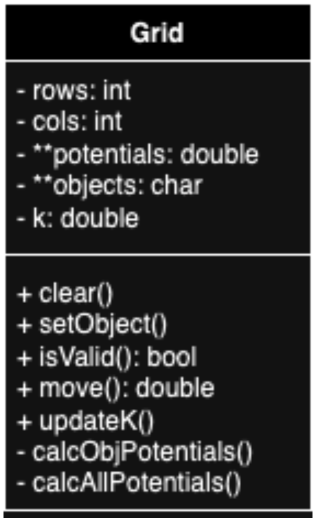
- There is a public method isValid() which checks if the x and y coordinates are within the range of the map
  - Could have made this method private and called it inside the other public methods but since main is supposed to fail on incorrect inputs, this would not always be practical.
  - For example, the move returns a double and if move was doing the validity check, main would not be able to determine the difference between a valid potential and a return error.
- setObject() and updateK() are responsible for triggering potential calculations
  - When setObject() is called, a new object is added to the map. The potential from this new object at each location is calculated and then added to the current potential at each location.
  - Calling update recalculates all the potentials from scratch based on the current object in the map
  - This means that when move is called, it is simply looking up the potential values.
  - I could have chosen to compute potentials when move is called but the project description requested that POINT and UPDATE recompute the potentials.

1.1.3 Private Methods

- calcObjPotential() calculates the potential impact of a single object on every other location in the map

    - This is called by setObject() to update the potentials when new object is added
- When updateK() is called and all the potentials need to be recalculated, calcAllPotentials() clears the map and calls calcObjPotential() for every object in the object array.

2.0 UML Diagram



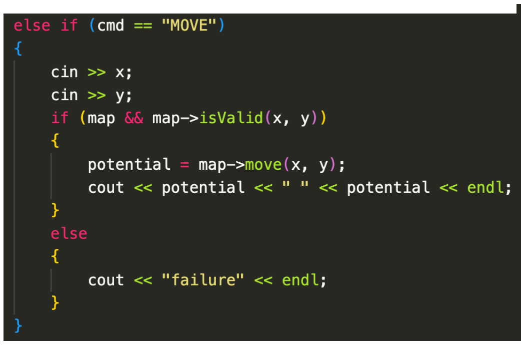
Grid

- rows: int
- cols: int
- **potentials: double
- **objects: char
- k: double

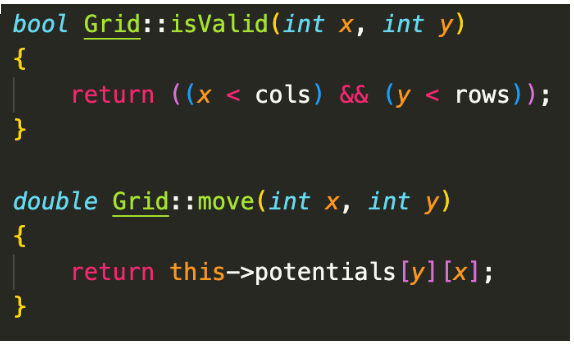
+ clear()
+ setObject()
+ isValid(): bool
+ move(): double
+ updateK()
- calcObjPotentials()
- calcAllPotentials()

3.0 Runtime

3.1 MOVE



```
else if (cmd == "MOVE")
{
    cin >> x;
    cin >> y;
    if (map && map->isValid(x, y))
    {
        potential = map->move(x, y);
        cout << potential << " " << potential << endl;
    }
    else
    {
        cout << "failure" << endl;
    }
}
```



```
bool Grid::isValid(int x, int y)
{
    return ((x < cols) && (y < rows));
}

double Grid::move(int x, int y)
{
    return this->potentials[y][x];
}
```

- cin input operations take constant time, O(1)
- isValid() is doing value comparisons which take constant time, O(1)
- move() is accessing an element in an array which for any element takes constant time, O(1)
- Since the potential variable is always a double, the cout operation takes constant time, O(1)
- Thus the overall worst case time complexity for the MOVE command is O(1)