Trek Wars

Purpose: classes, inheritance, class diagrams, virtual

Due: Nov 22nd

Description

Universal Merriment Development (UMD) is creating an online space battle game called (Trek Wars). You are tasked with creating some classes for this game. Again you have to just write the class and test it. The autograder will supply a main, and a sample main is included. A description of the classes is given below:

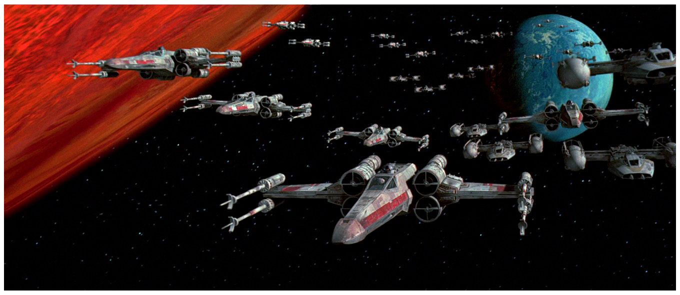

Figure 1: Graphics from the game.

```
enum Alignment{us, them, chaotic};
```
Ship (the base class)

attributes
```
   name : string
   align : Alignment
   xLoc : integer
   yLoc : integer
   range: int
   currHealth : integer
   attackPower: integer
   maxHealth: integer
methods
   public Ship(name:string, x : integer , y : integer , align : Alignment ,
            maxHeal : integer, rng :integer, attackPwr : integer );
                      // currHealth is set to maximum
   public virtual attack(target: * Ship) : void //
   private virtual getType() : string//''Battleship'', ''Cruiser'', ''Corvette'', ''Repairship''
   public getX():int // returns the x coordinate
   public getY():int // returns the y coordinate
   public getAlign():Alignment // returns the alignment
   public status () : string // see format below
   public virtual move() : void // changes position by the amount of that type of ship,
                      // increases health by 1 (until max reached)
   public changeAlign() :void // changes the alignment.
   public accessDamage(amt : int) : void // changes the health by amt,
   // (keeping it within bounds [0,maxHealth])
```

Battle(derived from the Ship class)

```
attributes //range is 10, maxHealth =100 ,attack = 10
   torpedoes // int initially 10
methods
// always moves along the vector (-1, -1)
move()
attack(target: Ship *) : void//attacks and fires torpedo >0 and
// does additional 10 damage, 1 less torpedo
status () // also indicates number of torpedoes
```

Cruiser (derived from Ship class) range is 50, maxHealth =50

```
method
// always moves along the vector (1, 2)
move()
attack(target: Ship *) : void//attacks 5
```

Corvette (derived from Ship class) range is 25, maxHealth =20

```
method
// always moves along the vector (5, 5)
move()
attack(target: Ship *) : void// Everyone loves corvettes so their attack
// flips ships in range to its state
// (if self is us, turns them to us,
// if self is them, turns us to them)
```

Repair (derived from Cruiser class) range is 25, maxHealth = 20

```
method
   attack(target: Ship *) : void //its attack repairs a ship of own kind to max health
```

Input

No input

Output

```
For status() print each
name : <name>
type : <Class name>
health:<healthStr>
location: (xLoc, yLoc)
torpedoes: // only if this is a battleship
```

Except for repair ships, attacks only work if the target of the attack is of the opposite alignment and in range. . By opposite alignment, we mean (us attacks them and them attacks us). Chaotic ships attack everyone. Repair ships only attack (in this case repair), ships of the same alignment.

How the program will be graded

You will turn in 2 documents. A memo to canvas (a .doc file) and the source code to mimir (this a zipped version of your source code which must be named trekWars.hpp .

Memo

What	pts
Name	1
Class diagram	10
Test plan with 3 unit tests for each of Battleships, Cruisers, Corvettes, and Repairships	16

Source Code Document (zipped version of trekWars.hpp)

What	**pts**
Name	1
Description (in your own words)	2
Style	10
pre/post conditions	10
Functionality	50

Sample main

```
int main() {
Ship** fleet;
int numShip; //cout<<"How many ships do you want ?";
    //cin>> numShip;
numShip = 6;
fleet= new Ship *[numShip];
fleet[0] = new Battle("Constitution", 0,  0, us);
fleet[1] = new Cruiser("Enterprize", 20, 20, them);
fleet[2] = new Corvette("1000Falcon", 0 5, chaotic);
fleet[3] = new  Repair("DocMcStuffings")
fleet[4] = new  Battle("BoatyMcBoatFace"
fleet[5] = new
for (int i = 0; i < numShip; i++) cout << fleet[i]->status();
for (int i = 0; i < numShip; i++) fleet[i]->move();
for (int i = 0; i < numShip; i++) cout << fleet[i]->status();
for (int i = 0; i < numShip; i++)
  for (int j = 0; j< numShip; j++)
fleet[i]->attack(fleet[j]);
for (int i = 0; i < numShip; i++)
    cout << fleet[i]->status();
return 0;
}
```