COSC 601 Assignment 2: Due Dec 12th

The Game of Dragons

You have been asked to come up with a prototype for a new Card Game, the game of dragons. This assignment will require you to implement the following classes in a package **`game`** and then use them to determine how the game might play out using a simulation.

You will be first required to implement the classes described in the accompanying UML diagram on the last page. A brief description of what will be required to implement the classes is provided as follows:

Dragon Class

Attributes

nAttackRank

This is a measure of the dragon's strength in Attack. The Attack Rating can vary from 36 to 70 depending upon the size of the dragon.

nDefenseRank

This is a measure of the Dragon's strength in defense. The defense ranking can vary from 10 to 35 depending upon the size of the Dragon.

nHitPoints

This is a measure of the Dragon's constitution; a dragon is alive in battle so long as the Dragon has hit points remaining. Hit points can range from 51 to 100.

nInitiative

This is used as part of the determination of which dragon goes first in a fight turn. This value can range from 0 to 60.

nSIze

This is used to record the size of the Dragon. An enumeration called DragonSize is used to indicate the size.

Methods:

Dragon(DragonSize nSize)

The constructor takes in the size of the Dragon You are creating. This value comes from the DragonSize enumeration. Dragons can be of Size Large, Medium, and Small. The attributes listed above are set according to the specified size and indicated in the following table. The Constructor must set the attribute in the given range using Math.Random to determine the exact value.

Size/Attribute	Small	Medium	Large
Hit Points	51-70	71-85	86-100
Initiative	40-60	20-40	0 – 10
Attack Ranking	36-50	51-60	61-70
Defense Ranking	10-19	20-29	29-35

getInitiative()

This is just returns the initiative value for the Dragon

getNumAttacksPerTurn()

This returns the number of times a dragon cat attack in a given turn. The formula is quite simple, small dragons can attack 3 times, medium dragons can attack twice and large dragons can only attack once.

getAttack()

The just is a getter for the returning the attack rating associated with the dragon

getDefense()

This is just a getter for returning the defense rating associated with the dragon.

isDead()

If the dragon hit points fall to 0 or lower than the Dragon is deal. This method would return true in that case.

resurrect()

This will resurrect the dragon by setting the hit points back to the original value.

toString()

Implement an appropriate toString method that will show the appropriate information for the Dragon.

defendAttack(Dragon obOther)

This is an abstract method (must be implemented in the sub-classes). This method is used to represent another dragon attacking this dragon. Essentially you will take the attack rating for the other dragon and subtract the defense ranking for the current dragon. The difference will be the amount of hit points of damage the other dragon causes during the attack. You will subtract this value from the current hit points for the dragon.

Note that different Dragon types have different modifiers applied to them depending upon the dragon that is attacking (see the descriptions for the sub-class dragon types).

You must include as part of your calculation a modifier based upon Dragon size. If the current Dragon is a small Dragon, attacks will miss 30% of the time. If the current dragon is a medium sized Dragon, attacks will miss 20% of the time. If the current Dragon is a large Dragon, an attack will always hit.

Subclasses

Each of the subclasses is similar in that it represents a different colored dragon. Each will have a constructor and store a name for a particular dragon ("SmallGreen1"). The main difference between them will be how they implement the **`defendAttack`** method.

Red

Red dragons are flame based dragons that are very effective against white (ice) dragons, but not very effective against Green dragons (poison).

Green

Green dragons are very effective against Red dragons but are weak against white Dragons.

White

White dragons are effective against Green dragons but are at a disadvantage when fighting Red Dragons.

Black

Black dragons are magical dragons that other dragons attack with their normal (100%) attack rating). Black dragons attack other dragons with 100% of their attack capability but are curiously weak against white dragons, who they can only attack with 75% of their attack ranking.

Color Modifiers

The following table illustrates how you modify the attack ranking of various dragons based upon the color of the dragon. Suppose you have a Green Dragon attacking a Red Dragon. The green will attack with 120% of its normal attack rating. As another example consider you have a Black dragon attacking a white dragon; it will only attack with 75% of its attack rating.

Defend/Attack	Red	Green	White	Black
Red	100%	80%	120%	100%
Green	120%	100%	90%	100%
White	80%	110%	100%	100%
Black	100%	100%	75%	100%

Game Testing

GameTest Class

To test the above Classes you will be required to create a **`GameTest`** class. This class will do the following:

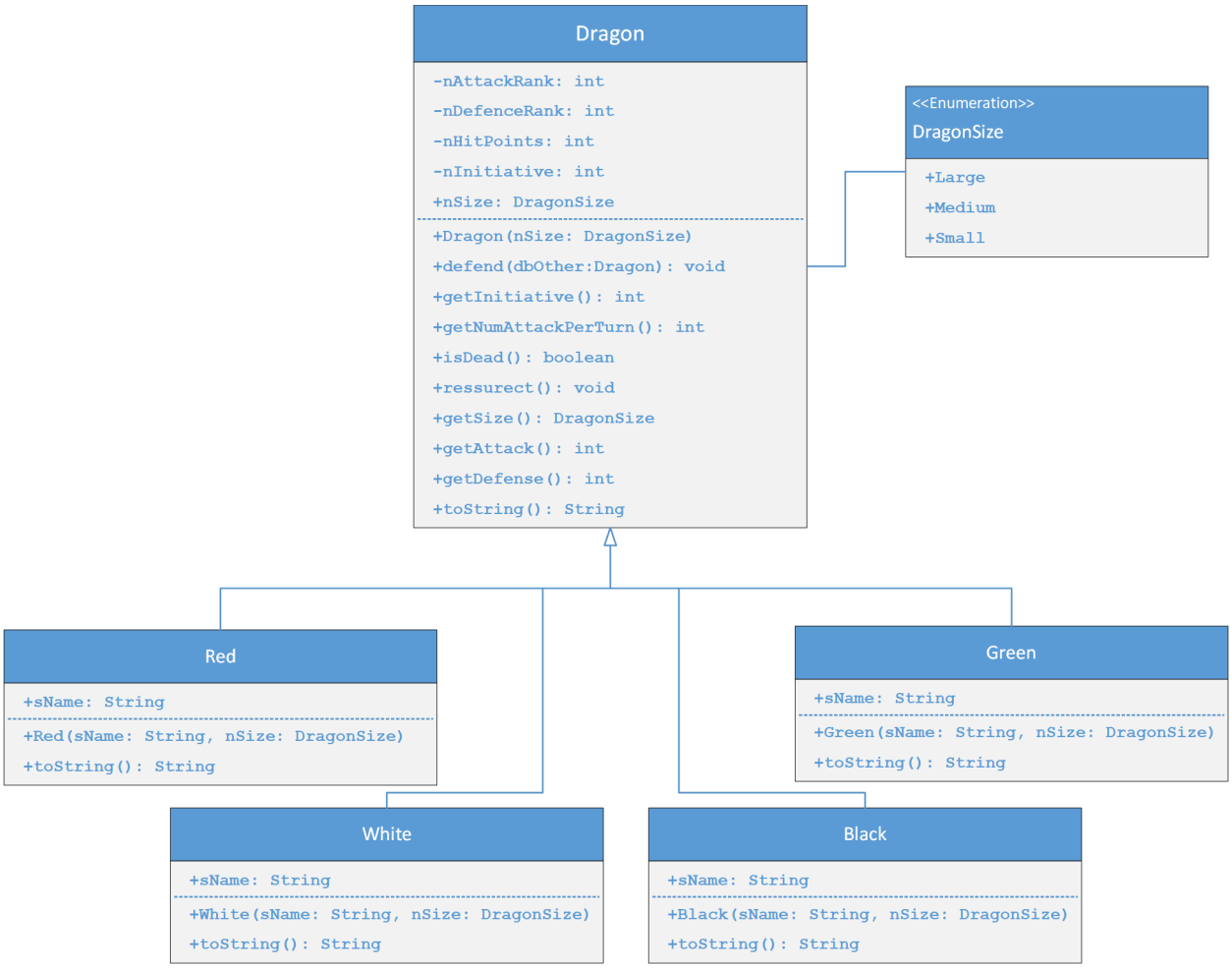
- Create an Array or an ArrayList to hold 36 Dragons
- Generate 3 sample Dragons for each Dragon Color and size and place them in the array. For example you will generate 3 Small Green Dragons, 3 Medium Green Dragons, etc.
- Have the Dragons battle each (every dragon must face every other dragon once) by calling the method **`battleRound`** repeatedly with different dragons. You must track how well each of the dragons does.
- After each battle round reset the called dragon's hit points by calling their resurrect method.
- After all the fights are over, Rank the dragons in terms of how many fights they have won. Sort the array with the dragons that have won the most coming first.
- Print this array out.

```
public static Dragon battleRound(Dragon obDragon1, Dragon obDragon2)
```

Create a method **`battleRound`** which takes 2 Dragon References as arguments. This method will return a Reference to the winning dragon based on the following Game Play

The method will simulate the Dragons fighting against each other using a Turns based approach. Essentially Dragons fight over several turns until one of the Dragons is food for crows (is killed). Each of the turns will follow this pattern:

1. The Dragon with the higher initiative will go first (if same initiative select one randomly).
2. The First Dragon will attack the other by calling the **`defendAttack`** method for the other dragon. This attack method is described else-where but will have the effect of reducing the attacked Dragon's hit points by a certain amount. If the attacked dragon is killed then the game is over and a Reference to the winning dragon returned.
3. The Second dragon will then attack the other using the same procedure as just described.
4. This will repeat for each dragon as long as they can still attack that turn. Small dragons can attack up to 3 times per turn while large dragons can attack only once. So if we have a small green dragon (3 Attacks Per turn, High initiative) fighting a large Red dragon (1 attack per turn, low initiative) the turn cycle would be (Green Attacks, Red Attacks, Green Attacks, Green Attacks).
5. The turn cycle as described above repeats until one of the dragons is killed.


Dragon
-nAttackRank: int
-nDefenceRank: int
-nHitPoints: int
-nInitiative: int
+nSize: DragonSize
+Dragon(nSize: DragonSize)
+defend(dbOther:Dragon): void
+getInitiative(): int
+getNumAttackPerTurn(): int
+isDead(): boolean
+ressurect(): void
+getSize(): DragonSize
+getAttack(): int
+getDefense(): int
+toString(): String
<<Enumeration>>
DragonSize
+Large
+Medium
+Small
Red
+sName: String
+Red(sName: String, nSize: DragonSize)
+toString(): String
Green
+sName: String
+Green(sName: String, nSize: DragonSize)
+toString(): String
White
+sName: String
+White(sName: String, nSize: DragonSize)
+toString(): String
Black
+sName: String
+Black(sName: String, nSize: DragonSize)
+toString(): String