CSC 260.002 Programming Assignment #8
Due date: Tuesday, November 15

In this assignment, you will write a Horse class and a HorseRace class. The former class will not have a main method so that you will have to create objects of type Horse in your HorseRace class. The Horse class represent a race horse and will have instance data to represent the Horse's name, how quickly it can start a race, its stamina during the race, how sturdy it is, and how quickly it runs the race, as well as a Random number generator, and values for where it is in the race and if the Horse is getting nervous because of other Horses around it. Use the following UML notation to construct your Horse class with the instance data and methods listed.

Horse:	default values
name: String	"unknown"
start: int	5
stamina: int	-2
sturdiness: int	15
minSpeed: int	8
maxSpeed: int	10
nervousness: int	5
location: int	0
g: Random	null

```
+Horse()
+Horse(g: Random)
+Horse(g: Random, name: String)
+Horse(g: Random, name: String, minSpeed: int,
     maxSpeed: int)
+Horse(g: Random, name: String, start: int,
     stamina: int, sturdiness: int, minSpeed: int,
     maxSpeed: int, nervousness: int)
+getName(): String
+getLocation(): int
+move(phase: int): void
-getsNervous(horse1: Horse, horse2: Horse): void
+toString(): String
```

The default values are for the constructors that do not receive those particular pieces of information. For instance, for the first three constructors, assign start to 5 and stamina to -2, and for the 0-arg and 1-arg constructors, use all of the default values. For all constructors, set location to 0 and nervousness to 5. The 0-arg constructor is the only one that does not receive a Random object and so should set its variable to null. Normally this variable should get a better name like generator but in this program, you can just use g.

The move method changes the Horse's location based on the int values (start, stamina, etc), and the phase of the race which is a parameter. If phase is 1-4, the Horse moves randomly between `start-1` and `start+1`. For instance, if start is 7, it will move between 6 and 8. In phases 5-15, the Horse moves between minSpeed and maxSpeed. For instance, if minSpeed is 10 and

maxSpeed is 15, the Horse would move a random amount between 10 and 15. You will have to work out the proper assignment statement to compute this given the variables minSpeed and maxSpeed. From phase 16 onward, the Horse moves between minSpeed and maxSpeed except that there is a 50% chance that the movement will be reduced by the stamina amount. This is the impact of the Horse slowing down because of a lack of stamina. Use the same Random object (g) to generate the 50% random chance (this can simply be a random number of 0 or 1). For every move, no matter what phase it is, generate a random number from 1-20 and if that value > sturdiness, the Horse only moves between 1-4 as the Horse "stumbles" during that turn. Finally, if the Horse does not stumble but the Horse's nervousness reaches 0, then it has become nervous because of the other nearby Horses (see the next method for how this is determined). In this case, subtract 3 from the amount it is moving and reset nervousness to 5. The move method should not only change the value of location based on the amount moved but also output the Horse's name, how much it moved, where it has moved to, and if it stumbled or is nervous. See the sample output at the end of the assignment.

The getsNervous method compares this Horse's location to the other two Horses' locations once the race has started (if location <= 5, this method does nothing). To compare this Horse's location to the other Horses' locations, you need to pass to this method the other two Horses as parameters. If those parameters are h1 and h2, then you compare `location` against `h1.getLocation()` and `h2.getLocation()`. Below is the logic for determining nervousness and its impact.

- If this Horse is at the same location or within 1 of both Horse h1 and Horse h2, subtract 4 from this Horse's nervousness value (values can become negative)
- If this Horse is within 3 of Horse h1, subtract 1 from its nervousness
- If this Horse is within 3 of Horse h2, subtract 1 from its nervousness

To determine "within 1" or "within 3", use Math.abs as in

```
Math.abs(location - h1.getLocation())<=...
```

The toString will return a String of the Horse's name and current location. Implement accessors as listed in the UML notation above. You do not need accessors for all instance data. The only mutator is move. Also notice that since getsNervous is only called from within the move method, getsNervous should be a private method, not public. All other methods should be public.

Implement the HorseRace class. This class has only a main method. It will create three Horse objects (shown below), all of which will share the same Random object (only create one Random variable in this class and pass it to all 3 Horse constructors). Since Random is used in both Horse and HorseRace, you need to import it into both files. This class will also need an int variable called phase, initialized to 0 (phase is the "turn" or the number of iterations to this point in the race).

After creating all three Horses, main uses a while loop that iterates while none of the three Horses has reached the finished line (250). You might implement it using code like the following:

```
while(h1.getLocation()<250&&h2.getLocation()<250&&
          h3.getLocation()<250)
```

In the while loop, increment phase and call each Horse's move method as in `h1.move(phase);` (remember that move outputs info about the Horse's move, so there is no other output required in this loop). After exiting the while loop, output the results of the race (the location of each Horse and who won). Assume there will be no tie to simplify the logic of determining who wins (there

may be a tie, but if so, just award one of the Horse's as the winner). If desired, you can add a Scanner to your code to input from the user after each turn so that the output pauses. This is not necessary, simply an option if you want to watch the game run turn-by-turn. Your Horses should be set up as follows:

name	start	stamina	sturdiness	minSpeed	maxSpeed
Speedy	7	-6	14	11	14
Steady	5	-1	17	9	11
Sturdy	4	-4	19	8	12

Here is a sample output from running the HorseRace program.

```
speedy moves to 6
steady moves to 6
sturdy moves to 3
speedy moves to 14
steady moves to 11
sturdy moves to 8
speedy moves to 20
steady moves to 15
sturdy moves to 11
speedy moves to 21 while stumbling
steady moves to 19
sturdy moves to 14
speedy moves to 29
steady moves to 21 while stumbling
sturdy moves to 19
speedy moves to 42
steady is nervous and falters, steady moves to 27
sturdy moves to 29
speedy moves to 54
steady moves to 31 while stumbling
sturdy moves to 40
…
speedy moves to 220
steady moves to 199 while stumbling
sturdy moves to 237
speedy moves to 232
steady moves to 201 while stumbling
sturdy moves to 246
speedy moves to 242
steady moves to 211
sturdy moves to 256

The race results are:
    speedy is at 242
    steady is at 211
    sturdy is at 256
sturdy wins!
```

Submit your two source code files (Horse and HorseRace) and one output from running HorseRace that includes some stumbles and nervousness and the race results at the end.