Isaac Heisdorffer - CSC - 325- FA25

I have three different threads within the program:

```
// Create character instances
        GameCharacter knight = new Knight();
        GameCharacter wizard = new Wizard();
        GameCharacter thief = new Thief();
```

These three are characters that are playable by the user. Each one runs the “run()” method within the game character class. They also run the “continuingAdventure()” method while using shared resources during the dragon fight.

```
// In main class or GameCharacter:
    protected static volatile int dragonHealth = 300;
    protected static final Object dragonLock = new Object();
    protected static AtomicInteger treasureCount = new AtomicInteger(3);
    protected static List<String> battleResults = new ArrayList<>();
    protected static int[] characterDamage = new int[3];
```

These variables are the shared resources that are used by the threads to trade for treasure with the villagers and fight the dragon.

One of the race conditions that I had experienced was that when characters would fight the dragon, the dragon's health would not decrease for the dragon when each player would attack it. The threads would also try to fight the dragon at the same time.

The solution was to put a synchronization block around the “dragonLock” object so that the threads would not try to access the object at the same time in order to have proper decrements of the health of the dragon.

Another issue was when the characters were trying to trade for treasure, it would not decrease when each player got a piece of the treasure, since they were accessing the treasure at the same time.

The solution was to use the Atomic Integer to make sure that the treasure would actually decrement.

Mechanisms

All of the threads join together.

```
protected static volatile int dragonHealth = 300;
```

Volatile keyword used so that all threads could see so that would prevent the threads from reading the same dragon health. There is also the synchronized block and atomic integer described above.

Lambdas are used to list the battle results from each thread.

```
battleResults.forEach(result -> System.out.println("- " + result));
```

Stream operations are used to calculate battle damage.

```
int totalDamage = Arrays.stream(characterDamage).sum();
```