CSCI 1110 – Fall 2020

Assignment 03

Please start this assignment early; programming and logic take time - if you leave it to the last minute, you might not have enough time to finish or might make silly mistakes that you otherwise could avoid. Note that TAs and Instructors will not be able to answer last-minute questions!

All work is to be handed in Mimir, our online code learning environment. You should, however, write your code on an IDE such as IntelliJ.

To complete this assignment, you will need to know about:

- Basic Java
- Conditionals
- Boolean Variables
- Loops
- Objects and Classes
- Inheritance

This is your first assignment with objects. The test cases from now on **are slightly different**. Some test cases will be regular input/input as before; however, **most tests cases** will be unit tests. In unit test cases there is no input from the user. The test can evaluate your code directly without any user input when needed. The tests can also test methods individually **as long as your code is compiling**. Before you can even try to test your code **it must be compiling**.

Your code **must compile**. If it does not compile, you will receive a 0 (zero) on that portion of the assignment, and no partial marks will be given.

Remember that students who ***hardcode*** their outputs to match the test cases in Mimir will receive a **zero** on the entire assignment.

Grading Scheme: Please see the grading scheme at the end of this document.

Coding Style: You must proper variable names and comments. Please follow the guidelines on https://web.cs.dal.ca/~franz/CodingStyle.html

Problem Overview

In OOP we model our problems using objects that we can manipulate. To create and use objects, we need to write classes. Classes are a sort of a blueprint that describes the properties and behaviour of the objects we want to create.

In this assignment we are going write classes representing **Characters** for RPG games similar to World of Warcraft and others. Our game will have different kinds of Characters and Attacks. We will use Inheritance to reduce duplicated code, allow for easy expansion of the game's base carachters (and attakcs), and explore polymorphic behaviour.

In a real-world scenario we would also be leveraging the concepts of Interfaces to write the code. We will see interfaces in the near future; however, because we are a little behind in the schedule we will not have interfaces on this assignment.

After you finish this assignment, try to answer the following questions:

- How diffictulry it would be to add a new type of character to the game?
- What code do you need to change to add the new character?
- How difficult would it be to implement our game without using inheritance?

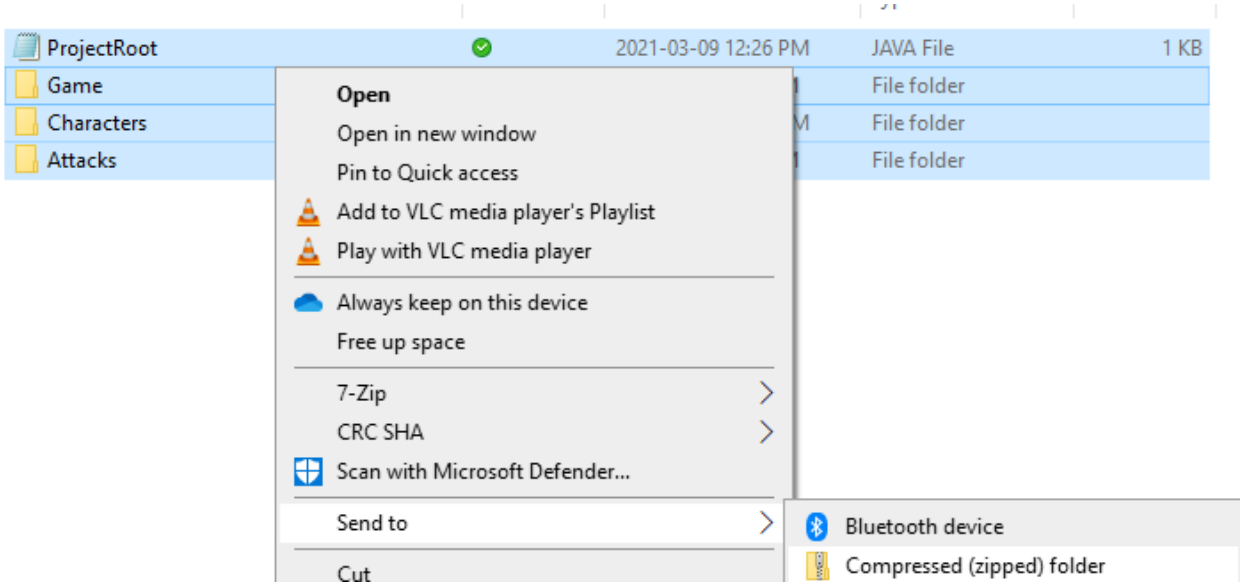
The starter code in Mimir comes with three packages: Attacks, Characters, and Game.

- **Attacks:** Contain the classes related to attacks in the game. These classes implementation is provided. **Study them before starting the assignment.** You don't have to edit the code inside this package.
- **Characters:** Contain the classes related to our game characters. We will have initially three concrete Characters: Mage, Priest, and Warrior
- **Game:** This package has an example code on how to use the Charcaters and Attacks to write a small turn-based duel of two game characters. You don't have to edit this code. You can run it once you have done the assignment. You can also study it to understand a bit more of the idea of using abstract types and polymorphism.

How to upload your solution to mimir: You must preserve the package structure when submitting your solution to mimir. The **only way** (that I know of) to preserve is to zip your files an upload the zip file.

On Windows: Select the three folders inside the src folder, right click on them, and "Send to compresed (zipped) folder".

On Mac: Select the three folders inside the src folder, right click (option click?), "Compress 3 items"



DO NOT WAIT UNTIL THE LAST WEEKEND TO START THIS ASSIGNMENT. Even if you don't have a lot of time now, at least download the starter code on Mimir, and try to create the zipped file.

"Help! Where do I begin?":

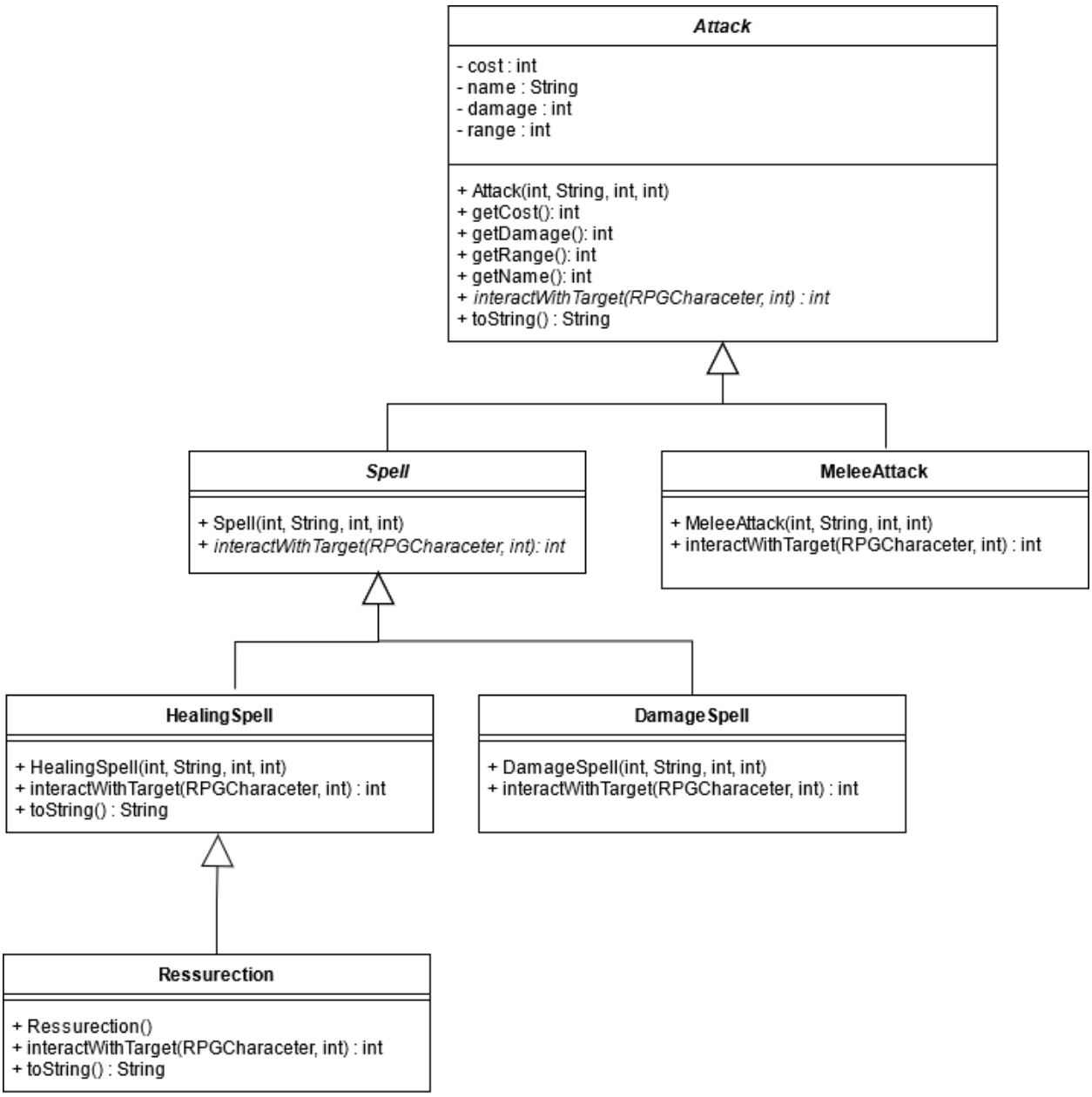
Review the classes inside the Attack package. Read the Point (see below) documentation.

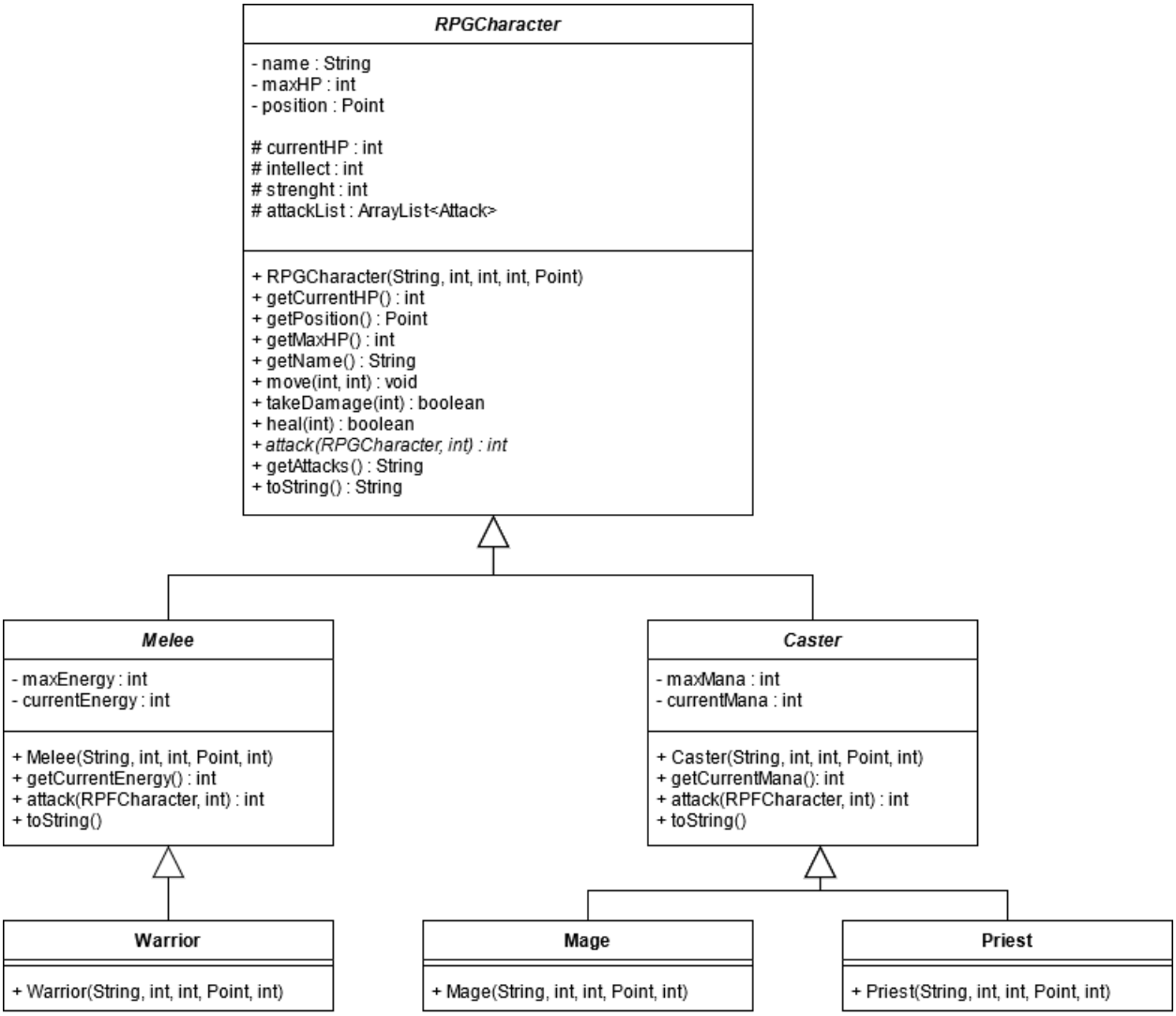
You always start with the most generic classes in the inheritance trees. Start with the **RPGCharacter**, then create the the more specialized classes.

This assignment is less complicated thant it seems codingwise. Review the concepts of inheritance before starting it.

The Point class: We will be using Java's Point class. You have to import java.awt.Point. More information on this class can be found here:
https://docs.oracle.com/javase/8/docs/api/java/awt/Point.html

The Assignments Classes and Inheritance Diagram





You don't have to write JavaDocs for: getCurrentHP, getPosition, getMaxHP, getName, toString, getCurrentEnergy, getCurrentMana

Implementation details for Attack

I've opted to reduce your workload a little. All the implementation for the attacks is provided.

Implementation details for RPGCharacter

RPGCharacter, Melee, and Caster should be abstract

RPGCharacter

- The constructor's parameters are in the following order: name, intellect, strength, maxHP, and position.
- The move method will move the character by calling the translate method form the Point class. The parameter order is x,y
- TakeDamage: deals damage to the character by deducting parameter to the character's current hp. If the HP falls bellow zero (included) the method should set the currentHP to zero and return false indicating that the character is dead.
- Heal: heals the character by adding the parameter to the character's current hp. A character **cannot** have more HP than the maxHP value. Returns true if the character is fully healed.
- Attack should be abstract
- GetAttacks will return a string with each of the character's attacks on a new line. Each line should contain the attack's index on the ArrayList and the attack's string representation. Example:

```
0 - MeleeAttack - Wand (0) - Damage: 3
1 - DamageSpell - Smite (10) - Damage: 10
2 - HealingSpell - Flash Heal (20) - Heal: 15
3 - Resurrection - Ressurection (50)
```

- ToString returns a String representing the Characters. The string should contain (in this order): the name, the type[1] , currentHP and maxHP:

```
Gandalf (Mage) - 100/100
```

Caster

- This class will represent all Casters in our game. Casters are characters who deal damage by casting spells primarly.

[1] You can retrieve the class of the implicit parameter using getClass(); You can retrieve the class' name using getSimplifiedName(). How can you use these two methods to reduce the numbers of toStrings in the subclasses?

- The caster's constructor will receive the parameters in this order: name, intellect, maxHP, position, and maxMana. Casters should be initialzed with a strength of 1 (one) and currentMana as maxMana.
- The attack method should work as follows:
    - This method will select the attack from the ArrayList and call the interactWithTarget method to inflict damage or heal the target.
    - The mehod will return a negative number in case the attack fails:
        - -1 attackIndex not in range of the ArrayList
        - -2 target out of range (check the Point's calss docs to see how you can calculate the distance between two points
        - -3 not enough mana
    - The method will return the target's currentHP if the attack is sucessifull
    - Casters can cast two types of spells and can also deal very basic melee attacks. You can see the Mage and Priest classes for example. The attack method will behave differently depending on the instance of the attack selected.
        - HealingSpells should heal the character itself (implicit parameter) and pass the character's intellect as the modifier value
        - For any other spell, the method will attack the target character and pass the intellect as the modifier
        - For non-spell attacks, it should attack the target and pass zero (0) as the modifier
    - You should use the interactWithTarget method from the attack that was selected via the index parameter
- The toString method should append the caster's mana underneath the character's toString:

```
Gandalf (Mage) - 100/100
Mana: 100/100
```

Melee

- This class represents all Melee characters. Instead of casting spells, these types of characters deal damage using physical attacks. Most of these attacks will consume energy.
- The constructor's parameters are, in this order, name, strength, maxHP, position, and maxEnergy. The currentEnergy should be initialized to the maxEnergy value and an intellect of 1
- The attack method is simpler than the caster's.
    - It will return a negative number if the attack fails:
        - -1 attackIndex not in the range of the ArrayList
        - -2 target out of range
        - -3 not enough energy
    - If the attack succeds, it will:

        - Deduct the energy cost
        - Call the interactWithTarget method from the chosen attack to inflict damage on the target
        - Return the target's currentHP after the attack
- The toString method will add the Characters current energy and max energy to the character's default toString implementation:

```
Magni Bronzebeard (Warrior) - 100/100
Energy 100/100
```

Mage, Priest, Warriors

- These are concrect classes (not abstract) in our game. Each class will create a character with a initial list of attacks (melee or spells).
- **Priest:**
    - Wand, cost 0, damage 3, range 3
    - Smite, cost 10, damage 10, range 7
    - Flash Heal, cost 20, heal 15, range 15
    - Ressurrection
- **Mage**
    - Staff, cost 0, damage 3, range 3
    - Fire Ball, cost 20, damage 10, range 20
    - Frost Ball, cost 15, damage 7, range 15
    - Lightning, cost 30, damage 15, range 20
- **Warrior**
    - Punch, cost 0, damage 5, range 3
    - Slam, cost 3, damage 5, range 3
    - Charge, cost 20, damage 30, range 15

How do I test an abstract class?

If you cannot create an object of a class how do you test it?

There are a couple of different ways but you should do as follows until you learn more about OOP:

- Let's assume you want to test your RPGCharacter class.

1. Create an empty class (e.g. ConcreteRPGCharacter)
2. Extend RPGCharacter
3. For each abstract method in RPGCharacter, write a "dumb" implementation.
    a. If the method return type is void, write "return;" as the implementation
    b. If the method is returning a primitive type, write "return 0;" as the implementation

    c. If the method is returing a reference type, write "return null;" as the implementation
4. Create a tester/runner class.
5. In the main method, create instances of ConcreteRPGCharacter.
6. You can now test the non-abstract methods of RPGCharaceter by calling them on the ConcreteRPGCharacter object.

Grading Scheme

Each problem on the assignment will be graded based on three criteria:

Functionality

"Does it work according to specifications?" This is determined in an automated fashion by running your program on a number of inputs and ensuring that the outputs match the expected outputs. The score is determined based on the number of tests that your program passes.

Quality of Solution

"Is it a good solution?" This considers whether the solution is correct, efficient, covers boundary conditions, does not have any obvious bugs, etc. This is determined by visual inspection of the code. Initially full marks are given to each solution and marks are deducted based on faults found in the solution.

Code Clarity

"Is it well written?" This considers whether the solution is properly formatted, well-documented, and follows coding style guidelines (https://web.cs.dal.ca/~franz/CodingStyle.html).

If your program does not compile, it is considered non-functional and of extremely poor quality, meaning you will receive 0 for the solution.

PROBLEM	POINTS
RPGCHARACTER CLASSES	70
QUALITY OF SOLUTION & CODE CLARITY	30
TOTAL	**100**