HW 5: Role Playing Game

(Deadline per Canvas)

This homework deals with the following topics:

- Setting up the Java programming environment and Eclipse (if you haven't done so already)
- An introduction to Java programming, syntax, and style
- Getting you started with class-based object oriented programming

Installing/Configuring Java & Eclipse

If needed, please reference the "Installing/Configuring Java & Eclipse" video for information on installing/configuring Java & Eclipse.

Preamble note on equality operators (IMPORTANT)

Given that this is the first assignment using Java, we want to highlight the difference between Python and Java when comparing two objects such as Strings. This is one of the few instances where the same syntax, in this case the "==" operator, results in different behavior between these two languages. Please see the following table that outlines the differences.

Equality Comparison	**Python**	**Java**
Values/contents	==	.equals()
Identities/memory locations	is	==
Strings Value Comparison Example (both will result in true/True)	str1 = "hello" str2 = "hello" is_equal = (str1 == str2)	String str1 = new String("hello"); String str2 = new String("hello"); boolean isEqual = str1**.equals(**str2**)**;

About the Assignment

Welcome to the final battle against the enemy's forces! Your mission is to defeat the computer's units before your units are knocked out. This is a strategic turn-based game where two players, one human and one computer, compete by controlling units with specific jobs and abilities. The goal is to defeat the opponent's units by reducing their Health Points (HP) to zero through a series of tactical moves and attacks.

At the start of the game, both the human and computer players are assigned three units. Each unit is randomly given a job, such as Knight, Mage or Archer, and a level, which determines its attributes like HP, Attack, Evasion, and Defense. These attributes play a crucial role in the units' performance during battles.

The game begins with the human player's turn. During their turn, the human player controls each of their units one by one. For each unit, the player is presented with its job type and level, indicating its readiness to act. The player can choose from two actions for each unit: attack or block. If "attack" is selected, the unit targets an opposing unit and deals damage based on the attacking unit's Attack stat value and the defending unit's Defense stat value. Before the damage is applied, the defending unit may attempt to evade the attack. If the defending unit's Evasion stat is high enough, the attack may be dodged entirely, resulting in no damage. If the evasion attempt is unsuccessful, the damage is then calculated by factoring in the defending unit's Defense stat and any temporary defense adjustments. If "block" is selected, the unit receives a temporary increase in the unit's Defense stat value to reduce incoming damage from the opponent's attacks.

Once the human player has completed actions for all of their three units, the game proceeds to the computer player's turn. The computer player automatically controls its units, performing similar actions to those of the human player. Each unit acts in sequence, with the computer deciding the best moves to defeat the human player's units.

The game is turn-based, with each player taking turns until a winner is determined or the turn limit (10 turns) is reached. Each player's turn involves taking actions with all their living units. The game continually checks for a winner at the end of each player's turn. The game ends when a player defeats all of the opponent's units. If the game reaches 10 turns without a winner, the player with the higher combined HP of their units wins.

Required Classes and Methods

The following methods must be present in your code with these given names and method signatures exactly. For the autograder to run properly, do not change the method names or the parameters. Do not add optional parameters or change the return types. If a method returns something, make sure what it returns is consistent with what's mentioned in the details below. Be sure to add javadocs to all methods that do not already have them and comments to your code. **Note: ComputerPlayer.Java has been fully implemented; you do not need to make any changes to this file. However, you may wish to thoroughly review ComputerPlayer.java, as well as HumanPlayer.java, Unit.java and GameControl.java, to understand what methods are available for your use before you begin the assignment.**

The following are the required methods you have to implement (Note: Some methods may be used in other methods):

GameControl.java

The *GameControl* class controls the entire game. It has a *main* method which serves as the entry point to the entire program and it will be the file you will "run".

getWinner(int turn):

This method checks if there is a winner in the game. It determines if either the human player or the computer player has been completely defeated. The method first checks if the current turn parameter is less than 10. This serves as an initial condition to determine the winner in the early stages of the game.

- Both Players Alive: If both the human player and the computer player still have units with more than 0 HP, the method returns null, indicating that there is no winner yet.
- Computer Defeated: If all of the computer's units have 0 or less HP, the method returns "human", indicating that the human player has won.
- Human Defeated: If all of the human's units have 0 or less HP, the method returns "computer", indicating that the computer player has won.

If the turn parameter is 10 or greater, the method evaluates the game based on the remaining HP (Health Points) of all units.

- Sum of Human Units' HP: Calculates the sum of the HP of all human units (Falia, Erom, Ama).
- Sum of Computer Units' HP: Calculates the sum of the HP of all computer units (Criati, Ledde, Tyllion).

Comparison:

- If the human units' combined HP is greater than the computer units' combined HP, the method returns "human".
- If the computer units' combined HP is greater than the human units' combined HP, the method returns "computer".
- If the combined HP of both sides is equal, the method returns "tie".

Return Value: A string ("human", "computer", “tie”, or null) indicating the winner if there is one.

takeHumanTurn(int turn):

This method handles the actions taken by the human player during their turn. This method does not return anything. Each of the human player’s units (Falia, Erom, and Ama) will take a turn to either attack or block.

- Unit Turn for Falia: Prints Falia's job and level, then allows Falia to take an action (attack or block) using the moveUnit method. Calls getWinner to check for a winner after Falia's action.
- Unit Turn for Erom: Prints Erom's job and level, then allows Erom to take an action using the moveUnit method. Calls getWinner to check for a winner after Erom's action.
- Unit Turn for Ama: Prints Ama's job and level, then allows Ama to take an action using the moveUnit method. Checking for a winner is not necessary after the third unit takes its turn because it will be checked in main right after the human turn ends.
- Reset Computer's Temporary Defenses: Resets the computer player's temporary defense buffs since they only last for one turn.

Print Statements: Throughout the method, prints blank lines and messages to guide the player and indicate which human unit is currently acting (as well as its job and level).

takeComputerTurn():

This method handles the actions taken by the computer player during their turn. This method does not take any parameters and does not return anything. The computer player uses a strategy to decide actions for each of its units.

- Computer Strategy: The computer player takes actions for each of its units (Criati, Ledde, Tyllion) against the human player's units (Falia, Erom, Ama).
- Reset Human's Temporary Defenses: Resets the human player's temporary defense buffs since they only last for one turn.

Print Statements: Messages are already included to indicate the computer's actions and any changes in the game state.

main(String[] args):

This is the entry point of the game, managing the flow of gameplay, turn-taking, and winner determination. The args parameter is not used (leave the method signature as is), and this method does not return anything.

- Create an instance of GameControl and call printInstructions to display the game's context and rules.
- The game loop runs for up to 10 turns and prints the current turn number.
- The human plays first. Call printStatus to show the current unit statuses and then call takeHumanTurn to execute the human player’s actions. Check for a winner after the human turn. Break out of the loop if a winner is found.
- Then the computer plays. Call printStatus to show updated unit statuses and then computerTurn to execute computer player actions. Again, check for a winner after the call takeComputerTurn. Break out of the loop if a winner is found.
- If no winner is determined after 10 turns, call getWinner to determine the winner based on combined HP of all units.

- Prints the result based on the outcome from the getWinner method:
    - "All your heroes have been defeated, enemy forces have won!" for computer win.
    - "You've defeated the enemy!" for human win.
    - "Nobody wins!" for a tie.

Unit.java

The following getters/setters are fully implemented. Please do not change their implementation.

setEvasion(int evasion):

This method sets the evasion value of the unit.

getTemporaryDefense():

This method retrieves the temporary defense value of the unit.

Please implement the following getters/setters:

getLevel():

This method retrieves the level of the unit.

getJob():

This method retrieves the job of the unit.

getHp():

This method retrieves the current HP (health points) of the unit.

setHp(int hp):

This method sets the HP (health points) of the unit to a specified value.

setTemporaryDefense(int temporaryDefense):

This method sets the temporary defense value of the unit and is used when blocking.

Unit(String name, String levelRange, String job):

This constructor initializes a Unit object with a given name, level range, and job. It also calculates and sets the unit's level and other stats based on the provided level range.

- Assigns the provided name and job to the unit.
- Randomly sets the unit's level based on the provided level range:
    - "low" results in a level between 1 and 3
    - "medium" results in a level between 4 and 6
    - "high" results in a level between 7 and 10
- Calculates and assigns the unit's stats (HP, attack, defense, evasion) based on the level using a multiplier.

printCurrentStatus():

This method prints the current status of the unit, including its name, level, job, and remaining HP. If the unit is knocked out (HP < 1), it prints a message indicating that the unit is knocked out and cannot move. This method does not take any parameters and does not return anything.

attack(String attackerStrength):

The purpose of the attack method is to compute the total damage a unit will inflict on a target, considering the attacker's strength relative to the target ("strong", "weak" or “same”) and applying appropriate multipliers to the damage calculation.

- Determine the value of the multiplier based on the attacker's strength. Initialize a multiplier variable to 1.0, assuming a neutral strength as the default.
    - Check the value of attackerStrength:
        - If attackerStrength is "strong", set the multiplier to 1.2 to represent a 20% increase in damage.
        - If attackerStrength is "weak", set the multiplier to 0.5 to represent a 50% reduction in damage.
- Define a maximum attack variable (attackMax) with a value of 50.0, representing the maximum potential damage before applying this unit's attack stat and multiplier.

- Calculate the raw damage based on the attacker's attack stat. The formula used is:
    - rawDamage = (this.attack / 30.0) * attackMax;
- Apply the multiplier to the raw damage to get the total damage and convert the calculated damage to an integer:
    - damage = (int) Math.round(rawDamage * multiplier)

Return the total damage value after applying any type-based adjustments.

block()

This method provides a temporary defensive buff to the unit that uses it. This method does not take any parameters and does not return anything. When a unit chooses the "Block" action during its turn, its defense is temporarily increased by adding 2 points to its temporaryDefense stat. This increased defense helps the unit reduce the damage taken from incoming attacks during the current turn. This increase only lasts for the duration of the current turn and is reset afterward.

This action should be chosen when the player anticipates receiving an attack from the opponent. By blocking, the unit becomes more resilient, making it a strategic option to mitigate damage.

receiveDamage(int damage):

The purpose of the receiveDamage method is to process incoming damage to a unit, considering evasion chances and defense adjustments, and update the unit's HP accordingly. This method does not return anything.

Evasion Check:

- The method first checks if the unit has any evasion stat greater than 0.
- If the evasion stat is greater than 0, it generates a random number between 0 and 20 (inclusive) using this.random.nextInt(21).
- If the generated random number is less than or equal to the unit's evasion stat, the unit successfully dodges or avoids the attack. The method prints "They dodged!" and returns without applying any damage to this unit.

Calculate Defense Adjustment:

- If the unit does not dodge the attack, the method calculates a defense adjustment factor (defenseAdjustment). This factor is computed as:
    - defenseAdjustment = (float) ((this.temporaryDefense + this.defense) / 10.0)

Calculate Actual Damage Received:

- The actual damage received (damageReceived) is calculated by dividing the incoming damage (damage) by the defense adjustment factor (defenseAdjustment). The result is then rounded to the nearest integer.

Update Unit's HP:

- The method subtracts the calculated damage (damageReceived) from the unit's current HP (this.hp).
- If the resulting HP is less than or equal to 0, the method sets this unit's HP to 0 to prevent negative values from being displayed.
- The method prints the actual damage received and the unit's remaining HP to the console.

HumanPlayer.java

The following methods are getters/setters. Please do not change their implementation.

getFalia():

This method returns the Falia unit of the HumanPlayer.

getErom():

This method returns the Erom unit of the HumanPlayer.

getAma():

This method returns the Ama unit of the HumanPlayer.

generateLevel():

This method generates a random level for a unit. The possible levels are "low", "medium", and "high". This method does not take any parameters.

- It generates a random integer between 0 and 2.
- Based on the value of the random integer, it assigns a string representing the level:
    - 0 -> "low" , 1 -> "medium" , 2 -> "high"

Return Value: A String representing the randomly generated level.

generateJob():

This method generates a random job for a unit. The possible jobs are "mage", "knight", and "archer". This method does not take any parameters.

- It generates a random integer between 0 and 2.
- Based on the value of the random integer, it assigns a string representing the job:
    - 0 -> "mage", 1 -> "knight" , 2 -> "archer"

Return Value: A String representing the randomly generated job.

HumanPlayer():

This constructor is responsible for creating an instance of a HumanPlayer and initializing its three units: falia, erom, and ama.

- The constructor initializes the three units with their names (Falia, Erom, Ama) and randomly generated levels and jobs.
- Each unit is created by calling the Unit constructor with parameters for the unit's name, level (generated by generateLevel()), and job (generated by generateJob()).

The following are the required methods you have to implement:

validateMove(String move):

The purpose of the validateMove method is to ensure that the move entered by the player is valid. It returns a standardized string representing the move if it is valid or prompts the player to enter a valid move if it is not.

- Convert the move string to lowercase to ensure the method is case-insensitive. This allows for inputs like "Attack", "attack", "A", or "a" to be treated equally.
- Check First Letter of Move: If the lower case of the first letter is 'a', the method recognizes the move as an *attack* and returns the lower-case string "attack". Else, if the lower case of the first letter is 'b', the method recognizes the move as a *block* and returns the lower-case string "block". Note that any string that begins with either the letters “a” or “b” of either case will be acceptable.

- If the first letter is neither “a” nor “b”, the method prints a friendly message such as "Please enter a valid move." to prompt the player to input a correct move.

The method then returns null to indicate that the move was invalid.

selectTarget(String targetName, ComputerPlayer computer):

This method checks if the selected target from the computer's units is valid and alive. If the target is valid and alive, it returns the target’s Unit object; otherwise, it prints an error message and returns null.

- Target Validation: The method compares the selected target name with the names of the computer’s units ("Criati", "Ledde", "Tyllion"). In this specific method case **does** matter in the sense that the units’ names begin with an uppercase letter.
- Alive Check: For each valid target name, it checks if the target’s HP is greater than 0.
- Return Value: If the target is valid and alive, the method returns the corresponding Unit object. Hint: Each computer unit has a getter method with the following functionality. If the target is invalid or already defeated, it prints a message stating that the target is not a member of the enemy’s forces or has already been vanquished, and returns null as the Return Value.

moveUnit(Unit unit, ComputerPlayer computer):

This method determines the action a human unit will take during its turn. This method does not return anything. The unit can either attack a target or block incoming attacks.

1. Alive Check: First, check if the unit is alive (HP > 0). If the unit is knocked out, print a message and the turn ends.
2. Move Selection: Prompt the human player to select a move ("attack" or "block"). The input is validated to ensure it is a valid move using validateMove. Reprompt if the input is not valid.
3. Attack:
    - If the move selected is "attack", prompt the player to select a target from the computer's units.
    - The selected target is validated using selectTarget to ensure it is alive. If not, print an error message and reiterate to the user to pick another target.

- Once a valid target is selected:
  - Call the determineAttackerStrength method to get the strength of the attacker relative to the target (e.g., "strong", "weak", "same").
  - Call the unit's attack method with the determined strength to calculate the damage the unit will deal.
  - Call the receiveDamage method with the calculated damage to apply the damage to the selected target.

4. Block:
   - If the move selected is "block", the unit's block ability is activated, which usually increases its defense during the opposing player's next turn.
     - Call the unit's block method to temporarily increase the temporaryDefense of the unit.

User Interface

You're free to create your own user interface for the game, as long as it makes sense for the user playing.

- Your program should run for 10 turns starting with the human player. A turn should proceed as follows:
- Print the turn number, and print the status of all units (both human and computer)
- For human player turn, for each unit, print the unit's type and level, then prompt the player for an action ('a' for attack or 'b' for block). If the player chooses to attack, prompt for a target, and print the result of the action.
- Print the updated status of the target.
- Before the computer player's turn, print the status of all units (both human and computer).
- After each player's turn, check if there is a winner and print the result if a winner is found.

If the game reaches the turn limit (10 turns) without a clear winner, determine the winner based on combined HP and print the result.

Program Output

Example 1:

```
==================================================
Turn 1

Your units:
Falia is a level 9 archer with 90 HP remaining.
Erom is a level 2 knight with 20 HP remaining.
Ama is a level 9 knight with 90 HP remaining.

Computer units:
Criati is a level 9 knight with 90 HP remaining.
Ledde is a level 4 knight with 40 HP remaining.
Tyllion is a level 10 mage with 100 HP remaining.

ARCHER Falia (Lv: 9) is ready to act:
Select a move:
a
Pick a target that is alive:
Ledde
Ledde takes 19 damage. Remaining HP: 21

KNIGHT Erom (Lv: 2) is ready to act:
Select a move:
A
Pick a target that is alive:
Tyllion
Tyllion takes 2 damage. Remaining HP: 98

KNIGHT Ama (Lv: 9) is ready to act:
Select a move:
a
Pick a target that is alive:
Ledde
Ledde dodged!


Your units:
Falia is a level 9 archer with 90 HP remaining.
Erom is a level 2 knight with 20 HP remaining.
Ama is a level 9 knight with 90 HP remaining.

Computer units:
Criati is a level 9 knight with 90 HP remaining.
Ledde is a level 4 knight with 21 HP remaining.
Tyllion is a level 10 mage with 98 HP remaining.

Criati attacks
Falia dodged!
Falia has 90 remaining.

Ledde attacks
Falia dodged!
Falia has 90 remaining.

Tyllion attacks
Ama takes 22 damage. Remaining HP: 68
Ama has 68 remaining.

==================================================
```

Example 2:

```
Welcome to the final battle against enemy forces. You will be facing off against the computer.
Each of you will have 3 units with randomly generated jobs and levels.
The jobs are: mage, knight, and archer. Archers are strong against mages, but weak against knights.
Mages are strong against knights, but weak against archers. Knights are strong against archers, but weak against mages.
There are two moves: attack (deal damage to one target) and block (temporarily increase defense).
Combat is turn based; all your love units will take a turn and then all the computer's live units will take a turn.
You have 10 turns to defeat the computer. If both players still have units standing, you only win
if the combined HP of your units exceeds the computer's.

==================================================
Turn 1

Your units:
Falia is a level 10 mage with 100 HP remaining.
Erom is a level 2 archer with 20 HP remaining.
Ama is a level 4 knight with 40 HP remaining.

Computer units:
Criati is a level 4 knight with 40 HP remaining.
Ledde is a level 6 mage with 60 HP remaining.
Tyllion is a level 6 archer with 60 HP remaining.

MAGE Falia (Lv: 10) is ready to act:
Select a move:
a
Pick a target that is alive:
Criati
Criati dodged!

ARCHER Erom (Lv: 2) is ready to act:
Select a move:
a
Pick a target that is alive:
Criati
Criati takes 4 damage. Remaining HP: 36

KNIGHT Ama (Lv: 4) is ready to act:
Select a move:
a
Pick a target that is alive:
Criati
Criati takes 16 damage. Remaining HP: 20


Your units:
Falia is a level 10 mage with 100 HP remaining.
Erom is a level 2 archer with 20 HP remaining.
Ama is a level 4 knight with 40 HP remaining.

Computer units:
Criati is a level 4 knight with 20 HP remaining.
Ledde is a level 6 mage with 60 HP remaining.
Tyllion is a level 6 archer with 60 HP remaining.

Criati attacks
Erom takes 40 damage. Remaining HP: 0
Erom has 0 remaining.

Ledde attacks
Ama takes 30 damage. Remaining HP: 10
Ama has 10 remaining.

Tyllion attacks
Falia takes 12 damage. Remaining HP: 88
Falia has 88 remaining.

==================================================
```

```
==================================================
Turn 2

Your units:
Falia is a level 10 mage with 88 HP remaining.
Erom is knocked out and cannot move.
Ama is a level 4 knight with 10 HP remaining.

Computer units:
Criati is a level 4 knight with 20 HP remaining.
Ledde is a level 6 mage with 60 HP remaining.
Tyllion is a level 6 archer with 60 HP remaining.

MAGE Falia (Lv: 10) is ready to act:
Select a move:
block

ARCHER Erom (Lv: 2) is ready to act:
Erom unit has 0 HP.

KNIGHT Ama (Lv: 4) is ready to act:
Select a move:
a
Pick a target that is alive:
Criati
Criati takes 16 damage. Remaining HP: 4


Your units:
Falia is a level 10 mage with 88 HP remaining.
Erom is knocked out and cannot move.
Ama is a level 4 knight with 10 HP remaining.

Computer units:
Criati is a level 4 knight with 4 HP remaining.
Ledde is a level 6 mage with 60 HP remaining.
Tyllion is a level 6 archer with 60 HP remaining.

Criati attacks
Ama takes 16 damage. Remaining HP: 0
Ama has 0 remaining.

Ledde attacks
Falia takes 9 damage. Remaining HP: 79
Falia has 79 remaining.

Tyllion attacks
Falia takes 11 damage. Remaining HP: 68
Falia has 68 remaining.

==================================================
```

```
==================================================
Turn 3

Your units:
Falia is a level 10 mage with 68 HP remaining.
Erom is knocked out and cannot move.
Ama is knocked out and cannot move.

Computer units:
Criati is a level 4 knight with 4 HP remaining.
Ledde is a level 6 mage with 60 HP remaining.
Tyllion is a level 6 archer with 60 HP remaining.

MAGE Falia (Lv: 10) is ready to act:
Select a move:
b

ARCHER Erom (Lv: 2) is ready to act:
Erom unit has 0 HP.

KNIGHT Ama (Lv: 4) is ready to act:
Ama unit has 0 HP.


Your units:
Falia is a level 10 mage with 68 HP remaining.
Erom is knocked out and cannot move.
Ama is knocked out and cannot move.

Computer units:
Criati is a level 4 knight with 4 HP remaining.
Ledde is a level 6 mage with 60 HP remaining.
Tyllion is a level 6 archer with 60 HP remaining.

Criati is blocking; their defense temporarily increases for the next turn!
Ledde attacks
Falia takes 9 damage. Remaining HP: 59
Falia has 59 remaining.

Tyllion attacks
Falia takes 11 damage. Remaining HP: 48
Falia has 48 remaining.

==================================================
```

```
==================================================
Turn 4

Your units:
Falia is a level 10 mage with 48 HP remaining.
Erom is knocked out and cannot move.
Ama is knocked out and cannot move.

Computer units:
Criati is a level 4 knight with 4 HP remaining.
Ledde is a level 6 mage with 60 HP remaining.
Tyllion is a level 6 archer with 60 HP remaining.

MAGE Falia (Lv: 10) is ready to act:
Select a move:
a
Pick a target that is alive:
Criati
Criati dodged!

ARCHER Erom (Lv: 2) is ready to act:
Erom unit has 0 HP.

KNIGHT Ama (Lv: 4) is ready to act:
Ama unit has 0 HP.


Your units:
Falia is a level 10 mage with 48 HP remaining.
Erom is knocked out and cannot move.
Ama is knocked out and cannot move.

Computer units:
Criati is a level 4 knight with 4 HP remaining.
Ledde is a level 6 mage with 60 HP remaining.
Tyllion is a level 6 archer with 60 HP remaining.

Criati is blocking; their defense temporarily increases for the next turn!
Ledde attacks
Falia takes 10 damage. Remaining HP: 38
Falia has 38 remaining.

Tyllion attacks
Falia takes 12 damage. Remaining HP: 26
Falia has 26 remaining.

==================================================
```

```
==================================================
Turn 5

Your units:
Falia is a level 10 mage with 26 HP remaining.
Erom is knocked out and cannot move.
Ama is knocked out and cannot move.

Computer units:
Criati is a level 4 knight with 4 HP remaining.
Ledde is a level 6 mage with 60 HP remaining.
Tyllion is a level 6 archer with 60 HP remaining.

MAGE Falia (Lv: 10) is ready to act:
Select a move:
a
Pick a target that is alive:
Criati
Criati takes 40 damage. Remaining HP: 0

ARCHER Erom (Lv: 2) is ready to act:
Erom unit has 0 HP.

KNIGHT Ama (Lv: 4) is ready to act:
Ama unit has 0 HP.


Your units:
Falia is a level 10 mage with 26 HP remaining.
Erom is knocked out and cannot move.
Ama is knocked out and cannot move.

Computer units:
Criati is knocked out and cannot move.
Ledde is a level 6 mage with 60 HP remaining.
Tyllion is a level 6 archer with 60 HP remaining.

Ledde attacks
Falia takes 10 damage. Remaining HP: 16
Falia has 16 remaining.

Tyllion attacks
Falia takes 12 damage. Remaining HP: 4
Falia has 4 remaining.

==================================================
```

```
==================================================
Turn 6

Your units:
Falia is a level 10 mage with 4 HP remaining.
Erom is knocked out and cannot move.
Ama is knocked out and cannot move.

Computer units:
Criati is knocked out and cannot move.
Ledde is a level 6 mage with 60 HP remaining.
Tyllion is a level 6 archer with 60 HP remaining.

MAGE Falia (Lv: 10) is ready to act:
Select a move:
a
Pick a target that is alive:
Ledde
Ledde takes 27 damage. Remaining HP: 33

ARCHER Erom (Lv: 2) is ready to act:
Erom unit has 0 HP.

KNIGHT Ama (Lv: 4) is ready to act:
Ama unit has 0 HP.


Your units:
Falia is a level 10 mage with 4 HP remaining.
Erom is knocked out and cannot move.
Ama is knocked out and cannot move.

Computer units:
Criati is knocked out and cannot move.
Ledde is a level 6 mage with 33 HP remaining.
Tyllion is a level 6 archer with 60 HP remaining.

Ledde attacks
Falia takes 10 damage. Remaining HP: 0
Falia has 0 remaining.

Falia, Erom, and Ama have all fallen. There cannot be targetted.
All your heroes have been defeated, enemy forces have won!
```

Comments & Style

In general, all of the comment and style conventions from Python also apply in Java, and comments are expected for any non-trivial lines of code and the length of your lines should not go off the right hand edge of the editing window. The following table outlines the differences between Python and Java comments and style, and for ***this and all other Java assignments the Java comments and style standards are expected***.

Comment/Style Comparison	**Python**	**Java**
Single Line Comment	# single line comment	// single line comment
Multi-line Comment	""" This is a multi-line comment. It can span multiple lines. """	/* This is a multi-line comment. It can span multiple lines. */
Variable Naming Convention	example_variable second_example	exampleVariable secondExample

JavaDocs

Similar to Python javadocs, which are used to document modules, classes, methods, and methods, Java uses a standard documentation convention called JavaDocs. The specified JavaDoc format has an additional benefit in that it can be used to generate HTML documentation automatically if used properly. If they do not already exist, please ***add JavaDocs to all class definitions and methods***.

JavaDoc Example

```
/**
 * A brief description of the class or method.
 *
 * A more detailed description of the class or method. This can span multiple
 * lines and provide comprehensive information about the functionality.
 *
 * @param paramName Description of the parameter(s).
 * @return Description of the return value.
 *
 */
```

Unit Testing

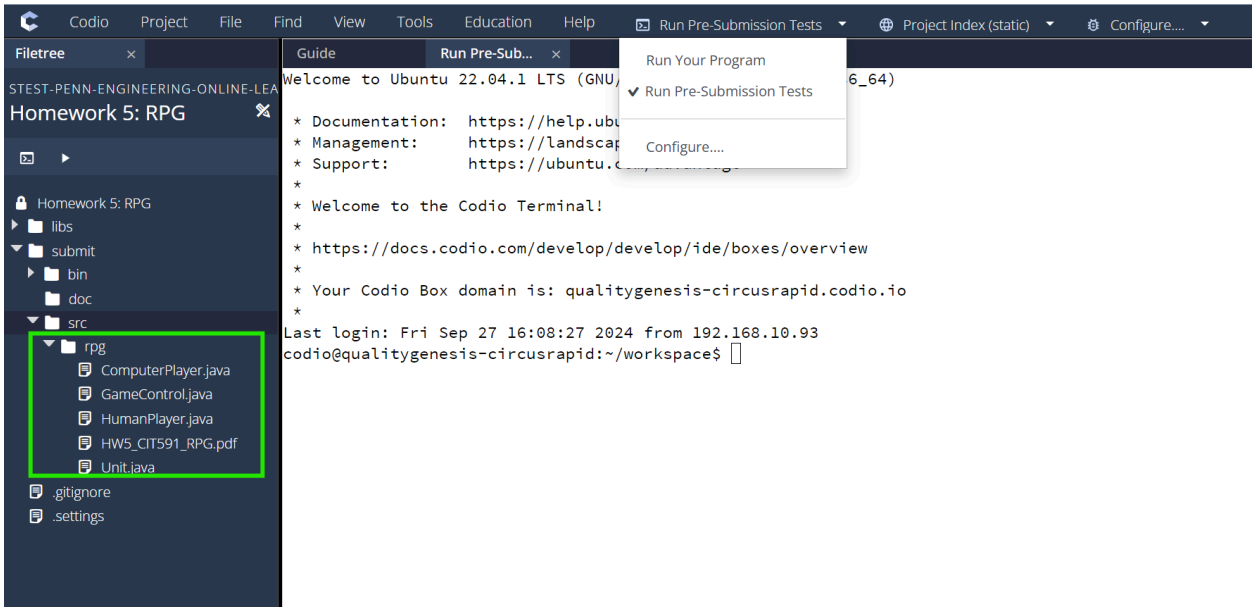
To test your code, **we have provided you with access to ALL of the unit tests for the autograded portion of this assignment**. Click "Test Submission" in Codio to run the tests against your code. Note that there will still be manually graded portions and passing all autograder tests does not guarantee getting full points on the manually graded section.

What to Submit

Please submit all the classes in your entire Java project. Make sure it includes everything in your "src" folder. **Do not remove the package declaration in each file and keep the entire program in the "rpg" subdirectory.**

Evaluation

1. Program correctness - 70 points (Autograded)
   This section will be fully autograded. All tests are available to you by choosing "Run Pre-Submission Tests." If there are indentation errors, syntax errors, or incorrect file names and signatures, the autograder will fail to run. Please read the error messages and resolve them, otherwise you will get a zero for the autograder portion. **You may need to scroll up the terminal window to see the error / failure messages**.



NOTE: Ensure you do a pre-submission tests check - we will not be accepting regrades for failures from the autograder.

2. Correctness of the game - 20 points (Manually Graded)
   This section is where we will evaluate your game flow as described in the instructions. This also includes calling in the appropriate required methods as needed (Example: When getting user input, invoke *get_user_input()* and not repeating those lines of code again.)

3. Style - 5 points (Manually Graded)
   Proper style means that your variable and method names are descriptive and related to the information they store and the camelcase naming convention is used. You should add in comments for all non-trivial lines of code. There will be no penalty for over-commenting, so if a particular line of code needs to be explained especially if it is doing some calculations or important flow control changes in your program, then put in comments on what that line of code is supposed to do. If you create your own methods, those should also be named descriptively and include comments and javadocs.

4. User Interface - 5 points (Manually Graded)
   As a general rule, a good user interface is easily understandable to users of this program who are not familiar with the assignment. This includes readability when printing information such as welcome and status messages (See the “User Interface Section”). The program should clearly indicate whose turn it is and what input is allowed from users when asking for user input. Since we are running the program only in the console, make sure the user can easily see the difference between different turns / steps in the game. Using sufficient whitespace will help with overall clarity.