Lab 2: Inheritance

Instruction

1. Click the provided link to automatically create your own repository for this lab
2. Using a git command to clone this repository to your local computer
3. Open Eclipse and then "import existing project" from the cloned folder (in Step2) into your Eclipse workspace
4. Rename an Eclipse Java project named "2110215_Lab2_2017_{ID}_{FIRSTNAME}" using this repository's root folder as the location of the project (Example: "2110215_Lab2_2017_5931234521_John")
5. Create the package "logic" and implement all the classes and methods following the details given
6. Check your codes with the provided JUnit Test Case
7. After implement all the class, create a UML Diagram using "ObjectAid" and put the result image (UML.png) in your project folder
8. After finishing the program, export your project into a runnable jar file called "Lab2_2017_{ID}.jar"
9. Commit and push it to your GitHub repository

Related concepts

- Git commands
- Using JUnit Test Case
- Class UML using ObjectAid
- Runnable JAR file
- Lecture2 concepts: class creation, inheritance

1. Problem Statement: Novice Arena

There is an event for novices to fight each other 1-1 on the arena. Each novice has 3 statuses: (1) HP (health point), (2) attack and (3) defense. For the game rule, each fighter takes turn doing their action until the opponent's HP reaches zero. The loser must retire from the event, while the winner will wait for the next opponent. Everyone expects a fair fight, but there may be some participants that pretend to be a novice and join this event. These pretenders can be either a mage or a summoner, which have extra skills that novices don't have. For each turn, the action of each character (novice, mage, summoner) is described below:

- Novice simply attacks their opponent**.** The opponent takes damage equal to the difference between the attacker's attack and the defender's defense**.**
- Mage can stop the time and attack twice per turn**.**
- Summoner can summon a monster, which can increase the attack and defense points of the summoner for 3 turns – this period is called "buff"; then it will disappear from the game. You must wait for 5 turns to summon a monster again – this is called "cooldown".

Write a program that simulates the arena and show the results of all the menu in the console. There are 4 input options as in the figure below (main menu).

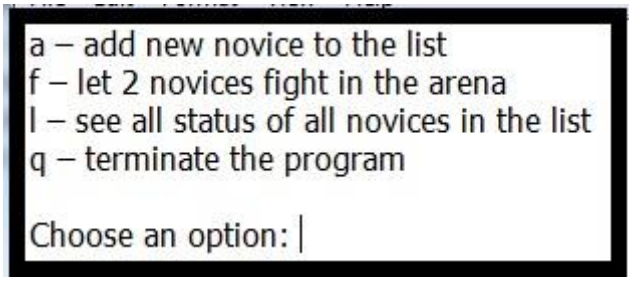


Figure 1 : Main menu

Create a character using the option 'a' in Figure1

When the user press 'a', the program should ask what class will you create (Figure 2). Type only number to choose. If the number is invalid, the program shows an error message "Wrong input" and return to main menu.

Next, if the selected class is valid, user must enter (1) a name and (2) 3 statuses including maxHp, attack and defense separated by "space" as shown in Figure 3.

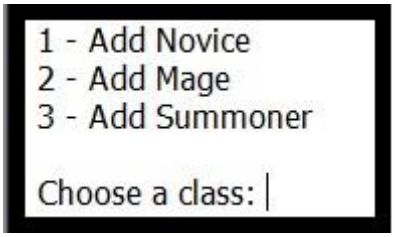


Figure 2 : Select a class for new novice

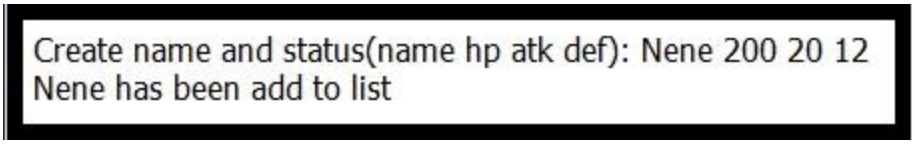


Figure 3 : Create name and status for new novice. Then add to list.

List all characters using the option 'l' in Figure1

When users presses 'l', the program will show all novices in the list (Figure 4) and return to main menu.

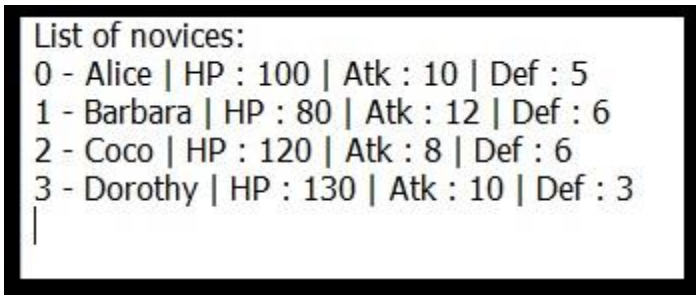


Figure 4 : Show all novices in the list.

Fight both characters using the option 'f' in Figure1

When user presses 'f', the program should show the list of all novices and ask the user to choose 2 novices to fight. If there is only 1 novice or less, the program should say "There is not enough novices to fight." and return to main menu.

Type only number shown in the list to select a character (novice). Suppose that two numbers are different. Example is shown in Figure 5. If number isn't in the list, the program should say "This novice isn't available." and return to main menu.

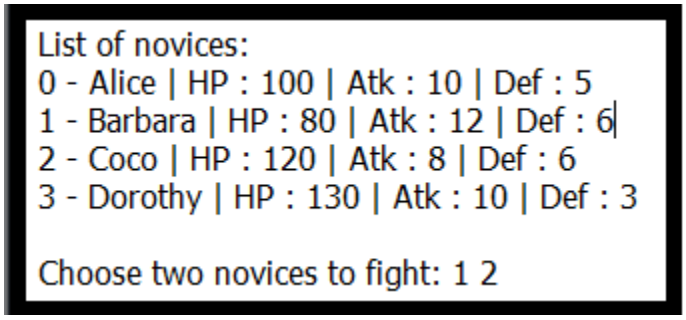


Figure 5 : Example of input when choosing two fighters

If those 2 numbers are valid, the program will simulate the fight. The result of each action must be shown in the console (Figure 6). When one of the fighter's HP reaches zero, the winner must be shown in the console (Figure 7) and then the program will remove the loser from the list, then return to main menu.

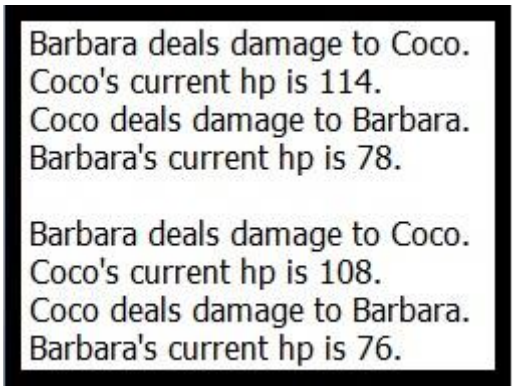


Figure 6 : Arena status

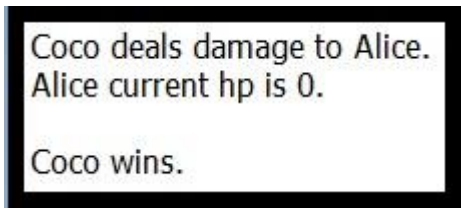
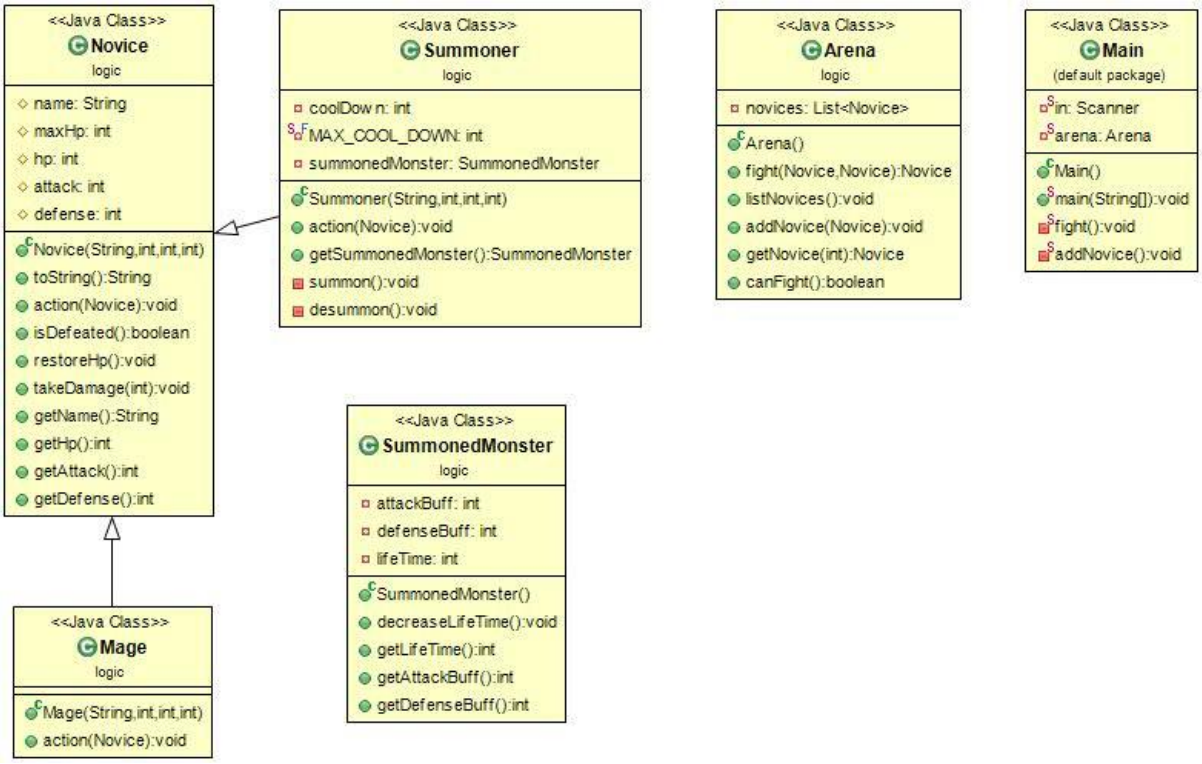


Figure 7 : Show the winner when the match end.

Quit the program using the option 'q' in Figure1

When user presses 'q', the program will stop.

2. Implement Details



2.1. Package logic

2.1.1. Class Novice

Field

- **protected String name;** Name of novice
- **protected int maxHp;** Maximum HP of novice
- **protected int hp;** Current HP of novice
- **protected int attack;** Attack points of novice
- **protected int defense;** Defense points of novice

Constructor

- **public Novice(String name, int maxHP, int attack, int defense);** Initialize name, maxHp, HP, attack and defense**.** Hp starts at the same as maxHp**.** If maxHp, attack or defense is less than 1, set it to 1.

Method

- **public String toString();** Return novice information as format "((name)) | HP **:** ((maxHp)) | Atk : ((attack)) | Def **:** ((defense))**".**
  - **(**Example**:** "Alice | HP **:** 100 | Atk **:** 10 | Def **:** 5"**)**

- **public void action(Novice opponent);** Opponent takes damage equal to the difference between this novice's attack and opponent defense.
- **public void takeDamage(Int damage);** Subtract current HP by damage. If damage is less than 1, set it to 1. After subtracting current HP, if it reaches below 0, set it to 0 (lose).
- **public boolean isDefeated();** Return true if current HP is not more than 0.
- **Public void restoreHp();** Set current HP to maxHp.
- **public String getName();** Getter of name.
- **public int getHp();** Getter of current HP.
- **public int getAttack();** Getter of attack points.
- **public int getDefense();** Getter of defense points.

2.1.2. Class Mage extends Novice

Constructor

- **public Mage(String name, int hp, int attack, int defense);** Initialize name, maxHp, hp, attack and defense. Hp starts at the same as maxHp. If maxHP, attack or defense is less than 1, set it to 1.

Method

- **public void action(Novice opponent);** Attack opponent. Note that mage can attack twice per turn.

2.1.3. Class SummonedMonster

Field

- **private int attackBuff;** An additional attack points for summoner.
- **private int defenseBuff;** An additional defense points for summoner.
- **private int lifeTime;** Number of turn(s) that this buff remains (status increasing period).

Constructor

- **public SummonedMonster();** Initialize attackBuff point to 8, defenseBuff point to 8 and lifeTime to 3.

Method

- **Public void decreaseLifeTime();** Decrease lifetime by 1**.**
- **Public getLifeTime();** Getter of lifeTime.
- **Public getAttackBuff();** Getter of attackBuff.
- **Public getDefenseBuff();** Getter of defenseBuff.

2.1.4. Class Summoner extends Novice

Field

- **private int coolDown;** Cooldown turn left before summoning a monster**.**
- **private static final int MAX_COOL_DOWN;** Number of max cooldown turn before summoning a monster**.** Set to 8**.**
- **private SummonedMonster summonedMonster;** A summoned monster**.**

Constructor

- **public Summoner(String name, int hp, int attack, int defense);** Initialize name, maxHp, hp, attack, defense and set coolDown to 0**.** Hp starts at the same as maxHp**.** If maxHP, attack or defense is less than 1, set it to 1.

Method

- **public void action(Novice opponent);** There are 4 steps to check every round for summoner**.**
  1. If there is a summoned monster, decrease life time of monster by 1**.** If life time is equal or less than 0, summoned monster will be removed.
  2. If there is no a summoned monster (null) and cooldown is 0, then this summoner can a monster**.**
  3. Take action. Hint you can use an action from its parent class.
  4. If cooldown is more than 0, then decrease by 1**.**

- **public SummonedMonster getSummonedMonster();** Getter of summonedMonster**.**
- **private void summon();** Summon a monster. Increase summoner's attack by summoned monster attackBuff and increase summoner's defense by summoned monster defenseBuff. Then set coolDown to max coolDown.
- **Private void desummon();** Subtract summoner's attackBuff and defenseBuff to original, then set summonedMonster to null.

2.1.5. Class Arena

Field

- **private List<Novice> novices;** List of novices in the arena**.**

Constructor

- **public Arena();** Initialize novices as an empty ArrayList**.**

Method

- **public Novice fight(Novice first, Novice second);** Let 2 novices fight. Returns the winner of the fight.
  - For each turn, each fighter takes their action**.** The first fighter takes an action first**.**
  - After each action, if the defender has lost the fight, return the winner**.**
  - Otherwise, the fight continues until one fighter loses**.** The loser must be removed from the list and restore the HP of the winner to maxHp.
- **public void listNovices();** Print all information of every novices in the list**.**
- **public void addNovice(Novice n);** Add new novice to the list**.**
- **public Novice getNovice(int number);** Return novice in the list corresponding to the number**.** If no valid novice in the list, return null.
- **public boolean canFight();** Return true if the number of novices in the list is 2 or more.

2.2. Default package

2.2.1. Class Main

Method

- **public static void main(String[] args);** Write the main menu of the program and call appropriate class corresponding to the menu.
- **private static void addNovice();** This method is called when user press “a”. It should ask a class to be created, then set name and status. If all input is valid, this new object will be added to the list in the arena.
- **private static void fight();** This method is called when user press “f”. It should show all novices in the list. If there are enough novices in the list, then it should ask user to choose 2 novices to fight. If the chosen novices are valid, it will simulate the fight.