SE116 – LAB#4 2024-2025 SPRING

Aim: Extending Classes (Inheritance), The Object Class, Annotations, Packages, Nested Classes

1. Write a Java Program to create an advanced role-playing game. Implement a `Character` hierarchy with following classes: `Character, Warrior, and Mage`. Also, implement a class `Player` to define the player who plays the game, a class `Party` to manage groups of characters, and a class `Battle` to simulate a battle in the game. Additionally, implement a nested class `Achievement` inside `Player` to track achievements. Finally, create a `Game` class to test different components of the game.

The class `Character` has the common information about a game character. These common data may be the following three `private` fields: String `name`, double `hitPoint` which is base damage a character can make, and String `gender`, `int level` (default = 1) and `int experience` which is the experience points needed to level up (default = 0). After defining set/get methods and non-parameterized/parameterized constructors, define the following methods as well: `calculateDamage, attack, regeneratePower, gainExperience(int xp), levelUp() and printInfo`. The method `calculateDamage` returns the value of `hitPoint`. The method `attack()` prints out "Attacking...Damage is: " and the damage value returned from the method `calculateDamage`. It will also allow characters to gain 20 points of xp. The method `regeneratePower` prints out "Regenerating Power". `gainExperience(int xp)` Increases experience and triggers level up if XP reaches 100. `levelUp()` increases level and resets experience. The method `printInfo` prints out the whole information of the character.

From the superclass `Character`, derive a subclass `Warrior` to represent a special type of a character which is a sword-using fighter. Thus, "`Warrior` **is a** `Character`". <u>Always keep in mind that a subclass inherits all the members from its superclass.</u> The class `Warrior` has extra `private` data members: `int energy` to be used to fight for determining if a warrior can attack (default = 20) and `int defense` used to reduce incoming damage (default = 5). The getter/setter methods are not needed for these data members. Create parameterized/non-parameterized constructors. Both constructors set `energy` to 20. Create a new `private` method `rest` that increases `energy` by 20 and prints out the updated `energy` value. Override the methods `calculateDamage, attack, regeneratePower`, and `printInfo`. The overridden method `regeneratePower` simply calls the `rest` method. The overridden method `calculateDamage` returns `hitPoint*1.2`. The overridden method `attack` prints out "Not enough energy. Get rest…" if `energy` is less than 10, otherwise the method first decreases `energy` by 10, then calls its parent's `attack` and finally prints out the remaining `energy`. The overridden method `printInfo` prints out the whole information of the warrior.

From the superclass `Character`, derive another subclass `Mage` to represent a special type of a character which uses magical powers to fight. Thus, "`Mage` **is a** `Character`". The class `Mage` has extra `private` data members: `int mana` that represents a spiritual power to be used to cast spells for fight and `double criticalChance` which determines the probability of a critical hit (default = 10%). In each attack, randomly decide whether `criticalChance` is exceeded by the character or not. If it is, then the damage is doubled. Create parameterized/non-parameterized constructors. In both constructors set `mana` to 10. Create a new `private` method `drinkPotion` that increases `mana` by 10 and prints out the updated `mana` value. Override the methods `calculateDamage, attack, regeneratePower`, and `printInfo`. The overridden method `regeneratePower` simply calls the `drinkPotion` method. The overridden method `calculateDamage` returns `hitPoint*0.8`. The overridden method `attack` prints out “Not enough mana. Drink potion…” if `mana` is less than 5, otherwise the method first decreases `mana` by 5, then calls its parent's `attack` and finally prints out the remaining `mana`. The overridden method `printInfo` prints out the whole information of the mage.

The class `Player` has the following three `private` data members: `String name`, `String password` and `ArrayList<Character> characters`. Thus, “`Player` **has a** `list of characters`”. Define set/get methods and non-parameterized/parameterized constructors. Define another method `printPlayerInfo` that takes no parameter, prints out the whole information of the player (including all character details) and returns no value. Also, create one other method that computes and returns the total damage of all characters the player has. The class `Player` will also include a nested class `Achievement`. It will keep track of unlocked achievements by the help of a list `List<String> unlockedAchievements`. Define a method `addAchievement()` that takes a parameter for achievement and add this to the list.

The class `Party` represents a group of characters working together. The class `Party` has the following attributes: `String partyName`, `ArrayList<Character> members`, `int powerBalance` which represents the total combined attack power of all members, `int reputation` which is the overall reputation of the party. Reputation of a party will be calculated by the summation of level values of each character in the party. Implement getters, setters, and both parameterized and non-parameterized constructors for these members. Define also the following methods: `addMember(Character character)` which adds a character to the party if the party has fewer than 10 members. Otherwise, print "Party is full!", `removeMember(Character character)`, `calculatePowerBalance()` which computes the total damage of all members, `calculateReputation()` which computes the reputation of all members, and `printPartyInfo()` which prints the details of the party, including members, power balance, and reputation.

The class `Battle` will simulate a battle between two parties. You will have two parties and two lists of characters as variables. These variables will be assigned to corresponding values by the parameterized constructor. `formTeams()` method will not get any parameters. It will shuffle the parties to select 3 random members from the party for assigning to battle teams. `startBattle()` method will start the battle and the battle will continue until one of the parties loses all its members. The fallen character during battle will be removed from the battle team. In each turn, the attacking side will be decided randomly. `attack()` method will get two parameters as characters and `hitPoint` of defender will be compared by the damage (taken from the `calculateDamage()` method) of the attacker. The difference will be returned to the attacker whether it is positive or negative. Finally, decide the winner by `declareWinner()` method according to surviving characters.

Finally, the class `Game` will contain the main method. Suppose that you have a game with players each owning various characters in the main method. To represent a game, use an instance from `ArrayList` of `Player`. Instantiate some `Player` objects each having a list of a few different characters of `Mage` and `Warrior`. Add the references of these `Player` objects into the `ArrayList`. Perform sample actions on the characters of the `Player` objects in the list. By the way, call the method `printPlayerInfo` before and after performing the actions. Create two different parties by `Party` class and start a battle between these parties. Finally, search the list of players to find and print out the `name` of the player whose characters' total damage is the highest. If there are duplicate results, the first one found may be used as the result of the search algorithm.