

Aim: Sample inheritance applications; IS-A relationship; HAS-A relationship.

TO DO @ LAB:

1. Write a Java Program to create a simple fantastic role-playing game. Implement a Character hierarchy with following classes: Character, Warrior, and Mage. Also, implement a class Test to test your game and characters.

The class Character has the common information about a game character. These common data may be the following three protected fields: String name, double hitPoint which is base damage a character can made, and String gender. After defining set/get methods and non-parameterized/parameterized constructors, define the following methods as well: calculateDamage, attack, regeneratePower, 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. The method regeneratePower prints out “Regenerating Power”. 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”. Always keep in mind that a subclass inherits all the members from its superclass. The class Warrior has an extra private data member: int energy to be used to fight. The getter/setter methods are not needed for energy. Create parameterized/non-parameterized constructors. In 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 an extra private data member: int mana that represents a spiritual power to be used to cast spells and fight. The getter/setter methods are not needed for mana. 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 Test will contain the method main. In main, construct an instance from Warrior and an instance from Mage. Then, simulate some actions (call the methods attack, regeneratePower, and printInfo) on these characters.

TO DO @ HOME:

2. Modify your project implemented in Question#1 in the following manner:

Change the access modifiers of the common data members in base class `Character` to `private`. Thus, hide the information of the superclass from other classes including its subclasses. So, you need to reconsider the method definitions in subclasses.

Implement another class `Player`. 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.

Modify the class `Test` in the following manner: Suppose that you have a game with players each owning various characters. In `main`, 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. 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.