RPG Character Design:
Using Inheritance

Objectives

- Implement a set of classes using inheritance principles that can represent different RPG (Role-Playing Game) characters and their unique skills.
- Implementation should utilize inheritance principles as discussed in class such as reducing code duplication, using protected variables, using abstract methods, using abstract classes, and designing proper sounding IS-A class relationships
- Test that the character classes are indeed receiving their proper skills by "asking" them to display their capabilities.

Background

You have been asked to work on a fantasy RPG video game. At this point, you are just getting the basic structures and classes in place in order to establish the foundation. You decide to use the concept of inheritance you learned in your computer science class.

Your job is to create six fantasy RPG character types: Knight, Barbarian, Ranger, Rogue, Cleric, and Wizard. Each will have a different capacity to avoid being hit (Armor Class) and a different capacity to withstand damage from being hit (Hit Points). In addition, each character type comes with certain abilities (some shared and some unique) including: the kind of armor they can wear, the type of weapons they can wield, fighting maneuvers, berserk raging, sneakiness, survival skills, lock-picking skills, spell-casting skills, healing skills, and problem-solving skills.

After writing your Java class hierarchy (trying to avoid code duplication wherever possible), you decide to create one of each character type in a driver application and "exercise" each one of the unique skills and print out the objects characteristics.

Tasks

1. Create six Java classes to represent the RPG character types using the following class names:
    - Knight
    - Barbarian
    - Ranger
    - Rogue
    - Cleric
    - Wizard
2. Using the following "character type to behavior/state table", design your class hierarchy accordingly using appropriate class property and class method names. HP and AC are object properties and the rest are behaviors. Utilize your knowledge of inheritance, protected variables, abstract classes, abstract methods, the **`super`** reference, and method overriding to create the best class hierarchy you can based upon the information given below. You will be expected to introduce new classes not mentioned in Step 1 to accomplish the desired solution. Each behavior should print an appropriate description to the console and the **`toString()`** should print out the character type description including the HP and AC values (see output below in Step 3). Do your best and have fun!

	armor	weapons	maneuvers	rages	sneaky	survival	lock-picking	spell-casting	healer	solver	HP	AC
Knight	heavy	martial	X								17	17
Barbarian	medium	martial		X							21	13
Ranger	medium	martial			X	X					13	15
Rogue	light	simple			X		X				11	16
Cleric	medium	simple						X	X		14	17
Wizard	none	staff						X		X	9	10

Appropriate Property Names: `hitPoints, armorClass`

Appropriate Behavior Names: `wearArmor(), useWeapon(), hasManeuvers(), tendsToRage(), sneaksAround(), willToSurvive(), pickslock(), castsSpell(), healsOthers(), solvesProblems()`

3. Create a program class called **`Adventure`** containing the **`main`** method. Within the **`main`** method, do the following:
   - Instantiate a character of each type and assign them to different variables of each type.
   - Print each character's description and "exercise" each method the class implements or inherits.
   - Your output should match the following:

```
I am a knight.
I have 17 HPs and 17 AC
I wear heavy armor!
I wield martial weapons against monsters.
I got some pretty cool fighting moves!

I am a barbarian.
I have 21 HPs and 13 AC
I wear medium armor!
I wield martial weapons against monsters.
When I get angry, I fight better!

I am a ranger.
I have 13 HPs and 15 AC
I wear medium armor!
I wield martial weapons against monsters.
I am very sneaky!
I will survive in the wild!

I am a rogue.
I have 11 HPs and 16 AC
I wear light armor!
I kill monsters with simple weapons.
I am very sneaky!
Picking locks is my specialty!

I am a cleric.
I have 14 HPs and 17 AC
I wear medium armor!
I kill monsters with simple weapons.
My real power is in my spells!
My friends rely upon my medical skills.

I am a wizard.
I have 9 HPs and 10 AC
I can hit monsters with my staff.
My real power is in my spells!
I am the best problem solver in the party!
```

4. Document your project:
    - Update the plain-text README file, following the format described below.
    - Comment your code appropriately; assume someone may be looking at your code and not know what you are trying to accomplish.

Documentation: README

Update the plain text file called README and write your name and class section on the top.

1. Fill out the **Overview** section
   Concisely explain what the program does. If this exceeds a couple of sentences, you are going too far. I do not want you to just cut and paste, but paraphrase what is stated in the project specification.
2. Fill out the **Compiling and Using** section
   This section should tell the user how to compile and run your code. It is also appropriate to instruct the user how to use your code. Does your program require user input? If so, what does your user need to know about it to use it as quickly as possible?
3. Fill out the **Discussion** section
   Think about and answer the following reflection questions as completely as you can. You may not have an “earth-shattering” reflection response to each one but you should be as thoughtful as you can.
   Reflections...
    - What problems did you have? What went well?
    - What process did you go through to create the program?
    - What did you have to research and learn that was new?
    - What kinds of errors did you get? How did you fix them?
    - What parts of the project did you find challenging?
    - Is there anything that finally "clicked" for you in the process of working on this code?
    - Is there anything that you would change about the lab?
    - Can you apply what you learned in this lab to future projects?
4. Fill out the **Testing** section
   You are expected to test your projects before submitting them for final grading. Pretend that your instructor is your manager or customer at work. How did you ensure that you are delivering a working solution to the requirements?

Grading

Points will be awarded according to the following breakdown:

Tasks	Points
Documentation: README file, comments	10
Appropriate class hierarchy utilizing abstract classes, abstract methods, etc.	30
Quality - code formatting, naming conventions, style, etc.	10

Required Files

Submit the following files within a zip file named: **Lab01_*LastName*_*FirstName*.zip**:

- **`README.txt`**
- **`Adventure.java`**
- **`Knight.java`**
- **`Barbarian.java`**
- **`Ranger.java`**
- **`Rogue.java`**
- **`Cleric.java`**
- **`Wizard.java`**
- Any other class needed in your class hierarchy