Changes:

1. Get rid of keyboard class, repacing it with System.in
    a. *choice = Keyboard.readInt();*
    b. This wrapper is now obsolete. The Scanner.in class is significantly easier to work with, requireing less user written code, and uses system functions, thus it is less prone to error.
2. Remove calls to Keyboard from entity classes, all user input will be handled by Dungeon.java
    a. *System.out.println("1. Attack Opponent");*
    *System.out.println("2. Surprise Attack");*
    *System.out.print("Choose an option: ");*
    *choice = Keyboard.readInt();*
    b. These subclasses should not know anything about user input. Having these classes handle input creates tighter coupling.
3. Dungeon character has unused compareTo and implements Comparable. Removing the superfluous code.
    a. *public int compareTo(Object o)*
    *{*
    *return 1;*
    *}*
    b. There is no need for the compareTo code, and keeping it there may cause confusion.
4. Hero gives a name to Dungeon Character (the player class name) and then promptly overwrites it with a user specified name. We have moved this input to Dungeon To keep the default name functionality, we added the name input to the Dungeon class. If the input is blank the default name is used via overloaded constructors.
    a. *public void readName()*
    *{*
    *System.out.print("Enter character name: ");*
    *name = Keyboard.readString();*
    *}//end readName method*
    b. This was confusing, as there was code that, in the end, did nothing. Unless there was no name sent in, where the default was set. Because this had to be done in the concrete classes, there was code that was the same in every hero.
5. Reduced the access to the class level variables in the Monster and Hero classes because there are no longer derived classes for these two classes, so the variables no longer need to be protected and they can be private.
    a. *protected double chanceToHeal;*
    *b. protected int minHeal, maxHeal;*
    *c. protected double chanceToBlock;*
6. Hero derived classes don't need to be there. A builder can create the unique behaviors for each hero. There is a lot of commonality between heroes so they can be pushed up

into the hero class and the hero class can just contain a couple more variables in order to add in the custom behaviors of each individual hero.
    a. *super("Warrior", 125, 4, .8, 35, 60, .2);*
    b. The code above shows that each hero class is in charge of setting its own stats, which isn't flexible. We put this into a builder class in order to change these values and create a warrior that has different stats but is still called a warrior.
7. Monster derived classes also don't need to be there. A builder can create the unique values for the monster class. The monsters can be pushed up into the Monster class for the same reason as the hero classes were pushed up.
    a. *super("Gnarltooth the Gremlin", 70, 5, .8, .4, 15, 30, 20, 40);*
    b. it is code like the example above that made me think of a builder class that will take care of building the object instead of letting the object build itself.
8. Character special attack/action is hardcoded into the classes themselves. Abstracting these means we can swap these out easily. Instead of using the functions below, we managed to extract the behavior from each function into separate classes that shared an interface (ISpecialMove). This allows for the special moves to switched out at runtime, allowing for greater functionality.
    a. *public void crushingBlow(DungeonCharacter opponent)*
    b. *public void surpriseAttack(DungeonCharacter opponent)*
    c. *public void increaseHitPoints()*
9. The 'theHero' variable within the chooseHero method inside of the Dungeon class is never used, therefore, we can get rid of it.
    a. *Hero theHero;*
    b. Having it there is just confusing for a developer because it makes it seem like it's there for a reason when it really isn't.
10. The pause variable in the battle method inside of the Dungeon class was a misleading variable name. I changed it to goAgain because that better explains what its purpose is because it is not pausing the gameplay but it is storing a value that determines whether or not the player wants to continue playing.
    a. *char pause = 'p';*
11. The methods in the Dungeon class: battle, playAgain, generateMonster and chooseHero all have public access modifiers. This is unnecessary and they all should be private because no other class needs to call any of the methods nor should they considering the Dungeon class is the tester class.

    The following are the function headers
    a. *public static Monster generateMonster()*
    b. *public static boolean playAgain()*
    c. *public static void battle(Hero theHero, Monster theMonster)*
    d. *public static Hero chooseHero()*