Refactors

1. Removed compareTo() and implementing comparable were unnecessary and dead code.

public int compareTo(Object o)

{

return 1;

}

1. Removed chooseHero() and generateMonster() methods from Dungeon class and opted for a factory system. Created CharacterFactory, HeroFactory and MonsterFactory.

public static Hero chooseHero()

{

int choice;

Hero theHero;

System.out.println("Choose a hero:\n" +

"1. Warrior\n" +

"2. Sorceress\n" +

"3. Thief");

choice = Keyboard.readInt();

switch(choice)

{

case 1: return new Warrior();

case 2: return new Sorceress();

case 3: return new Thief();

default: System.out.println("invalid choice, returning Thief");

return new Thief();

}//end switch

}//end chooseHero method

public static Monster generateMonster()

{

int choice;

choice = (int)(Math.random() \* 3) + 1;

switch(choice)

{

case 1: return new Ogre();

case 2: return new Gremlin();

case 3: return new Skeleton();

default: System.out.println("invalid choice, returning Skeleton");

return new Skeleton();

}//end switch

}//end generateMonster method

1. Member fields in DungeonCharacter, Monster and Hero made private and getters/setters created and used accordingly. (Not necessarily a code smell but just something I like to do to maintain encapsulation)

protected String name;

protected int hitPoints;

protected int attackSpeed;

protected double chanceToHit;

protected int damageMin, damageMax;

1. Split the battle method into multiple methods. runGame(), doBattle() and printResults(). These methods have the same functionality as before but this shortens the methods to make the code easier to read and understand.
   * Offending Code:

public static void battle(Hero theHero, Monster theMonster)

{

char pause = 'p';

System.out.println(theHero.getName() + " battles " +

theMonster.getName());

System.out.println("---------------------------------------------");

//do battle

while (theHero.isAlive() && theMonster.isAlive() && pause != 'q')

{

//hero goes first

theHero.battleChoices(theMonster);

//monster's turn (provided it's still alive!)

if (theMonster.isAlive())

theMonster.attack(theHero);

//let the player bail out if desired

System.out.print("\n-->q to quit, anything else to continue: ");

pause = Keyboard.readChar();

}//end battle loop

if (!theMonster.isAlive())

System.out.println(theHero.getName() + " was victorious!");

else if (!theHero.isAlive())

System.out.println(theHero.getName() + " was defeated :-(");

else//both are alive so user quit the game

System.out.println("Quitters never win ;-)");

}//end battle method

1. Removed the comments within the methods in Dungeon class. The code is easy enough to read without them cluttering things up.
2. Pushed battleChoices() method up to hero from the individual subclasses because all of them used similar structure besides individual specials that are implemented by an abstract method in Hero. An example of the offending code from one of the heroes (Warrior):

public void battleChoices(DungeonCharacter opponent)

{

int choice;

super.battleChoices(opponent);

do

{

System.out.println("1. Attack Opponent");

System.out.println("2. Crushing Blow on Opponent");

System.out.print("Choose an option: ");

choice = Keyboard.readInt();

switch (choice)

{

case 1: attack(opponent);

break;

case 2: crushingBlow(opponent);

break;

default:

System.out.println("invalid choice!");

}//end switch

numTurns--;

if (numTurns > 0)

System.out.println("Number of turns remaining is: " + numTurns);

} while(numTurns > 0);

}//end battleChoices method