Refactor List

Refactor #1: Get rid of the Keyboard class.

Since Java has Scanner now, the Keyboard class isn't needed. I replaced all of the Keyboard class calls with appropriate Scanner methods.

Ex:

```
public void readName()
 {
        System.out.print("Enter character name: ");
        name = Keyboard.readString();
 }//end readName method
```

Is now:

```
public void readName()
 {
        Scanner kb = new Scanner(System.in);
        System.out.print("Enter character name: ");
        name = kb.nextLine();
 }//end readName method
```

Refactor #2: Removed unneeded comments

There were several instances of comments stating the obvious, or stating the job of a variable/method, so I removed comments and renamed variables/methods to be intent-revealing.

Refactor #3: Remove Comparable interface on the DungeonCharacter class.

The overridden compareTo was not functional, and the game did not utilize DungeonCharacter comparison.

Before:

```
public abstract class DungeonCharacter implements Comparable
{

    protected String characterName;
    protected int healthPoints;
    protected int attackSpeed;
    protected double chanceToHit;
    protected int damageMin, damageMax;

    public int compareTo(Object o)
    {
        return 1;
    }
```

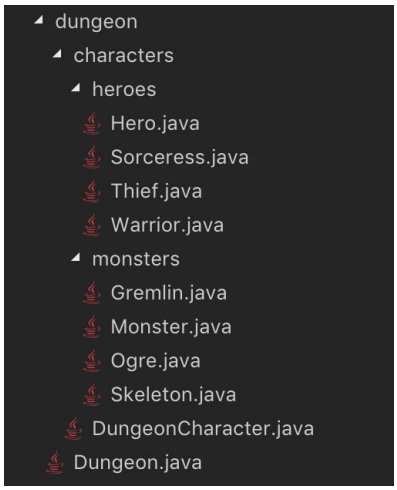
After:

```
public abstract class DungeonCharacter
{

    protected String characterName;
    protected int healthPoints;
    protected int attackSpeed;
    protected double chanceToHit;
    protected int damageMin, damageMax;
```

Refactor #4: Package Organization

I organized the project into packages, so as to be better organized. Before everything was in the root folder. Now it looks like this:



Refactor #5: Factories.
So as to create more cohesive code, and to make character creation more extensible, I added factories for both Heroes and Monsters.
HeroFactory Class:

```
public class HeroFactory {
   public Hero createHero(String type) {
       Hero newHero;
       type = type.toLowerCase();

       if (type.equals("sorceress")) {
           newHero = new Sorceress();
       } else if (type.equals("thief")) {
           newHero = new Thief();
       } else if (type.equals("warrior")) {
           newHero = new Warrior();
       } else {
           throw new IllegalArgumentException("Type " + type + " not found");
       }
       return newHero;
   }
}
```

The Dungeon class now uses this factory to create Heroes:

```
switch(choice)
{
    case 1: return hf.createHero("Warrior");

    case 2: return hf.createHero("Sorceress");

    case 3: return hf.createHero("Thief");

    default: System.out.println("invalid choice, returning Thief");
          return hf.createHero("Thief");
}//end switch
```

Instead of calling their constructors.
The same is true for Monsters and their creation.

Refactor #6: Dependency Inversion/Removal of duplicate code
The *battleChoices* method in the Hero class and its concrete implementations was messy. Each did nearly the same thing except for the “special” attack. I created an abstract method in hero called *specialAction* and called it from the *battleChoices* method in Hero. This means each concrete implimentation does not need its own *battleChoices* implimentation.
New generic *battleChoices* method:

```
public void battleChoices(DungeonCharacter opponent) {
        numTurns = attackSpeed / opponent.getAttackSpeed();

        if (numTurns == 0)
            numTurns++;

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

        int choice;

        Scanner kb = new Scanner(System.in);

        do
        {
          System.out.println("1. Attack Opponent");
          System.out.println("2. " + specialActionName);
          System.out.print("Choose an option: ");
          choice = Integer.parseInt(kb.nextLine());

          switch (choice)
          {
                  case 1: attack(opponent);
                      break;
                  case 2: specialAction(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
```