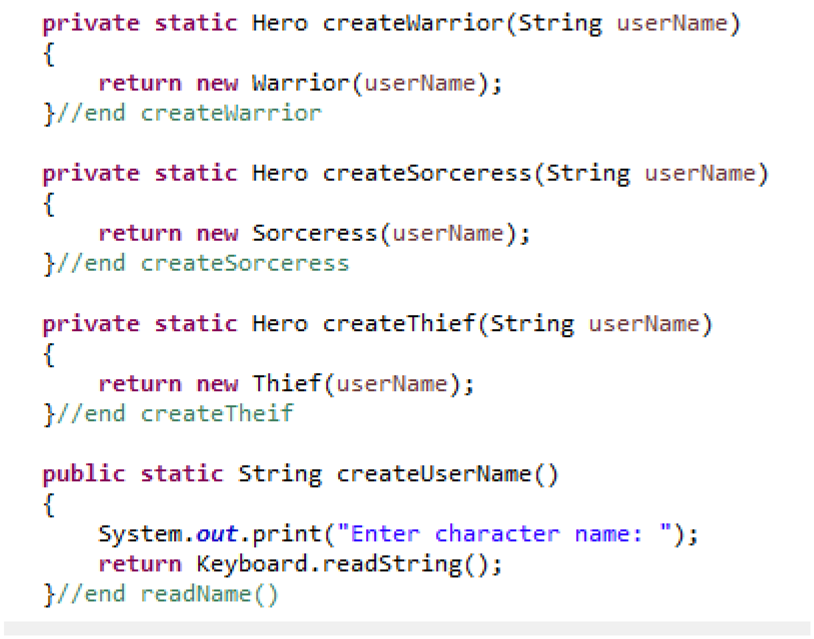
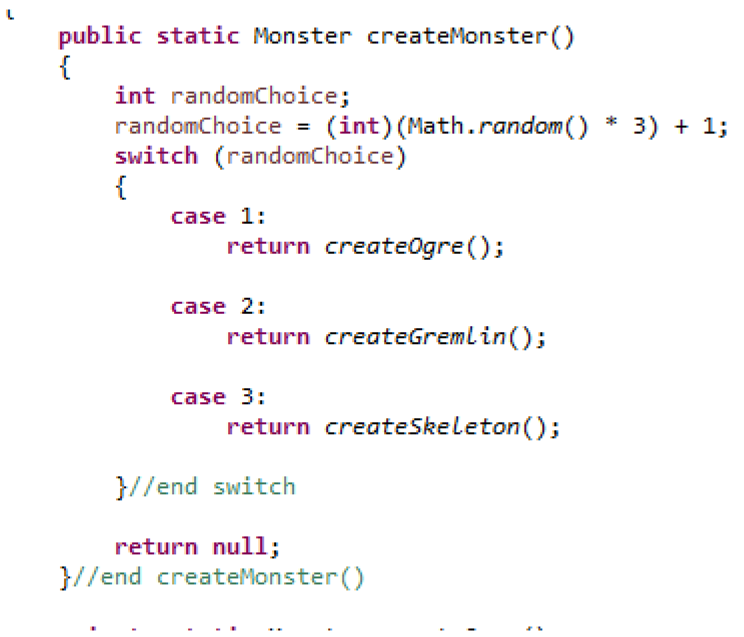
1. Created HeroFactory



```
private static Hero createWarrior(String userName)
{
    return new Warrior(userName);
}//end createWarrior

private static Hero createSorceress(String userName)
{
    return new Sorceress(userName);
}//end createSorceress

private static Hero createThief(String userName)
{
    return new Thief(userName);
}//end createTheif

public static String createUserName()
{
    System.out.print("Enter character name: ");
    return Keyboard.readString();
}//end readName()
```

Created to handle the creation of heroes and limit outside visibility

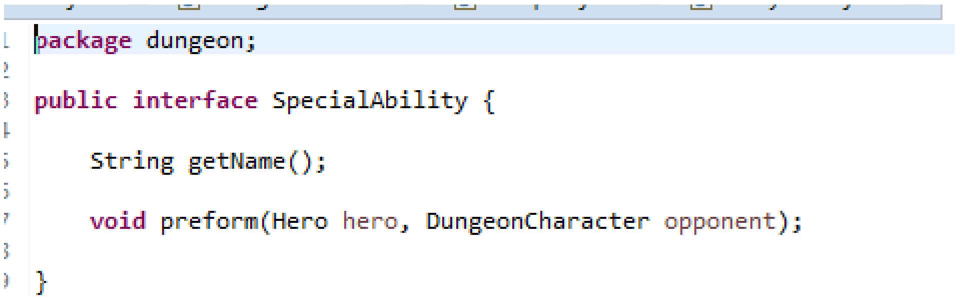
2. Created MonsterFactory



```
public static Monster createMonster()
{
    int randomChoice;
    randomChoice = (int)(Math.random() * 3) + 1;
    switch (randomChoice)
    {
        case 1:
            return createOgre();

        case 2:
            return createGremlin();

        case 3:
            return createSkeleton();

    }//end switch

    return null;
}//end createMonster()
```

Created to handle the creation of monsters and limit outside visibility

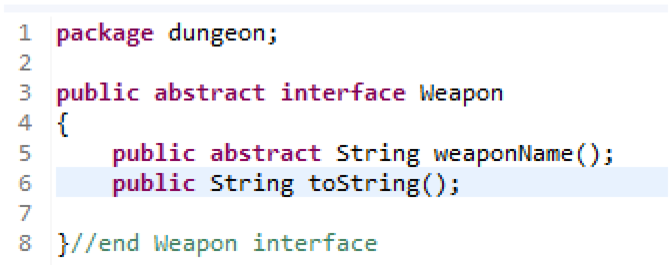
3. Created SpecialAblity Interface



```java
package dungeon;

public interface SpecialAbility {

    String getName();

    void preform(Hero hero, DungeonCharacter opponent);

}
```

Allowed for the moving of the battle choice to the hero class to remove duplicate code from the subclasses

4. Created Weapon Interface



```java
package dungeon;

public abstract interface Weapon
{
    public abstract String weaponName();
    public String toString();

}//end Weapon interface
```

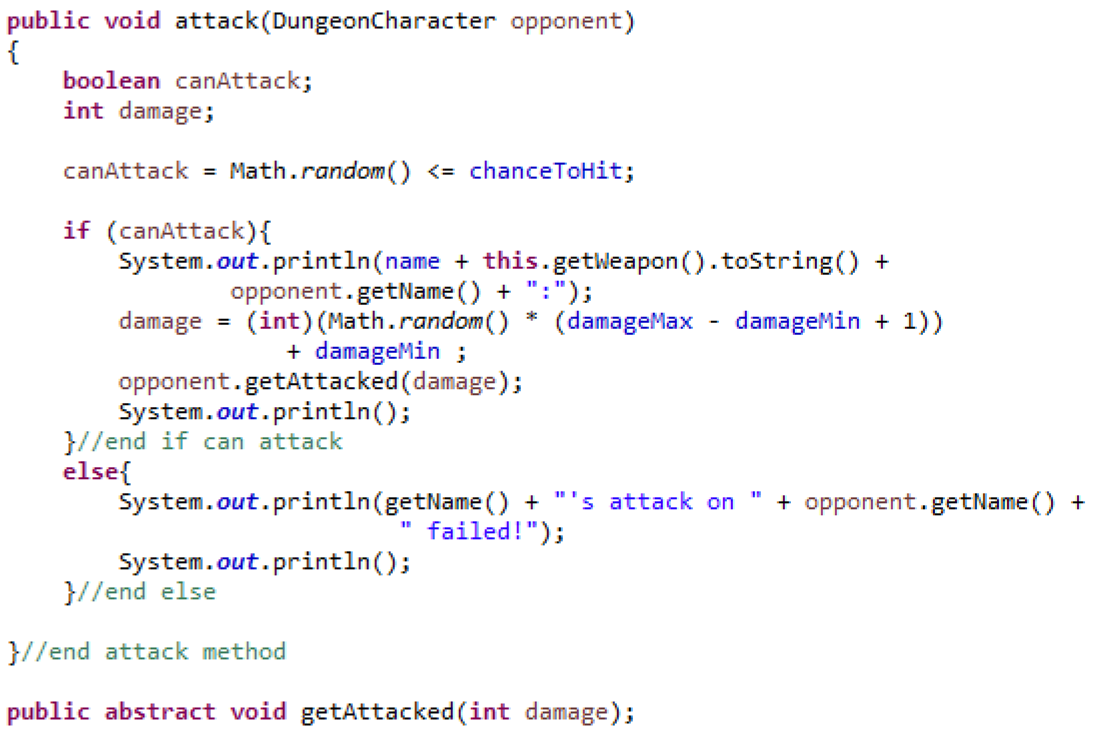
Allowed for the moving of attack to the dungeon character class to remove duplicate code from the subclasses. Also, allowed for the cleaning of the Attack method.

5. Cleaned up un-needed comments.

6. Updated attack method, only DungeonCharcter has one now

```
public void attack(DungeonCharacter opponent)
{
        System.out.println(name + " swings a mighty sword at " +
                                                opponent.getName() + ":");
        super.attack(opponent);
}//end override of attack method
```

OLD from Warrior

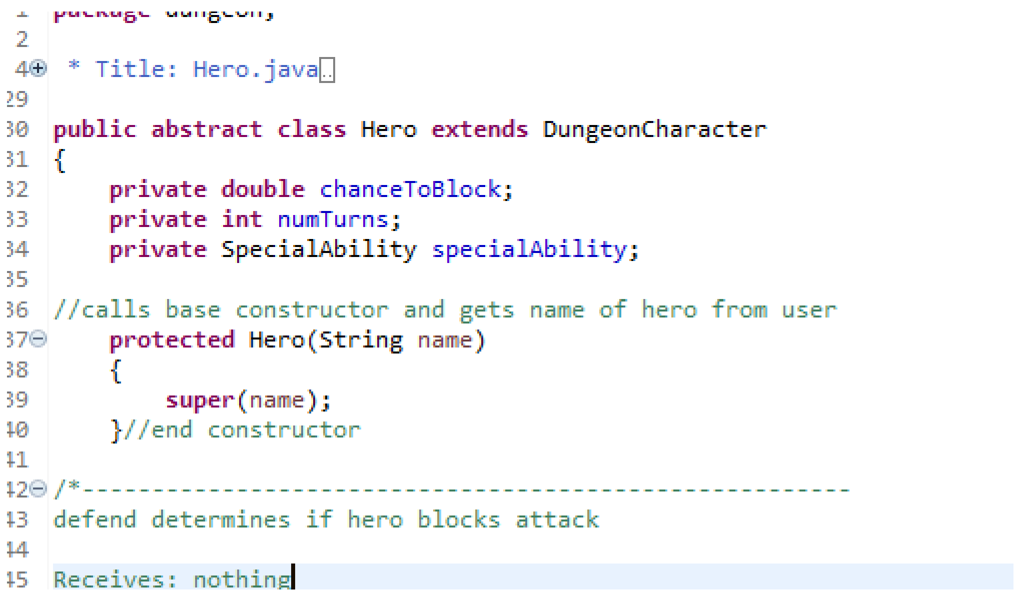


```
public void attack(DungeonCharacter opponent)
{
    boolean canAttack;
    int damage;

    canAttack = Math.random() <= chanceToHit;

    if (canAttack){
        System.out.println(name + this.getWeapon().toString() +
                opponent.getName() + ":");
        damage = (int)(Math.random() * (damageMax - damageMin + 1))
                  + damageMin ;
        opponent.getAttacked(damage);
        System.out.println();
    }//end if can attack
    else{
        System.out.println(getName() + "'s attack on " + opponent.getName() +
                          " failed!");
        System.out.println();
    }//end else

}//end attack method

public abstract void getAttacked(int damage);
```

NEW

As stated above we created the weapon class to allow for this clean up. We were able to move this method to the Super class and remove Overridden methods in the sub classes.

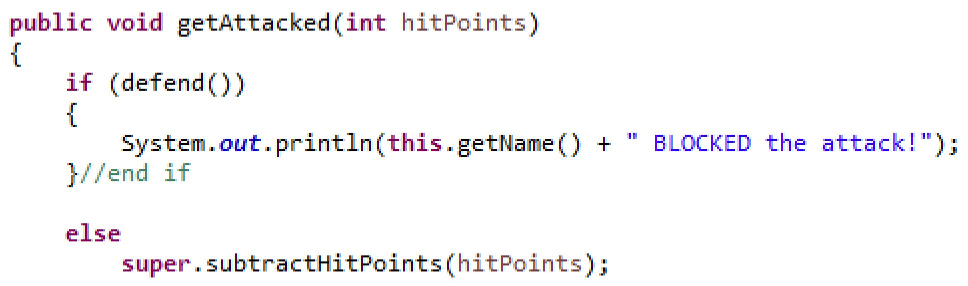
7. Changed visibility on classes and fields to package and private



```
package dungeon;

 * Title: Hero.java

public abstract class Hero extends DungeonCharacter
{
    private double chanceToBlock;
    private int numTurns;
    private SpecialAbility specialAbility;

//calls base constructor and gets name of hero from user
    protected Hero(String name)
    {
        super(name);
    }//end constructor

/*-----------------------------------------------------------
defend determines if hero blocks attack

Receives: nothing
```

Changed all fields to private and created getters and setters as needed. We also changed constructors to package or protected visibility as made sense

8. Changed methods in Hero/Monster classes to have a more intent revealing name



```
public void getAttacked(int hitPoints)
{
    if (defend())
    {
        System.out.println(this.getName() + " BLOCKED the attack!");
    }//end if

    else
        super.subtractHitPoints(hitPoints);
```

Changed SubtractHitpoints in the child classes to getAttacked, also created the method as an abstract method in the parent class to ensure future classes will have this field. Named fields with names that reveal intent. Simplified the constructors by only passing in a name.

9. Remove comparable & compareTo from DungeonCharacter

Removed unimplemented code and dead code