

Strategy HW02: Full Implementation

Assignment Goals

The goal of this assignment is to:

1. Put together the ideas from the previous two home works
2. Get comfortable with the idea of refactoring
3. Develop a better understanding of the Strategy pattern

Table of Contents

Assignment Goals.....	<i>i</i>
Figures.....	<i>i</i>
Deliverables.....	<i>1</i>
Structural Diagrams.....	<i>2</i>
Project Layout.....	<i>2</i>
Monster Package Structure	<i>3</i>
Abilities Package Structure	<i>4</i>
Monster Package.....	<i>5</i>
Details of Monster	<i>6</i>
Details of the Imp and Kobold classes	<i>9</i>
Ability Package.....	<i>10</i>
Details of MeleeAttack.java	<i>10</i>
The Driver class and the final output.....	<i>11</i>

Figures

Figure 1: Homework Layout	<i>2</i>
Figure 2: Adding a Package.....	<i>2</i>
Figure 3: Monster Diagram.....	<i>3</i>
Figure 4: Abilities Package Structure	<i>4</i>
Figure 5: Monster Class Structure	<i>6</i>
Figure 6: Details of the getAttribute method	<i>7</i>
Figure 7 The completed Imp class	<i>9</i>
Figure 8: Details of MeleeAttack.java	<i>10</i>
Figure 9: The Driver.java class	<i>11</i>
Figure 10: The correct output from the Driver file.....	<i>11</i>

Deliverables

1. Take a screenshot of the output and include it in a zip file
2. Turn in a zip file with all seven classes implemented.

Structural Diagrams

Project Layout

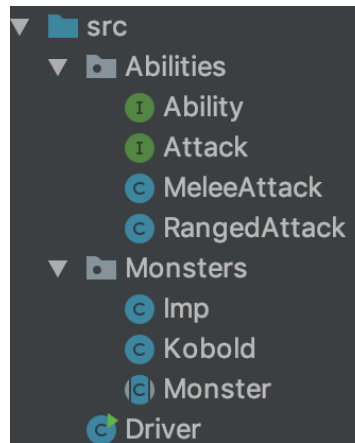


Figure 1: Homework Layout

For this homework I have rearranged the structure a little bit. I have created two packages, Abilities and Monsters.

This was done as follows:

- right click on the src folder
- selecting new
- clicking on package.

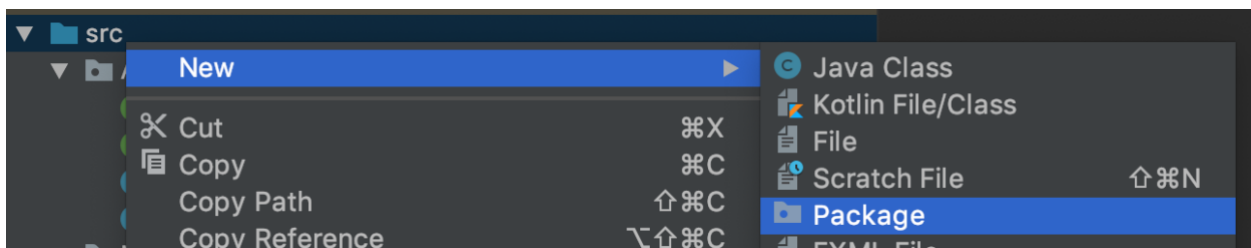


Figure 2: Adding a Package

Once the packages are created, the files may be dragged into them and IntelliJ will refactor as needed.

Within the packages some of the classes have changed. The following diagrams show the overall change, and the details are recorded in the sections of this document specific to each package.

Monster Package Structure

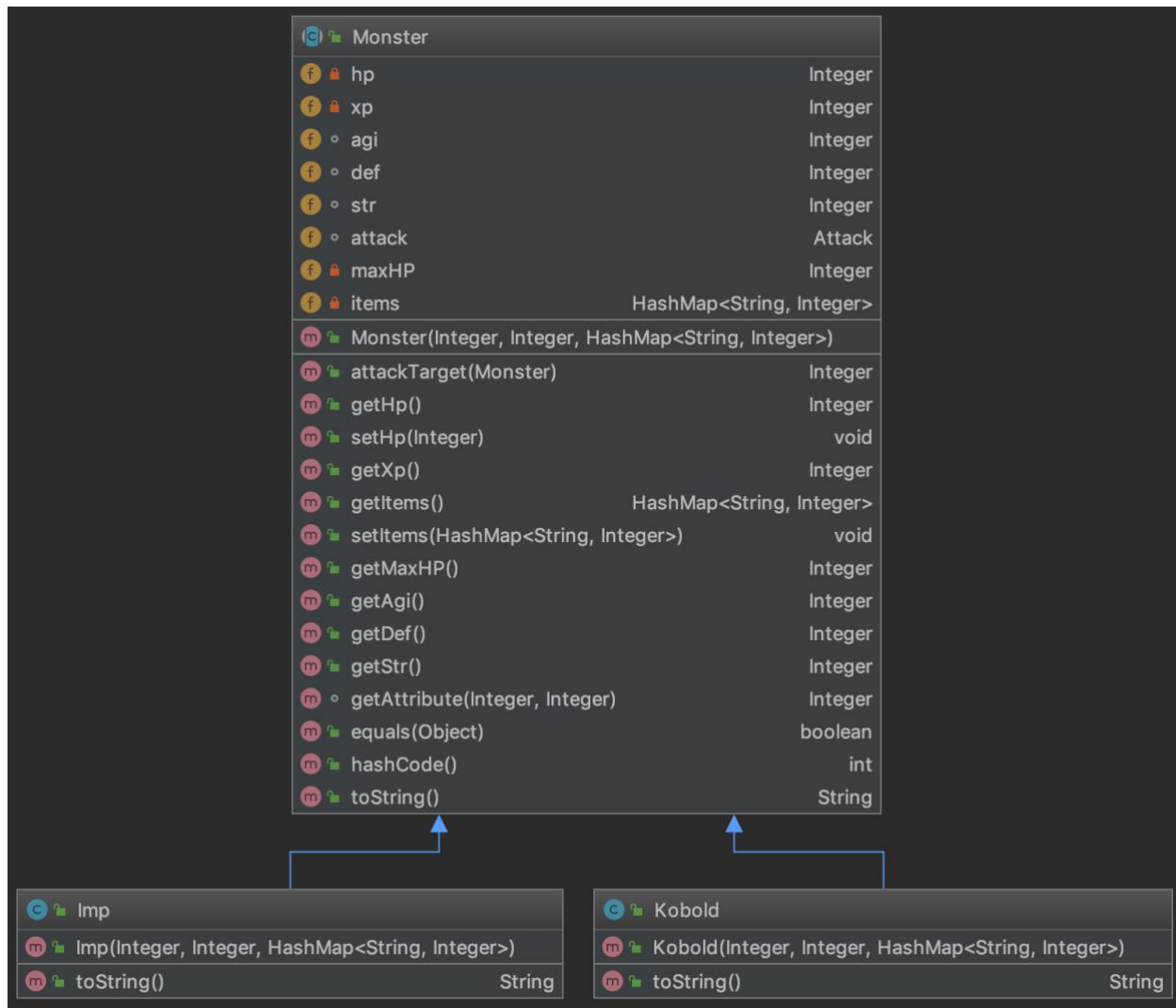
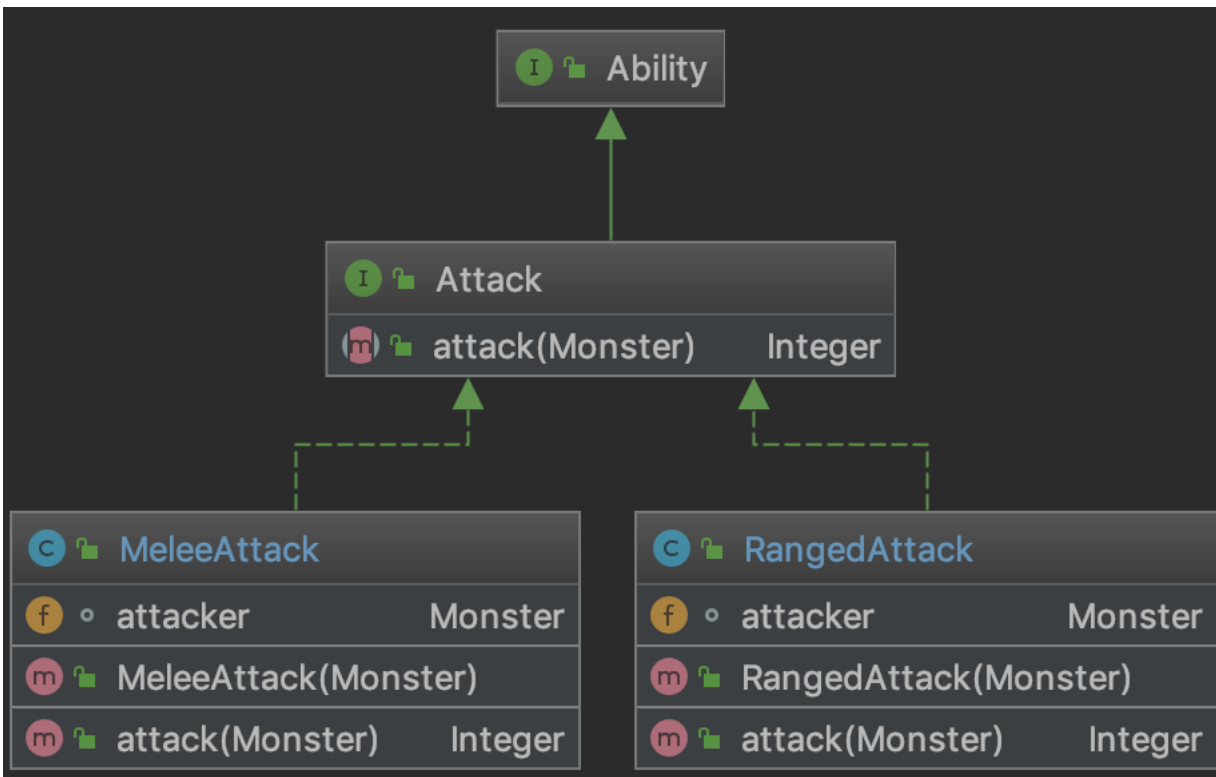


Figure 3: Monster Diagram

Several fields have been added to Monster these are detailed below.

In addition to adding the fields the `equals()` and `hashCode()` methods have been updated as well.

Abilities Package Structure

*Figure 4: Abilities Package Structure*

The structure of the Abilities package has not changed.

Monster Package

The monster package has seen the most change. `Monster.java` has had four fields and one method added. The individual implementations of `Monster`, `Imp.java` and `Kobold.java`, have both had details added to their constructors.

Details of Monster

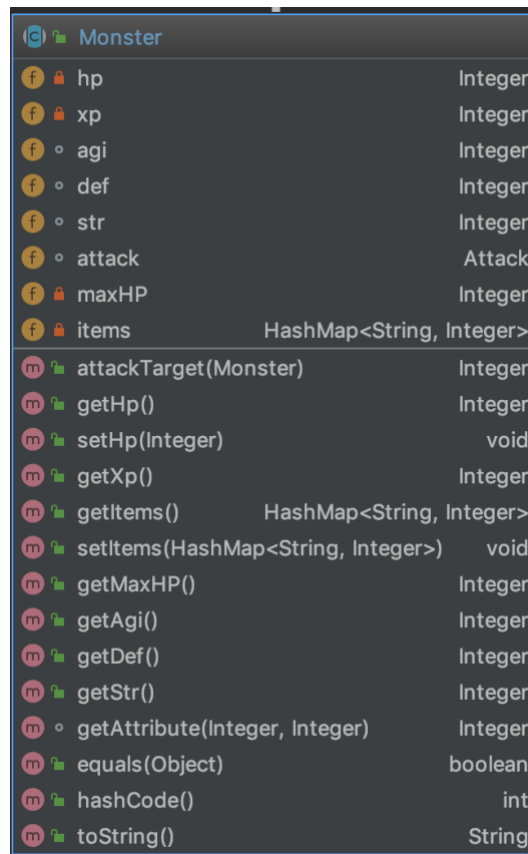


Figure 5: Monster Class Structure

Four new package private fields have been added to the monster class. These are detailed below. For now, these values will be set in the specific implementations of Monster. In general, to prevent repeated code, it would be better to implement methods to set these in the super class, Monster, and call that class when the individual extensions are instantiated.

Field : type	Access Level	Setter?	Getter?	Default Value
agi : Integer	Package Private	no	yes	5
def : Integer	Package Private	no	yes	5
str: Integer	Package Private	no	yes	5
attack : Attack	Package Private	no	yes	

Table 1: Details of the new fields in Monster

In addition to these fields one new method has been added
 getAttribute(Integer min, Integer max)
 Get attribute is used for setting the str, def, and agi values.

NOTE: attack is of type Attack.java

```
/**
 * This methods returns an integer value between min and max.
 * This is goofy. rand.nextInt(n) returns a number between 0-n
 * to get the value we want, a value between str - maxStr,
 * We need to get a random number from maxStr-str and
 * add str back in.
 * @param min an integer
 * @param max an integer
 * @return a random integer between min and max
 */
Integer getAttribute(Integer min, Integer max){
    Random rand = new Random();
    if(min > max){
        Integer temp = min;
        min = max;
        max = temp;
    }
    //returns a random number between min and max inclusive
    return rand.nextInt(max-min) + min;
}
```

Figure 6: Details of the getAttribute method

The comments should explain what this method does.

attackTarget(Monster target)

This method returns the results of the attack method of the Attack object of the current monster on the target.

That is to say that Monster.java has a field of type Attack.java. Attack.java has a method named attack(Monster target). This new method, attackTarget(Monster target), uses the attack *method* of the Attack object assigned to the current monster on the target that was passed in to the method attackTarget(Monster).

Details of the Imp and Kobold classes

```
public class Imp extends Monster {  
  
    public Imp(Integer maxHP, Integer xp, HashMap<String, Integer> items){  
        super(maxHP, xp, items);  
        //These should be stored in a HashMap  
        // that way we can use an iterator.  
        Integer maxStr = 15;  
        Integer maxDef = 6;  
        Integer maxAgi = 3;  
  
        attack = new MeleeAttack(this);  
        //this should use a data structure  
        str = super.getAttribute(str, maxStr);  
        def = super.getAttribute(def, maxDef);  
        agi = super.getAttribute(agi, maxAgi);  
    }  
  
    @Override  
    public String toString() {  
        return "Monsters.Imp has : " + super.toString();  
    }  
}
```

Figure 7 The completed Imp class

The Imp and Kobold class take advantage of the new fields declared in Monster. To make our code more extensible it would make sense to include methods for setting str, def, and agi in Monster. In a future version these will be placed in either a HashMap or an inner class.

Ability Package

The ability. package has not been changed much with the exception of more detail added to the specific definitions of attack.

Details of MeleeAttack.java

```
public class MeleeAttack implements Attack {  
  
    Monster attacker;  
  
    public MeleeAttack(Monster attacker) {  
        this.attacker = attacker;  
    }  
  
    @Override  
    public Integer attack(Monster target) {  
        String message = attacker + " uses a melee attack on " + target;  
        System.out.println(message);  
        return attacker.getStr() - target.getDef();  
    }  
}
```

Figure 8: Details of MeleeAttack.java

MeleeAttack.java and RangedAttack.java are, again, almost identical. The only place they differ are in the messages that are printed and the values that are returned.

MeleeAttack will return the difference of the attackers str and the targets def.

RangedAttack will return the difference of the attackers agi and the targets agi.

The Driver class and the final output

```
public class Driver {  
    public static void main(String[] args){  
        HashMap<String, Integer> items = new HashMap<>();  
        items.put("gold",5);  
        List<Monster> monsters = new ArrayList<>();  
        monsters.add(new Imp(15,20,items));  
        monsters.add(new Kobold(1,5,items));  
  
        for (Monster m : monsters){  
            System.out.println(m);  
        }  
  
        System.out.println(monsters.get(0).attackTarget(monsters.get(1)));  
        System.out.println(monsters.get(1).attackTarget(monsters.get(0)));  
    }  
}
```

Figure 9: The Driver.java class

The driver class is detailed in Figure 9

The driver has been updated to include

The output of the Driver class is shown in Figure 10. Note that the numbers shown after the attacks picture below, 9 and 3 respectively, are random. These are the results of the 'attack roll' and would be used to calculate a hit.

```
Monsters.Imp has : hp=15/15  
Monsters.Kobold has : hp=1/1  
Monsters.Imp has : hp=15/15 uses a melee attack on Monsters.Kobold has : hp=1/1  
9  
Monsters.Kobold has : hp=1/1 uses a ranged attack on Monsters.Imp has : hp=15/15  
3
```

Figure 10: The correct output from the Driver file.