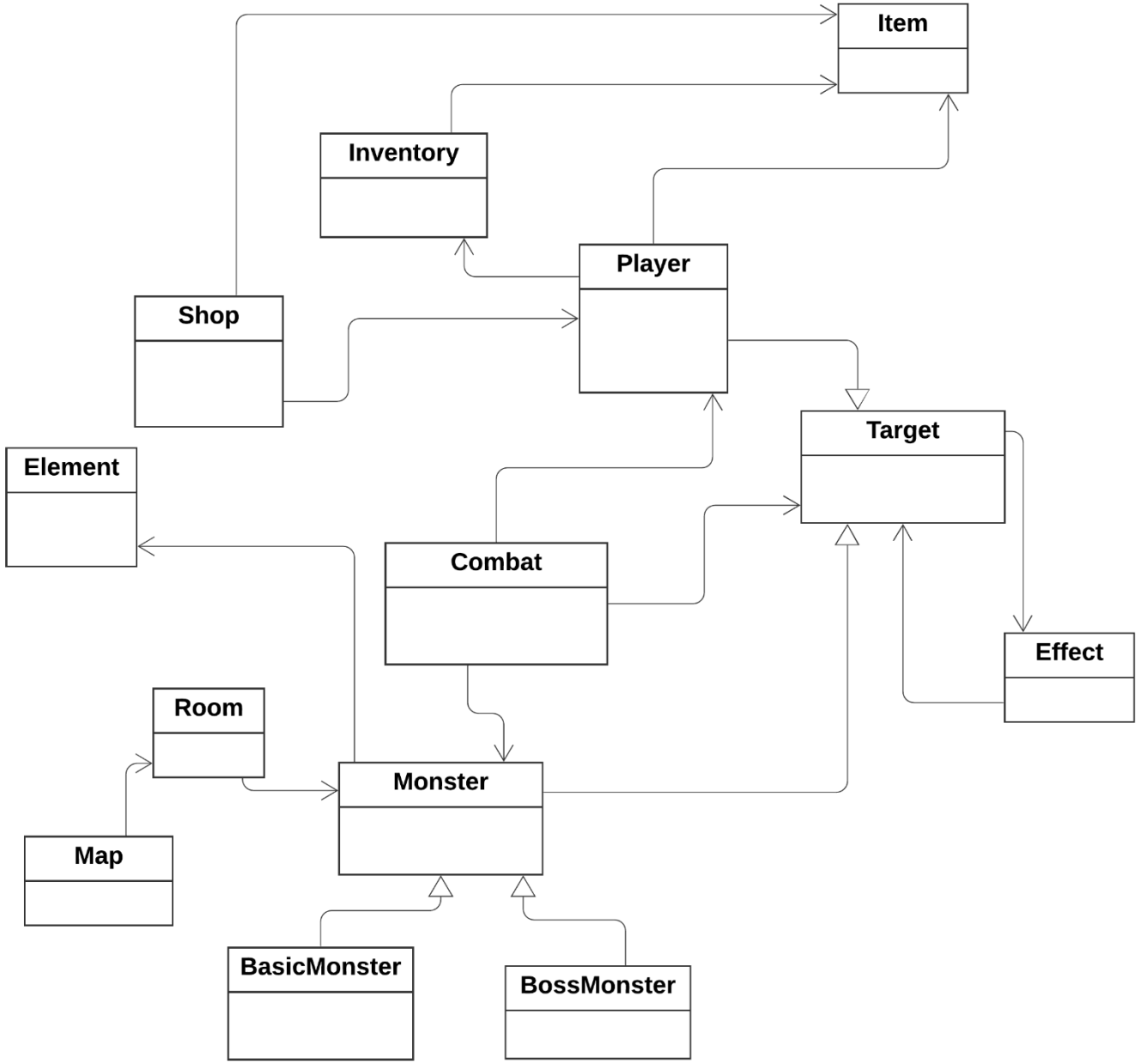
Grupp nr: 11

- Bjargey Ingólfsdóttir, bjin3492@student.su.se
- Linnéa Palmgren, lipa7972@student.su.se
- Karl Gustafsson, kagu9654@student.su.se
- Magnus Palmstierna, mapa7956@student.su.se
- Moa Hoffström, moho8130@student.su.se

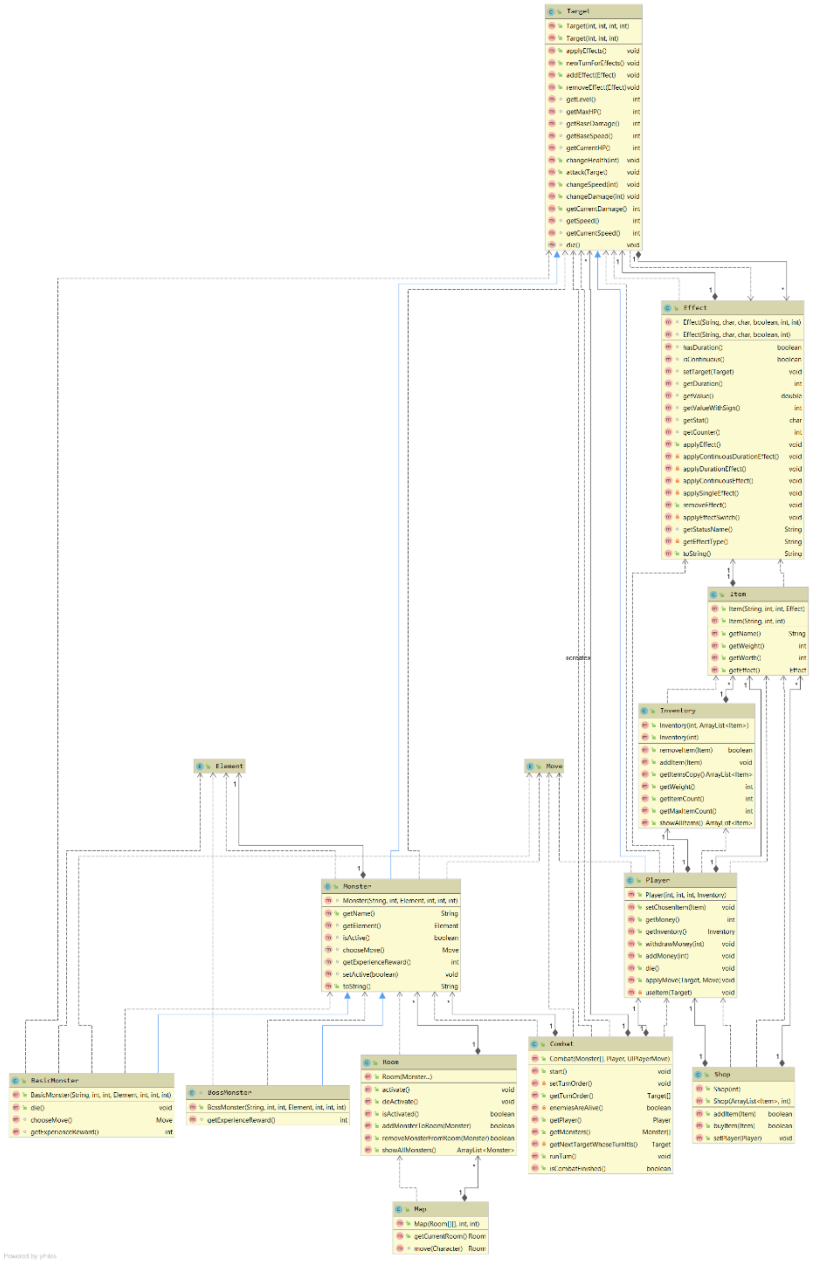
Verktyg

- Vilka verktyg använde ni?
- JUnit
- Ant
- Java Flight Recorder (IntelliJ Plugin)
- CodeMR (IntelliJ Plugin)
- Statistic (IntelliJ Plugin)
- MetricsReloaded (IntelliJ Plugin)

Slutlig design



Slutlig design



TDD-exempel: Karl Gustafsson

Testkod

```
7   class MapTest {
8       Monster anka = new BasicMonster("Anka", 3, 1, Element.FIRE, 4, 2, 1);
9       Room room00 = new Room(anka);
10      Room room01 = new Room(anka);
11      Room room02 = new Room(anka);
12      Room room10 = new Room(anka);
13      Room room20 = new Room(anka);
14      Room room11 = new Room(anka);
15      Room room12 = new Room(anka);
16      Room room21 = new Room(anka);
17      Room room22 = new Room(anka);
18
19      Room[][] roomArray = {{room00, room01, room02}, {room10, room11, room12}, {room20, room21, room22}};
20      Map map = new Map(roomArray, 1, 1);
21
22      @Test
23      void testGetCurrentRoom(){
24          assertEquals(room11, map.getCurrentRoom());
25      }
26
27      @Test
28      void testMoveUp(){
29          assertTrue(map.move('u') == room12);
30      }
31
32      @Test
33      void testMoveDown(){
34          assertTrue(map.move('d') == room10);
35      }
36
37      @Test
38      void testMoveLeft(){
39          assertTrue(map.move('l') == room01);
40      }
41
42      @Test
43      void testMoveRight(){
44          assertTrue(map.move('r') == room21);
45      }
```

Koden som testas

```
20      Room move(Character dir) throws IllegalArgumentException, ArrayIndexOutOfBoundsException {
21          Room newRoom;
22          switch (dir){
23              case 'u':
24                  try{
25                      newRoom = map[playerCordX][playerCordY + 1];
26                      playerCordY += 1;
27                      return newRoom;
28                  } catch (ArrayIndexOutOfBoundsException e){
29                      return null;
30                  }
31              case 'd':
32                  try{
33                      newRoom = map[playerCordX][playerCordY - 1];
34                      playerCordY -= 1;
35                      return newRoom;
36                  } catch (ArrayIndexOutOfBoundsException e){
37                      return null;
38                  }
39              case 'r':
40                  try{
41                      newRoom = map[playerCordX + 1][playerCordY];
42                      playerCordX += 1;
43                      return newRoom;
44                  } catch (ArrayIndexOutOfBoundsException e){
45                      return null;
46                  }
47              case 'l':
48                  try {
49                      newRoom = map[playerCordX - 1][playerCordY];
50                      playerCordX -= 1;
51                      return newRoom;
52                  } catch (ArrayIndexOutOfBoundsException e){
53                  return null;
54              }
55          }
56          throw new IllegalArgumentException("only: u, d, r, l are valid inputs.");
57      }
58  }
```

TDD-exempel: Karl Gustafsson

Testkod

```
47      @Test
48      void testMoveUpNull(){
49          map.move('u');
50          assertTrue(map.move('u') == null);
51      }
52
53      @Test
54      void testMoveDownNull(){
55          map.move('d');
56          assertTrue(map.move('d') == null);
57      }
58
59
60      @Test
61      void testMoveLeftNull(){
62          map.move('l');
63          assertTrue(map.move('l') == null);
64      }
65
66
67      @Test
68      void testMoveRightNull(){
69          map.move('r');
70          assertTrue(map.move('r') == null);
71      }
72
73      @Test
74      void testIllegalMove(){
75          assertThrows(IllegalArgumentException.class, () -> map.move('k'));
76      }
77  }
```

Koden som testas

```
3   public class Map{
4       private int playerCordX;
5       private int playerCordY;
6
7       private Room[][] map;
8
9       public Map(Room[][] map, int playerCordX, int playerCordY){
10          this.map = map;
11          this.playerCordX = playerCordX;
12          this.playerCordY = playerCordY;
13      }
14
15
16      public Room getCurrentRoom(){
17          return map[playerCordX][playerCordY];
18      }
```

TDD-exempel: Magnus Palmstierna

Testkod

```
  import org.junit.jupiter.api.Test;

+ import java.util.ArrayList;
  import java.util.HashMap;
  import static org.junit.jupiter.api.Assertions.*;

  class CombatTest {
-     Player player = new Player(1,12);
+     Player player = new Player(1,12, 2, new Inventory(10, new ArrayList<Item>()));
      Monster spider = new BasicMonster("Spider", 5, 5, Element.FIRE, 10, 2, 5, 2);
      Monster[] monsters = {spider};
      Combat validCombat = new Combat(monsters, player);
-     validCombat.initialize();

      //Check that fetched player is not null
      @Test
@@ -30,8 +30,8 @@ void checkMonsterArray(){
      //Check that turn order is correct
      @Test
      void checkTurnOrder(){
-         HashMap<Integer, Object> turnOrder = validCombat.getOrder();
+         Target[] turnOrder = validCombat.getTurnOrder();
          //SKAPA EQUALS FÖR PLAYER
-         AssertTrue(turnOrder.get(1).getSpeed() > turnOrder.get(2).getSpeed());
+         assertTrue(turnOrder[0].getSpeed() >= turnOrder[1].getSpeed());
      }
  }
```

Koden som testas

```
1  1  package com.interoligt.rougelike.Main;
2  2
3  3  public class Combat {
4  4
5  5      private Monster[] enemies;
6  6      private Player player;
7  7      private Target[] turnOrder;
8  8
9  9      public Combat(Monster[] enemies, Player player){
10 10         this.enemies = enemies;
11 11         this.player = player;
12 12         turnOrder = new Target[enemies.length+1];
13 13     }
14 14
15 15     private void setTargetOrder(){
16 16         for(int i = 0; i < enemies.length; i++){
17 17             turnOrder[i] = enemies[i];
18 18         }
19 19         turnOrder[enemies.length] = player;
20 20
21    -         for(Target i : enemies) {
22    -             for (Monster j : enemies) {
23    -                 if (j.getSpeed() > i.getSpeed()){
24    -                     Target temp = i;
25    -                     i = j;
26    -                     j = temp;
   21 +         for(int i = 0; i < turnOrder.length; i++) {
   22 +             for (int j = 1; j < turnOrder.length-i; j++) {
   23 +                 if (turnOrder[j-1].getSpeed() < turnOrder[j].getSpeed()){
   24 +                     Target temp = turnOrder[j-1];
   25 +                     turnOrder[j-1] = turnOrder[j];
   26 +                     turnOrder[j] = temp;
27 27                 }
```

TDD-exempel: Magnus Palmstierna

Testkod

```java
package com.interoligt.rougelike.Main;

import ...

public class TestItem {

    Item itemToTest = new Item( name: "qwer",  weight: 1,  worth: 1,  effect: null);

    //Check name contains letters
    @Test
    void testName(){
        String name = itemToTest.getName();
        assertFalse( condition: name.isBlank() && name.isEmpty());
    }
    //Check item has worth
    @Test
    void testWorth(){
        int worth = itemToTest.getWorth();
        assertTrue( condition: worth > 0);
    }
    //Check item has weight
    @Test
    void testWeight(){
        int weight = itemToTest.getWeight();
        assertTrue( condition: weight > 0);
    }

    //Assert throws
    @Test
    void assertEmptyNameThrow() { assertThrows(IllegalArgumentException.class, () -> new Item( name: "",  weight: 1,  worth: 1,  effect: null)); }
    @Test
    void assertBlankNameThrow() { assertThrows(IllegalArgumentException.class, () -> new Item( name: " ",  weight: 1,  worth: 1,  effect: null)); }

    @Test
    void assertWorthThrow() { assertThrows(IllegalArgumentException.class, () -> new Item( name: "qwer",  weight: 1,  worth: 0,  effect: null)); }
    @Test
    void assertWeightThrow() { assertThrows(IllegalArgumentException.class, () -> new Item( name: "qwer",  weight: 0,  worth: 1,  effect: null)); }
}
```

Koden som testas

```java
package com.interoligt.rougelike.Main;

public class Item {
    String name;
    int weight;
    int worth;
    Effect effect;

    public Item(String name, int weight, int worth, Effect effect){
        if(!name.isBlank() && !name.isBlank()){
            this.name = name;
        }else{
            throw new IllegalArgumentException("Name must have letters");
        }
        if(weight > 0){
            this.weight = weight;
        }else{
            throw new IllegalArgumentException("Item must have a weight");
        }
        if(worth > 0){
            this.worth = worth;
        }else{
            throw new IllegalArgumentException("Item must have worth");
        }
        this.effect = effect;
    }

    public String getName() { return name; }

    public int getWeight() { return weight; }

    public int getWorth() { return worth; }

    public Effect getEffect() { return effect; }
}
```

TDD-exempel: Moa Hoffström

Testkod

```
//T3
@Test
void testPurchaseMoneyEqualsItemWorth() {
    shop.addItem(fourthItem);
    player.withdrawMoney( amount: 400);
    shop.buyItem(fourthItem);
    assertEquals( expected: 0, player.getMoney());
}

//T4
@Test
void testPurchaseWhileItemIsNull(){
    assertThrows(NullPointerException.class,
            ()->{
                shop.buyItem(null);
            });
}

//T1
@Test
void testPurchaseWithMoneyAmount0(){
    player.withdrawMoney( amount: 500);
    shop.addItem(firstItem);
    assertFalse(shop.buyItem(firstItem));
}
```

Koden som testas

```
public boolean buyItem(Item item){
    if(item != null){
        int itemWorth = item.getWorth();

        if(player.getMoney() < itemWorth) {
            return false;
        }else{
            try {
                player.getInventory().addItem(item);
                player.withdrawMoney(itemWorth);
                return true;

            }catch(Exception e){
                return false;
            }
        }
    }else {
        throw new NullPointerException();
    }
}
```

TDD-exempel: Moa Hoffström

Testkod

```
@Test
void testExceededMaximumItems(){
    Shop emptyShop = new Shop( maxItems: 2);
    emptyShop.addItem(firstItem);
    emptyShop.addItem(secondItem);
    assertFalse(emptyShop.addItem(thirdItem));

}

@Test
void testExceedMaximumItemsWithArrayList(){
    items.add(firstItem);
    items.add(secondItem);
    items.add(thirdItem);
    assertThrows(IllegalArgumentException.class, () -> new Shop(items,  maxItems: 2));
}
```

Koden som testas

```
public Shop(int maxItems) { this.maxItems=maxItems; }

public Shop(ArrayList<Item> items, int maxItems) throws IllegalArgumentException {
    if(items.size() <= maxItems){
        this.maxItems=maxItems;
        this.items = items;
    }else {
        throw new IllegalArgumentException("Max Items Is Exceeded. No shop was created");
    }
}
```

TDD-exempel: Moa Hoffström

Testkod

```
@Test
void testAssertActivatedIsInitiallyFalse() {
    assertFalse(roomToTest.isActivated());
}

@Test
void testActivation(){
    roomToTest.activate();
    assertTrue(roomToTest.isActivated());
}

@Test
void testDeActivate(){
    roomToTest.activate();
    roomToTest.deActivate();
    assertFalse(roomToTest.isActivated());
}
```

Koden som testas

```
private boolean activated = false;
private ArrayList<Monster> monsters;

public Room(Monster... m) {
    this.monsters = new ArrayList<Monster>(Arrays.asList(m));
}

public void activate(){
    activated = true;
}

public void deActivate(){
    activated = false;
}

public boolean isActivated() { return activated; }
```

TDD-exempel: Bjargey Ingólfsdóttir

Testkod

```
//Test to see how many Items are in inventory
@Test
void testGetItemCount() {
    int itemCount = inventoryToTest.getItemCount();
    assertTrue( condition: itemCount == 3);
}
```

Koden som testas

```
public int getItemCount() {
    return items.size();
}
```

TDD-exempel: Bjargey Ingólfsdóttir

Testkod

```java
// Test to add too low items
@Test
void testTooLowNewItemsConstructor(){
    assertThrows(Exception.class, () -> new Inventory( maxItemCount: -3, items));
}
```

Koden som testas

```java
public Inventory(int maxItemCount, ArrayList<Item> items){
    if(maxItemCount > 0 ){
        if (items.size() > maxItemCount){
            throw new IllegalArgumentException();
        } else {
            this.maxItemCount = maxItemCount;
            this.items = items;
        }
    }else{
        throw new IllegalArgumentException("Max Item Count needs to be over zero");
    }
}
```

TDD-exempel: Bjargey Ingólfsdóttir

Testkod

```
@Test
void testRemoveMonsterFromRoom(){
    roomToTest.addMonsterToRoom(monsterToTest);
    Monster monsterToRemove = new BasicMonster( name: "Spider", level: 5, expPerLevel: 12,
            Element.FIRE, baseHealth: 50, baseArmour: 8, baseDamage: 4, baseSpeed: 2);
    roomToTest.addMonsterToRoom(monsterToRemove);
    ArrayList<Monster> whatShouldRemain = new ArrayList<>(Arrays.asList(monsterToTest));
    roomToTest.removeMonsterFromRoom(monsterToRemove);
    assertEquals(whatShouldRemain,        roomToTest.showAllMonsters());
}
```

Koden som testas

```
public boolean removeMonsterFromRoom(Monster monster){
    return monsters.remove(monster);
}
```

TDD-exempel: Linnéa Palmgren

Testkod

```
@Test
void effectConstructorContTest(){
    Effect health = new Effect("Health Regen",'h', '+', true,
            5, 5);
    assertEquals(health.toString(), "Name: Health Regen, Stat: " +
            "Health, Effect: Add 5 each " +
            "turn, Duration: 5 turns");
}
```

Koden som testas

```
Effect(String name, char stat, char operator, boolean isContinuous,
    double value, int duration) {
    this(name, stat,operator,isContinuous,value);
    this.duration = duration;
    hasDuration = true;
}
Effect (String name, char stat, char operator, boolean isContinuous,
    double value){
    this.name = name;
    this.stat = stat;
    this.operator = operator;
    this.isContinuous = isContinuous;
    this.value = value;
}
@Override
public String toString(){
    String str = ", Name: " + name + ", Stat: " + getStatusName() + ", Effect: " +
    getEffectType() + " " + value;
    if(isContinuous){
        str += " each turn";
    }
    if(hasDuration){
        str += ", Duration: " + duration + " turns";
    }
    return str;
}
```

TDD-exempel: Linnéa Palmgren

Testkod

```java
@Test
void constructorNameNullTest() {
    assertThrows(NullPointerException.class, () -> {
        new BasicMonster(null, 5, 12, Element.FIRE, 50, 8, 4, 2);
    });
}

@Test
void constructorElementNullTest() {
    assertThrows(NullPointerException.class, () -> {
        new BasicMonster("Spider", 5, 12,null, 50, 8, 4, 2);
    });
}
```

Koden som testas

```java
Monster(String name, int level, Element element, int baseHealth,
        int baseArmour, int baseDamage, int baseSpeed)
        throws NullPointerException, IllegalArgumentException{
    if(name == null || element == null) {
        throw new NullPointerException("Neither name nor " +
                "element is allowed to be null");
    }
…
}
```

TDD-exempel: Linnéa Palmgren

Testkod

```
@Test
void applyEffectHealthIncreaseWithDurationTest(){
    Effect damage = new Effect(target,"Decrease health",
            'h', '-', false, 1000);
    damage.applyEffect();
    Effect health = new Effect(target, "Health boost",
            'h','+',false,500,2);
    health.applyEffect();
    health.applyEffect();
    assertEquals(target.getCurrentHP(), 6500);
    health.applyEffect();
    assertEquals(target.getCurrentHP(), 6000);
}
```

Koden som testas

```
public void applyEffect(){
    if(isActive) {
        if (hasDuration && isContinuous) {
            applyContinuousDurationEffect();
        } else if (hasDuration) {
            applyDurationEffect();
        } else if (isContinuous) {
            applyContinuousEffect();
        } else {
            applySingleEffect();
        }
    }
}
public void applyDurationEffect(){
    if(counter > 0){
        applySingleEffect();
    }
    else{
        removeEffect();
    }
    --counter;
}

public void applySingleEffect(){
    if(!hasBeenApplied) {
        applyEffectSwitch();
    }
    hasBeenApplied = true;
}
```

TDD erfarenheter

- TDD Fungerade bra i början innan vi behövde integrera klasserna med varann för då så kunde olika personer ha olika idéer på hur kommunikationen skulle ske vilket ledde till att vi fick göra stora refaktoreringar. T.ex. Target-klassen som vi inte har specifika tester för utan de täcks upp av monster- och player-testerna.

- För att lättare kunna tillämpa TDD, krävs bra diskussioner och tidig generell struktur för hur programmet ska fungera, och vilka klasser som ska ansvara för vad.

Testfallsdesign ekvivalensklasser

- Vi har valt att tillämpa ekvivalensklassuppdelning för metoden buyItem() i Shop-klassen, eftersom att metoden har regler som appliceras efter varandra – buyItem() kräver dels att det finns utrymme i Inventory och dels att Player har tillräckligt mycket pengar för att ha råd med ett Item.

Ekvivalensklasserna

Aspekt	ID	Ekvivalensklasser	Valid
Money	M1	player.getMoney() <= 0	Yes
	M2	player.getMoney() < ItemWorth	No
	M3	player.getMoney() = ItemWorth	Yes
	M4	player.getMoney() > ItemWorth	Yes
Inventory	IN1	inventory.maxItemCount() > inventory.getItemCount()	Yes
	IN2	inventory.maxItemCount() <= inventory.getItemCount()	No
Item	IM1	Item = null	No
	IM2	Item != null	Yes

Testfall

ID	Indata	Förväntat resultat	Täckta klasser
T1	Money = 0 ItemWorth = 100 Item != null inventory.maxItemCount() > inventory.getItemCount()	Fail	M1
T2	Money = 50 ItemWorth = 100 Item != null inventory.maxItemCount() > inventory.getItemCount()	Fail	M2
T3	Money = 100 ItemWorth = 100 Item != null inventory.maxItemCount() > inventory.getItemCount()	OK	M3, IM2, IN1
T4	Money = 500 Item = null inventory.maxItemCount() > inventory.getItemCount()	Fail	IM1
T5	Money = 500 ItemWorth = 100 Item != null inventory.maxItemCount() > inventory.getItemCount()	OK	IN1, IM2, M4
T6	Money = 500 ItemWorth = 100 Item != null inventory.maxItemCount() <= inventory.getItemCount()	Fail	IN2

Testmatris

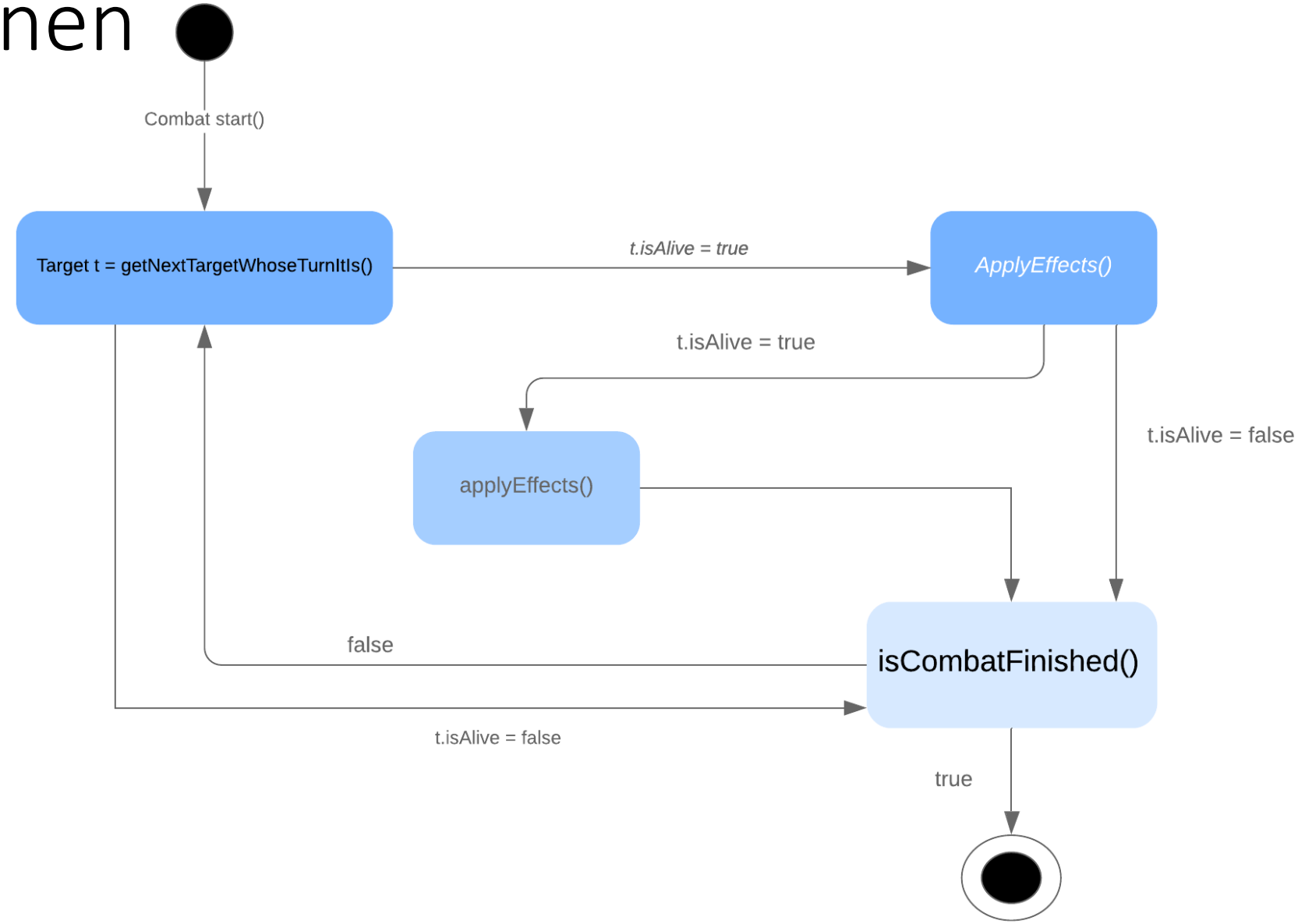
	M1	M2	M3	M4	IN1	IN2	IM1	IM2
T1	X							
T2		X						
T3			X		X			X
T4							X	
T5				X	X			X
T6						X		

Tillståndsmaskiner

- I Combat finns det flera tillstånd som en tur går igenom därav valde vi att använda en tillståndsmaskin för att ta fram testfall för den klassen.

- Diagrammet hjälper oss ha en visuell uppsikt över de möjliga stadier klassen kan gå igenom så att vi kan utforma tester som grundligt går igenom tillstånden som potentiellt kan orsaka fel om de inte hanteras korrekt.

Tillståndsmaskinen



Testfall

ID	Beskrivning	Täckta tillstånd	Täckta övergångar
1 (testAllTrueStateDiagram)	1. Target lever 2. Target överlever potential effekt 3. Combat är slut	getNextTargetWhoseTurnItIs(), ApplyEffects(), applyEffects(), isCombatFinished()	t.isAlive = true, t.isAlive = true, isCombatFinished() = false
2 (testTrueFalseStateDiagram)	1. Target lever 2. Target överlever inte potentiell effect 3. Combat är slut med död spelare	getNextTargetWhoseTurnItIs(), ApplyEffects(), isCombatFinished()	t.isAlive = true, t.isAlive = false
3 (testFalseStateDiagram)	1. Target lever inte 2. Combat startar och slutar omedelbart.	getNextTargetWhoseTurnItIs(), isCombatFinished()	t.isAlive = false

Granskning

- Vi valde att granska metoden Effect, då den är den största metoden i programmet, samt innehåller en del if- och switch-satser.
- Då vi inte har ett färdigt program, är det svårt att göra en inspektion utifrån ett scenario, så vi valde att göra den utifrån en checklista. Vi utgick ifrån den checklista som använts i kursen och anpassade den till den kod vi skulle granska.

Granskning

Checklist used for Inspection

Class Definition

1. Does each class have a descriptive name used in accord with naming conventions?
2. For each class and interface: Are the declarations ordered as follows: Class/interface documentation comment, class or interface statement, Class/interface implementation comment, Class (static) variables and constants, Instance variables, Constructors, Methods? 3. Does each class have an appropriate constructor?
4. For each member of every class: Could access to the member be further restricted?

Method

7. Are descriptive method names used in accord with naming conventions?
8. Is every method parameter value checked before being used?
9. For every method: Does it return the correct value at every method return point?
12. Are there static methods that should be non-static or vice-versa?

Declarations

13. Are descriptive variable and constant names used in accord with naming conventions?
14. Are there variables with confusingly similar names?
15. Is every variable and attribute correctly typed?
16. Is every variable and attribute properly initialized?
17. Could any non-local variables be made local?

Computation & Comparison

22. Are parentheses used to avoid ambiguity?
23. Are the comparison operators correct?
24. Is each boolean expression correct?
25. Are there improper and unnoticed side-effects of a comparison?

Control Flow

29. Does each switch statement have a default case?
30. Are missing switch case break statements correct and marked with a comment?
31. Is the nesting of loops and branches too deep, and is it correct?
32. Can any nested if statements be converted into a switch statement?
34. Does every method terminate?

Performance

36. Are logical tests arranged such that the often successful and inexpensive tests precede the more expensive and less frequently successful tests?
38. Is every result that is computed and stored actually used?

Packaging

54. Does each line contain at most one statement?
55. For each line: Is it no more than about 80 characters long?
56. For each method: Is it no more than about 60 lines long?

Modularity

60. Is there duplicate code that could be replaced by a call to a method that provides the behavior of the duplicate code?

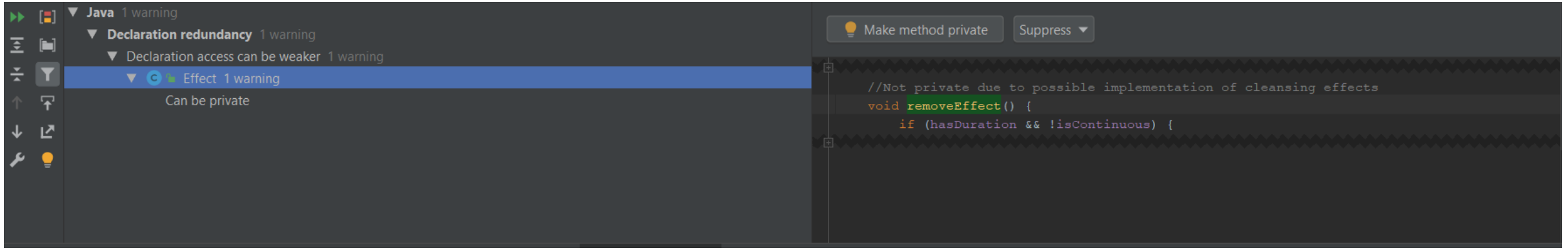
Granskningsrapport

Fel hittade i koden utifrån checklistan	Bedömd nivå
Some methods could be further restricted	Minor
One method doesn’t check if method parameter ”target” is null	Minor
No comments when break isn’t used in switch statements	Minor
5 lines are too long	Minor

Erfarenheter av granskning

- Vi anser att det är bättre med informell granskning när det gäller kod av denna storlek och på denna nivå.
- Det bästa med inspektionen var att man fick en idé om vilka saker man själv behöver tänka på i framtiden. Möjligtvis att man gör en egen lista med saker man har svårt med eller brukar missa, så att man själv kan hålla koll på att man kodar på ett bra sätt och får bra vanor.

Kodkritiksystem

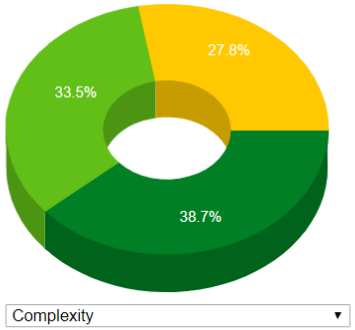


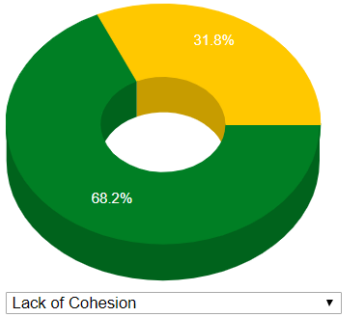
Kodkritiksystemet från IntelliJ visade samma resultat innan som efter vår granskning, och felet den uppvisade var inte ett fel i sig då vår access modifier kan inte vara private då det låser ut en av de potentiella scenario där den kan behöva vara tillgänglig för andra klasser.

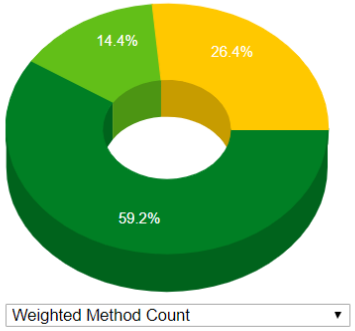
På en mindre skala där det skett mycket informella granskningar löpande då vi alla kodat i samma rum oftast och kunnat fråga varandra om hjälp kan ha vart bidragande till att kodkritiksystemet ej hittade någon större mängd fel, och med tanke på att fel hittades av oss under granskningen som inte plockades upp här påvisar den potentiella nödvändigheten av både statisk analys och formell granskning, även om bara för att bekräfta att fel inte är där.

Statiska mått

ID	CLASS	COUPLING	COMPLEXITY	LACK OF COHESION	SIZE	LOC	COMPLEXITY	COUPLING	LACK OF COHESION	SIZE
1	Effect					138	medium-high	low	medium-high	low-medium
2	SampleBossMonster					7	medium-high	low	low	low
3	Target					75	low-medium	low	low	low-medium
4	Player					31	low-medium	low	low	low
5	Monster					28	low-medium	low	medium-high	low







Måtten utvalda är LOC, brist på cohesion, weighted method count och cyclomatisk komplexitet.

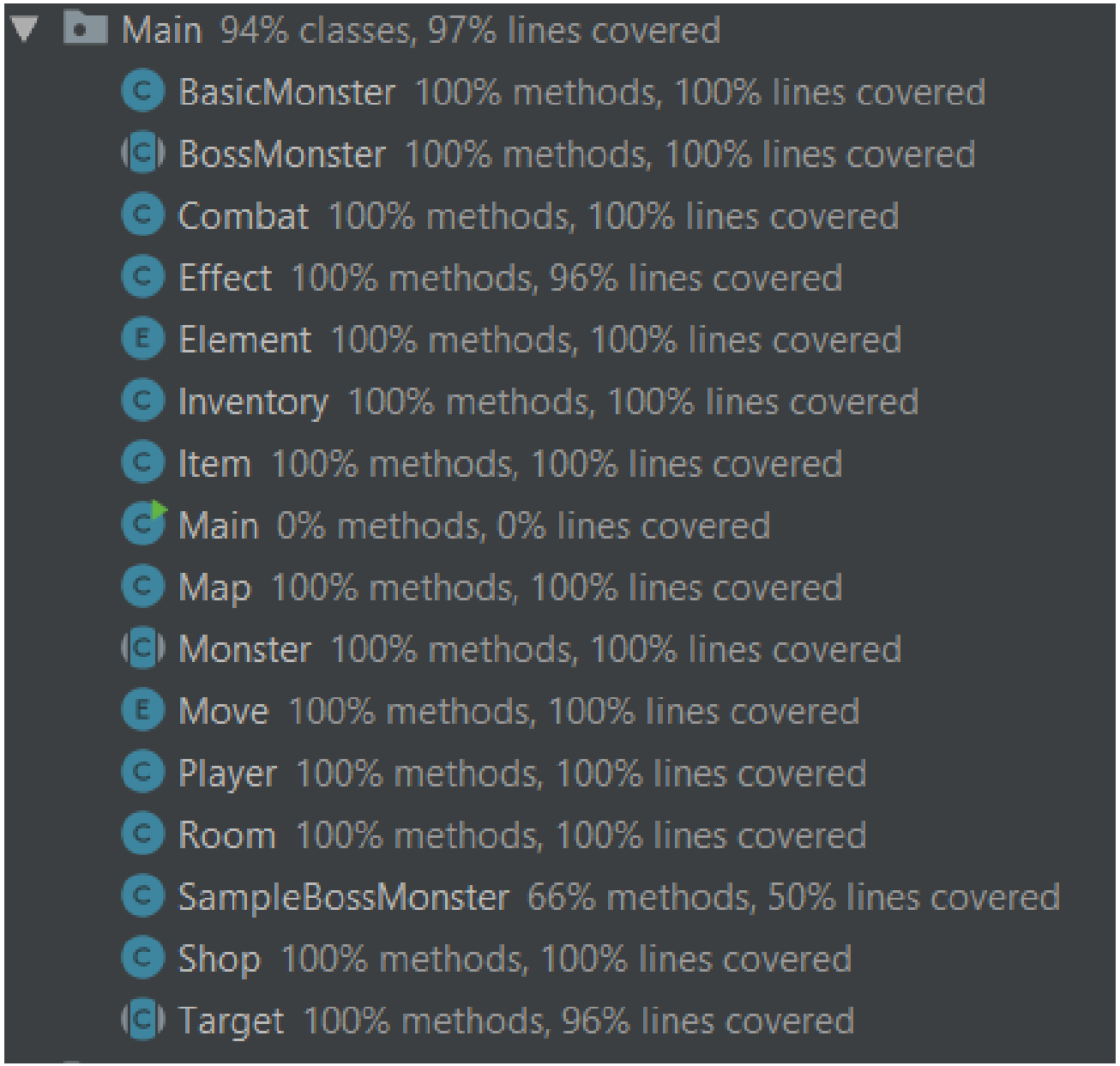
Komplexiteten är låg på de flesta använda klasser med undantag för Effect som pga de möjliga utfallen av tillstånd på effekterna kräver en högre komplexitet för att kunna täcka typerna vi ville ha med i biblioteket.

Brist på cohesion är också låg i alla förutom Effect och Monster, vilket kan förklaras med att Effect aldrig används fristående utan alltid appliceras på andra klasser, och att Monster är en abstract klass som är beroende av andra abstract klasser.

Weighted method count kan vi i grafen se att i majoriteten av klasser så är de bestående av fåtal metoder. Med största sannolikhet är de klasser med högre count Effect och Target, då de har högst LOC och vi för det mesta har försökt hålla metoder i liknande mängd rader. Anledningen är mängden variationer i Effect som behövs, och hur central Target är för hela programmet.

Täckningsgrad

Main har 0% täckningsgrad då den inte används. SampleBoxMonster har en 66% täckningsgrad då den endast används för testing och inte är ett del av programmet



Profiler

- Vi skapade ett program som testade combat mellan player och monsters i olika rum, och testade den koden med en profiler. Då programmet inte körs under någon längre tid, och endast testar ett scenario, valde vi att använda tracing istället för sampling.
- Våra mått visade att våra metoder var väldigt små och snabba, samt att vi inte hade några problem. Vi hade svårt att se något alls som hade med vår kod att göra, då våra metoder är korta och okomplicerade. Det tog därför ingen tid att köra programmet. Då vi varken använder threading, instansierar stora mängder objekt samtidigt eller tar bort objekt, finns det egentligen ingenting att mäta.

Byggscript

```
<project name="junit5-jupiter-starter-ant" default="build" basedir=".">
  <!--
        <fail message="Ant 1.10.4+ is required!">
            <condition>
                <not>
                    <antversion atleast="1.10.4"/>
                </not>
            </condition>
        </fail>
  -->
  <property name="classpath" value="${basedir}/lib"/>
  <path id="test.classpath">
    <pathelement path="build/test"/>
    <pathelement path="build/main"/>
    <!--         <fileset dir="${ant.home}/lib" includes="*.jar" /> -->
    <fileset dir="lib" includes="**/*.jar"/>
    <fileset dir="${basedir}/lib"/>
  </path>
  <target name="build" description="clean build" depends="clean, test"/>
  <target name="clean">
    <delete dir="build"/>
  </target>
  <target name="init">
    <mkdir dir="build/main"/>
    <mkdir dir="build/test"/>
    <mkdir dir="build/test-report"/>
  </target>
  <target name="compile" depends="init">
    <javac destdir="build/main" srcdir="src/" includeantruntime="false"/>
    <javac destdir="build/test" classpathref="test.classpath" srcdir="test/" includeantruntime="false"/>
  </target>
  <!--
     https://junit.org/junit5/docs/snapshot/user-guide/#running-tests-build-ant
  -->
  <target name="test.junit.launcher" depends="compile">
    <junitlauncher haltOnFailure="false" printSummary="true">
      <classpath refid="test.classpath"/>
      <testclasses outputdir="build/test-report">
        <fileset dir="build/test">
          <include name="**/*Test.class"/>
        </fileset>
        <listener type="legacy-xml" sendSysOut="true" sendSysErr="true"/>
        <listener type="legacy-plain" sendSysOut="true"/>
      </testclasses>
    </junitlauncher>
  </target>
  <!--
        &lt;!&ndash; https://junit.org/junit5/docs/current/user-guide/#running-tests-console-launcher &ndash;&gt;
        <target name="test.console.launcher" depends="compile">
            <java classpathref="test.classpath" classname="org.junit.platform.console.ConsoleLauncher" fork="true" failonerror="true">
                <arg value="&#45;&#45;scan-classpath"/>
                <arg line="&#45;&#45;reports-dir build/test-report"/>
            </java>
            <junitreport todir="build/test-report">
                <fileset dir="build/test-report">
                    <include name="TEST-*.xml"/>
                </fileset>
                <report format="frames" todir="build/test-report/html"/>
            </junitreport>
        </target>
  -->
  <target name="test" depends="test.junit.launcher"/>
</project>
```

Varsågod för oss!