

Douglas Sangster – Implementation & Testing Unit (PDA)

- I.T. 1 – Use of Encapsulation

```
public class Card implements Serializable {

    private CardName    name;
    private FactionColour colour;
    private int          imageBlueViewInt;
    private int          imageRedViewInt;
    private HashMap<Integer, ArrayList<Coordinate>> moveset;

    public Card(CardName name, FactionColour colour, int imageBlueViewInt, int imageRedViewInt, HashMap<Integer, ArrayList<Coordinate>> moveset) {

        this.name          = name;
        this.colour         = colour;
        this.imageBlueViewInt = imageBlueViewInt;
        this.imageRedViewInt = imageRedViewInt;
        this.moveset        = moveset;
    }

    public CardName getName() {

        return name;
    }

    public FactionColour getCardColour() {

        return this.colour;
    }

    public int getImageBlueViewInt(){

        return this.imageBlueViewInt;
    }

    public int getImageRedViewInt(){

        return this.imageRedViewInt;
    }
}
```

- I.T. 2 – Use of Inheritance

```
public abstract class Being {

    private String name;
    private int maxHealth;
    private int currentHealth;
    protected IDefend defender;
    protected ArrayList<IAbsorb> protection;

    public Being(String name, int maxHealth) {
        this.name          = name;
        this.maxHealth      = maxHealth;
        this.currentHealth  = maxHealth;
        this.protection     = new ArrayList<>();
    }

    public void receiveAttack(int damageReceived){
        if(defender != null){
            ((Being) defender).receiveAttack(damageReceived);
        } else {
            for(IAbsorb element: protection){
                damageReceived = element.block(damageReceived);
            }
            takedamage(damageReceived);
        }
    }

    public String getName() { return name; }

    public int getMaxHealth() { return this.maxHealth; }

    public int getCurrentHealth() { return this.currentHealth; }
}

public abstract class Adventurer<T> extends Being {

    protected T mainItem;

    public Adventurer (String name, int maxHealth, T mainItem) {
        super(name, maxHealth);
        this.mainItem = mainItem;
    }

    public abstract void useMainItem(Being target);

    public T getMainItem(){
        return this.mainItem;
    }

    public void wieldItem(T newMainItem) {
        this.mainItem = newMainItem;
    }
}
```

```

public class Knight extends Adventurer<Weapon> implements IDefend {

    private Being beingUnderProtection;

    public Knight(String name, int maxHealth, Weapon mainWeapon, Armour shield) {
        super(name, maxHealth, mainWeapon);
        this.protection.add(shield);
    }

    public void useMainItem(Being target) {
        this.mainItem.dealDamage(target);
    }

    public void defend(Being target){
        if(this.defender != null){
            this.defender.stopDefending();
        }

        if(beingUnderProtection != null) {
            beingUnderProtection.removeDefender();
        }

        beingUnderProtection = target;
        target.addDefender(this);
    }

    public void stopDefending(){
        beingUnderProtection.removeDefender();
        beingUnderProtection = null;
    }
}

```

The screenshot shows an IDE with the following components:

- Package Explorer (Left):** A tree view showing packages like `beings`, `adventurers`, `creatures`, `tools`, and `test`. The `test` package is expanded, showing `java` and `adventurerTests` sub-packages.
- Code Editor (Center):** Displays the `BeingTest` class with several JUnit tests. The tests include:
 - `@Before` method for setting up test data (shield, helmet, crossbow, knight, knightBeingDefended).
 - `@Test` methods: `canGetName()`, `canGetMaxHealth()`, `canGetMainWeaponValue()`, and `canReceiveAttack()`.
- Test Runner (Bottom):** A table showing the results of 7 tests, all of which passed.

Test Name	Duration	Status
BeingTest (BeingTest)	15ms	Passed
canAddDefenderWhichThenReceivesAttackInste	0ms	Passed
canAddHealthUpToMax	14ms	Passed
canAddMoreProtection	0ms	Passed
canGetMainWeaponValue	1ms	Passed
canGetMaxHealth	0ms	Passed
canGetName	0ms	Passed
canReceiveAttack	0ms	Passed

- I.T. 3 – Searching Data

```

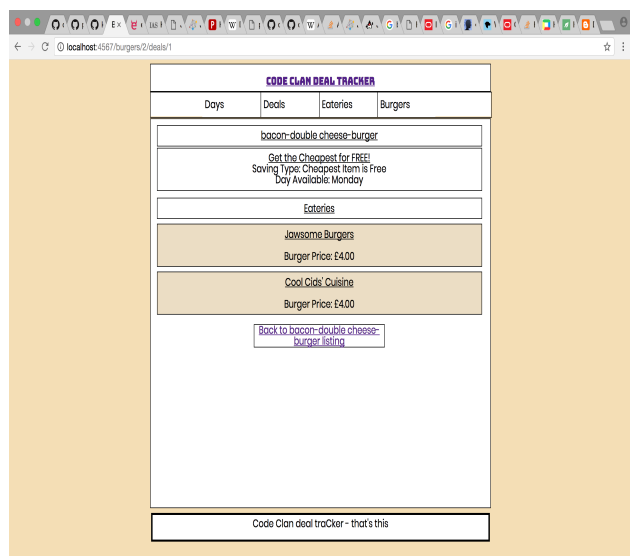
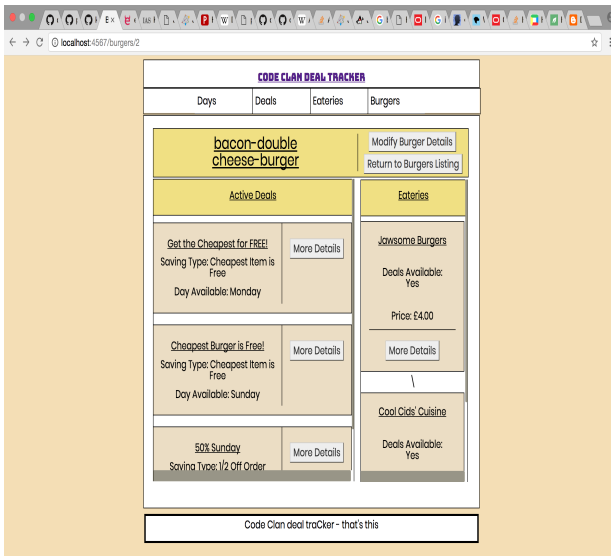
def Eatery.find_by_burger_deal(options)
  sql = "
  SELECT eateries.id, eateries.name
  FROM eateries INNER JOIN active_deals ON
  active_deals.eatery_id = eateries.id WHERE
  active_deals.burger_id = $1 AND
  active_deals.deal_id = $2;
  "
  burger_id = options['burger'].to_i
  deal_id = options['deal'].to_i
  values = [burger_id, deal_id]
  eatery_hashes = SqlRunner.run(sql, values)
  return mapper_aid(eatery_hashes)
end

```

```

get('/burgers/:burger_id/deals/:deal_id')
do
  burger_id = params['burger_id'].to_i
  deal_id = params['deal_id'].to_i
  @burger = Burger.find(burger_id)
  @deal = Deal.find(deal_id)
  @eateries =
    Eatery.find_by_burger_deal({'burger' =>
      burger_id , 'deal' => deal_id})
  erb(:"burgers/deals/show")
end

```



- I.T. 4 – Sorting Data

```

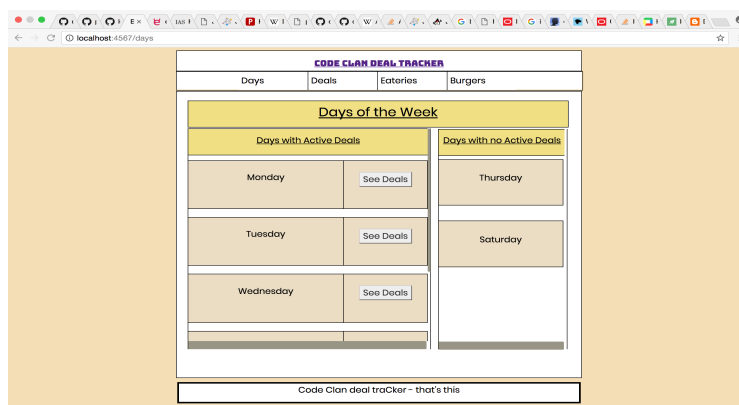
def Day.find_all_active
  sql = "
    SELECT DISTINCT days.id, days.day FROM
    days INNER JOIN
    deals ON days.id = deals.day_id INNER
    JOIN
    active_deals ON deals.id =
    active_deals.deal_id
    ORDER BY days.id ASC;
  "
  day_hashes = SqlRunner.run(sql)
  return mapper_aid(day_hashes)
end

```

```

get('/days') do
  @active_days = Day.find_all_active
  @inactive_days = Day.find_all_inactive
  erb(:"days/index")
end

```



- I.T. 5 – Use of an Array

```
public Deck(){
    cards = new ArrayList<>();
    createDeck();
}

private void createDeck() {
    Suit[] suits = Suit.values();
    Rank[] ranks = Rank.values();
    int indexSuit;
    int indexRank;
    for (indexSuit = 0; indexSuit < suits.length; indexSuit += 1) {
        for (indexRank = 0; indexRank < ranks.length; indexRank += 1) {
            Card card = new Card(suits[indexSuit], ranks[indexRank]);
            cards.add(card);
        }
    }
}

public int getSize() {
    return this.cards.size();
}
```

```
@Before
public void before(){
    deck = new Deck(); // deck: Deck@832
    player = new Player("name: " + "Kaius"); // player: null
    dealer = new Dealer("name: " + "Joker");
}

@Test
public void deckHas52Cards(){
    assertEquals("expected: 52, deck.getSize()", 52, deck.getSize());
}

DeckTest > before()

Variables
this = {DeckTest@827}
deck = {Deck@832}
cards = {ArrayList@841} size = 52
```

- I.T. 6 – Use of a Hash

```
public HashMap<Player, GameResultType> decideResult(Player player1, Player
player2) {
    HashMap<Player, GameResultType> outcome;
    outcome = new HashMap<>();
    if (player1.getHandValue() == player2.getHandValue()){
        outcome.put(player1, GameResultType.DRAW);
        outcome.put(player2, GameResultType.DRAW);
    } else if (player1.getHandValue() > player2.getHandValue()){
        outcome.put(player1, GameResultType.WIN);
        outcome.put(player2, GameResultType.LOSE);
    } else {
        outcome.put(player1, GameResultType.LOSE);
        outcome.put(player2, GameResultType.WIN);
    }

    return outcome;
}
```

```
adjudicationResultHash = adjudicator.decideResult(this.playerHuman,
this.playerComputer);
GameResultType playerHumanResult = adjudicationResultHash.get(this
.playerHuman);

if (playerHumanResult.equals(GameResultType.DRAW)){
    resultString = "The game is a draw!";
} else if (playerHumanResult.equals(GameResultType.WIN)) {
    resultString = "You won the game!";
} else {
    resultString = String.format("You lost the game, %s had a better
hand.", opponentName);
}

System.out.println(resultString);
}
```

```
classes — java Runner — java — java — 80x24

Your hand has a value of 19, Alex's hand has a value of 11.

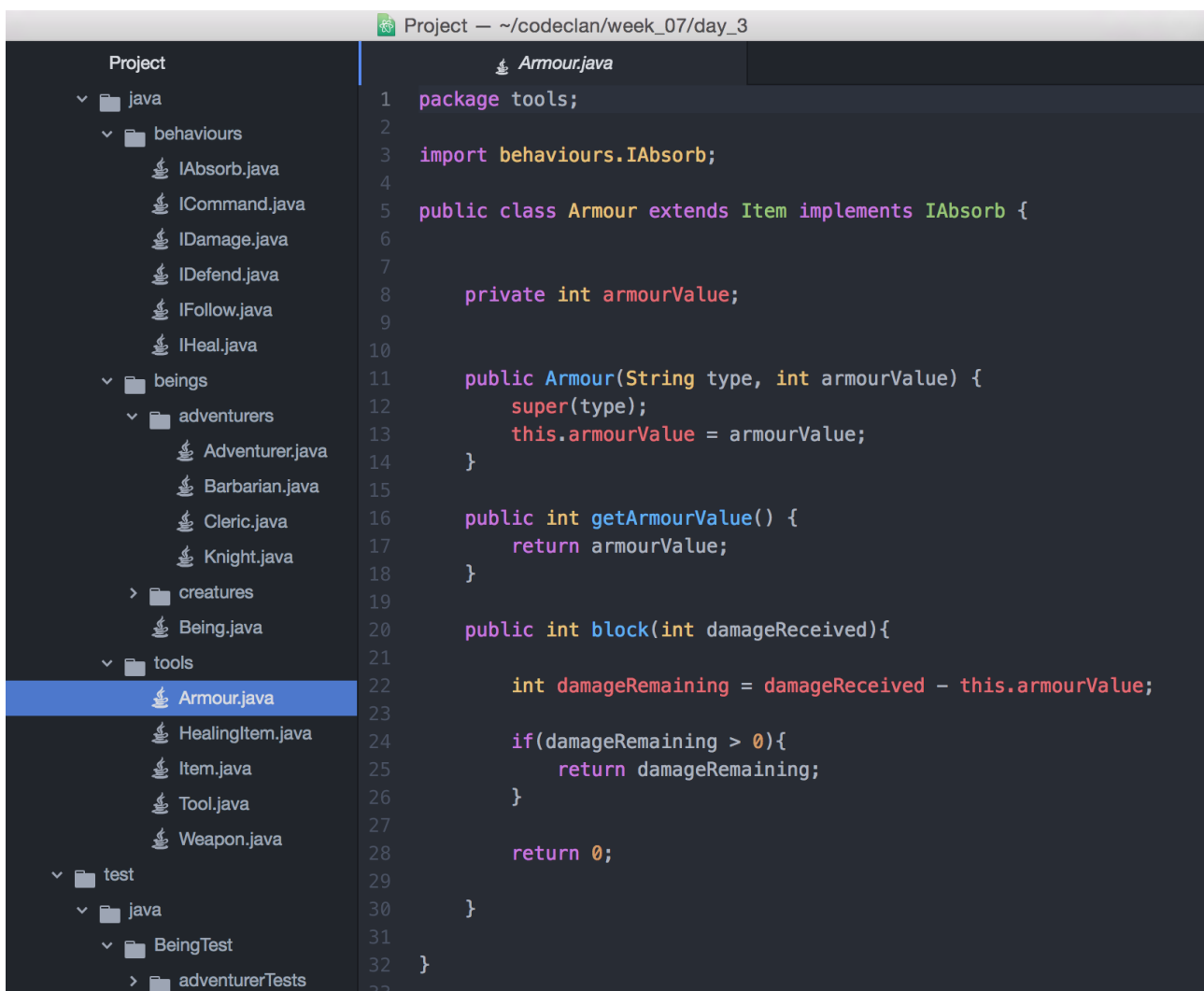
You won the game!

Start another game?
(Y)es/(N)o: █
```

- I.T 7 – Use of Polymorphism



```
IAbsorb.java — ~/codeclan/week_07/day_3
1 package behaviours;
2
3 public interface IAbsorb {
4
5     int block(int damage);
6
7 }
8
```



```
Project — ~/codeclan/week_07/day_3
1 package tools;
2
3 import behaviours.IAbsorb;
4
5 public class Armour extends Item implements IAbsorb {
6
7     private int armourValue;
8
9
10
11     public Armour(String type, int armourValue) {
12         super(type);
13         this.armourValue = armourValue;
14     }
15
16     public int getArmourValue() {
17         return armourValue;
18     }
19
20     public int block(int damageReceived){
21
22         int damageRemaining = damageReceived - this.armourValue;
23
24         if(damageRemaining > 0){
25             return damageRemaining;
26         }
27
28         return 0;
29
30     }
31
32 }
33
```

Project — ~/codeclan/week_07/day_3

Project

- IDEfend.java
- IFollow.java
- IHeal.java
- beings
 - adventurers
 - Adventurer.java
 - Barbarian.java
 - Cleric.java
 - Knight.java
 - creatures
 - Being.java
- tools
 - Armour.java
 - HealingItem.java
 - Item.java
 - Tool.java
 - Weapon.java
- test
 - java
 - BeingTest
 - itemTests
 - ArmourTest.java
 - HealingItemTest.java

ArmourTest.java

```
1 package itemTests;
2
3 import tools.Armour;
4 import org.junit.Before;
5 import org.junit.Test;
6
7 import static org.junit.Assert.assertEquals;
8
9 public class ArmourTest {
10
11     Armour armour;
12
13     @Before
14     public void before() {
15         armour = new Armour("Shield", 2);
16     }
17
18     @Test
19     public void canGetArmourValue(){
20         assertEquals(2, armour.getArmourValue());
21     }
22
23     @Test
24     public void canBlockDamage(){
25         assertEquals(1, armour.block(3));
26     }
27
28 }
29
```

java

- behaviours
 - IAbsorb
 - ICommand
 - IDamage
 - IDEfend
 - IFollow
 - IHeal
- beings
 - tools
 - Armour
 - HealingItem
 - Item
 - Tool
 - Weapon
- test
 - java
 - BeingTest
 - itemTests
 - ArmourTest
 - HealingItemTest
 - ItemTest
 - WeaponTest

build.gradle

gradlew

```
1 package itemTests;
2
3 import tools.Armour;
4 import org.junit.Before;
5 import org.junit.Test;
6
7 import static org.junit.Assert.assertEquals;
8
9 public class ArmourTest {
10
11     Armour armour;
12
13     @Before
14     public void before() {
15         armour = new Armour( type: "Shield", armourValue: 2);
16     }
17
18     @Test
19     public void canGetArmourValue(){
20         assertEquals( expected: 2, armour.getArmourValue());
21     }
22
23     @Test
24     public void canBlockDamage(){
25         assertEquals( expected: 1, armour.block( damageReceived: 3));
26     }
27
28 }
29
```

ArmourTest · before()

Run: ArmourTest

All 2 tests passed - 3ms

ArmourTest (itemTests) 3ms /Library/Java/JavaVirtualMachines/jdk1.8.0_152.jdk/Contents/Home/bin/java ...

canBlockDamage 1ms

canGetArmourValue 2ms

Process finished with exit code 0

Tests Passed: 2 passed (3 minutes ago)

14:1 LF UTF-8