Defense Against the Dark Arts Code Smell Write Up

Team Members: Ananya Kumar, Shreya Malpani, Ayushi Rajpoot, Dristi Shah, & Ishita Verma

GT GitHub Repo: https://github.gatech.edu/Defense-Against-The-Dark-Arts/M2

1) Couplers- Feature Envy Code Smell

BEFORE CHANGE:

GameScreen.java

```java
        });
    }

    public static int calcTowerPrice(String type, String difficulty) {
        int basePrice;
        if (type.equals("Gryffindor")) {
            basePrice = 10;
        } else if (type.equals("Hufflepuff")) {
            basePrice = 8;
        } else if (type.equals("Ravenclaw")) {
            basePrice = 8;
        } else {
            basePrice = 6;
        }

        if (difficulty.equals("Easy")) {
            return basePrice;
        } else if (difficulty.equals("Medium")) {
            return (int) (basePrice * 1.5);
        } else {
            return basePrice * 2;
        }
    }

}
```

A feature envy is a code smell that is characterized as part of the coupler group. A method guilty of feature envy is more interested in another class than the one it is in. The calcTowerPrice() method in the above screenshot is in the GameScreen class, but it is more interested in Tower and Tower's instance variables (tower type and tower buy price).

AFTER CHANGE:

```java
Tower.java

    public static int calculateTowerPrice(String type, String difficulty) {
        int basePrice;
        if (type.equals("Gryffindor")) {
            basePrice = 10;
        } else if (type.equals("Hufflepuff")) {
            basePrice = 8;
        } else if (type.equals("Ravenclaw")) {
            basePrice = 8;
        } else {
            basePrice = 6;
        }

        int buyPrice = basePrice;
        if (difficulty.equals("Medium")) {
            buyPrice = (int) (basePrice * 1.5);
        } else if (difficulty.equals("Hard")) {
            buyPrice = basePrice * 2;
        }

        return buyPrice;
    }
```

This feature envy code smell was fixed by moving this method to the Tower class. It no longer classifies as a feature envy code smell because the method is now interested in the current Tower class it is in rather than being in a different class with no relation.

2) Dispensables- Data Class Code Smell

BEFORE CHANGE:

```java
package com.example.milestone2game.entities;
import java.io.Serializable;

public class Tower implements Serializable {
    private String type;
    private int buyPrice;
    private int upgradePrice;
    private int firingRate;
    private int attackStrength;
    //private int attackRadius;

    public Tower(String type) {
        this.type = type;
        this.buyPrice = 10;
        this.upgradePrice = 2;
    }

    public int getBuyPrice() {
        return this.buyPrice;
    }

    public int getUpgradePrice() {
        return this.upgradePrice;
    }
}
```

A data class code smell is characterized as a dispensable code smell. A data class is a class that contains only fields and methods for accessing them (getters and setters). In the screenshot above, the Tower class is a data class because it only has a constructor, instance variables for its data and getters/setters for these variables.

AFTER CHANGE:

```java
public class Tower implements Serializable {
    private String type;
    private int buyPrice;
    private int upgradePrice;
    private int firingRate;
    private int attackStrength;
    private int attackRadius;

    public Tower(String type, String difficulty) {
        this.type = type;
        this.buyPrice = calculateTowerPrice(type, difficulty);
        this.upgradePrice = (int) (buyPrice / 2);
        this.firingRate = (int) (buyPrice / 3);
        this.attackStrength = (int) (buyPrice / 3);
        this.attackRadius = (int) (buyPrice / 3);
    }

    public static int calculateTowerPrice(String type, String difficulty) {
        int basePrice;
        if (type.equals("Gryffindor")) {
            basePrice = 10;
        } else if (type.equals("Hufflepuff")) {
            basePrice = 8;
        } else if (type.equals("Ravenclaw")) {
            basePrice = 8;
        } else {
            basePrice = 6;
        }

        int buyPrice = basePrice;
        if (difficulty.equals("Medium")) {
            buyPrice = (int) (basePrice * 1.5);
        } else if (difficulty.equals("Hard")) {
            buyPrice = basePrice * 2;
        }

        return buyPrice;
    }

    public void upgradeTower() {
        firingRate++;
        attackStrength++;
        attackRadius++;
    }

    public String getType() {
        return type;
    }

    public int getBuyPrice() {
        return buyPrice;
    }

    public int getUpgradePrice() {
        return upgradePrice;
    }
}
```

This class no longer classifies as a dispensable data class because it was fixed by adding additional methods to it that were not getters and setters. As seen above, the upgradeTower() and calculateTowerPrice() methods were added. These methods contain more functionalities related to Towers.

3) Bloater- Long Method

BEFORE CHANGE:

```
193     @Override
194     protected void onCreate(Bundle savedInstanceState) {
195         super.onCreate(savedInstanceState);
196         setContentView(R.layout.gamemap);
197         opengamescreen();
198         Intent intent = getIntent();
199         Player player = (Player) intent.getSerializableExtra("currentPlayer");
200         Monument monument = new Monument();
201         String difficulty = player.getDifficulty();
202         monument.initializeHealth(difficulty);
203         TextView moneyText = (TextView) findViewById(R.id.money);
204         TextView healthText = (TextView) findViewById(R.id.health);
205         moneyText.setText("Money: " + player.getMoney());
206         healthText.setText("Health: " + monument.getHealth());
207         ImageButton tile1Button = (ImageButton) findViewById(R.id.tile_1);
208         Tile tile1 = new Tile(tile1Button);
209         tile1Button.setOnClickListener(new View.OnClickListener() {
210             @Override
211             public void onClick(View view) {
212                 if (!tile1.getClicked()) {
213                     tile1.setClicked(true);
214                     showDialog(player, tile1, moneyText);
215                     moneyText.setText("Money:" + player.getMoney());
216                 } else {
217                     showDialog2(player, moneyText);
218                 }
219             }
220         });
221         ImageButton tile2Button = (ImageButton) findViewById(R.id.tile_2);
222         Tile tile2 = new Tile(tile2Button);
223         tile2Button.setOnClickListener(new View.OnClickListener() {
224             @Override
225             public void onClick(View view) {
226                 if (!tile2.getClicked()) {
227                     tile2.setClicked(true);
228                     showDialog(player, tile2, moneyText);
229                     moneyText.setText("Money:" + player.getMoney());
230                 } else {
231                     showDialog2(player, moneyText);
232                 }
233             }
234         });
235
236         ImageButton tile3Button = (ImageButton) findViewById(R.id.tile_3);
237         Tile tile3 = new Tile(tile3Button);
238         tile3Button.setOnClickListener(new View.OnClickListener() {
239             @Override
240             public void onClick(View view) {
241                 if (!tile3.getClicked()) {
242                     tile3.setClicked(true);
243                     showDialog(player, tile3, moneyText);
244                     moneyText.setText("Money:" + player.getMoney());
245                 } else {
246                     showDialog2(player, moneyText);
247                 }
248             }
249         });   Method ends at line 425. Cannot fit in screenshot
250
```

A long method code smell is in the bloaters group of code smells. The method in the screenshot above is 232 lines long and would be considered a long method because of this length.

AFTER CHANGE:

```java
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.gamemap);
        opengamescreen();
        Intent intent = getIntent();
        player = (Player) intent.getSerializableExtra("currentPlayer");
        shop = new Shop(GameScreen.this, player);
        Monument monument = new Monument();
        String difficulty = player.getDifficulty();
        monument.initializeHealth(difficulty);
        TextView moneyText = (TextView) findViewById(R.id.money);
        TextView healthText = (TextView) findViewById(R.id.health);
        moneyText.setText("Money: " + player.getMoney());
        healthText.setText("Health: " + monument.getHealth());
        ImageButton tile1Button = (ImageButton) findViewById(R.id.tile_1);
        Tile tile1 = new Tile(tile1Button);
        tile1Button.setOnClickListener(new View.OnClickListener() {
            @Override public void onClick(View view) {
                onTileInteraction(tile1);
            }
        });
        ImageButton tile2Button = (ImageButton) findViewById(R.id.tile_2);
        Tile tile2 = new Tile(tile2Button);
        tile2Button.setOnClickListener(new View.OnClickListener() {
            @Override public void onClick(View view) {
                onTileInteraction(tile2);
            }
        });
        ImageButton tile3Button = (ImageButton) findViewById(R.id.tile_3);
        Tile tile3 = new Tile(tile3Button);
        tile3Button.setOnClickListener(new View.OnClickListener() {
            @Override public void onClick(View view) {
                onTileInteraction(tile3);
            }
        });
```

By removing white space and removing repeated statements, the total length of the onCreate() method was reduced by 35%. This was done by also moving parts of this method into the Tower class.

4) Bloater- Long Parameter List

BEFORE CHANGE:

```java
public void startCombat(GameScreen gameScreen, Path path,
                        Monument monument, TextView monHealthText, TextView moneyText) {
    int startDelay = (3000 * (id - 1)) + 1;

    ImageView enemyIcon = gameScreen.findViewById(imageSrc);
    TextView enemyHealthText = gameScreen.findViewById(healthTxtSrc);
    enemyHealthText.setText("Health: " + health);

    ObjectAnimator pathAnimatorEn =
            ObjectAnimator.ofFloat(enemyIcon, View.TRANSLATION_X, View.TRANSLATION_Y, path);
    ObjectAnimator pathAnimatorTxt =
            ObjectAnimator.ofFloat(enemyHealthText, View.TRANSLATION_X, View.TRANSLATION_Y, path)

    pathAnimatorEn.setStartDelay(startDelay);
    pathAnimatorTxt.setStartDelay(startDelay);
    pathAnimatorEn.setDuration(pathDuration);
    pathAnimatorTxt.setDuration(pathDuration);
    pathAnimatorEn.start();
    pathAnimatorTxt.start();
                                                    Method continues below...(Cannot fit in screenshot)
```

A method with a long parameter list is part of the bloaters group of code smells. A method with a long parameter list has more than three or four parameters. The startCombat() method in the screenshot above has 5 parameters, which makes it qualify as a method with a long parameter list.

AFTER CHANGE:

```java
60 @    public void startCombat(GameScreen gameScreen, Path path, Monument monument) {
61          int startDelay = (3000 * (id - 1)) + 1;
62          ImageView enemyIcon = gameScreen.findViewById(imageSrc);
63          TextView enemyHealthText = gameScreen.findViewById(healthTxtSrc);
64          TextView moneyText = gameScreen.findViewById(R.id.money);
65          TextView monHealthText = (TextView) gameScreen.findViewById(R.id.health);
66          enemyHealthText.setText("Health: " + health);
67
68          ObjectAnimator pathAnimatorEn =
69                  ObjectAnimator.ofFloat(enemyIcon, View.TRANSLATION_X, View.TRANSLATION_Y, path);
70          ObjectAnimator pathAnimatorTxt =
71                  ObjectAnimator.ofFloat(enemyHealthText, View.TRANSLATION_X, View.TRANSLATION_Y, pat
72
73          pathAnimatorEn.setStartDelay(startDelay);
74          pathAnimatorTxt.setStartDelay(startDelay);
75          pathAnimatorEn.setDuration(pathDuration);
76          pathAnimatorTxt.setDuration(pathDuration);
77          pathAnimatorEn.start();
78          pathAnimatorTxt.start();
```

This is the fixed startCombat() method. It now has 3 parameters and is no longer hard to understand. It is no longer a method with a long parameter list.

5) Dispensibles- Duplicated Code

BEFORE CHANGE:

```
193     @Override
194     protected void onCreate(Bundle savedInstanceState) {
195         super.onCreate(savedInstanceState);
196         setContentView(R.layout.gamemap);
197         opengamescreen();
198         Intent intent = getIntent();
199         Player player = (Player) intent.getSerializableExtra("currentPlayer");
200         Monument monument = new Monument();
201         String difficulty = player.getDifficulty();
202         monument.initializeHealth(difficulty);
203         TextView moneyText = (TextView) findViewById(R.id.money);
204         TextView healthText = (TextView) findViewById(R.id.health);
205         moneyText.setText("Money: " + player.getMoney());
206         healthText.setText("Health: " + monument.getHealth());
207         ImageButton tile1Button = (ImageButton) findViewById(R.id.tile_1);
208         Tile tile1 = new Tile(tile1Button);
209         tile1Button.setOnClickListener(new View.OnClickListener() {
210             @Override
211             public void onClick(View view) {
212                 if (!tile1.getClicked()) {
213                     tile1.setClicked(true);
214                     showDialog(player, tile1, moneyText);
215                     moneyText.setText("Money:" + player.getMoney());
216                 } else {
217                     showDialog2(player, moneyText);
218                 }
219             }
220         });
221         ImageButton tile2Button = (ImageButton) findViewById(R.id.tile_2);
222         Tile tile2 = new Tile(tile2Button);
223         tile2Button.setOnClickListener(new View.OnClickListener() {
224             @Override
225             public void onClick(View view) {
226                 if (!tile2.getClicked()) {
227                     tile2.setClicked(true);
228                     showDialog(player, tile2, moneyText);
229                     moneyText.setText("Money:" + player.getMoney());
230                 } else {
231                     showDialog2(player, moneyText);
232                 }
233             }
234         });
235
236         ImageButton tile3Button = (ImageButton) findViewById(R.id.tile_3);
237         Tile tile3 = new Tile(tile3Button);
238         tile3Button.setOnClickListener(new View.OnClickListener() {
239             @Override
240             public void onClick(View view) {
241                 if (!tile3.getClicked()) {
242                     tile3.setClicked(true);
243                     showDialog(player, tile3, moneyText);
244                     moneyText.setText("Money:" + player.getMoney());
245                 } else {
246                     showDialog2(player, moneyText);
247                 }
248             }
249         });   Method ends at line 425. Cannot fit in screenshot
250
```

The above screenshot displays duplicated code because there are sections of code that are duplicated. This duplicated code is in the onClick() method for each of the 15 tiles as seen for the first three in the above screenshot.

AFTER CHANGE:

```
28      @Override
29      protected void onCreate(Bundle savedInstanceState) {
30          super.onCreate(savedInstanceState);
31          setContentView(R.layout.gamemap);
32          opengamescreen();
33          Intent intent = getIntent();
34          player = (Player) intent.getSerializableExtra("currentPlayer");
35          shop = new Shop(GameScreen.this, player);
36          Monument monument = new Monument();
37          String difficulty = player.getDifficulty();
38          monument.initializeHealth(difficulty);
39          TextView moneyText = (TextView) findViewById(R.id.money);
40          TextView healthText = (TextView) findViewById(R.id.health);
41          moneyText.setText("Money: " + player.getMoney());
42          healthText.setText("Health: " + monument.getHealth());
43          ImageButton tile1Button = (ImageButton) findViewById(R.id.tile_1);
44          Tile tile1 = new Tile(tile1Button);
45          tile1Button.setOnClickListener(new View.OnClickListener() {
46              @Override public void onClick(View view) {
47                  onTileInteraction(tile1);
48              }
49          });
50          ImageButton tile2Button = (ImageButton) findViewById(R.id.tile_2);
51          Tile tile2 = new Tile(tile2Button);
52          tile2Button.setOnClickListener(new View.OnClickListener() {
53              @Override public void onClick(View view) {
54                  onTileInteraction(tile2);
55              }
56          });
57          ImageButton tile3Button = (ImageButton) findViewById(R.id.tile_3);
58          Tile tile3 = new Tile(tile3Button);
59          tile3Button.setOnClickListener(new View.OnClickListener() {
60              @Override public void onClick(View view) {
61                  onTileInteraction(tile3);
62              }
63          });
```

The duplicated code code smell was fixed as seen in the above screenshot by condensing the code in the onClick() method into a single helper method called onTile interaction(). This method now has 8 lines less of duplicated code.