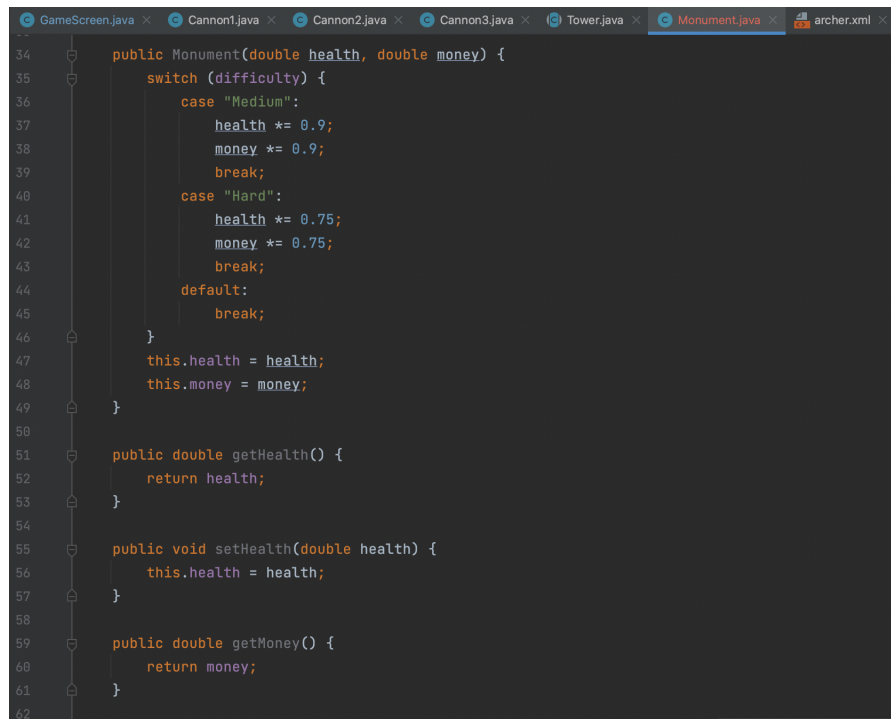


Code Smell: Lazy Class

At first, we created a Monument Class to store monument attributes and also change the attributes of the Monument Class. However, over time, we needed the class less and less as the class wasn't really being used as a class.

A screenshot of an IDE showing the Monument.java file. The code defines a Monument class with a constructor that takes health and money, and a switch statement for difficulty levels (Medium, Hard, default). It also has getter and setter methods for health and money.

```
34 public Monument(double health, double money) {
35     switch (difficulty) {
36         case "Medium":
37             health *= 0.9;
38             money *= 0.9;
39             break;
40         case "Hard":
41             health *= 0.75;
42             money *= 0.75;
43             break;
44         default:
45             break;
46     }
47     this.health = health;
48     this.money = money;
49 }
50
51 public double getHealth() {
52     return health;
53 }
54
55 public void setHealth(double health) {
56     this.health = health;
57 }
58
59 public double getMoney() {
60     return money;
61 }
62 }
```

To fix the code smell, we just added the texts that needed to be changed to be changed based off difficulty within the Player class. This also combatted duplicate code in a way and we were also able to just use monument and set its old attributes as attributes off the Player class.

A screenshot of an IDE showing the Player.java file. The code defines a Player class with private attributes for balance, name, difficulty, and monumentHealth. The constructor sets difficulty and name, and then uses if-else statements to set balance and monumentHealth based on the difficulty level (easy, medium, hard).

```
package com.example.towerdefense;

public class Player {
    private int balance;
    private String name;
    private String difficulty;
    private int monumentHealth;

    Player(String difficulty, String name) {
        this.setDifficulty(difficulty);
        this.setName(name);
        if (difficulty.equals("easy")) {
            setBalance(500);
            setMonumentHealth(100);
        } else if (difficulty.equals("medium")) {
            setBalance(750);
            setMonumentHealth(90);
        } else if (difficulty.equals("hard")) {
            setBalance(1000);
            setMonumentHealth(80);
        }
    }
}
```

Code Smell: Data Clumps

At first, we created 3 cannon classes to represent the different types of towers. We still have those classes, but we realize we used the same parameters for each of the cannon classes, like cost, upgradeMultiplier, etc. Hence, there are clumps of the same data in different classes.

```
15 public Cannon1(Player player, ImageButton button, int cost, int level, double upgradeMultiplier, int attackSpeed,
16               int attackDamage, String location, String imageString) {
17     this.setPlayer(player);
18     this.setButton(button);
19     this.setCost(50);
20     this.setUpgradeMultiplier(1);
21
22     if (player.getDifficulty().equals("medium")) {
23         this.setCost(75);
24         this.setUpgradeMultiplier(1.2);
25     } else if (player.getDifficulty().equals("hard")) {
26         this.setCost(100);
27         this.setUpgradeMultiplier(1.5);
28     }
29     //might need to fix imageString
30     //super(cost, 1, upgradeMultiplier, 10, 3, "", "../../res/drawable/cannon1new.jpg");
31     this.setAttackSpeed(10);
32     this.setAttackDamage(3);
33 }
```

To fix this code smell, we simply made a new tower class that all the cannon classes would inherit. And this tower class would have all the attributes in just one class, and it be reused by many classes.

```
4
5 public abstract class Tower {
6     private int cost;
7     private int level;
8     private double upgradeMultiplier;
9     private int attackSpeed;
10    private int attackDamage;
11
12    private ImageButton button;
13
14    public Tower(int cost, int level, double upgradeMultiplier, int attackSpeed,
15               int attackDamage, String location, String imageString) {
16        this.setCost(cost);
17        this.setLevel(level);
18        this.setUpgradeMultiplier(upgradeMultiplier);
19        this.setAttackSpeed(attackSpeed);
20        this.setAttackDamage(attackDamage);
21    }
22
23    public Tower() { this( cost: 100, level: 1, upgradeMultiplier: 1, attackSpeed: 20, attackDamage: 1, location: "", imageString: ""); }
24
25    public int getCost() { return cost; }
26
27    public void setCost(int cost) { this.cost = cost; }
28
29    public int getLevel() { return level; }
30
31    public void setLevel(int level) { this.level = level; }
32
33    public double getUpgradeMultiplier() { return upgradeMultiplier; }
34}
```

Code Smell: Duplicated Code

At first, we had a series of 5 `setOnClickListener` calls that all used similar code. The code starts from the `Integer` resource line and goes to the `towerAlreadyExists()` line which would display a message for each tower if the condition is met. This is an example of duplicate code

```
place4ImageButton.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        places.remove(place4);  
        visibilityOff();  
        Integer resource = (Integer) place4ImageButton.getTag();  
        if (resource != null) {  
            towerAlreadyExists();  
        }  
    }  
});  
  
place5ImageButton.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        places.remove(place5);  
        visibilityOff();  
        Integer resource = (Integer) place5ImageButton.getTag();  
        if (resource != null) {  
            towerAlreadyExists();  
        }  
    }  
});
```

.To combat this, we just took the code that was duplicated in all of the 5 `setOnClickListener` functions and pasted it into a new function. We also made sure that function took in the necessary parameters needed and then called that function in all of the 5 `setOnClickListener` functions.

```
private boolean placeTower(ImageButton button, int imgRes) {  
    Integer resource = (Integer) button.getTag();  
    if (resource != null) {  
        towerAlreadyExists();  
        return false;  
    }  
}
```

Code Smell: Long Method

In the Game Screen Class, we had one huge `onClickListener` function that basically ran the game and focused on a couple Game Screen changes. However, the method was way too long to even pass checkstyle and so the method was essentially a code smell.

```
//duration should be movementSpeed of enemy object
animator.setDuration(temp.getMovementSpeed());
animator.start();
// for each value in witches
// check if witch.x and witch.y is equal to end coordinates
// if code: delete witch from arraylist and reduce monument health
if (player.getMonumentHealth() > 0) {
    for (Enemy enemy: enemies) {
        if (player.getMonumentHealth() == 0) {
            gameOver();
        }
        View enemyView = enemy.getView();
        System.out.println(enemyView.getX() + " " + enemyView.getY());
        if (enemyView.getX() == difficultyObj.getMonumentCoords()[0]
            && enemyView.getY() == difficultyObj.getMonumentCoords()[1]) {

            if (enemyView.getVisibility() == View.VISIBLE) {
                enemy.attack(player);
                health.setText("Health: " + player.getMonumentHealth());
                if (player.getMonumentHealth() <= 0) {
                    player.setMonumentHealth(100);
                    gameOver();
                    return;
                }
            }
            enemyView.setVisibility(View.GONE);
        }
    }
} else {
    gameOver();
}
```

To combat this, we just took a snippet of code that was all related to a single functionality and separated the long method into two methods. In this case, we took the portion of the long code pertaining to checking the enemies and changing the attributed accordingly and just made a 2nd method out of it and called the function in the previous method.

```
private void checkEnemies(ArrayList<Enemy> enemies, Difficulty difficultyObj) {
    if (player.getMonumentHealth() > 0) {
        for (Enemy enemy: enemies) {
            if (player.getMonumentHealth() == 0) {
                gameOver();
            }
            View enemyView = enemy.getView();
            System.out.println(enemyView.getX() + " " + enemyView.getY());
            if (enemyView.getX() == difficultyObj.getMonumentCoords()[0]
                && enemyView.getY() == difficultyObj.getMonumentCoords()[1]) {

                if (enemyView.getVisibility() == View.VISIBLE) {
                    enemy.attack(player);
                    health.setText("Health: " + player.getMonumentHealth());
                    if (player.getMonumentHealth() <= 0) {
                        player.setMonumentHealth(100);
                        gameOver();
                        return;
                    }
                }
                enemyView.setVisibility(View.GONE);
            }
        }
    }
} else {
    gameOver();
}
```

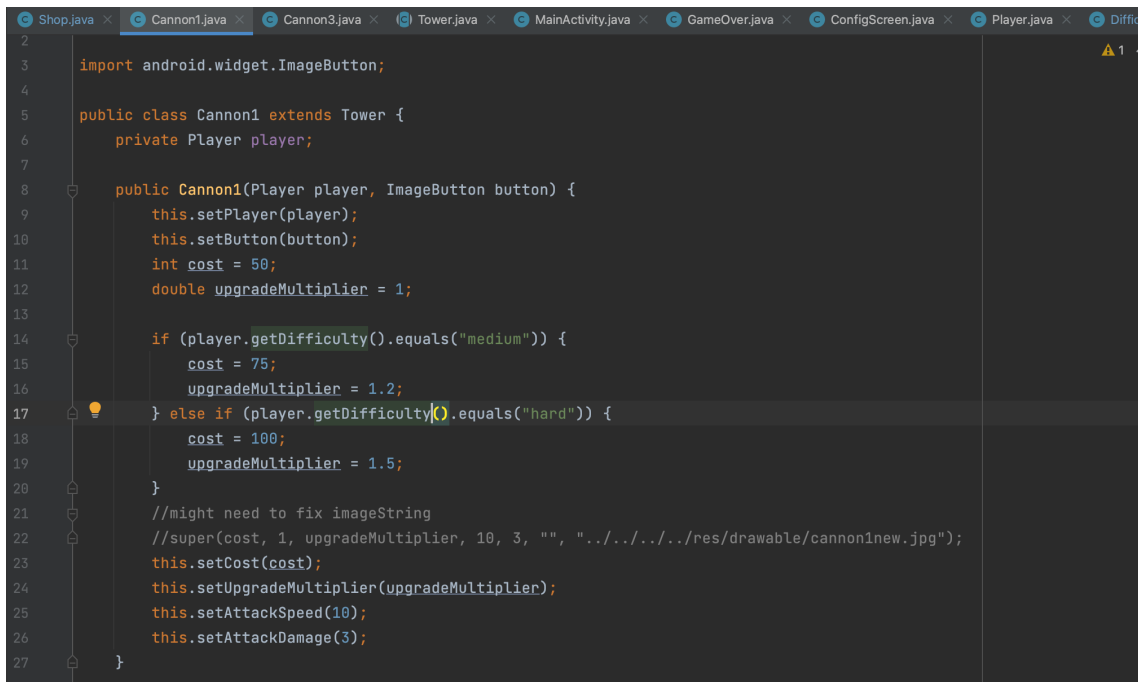
Code Smell: Long Parameter List

This cannon class had too many parameters that didn't all need to be defined here. Some of them weren't even used as well. Hence, this was a long parameter list code smell.



```
15 public Cannon1(Player player, ImageButton button, int cost, int level, double upgradeMultiplier, int attackDamage, String location, String imageString) {
16     this.setPlayer(player);
17     this.setButton(button);
18     this.setCost(50);
19     this.setUpgradeMultiplier(1);
20
21     if (player.getDifficulty().equals("medium")) {
22         this.setCost(75);
23         this.setUpgradeMultiplier(1.2);
24     } else if (player.getDifficulty().equals("hard")) {
25         this.setCost(100);
26         this.setUpgradeMultiplier(1.5);
27     }
28
29     //might need to fix imageString
30     //super(cost, 1, upgradeMultiplier, 10, 3, "", "../../res/drawable/cannon1new.jpg");
31     this.setAttackSpeed(10);
32     this.setAttackDamage(3);
33 }
```

To fix this code smell, we simply removed the parameters that weren't needed for the cannon class. We still kept some of the parameters needed to make an efficient method with less parameters.



```
2 import android.widget.ImageButton;
3
4 public class Cannon1 extends Tower {
5     private Player player;
6
7     public Cannon1(Player player, ImageButton button) {
8         this.setPlayer(player);
9         this.setButton(button);
10         int cost = 50;
11         double upgradeMultiplier = 1;
12
13         if (player.getDifficulty().equals("medium")) {
14             cost = 75;
15             upgradeMultiplier = 1.2;
16         } else if (player.getDifficulty().equals("hard")) {
17             cost = 100;
18             upgradeMultiplier = 1.5;
19         }
20
21         //might need to fix imageString
22         //super(cost, 1, upgradeMultiplier, 10, 3, "", "../../res/drawable/cannon1new.jpg");
23         this.setCost(cost);
24         this.setUpgradeMultiplier(upgradeMultiplier);
25         this.setAttackSpeed(10);
26         this.setAttackDamage(3);
27     }
28 }
```