# M5 - Code Smells

M5-smelly.md

## Long Method - enterBuyTowerMode() (M3)

```
public void enterTowerPlacementMode(TowerEnum tower) {
    Scene scene = ((Stage) gamePane.getScene().getWindow()).getScene();

    if (buyModeHandler != null) {
        TDApp.showErrorMsg("Invalid State", "You're already buying something else");
        return;
    }


    buyModeHandler = new EventHandler<MouseEvent>() {
        @Override
        public void handle(MouseEvent mouseEvent) {
            int x = (int) mouseEvent.getSceneX();
            int y = (int) mouseEvent.getSceneY();


            x -= Tower.SIZE / 2; // This is so that where you click is the center of where
            y -= Tower.SIZE / 2; // the tower is placed


            Tower newTower;
            switch (tower) {
                case BASIC:
                    newTower = new BasicTower(x, y);
                    break;
                case CANNON:
                    newTower = new CannonTower(x, y);
                    break;
                case SPIKE:
                    newTower = new SpikeTower(x, y);
                    break;
                default:
                    newTower = null;
            }


            if (isInvalidTowerLocation(newTower)) {
                TDApp.showErrorMsg("Invalid Location", "You can't place a tower there");
                return;
```

```
                }

                game.getTowers().add(newTower);
                gameObjects.add(newTower);
                game.setMoney(game.getMoney() - newTower.getPrice(game.getDifficulty()));
                scene.removeEventHandler(MouseEvent.MOUSE_CLICKED, this);
                buyModeHandler = null;
                exitTowerPlacementMode();
            }
        };

        scene.addEventHandler(MouseEvent.MOUSE_CLICKED, buyModeHandler);

        Image image;
        switch (tower) {
        case BASIC:
            image = new Image(TDApp.getResourcePath(BasicTower.BASIC_TOWER_IMAGE));
            break;
        case CANNON:
            image = new Image(TDApp.getResourcePath(CannonTower.CANNON_TOWER_IMAGE));
            break;
        case SPIKE:
            image = new Image(TDApp.getResourcePath(SpikeTower.SPIKE_TOWER_IMAGE));
            break;
        default:
            image = null;
        }
        scene.setCursor(new ImageCursor(image));
    }
```

In our code, the enterBuyTowerMode() method handled creating and storing an event handler, creating a tower object and updating it based on an input enumerator, and retrieved resources on its own. This left a lot of responsibility on this one method which would make it hard to update. Notably, there are 2 cases per tower type in the method, which would cause this method to very quickly become unweildy once we added more towers.

## Fixed version

```
  public void enterTowerPlacementMode(Tower tower) {
        Scene scene = ((Stage) gamePane.getScene().getWindow()).getScene();
        if (buyModeHandler != null) {
            TDApp.showErrorMsg("Invalid State", "You're already buying something else");
            return;
        }
        buyModeHandler = handle(mouseEvent) -> {
                int x = (int) mouseEvent.getSceneX();
```

```
                int y = (int) mouseEvent.getSceneY();
                x -= Tower.SIZE / 2; // This is so that where you click is the center of where
                y -= Tower.SIZE / 2; // the tower is placed
                tower.setX(x);
                tower.setY(y);
                if (game.getGame().isInvalidTowerLocation(tower)) {
                    TDApp.showErrorMsg("Invalid Location", "You can't place a tower there");
                    return;
                }
                game.addTower(tower);
                game.setMoney(game.getGame().getMoney() - tower.getPrice(game.getGame().getDiffic
                scene.removeEventHandler(MouseEvent.MOUSE_CLICKED, this);
                buyModeHandler = null;
                exitTowerPlacementMode();
            }
        };
        scene.addEventHandler(MouseEvent.MOUSE_CLICKED, buyModeHandler);
        Image image = new Image(TDApp.getResourcePath(tower.getImgPath()));
        scene.setCursor(new ImageCursor(image));
    }
```

I fixed this code smell by offloading much of the work to other methods which vastly reduces the method length, but also increases cohesion. The tower object is passed into the method, which removes the need for the method to determine what type of tower to create using the enumerator, and instead of having to create it, it only needs to update a couple of values. It also offloads getting the image resource to the tower object instead of handling that itself.

## Middle Man - TowerUI (M4)

```
public abstract class TowerUI extends Rectangle {

    private final Tower tower;

    public TowerUI(Tower tower) {
        super(tower.getX(),
              tower.getY(),
              tower.getWidth(),
              tower.getHeight());
        setFill(new ImagePattern(new Image(TDApp.getResourcePath(tower.getImgPath()))));
        this.tower = tower;
    }

    public Tower getTower() {
        return tower;
    }
}
```

The TowerUI class extends the rectangle class, and contains a tower object which contains all the information about the tower in question. This code smell exists to further decouple the UI from the game logic. However, this just adds extra complexity for little benefit.

### Fixed Version

```
public abstract class Tower extends Rectangle {

    public static final int SIZE = 50;
    protected int attackSpeed;
    protected int attackDamage;
    protected Map<GameDifficulty, Integer> statsPerDifficulty;

    public Tower(int locationX, int locationY, int attackSpeed, int attackDamage, String imageLoc
        super(locationX, locationY, SIZE, SIZE);
        setFill(new ImagePattern(new Image(TDApp.getResourcePath(imageLocation))));
        this.attackSpeed = attackSpeed;
        this.attackDamage = attackDamage;
        this.statsPerDifficulty = new HashMap<>();
    }

    public int getPrice(GameDifficulty difficulty) { return statsPerDifficulty.get(difficulty); }

    public int getAttackSpeed() { return attackSpeed; }

    public int getAttackDamage() { return attackDamage; }
}
```

I fixed this code smell by removing the extra layer between the Tower object and the data behind it. Now, the Tower itself extends the rectangle and contains all the relevant information that needs to be accessed instead of being locked behind another layer.

## Switch statements - getPrice() (M3)

```
public class BasicTower extends Tower {

    public static final String BASIC_TOWER_IMAGE = "images/obamaBasic.png";
    public static final int BASIC_EASY_PRICE = 50;
    public static final int BASIC_MEDIUM_PRICE = 75;
    public static final int BASIC_HARD_PRICE = 100;
    private static final int INIT_ATTACK_SPEED = 0;
    private static final int INIT_ATTACK_DAMAGE = 0;
```

```
    ...

    @Override
    public int getPrice(GameDifficulty difficulty) {
        switch (difficulty) {
        case EASY:
            return BASIC_EASY_PRICE;
        case MEDIUM:
            return BASIC_MEDIUM_PRICE;
        case HARD:
            return BASIC_HARD_PRICE;
        default:
            return -1; // This shouldn't happen.
        }
    }
}
```

To get the price based on the difficulty, our code used a switch statement based on an input which contained the difficulty of the game. Then the method would return a public static constant. This switch statement adds needless complexity to the `getPrice()` method, and makes the method much longer than it needs to be.

## Fixed Version

```
public abstract class Tower extends SymbolicGameObject {
    // other code here

    protected Map<GameDifficulty, Integer> statsPerDifficulty;

    // other code here

    public int getPrice(GameDifficulty difficulty) {
        return statsPerDifficulty.get(difficulty);
    }

    // other code here
}



public class BasicTower extends Tower {

    // other code here

    private static final HashMap<GameDifficulty, Integer> basicStatsPerDifficulty;
    static {
        basicStatsPerDifficulty = new HashMap<>();
```

```
        basicStatsPerDifficulty.put(GameDifficulty.EASY, 50);
        basicStatsPerDifficulty.put(GameDifficulty.MEDIUM, 75);
        basicStatsPerDifficulty.put(GameDifficulty.HARD, 100);
    }

    public BasicTower(int locationX, int locationY) {
        super(locationX, locationY, INIT_ATTACK_SPEED, INIT_ATTACK_DAMAGE, BASIC_TOWER_IMAGE);
        statsPerDifficulty = basicStatsPerDifficulty;
    }

    // other code here
}
```

I fixed this code smell by replacing the switch case blocks with a Hashmap. Towers no longer need to override the getPrice() by using a cascading if block or switch statement, but just need to update a HashMap when the object is initialized. I did this by creating a static private variable with the price information and adding the values using a static initializer. After this, the getPrice() method can get the price from the map using a difficulty value.

## Lazy Class - Monument (M4)

```
public class Monument extends SymbolicGameObject {

    public static final String IMG_PATH = "images/monument.jpg";
    private static final int SIZE = 200;

    public Monument(int x, int y) {
        super(x, y, SIZE, SIZE, IMG_PATH);
    }

    public boolean collidesWithMonument(SymbolicGameObject gameObj) {
        return hasCollision(gameObj);
    }
}
```

We represent the monument with a Monument class. It extends the SymbolicGameObject class which contains information like size, positiong, image resources, detecting collisions and more. However, the Monument basically nothing on top of SymbolicGameObject and functionally has no use.

### Fixed Version

```
// Delete the file lol
```

I fixed this code smell by deleting the class. To replace the functionality of this class, we can use a generic SymbolicGameObject object to represent the Monument. To detect collisions with the monument, we can use SymbolicGameObject's hasCollision method (as that's what colliesWithMonument was doing anyways).