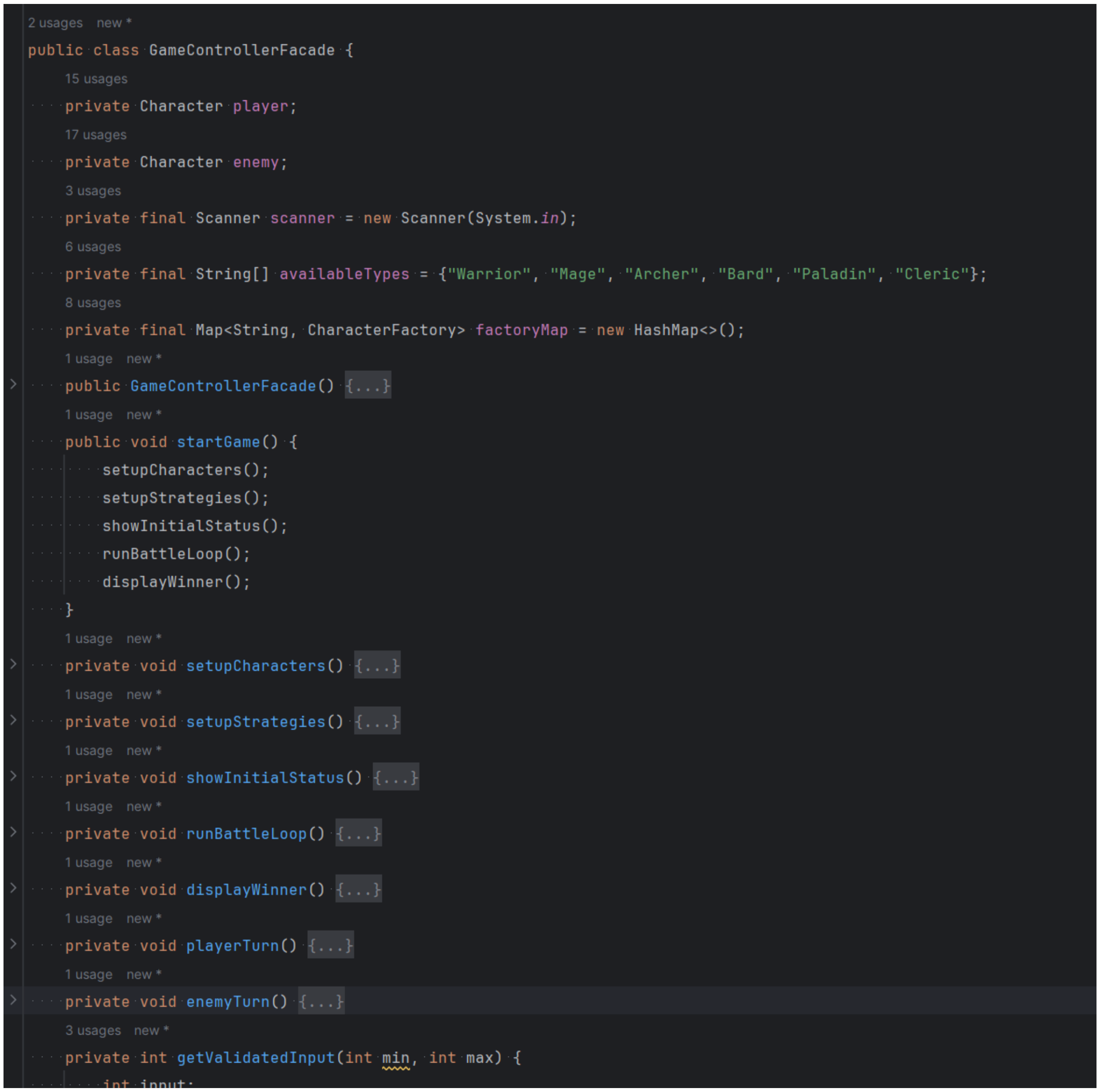
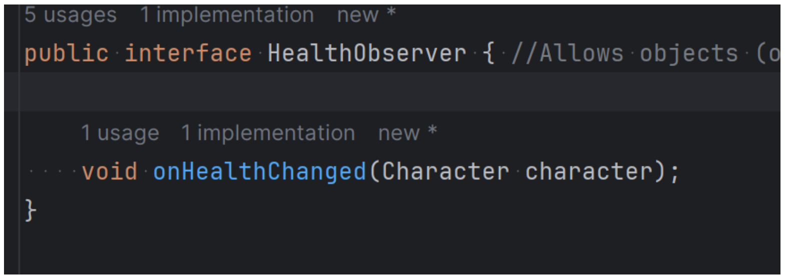
Text-Based Battle Game

Façade Pattern – Game Controller



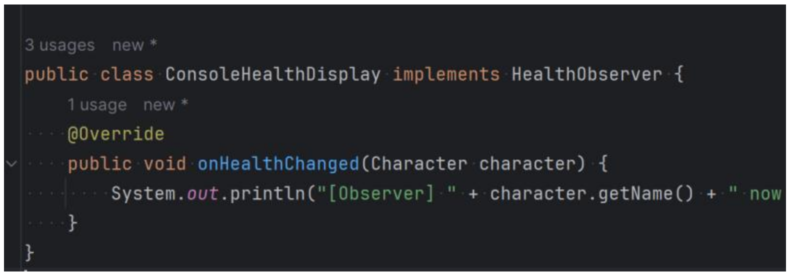
```
2 usages   new *
public class GameControllerFacade {
    15 usages
    private Character player;
    17 usages
    private Character enemy;
    3 usages
    private final Scanner scanner = new Scanner(System.in);
    6 usages
    private final String[] availableTypes = {"Warrior", "Mage", "Archer", "Bard", "Paladin", "Cleric"};
    8 usages
    private final Map<String, CharacterFactory> factoryMap = new HashMap<>();
    1 usage   new *
    public GameControllerFacade() {...}
    1 usage   new *
    public void startGame() {
        setupCharacters();
        setupStrategies();
        showInitialStatus();
        runBattleLoop();
        displayWinner();
    }
    1 usage   new *
    private void setupCharacters() {...}
    1 usage   new *
    private void setupStrategies() {...}
    1 usage   new *
    private void showInitialStatus() {...}
    1 usage   new *
    private void runBattleLoop() {...}
    1 usage   new *
    private void displayWinner() {...}
    1 usage   new *
    private void playerTurn() {...}
    1 usage   new *
    private void enemyTurn() {...}
    3 usages   new *
    private int getValidatedInput(int min, int max) {
        int input;
```

Observer – Player Health



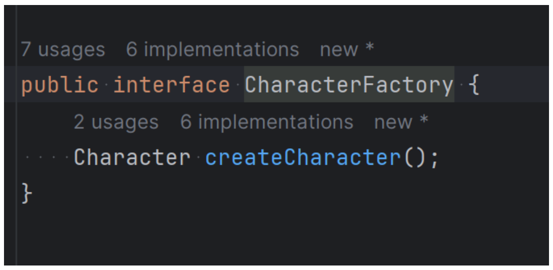
```
5 usages   1 implementation   new *
public interface HealthObserver { //Allows objects (o

    1 usage   1 implementation   new *
    void onHealthChanged(Character character);
}
```

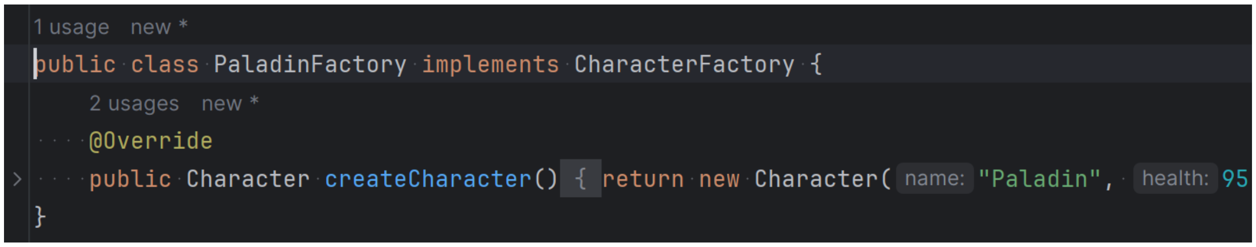


```
3 usages   new *
public class ConsoleHealthDisplay implements HealthObserver {
    1 usage   new *
    @Override
    public void onHealthChanged(Character character) {
        System.out.println("[Observer] " + character.getName() + " now
    }
}
```

Factory – Character Creation

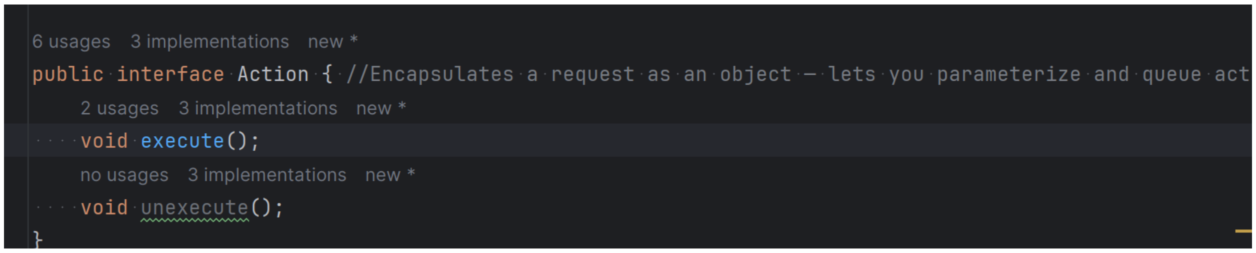


```
7 usages   6 implementations   new *
public interface CharacterFactory {
    2 usages   6 implementations   new *
    Character createCharacter();
}
```

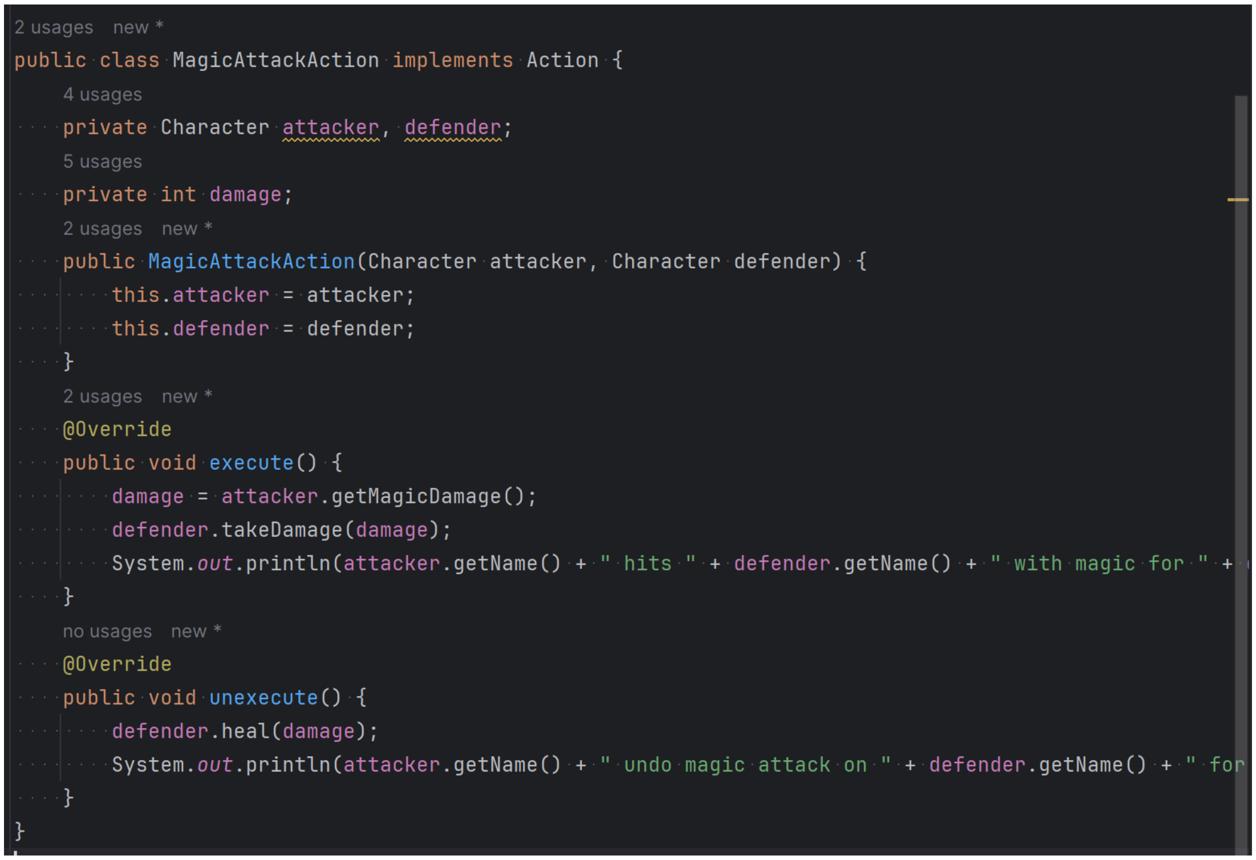


```
1 usage   new *
public class PaladinFactory implements CharacterFactory {
    2 usages   new *
    @Override
    public Character createCharacter() { return new Character( name: "Paladin",  health: 95
}
```

Command – Actions

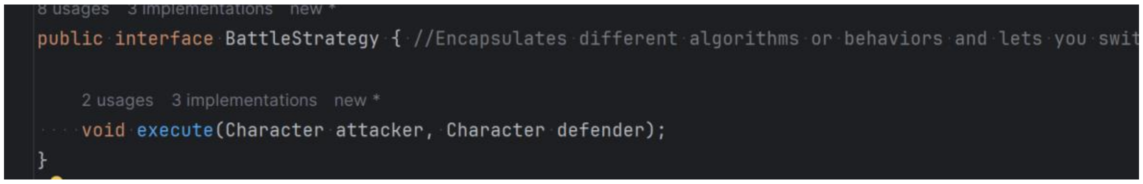


```
6 usages   3 implementations   new *
public interface Action { //Encapsulates a request as an object — lets you parameterize and queue act
    2 usages   3 implementations   new *
    void execute();
    no usages   3 implementations   new *
    void unexecute();
}
```



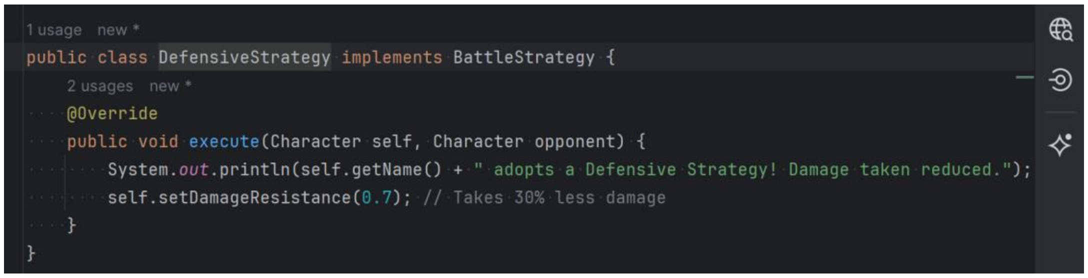
```
2 usages   new *
public class MagicAttackAction implements Action {
    4 usages
    private Character attacker, defender;
    5 usages
    private int damage;
    2 usages   new *
    public MagicAttackAction(Character attacker, Character defender) {
        this.attacker = attacker;
        this.defender = defender;
    }
    2 usages   new *
    @Override
    public void execute() {
        damage = attacker.getMagicDamage();
        defender.takeDamage(damage);
        System.out.println(attacker.getName() + " hits " + defender.getName() + " with magic for " +
    }
    no usages   new *
    @Override
    public void unexecute() {
        defender.heal(damage);
        System.out.println(attacker.getName() + " undo magic attack on " + defender.getName() + " for
    }
}
```

Strategy – Battle Strategy



```
8 usages   3 implementations   new *
public interface BattleStrategy { //Encapsulates different algorithms or behaviors and lets you swit

    2 usages   3 implementations   new *
    void execute(Character attacker, Character defender);
}
```



```
1 usage   new *
public class DefensiveStrategy implements BattleStrategy {
    2 usages   new *
    @Override
    public void execute(Character self, Character opponent) {
        System.out.println(self.getName() + " adopts a Defensive Strategy! Damage taken reduced.");
        self.setDamageResistance(0.7); // Takes 30% less damage
    }
}
```