

# **LAPORAN UTS PRAKTIKUM**

Mata Kuliah : Praktikum Pemrograman Berbasis Objek

Dosen : Irsyad Arif Mashudi, S.Kom., M.Kom



**Ilham Dharma Atmaja**

**24410702020**

**Kelas :TI 2D**

**PROGRAM STUDI TEKNIK INFORMATIKA  
JURUSAN TEKNOLOGI INFORMASI  
POLITEKNIK NEGERI MALANG TAHUN 2025**

[illegible]

2. **Enemy.java** (Abstract Class): *Extends* Character (Inheritance); menambahkan threatLevel tervalidasi dan *field* AttackStrategy untuk perilaku serangan.

```

1 public abstract class Enemy extends Character {
2     private int threatLevel; // 0..5 (1000..30)
3     protected AttackStrategy strategy; // 1000..30
4
5     protected Enemy(String name, int hp, int mp, int threatLevel, AttackStrategy strategy) {
6         setName(name, hp, mp);
7         if (threatLevel < 1 || threatLevel > 5) { // validate threatLevel (size: 30)
8             throw new IllegalArgumentException("Threat level must be between 1 and 5.");
9         }
10        this.threatLevel = threatLevel;
11        setStrategy(strategy);
12    }
13
14    public final int getThreatLevel() { return threatLevel; }
15
16    public final void setStrategy(AttackStrategy s) {
17        if (s == null) {
18            throw new IllegalArgumentException("AttackStrategy cannot be null.");
19        }
20        this.strategy = s;
21    }
22 }

```

3. **Player.java** (Concrete Class): *Extends* Character; menerapkan Polymorphism dengan menggunakan AttackStrategy dan logika kondisional untuk menggunakan *skill* (HealSkill/PiercingStrike) dalam attack().

[illegible]

4. **Monster.java** (Concrete Class): *Extends* Enemy; mengimplementasikan attack() dengan *damage* berbasis *strategy* dan *randomization*, mewujudkan Polymorphism.

```
1 import java.util.Random;
2
3 public class Monster extends Enemy {
4     private final Random random = new Random();
5
6     public Monster(String name, int hp, int ap, int threatlevel, AttackStrategy strategy) {
7         super(name, hp, ap, threatlevel, strategy);
8     }
9
10    @Override
11    public void attack(Character target) {
12        // Damage dihitung menggunakan strategy [cite: 11]
13        int baseDamage = strategy.computeDamage(this, target);
14
15        // Randomisasi damage (rata-rata 80% - 100%) [cite: 12]
16        int minDamage = (int) (baseDamage * 0.8);
17        int maxDamage = (int) (baseDamage * 1.1);
18        int finalDamage = random.nextInt(maxDamage - minDamage + 1) + minDamage;
19
20        System.out.printf("Monster attacks (Base: %d, Final: %d dmg)", baseDamage, finalDamage);
21        target.takeDamage(finalDamage);
22    }
23
24    @Override
25    public String toString() {
26        return String.format("Monster(name=%s, HP=%d/%d, AP=%d, Threat=%d)",
27            getName(), getHealth(), getMaxHealth(), getAttackPower(), getThreatLevel());
28    }
29 }
```

5. **BossMonster.java** (Concrete Class): *Extends* Enemy; mengimplementasikan Polymorphism di attack() dengan logika *Rage Strike* ( $2 \times$  damage) yang dipicu oleh HP  $< 50\%$  atau setiap 3 giliran.

```
1 public class BossMonster extends Enemy {
2     private int turnCounter = 0; // State internal untuk Rage Strike [cite: 18]
3
4     public BossMonster(String name, int hp, int ap, int threatlevel, AttackStrategy strategy) {
5         super(name, hp, ap, threatlevel, strategy);
6     }
7
8     @Override
9     public void attack(Character target) {
10         turnCounter++; // Hitung giliran
11
12         boolean isLowHealth = getHealth() < (getMaxHealth() * 0.5);
13         boolean isThirdTurn = (turnCounter % 3 == 0);
14
15         // Cek Rage Strike trigger
16         if (isLowHealth || isThirdTurn) {
17             // Rage Strike: 2x damage [cite: 19]
18             int baseDamage = strategy.computeDamage(this, target);
19             int rageDamage = (int) (baseDamage * 2.0);
20
21             String trigger = isLowHealth ? "HP < 50% : " : "3rd turn";
22             System.out.printf("Boss attacks (RAGE STRIKE: %s) -> %d dmg", trigger, rageDamage);
23             target.takeDamage(rageDamage);
24         } else {
25             // Serangan normal
26             int damage = strategy.computeDamage(this, target);
27             System.out.printf("Boss attacks (Normal) -> %d dmg", damage);
28             target.takeDamage(damage);
29         }
30     }
31
32     @Override
33     public String toString() {
34         return String.format("BossMonster(name=%s, HP=%d/%d, AP=%d, Threat=%d)",
35             getName(), getHealth(), getMaxHealth(), getAttackPower(), getThreatLevel());
36     }
37 }
```

## B. Interface dan Implementasi

6. **Skill.java (Interface):** Mendefinisikan kontrak Abstraction (name, apply) untuk kemampuan aktif yang harus diimplementasikan.

A screenshot of a code editor showing the definition of the Skill interface. The code is as follows:

```
1 public interface Skill {
2     String name();
3     void apply(Character self, Character target);
4 }
```

7. **HealSkill.java (Implementasi Skill):** Mengimplementasikan Skill, memulihkan HP target/diri sendiri sambil divalidasi agar  $\leq \text{maxHealth}$  (Polymorphism).

A screenshot of a code editor showing the implementation of the HealSkill class. The code is as follows:

```
1 public class HealSkill implements Skill {
2     private final int amount;
3
4     public HealSkill(int amount) {
5         this.amount = Math.max(0, amount);
6     }
7
8     @Override
9     public String name() {
10        return "HealSkill (" + amount + " HP)";
11    }
12
13    @Override
14    public void apply(Character self, Character target) {
15        // Validasi kondisi diri sendiri (self) jika self
16        int oldHealth = self.getHealth();
17        self.heal(amount);
18        System.out.printf("Heals for %d HP, HP: %d to %d\n", amount, oldHealth, self.getHealth());
19    }
20 }
```

8. **PiercingStrikeSkill.java (Implementasi Skill):** Mengimplementasikan Skill; menyediakan *damage multiplier* yang digunakan Player untuk *attack*.

A screenshot of a code editor showing the implementation of the PiercingStrikeSkill class. The code is as follows:

```
1 public class PiercingStrikeSkill implements Skill {
2     private final double multiplier; // Contoh: 1.2
3
4     public PiercingStrikeSkill(double multiplier) {
5         this.multiplier = multiplier;
6     }
7
8     @Override
9     public String name() {
10        return "PiercingStrike (" + multiplier + ")";
11    }
12
13    @Override
14    public void apply(Character self, Character target) {
15        // Skill ini diterapkan langsung di Player.attack()
16        // Tidak ada aksi langsung di sini
17    }
18
19    public double getMultiplier() {
20        return multiplier;
21    }
22 }
```

9. **StatusEffect.java** (Interface): Mendefinisikan kontrak Abstraction (onTurnStart, onTurnEnd, isExpired) untuk efek berbasis giliran.

```
1 public interface StatusEffect {
2     String name();
3     void onTurnStart(Character self);
4     void onTurnEnd(Character self);
5     boolean isExpired();
6 }
```

10. **RegenEffect.java** (Implementasi StatusEffect): Mengimplementasikan StatusEffect; memulihkan HP di onTurnEnd selama durasi tertentu.

```
1 public class RegenEffect implements StatusEffect {
2     private final int perTurn;
3     private int duration;
4
5     public RegenEffect(int perTurn, int duration) {
6         this.perTurn = perTurn;
7         this.duration = duration;
8     }
9
10    @Override
11    public String name() {
12        return "Regen (" + perTurn + " HP, " + duration + " turns)";
13    }
14
15    @Override
16    public void onTurnStart(Character self) {
17        // TODO: add logic if need giliran
18    }
19
20    @Override
21    public void onTurnEnd(Character self) {
22        if (isExpired()) return;
23
24        int oldHealth = self.getHealth();
25        self.heal(perTurn);
26        duration--;
27        System.out.printf(" [Regen] %s heals for %d HP, HP: %d --> %d, Remaining: %d turns\n",
28            self.getName(), perTurn, oldHealth, self.getHealth(), duration);
29    }
30
31    @Override
32    public boolean isExpired() {
33        return duration <= 0;
34    }
35 }
```

11. **ShieldEffect.java** (Implementasi StatusEffect): Mengimplementasikan StatusEffect; efeknya dihitung oleh Character.onIncomingDamage untuk mengurangi *flat damage* selama durasi.

```

1 public class ShieldEffect implements StatusEffect {
2     private final int flatReduce;
3     private int duration;
4
5     public ShieldEffect(int flatReduce, int duration) {
6         this.flatReduce = flatReduce;
7         this.duration = duration;
8     }
9
10    public int getFlatReduce() { return flatReduce; }
11
12    @Override
13    public String name() {
14        return "Shield (" + flatReduce + " dmg, " + duration + " turns)";
15    }
16
17    // Shield diterapkan di blok Character.unblockDamage.
18
19    @Override
20    public void onStart(Character self) {
21        // Tidak ada aksi di awal giliran
22    }
23
24    @Override
25    public void onTurnEnd(Character self) {
26        duration--;
27        System.out.printf(" [Shield] Remaining: %d turns\n", duration);
28    }
29
30    @Override
31    public boolean isExpired() {
32        return duration <= 0;
33    }
34 }

```

### C. Strategy Pattern

12. **AttackStrategy.java** (Interface): Mendefinisikan kontrak Abstraction (computeDamage) untuk memisahkan algoritma perhitungan *damage* dari karakter.

```

1 public interface AttackStrategy {
2     int computeDamage(Character self, Character target);
3 }

```

13. **FixedStrategy.java** (Implementasi Strategy): Mengimplementasikan AttackStrategy dengan *damage* tetap (hanya Attack Power).

```

1 public class FixedStrategy implements AttackStrategy {
2     @Override
3     public int computeDamage(Character self, Character target) {
4         return self.getAttackPower();
5     }
6
7     @Override
8     public String toString() {
9         return "FixedStrategy";
10    }
11 }

```

14. **LevelScaledStrategy.java** (Implementasi Strategy): Mengimplementasikan AttackStrategy dengan *damage* yang diskalakan oleh level karakter (bonusPerLevel), menunjukkan Polymorphism.



```
1 public class LevelScaledStrategy implements AttackStrategy {
2     private final int bonusPerLevel;
3
4     public LevelScaledStrategy(int bonusPerLevel) {
5         this.bonusPerLevel = bonusPerLevel;
6     }
7
8     @Override
9     public int computeDamage(Character self, Character target) {
10         int baseDamage = self.getAttackPower();
11
12         if (self instanceof Player player) {
13             // Tambah bonus per level [cite: 38]
14             baseDamage += player.getLevel() * bonusPerLevel;
15         }
16
17         return baseDamage;
18     }
19
20     @Override
21     public String toString() {
22         return String.format("LevelScaledStrategy (%d/level)", bonusPerLevel);
23     }
24 }
```

#### D. Simulasi dan Uji

15. **Battle.java** (Simulasi): Mengelola loop utama run() hingga satu tim kalah; menerapkan logika *auto-targeting* yang kompleks (Enemy \$to\$ HP tertinggi, Player \$to\$ Threat tertinggi/HP terendah).



```

1 import java.util.Comparator;
2 import java.util.List;
3 import java.util.stream.Collectors;
4
5 public class Battle {
6     private final List<Character> teamA;
7     private final List<Character> teamB;
8     private int turn = 0;
9
10    public Battle(List<Character> teamA, List<Character> teamB) {
11        this.teamA = teamA.stream().filter(Character::isAlive).collect(Collectors.toList());
12        this.teamB = teamB.stream().filter(Character::isAlive).collect(Collectors.toList());
13    }
14
15    private boolean isTeamADefeated() {
16        return teamA.stream().noneMatch(Character::isAlive);
17    }
18
19    private boolean isTeamBDefeated() {
20        return teamB.stream().noneMatch(Character::isAlive);
21    }
22
23    private Character findTarget(Character attacker, List<Character> defenders) {
24        List<Character> aliveDefenders = defenders.stream().filter(Character::isAlive).toList();
25        if (aliveDefenders.isEmpty()) return null;
26
27        if (attacker instanceof Player) {
28            // Player target: enemy dengan threatlevel tertinggi, lalu HP terendah [rule: 34]
29            return aliveDefenders.stream()
30                .filter(c -> c instanceof Enemy)
31                .max(c -> {Enemy e}
32                    .max(Comparator
33                        .comparingInt(Enemy::getThreatLevel)
34                        .thenComparingInt(Enemy::getHealth)
35                        .reversed()) // HP terendah
36                )
37                .orElse(null);
38        } else if (attacker instanceof Enemy) {
39            // Enemy target: Player dengan HP tertinggi [rule: 34]
40            return aliveDefenders.stream()
41                .filter(c -> c instanceof Player)
42                .max(Comparator.comparingInt(Character::getHealth))
43                .orElse(null);
44        }
45        return aliveDefenders.get(0); // Default
46    }
47
48    public void run() {
49        System.out.println("=== BATTLE START ===");
50
51        while (!isTeamADefeated() && !isTeamBDefeated()) {
52            turn++;
53            System.out.println("\n=== TURN " + turn + " ===");
54
55            // Turn Team A
56            for (Character a : teamA) {
57                if (a.isAlive()) {
58                    Character target = findTarget(a, teamB);
59                    if (target != null) {
60                        a.performTurn(target);
61                    }
62                }
63            }
64            // Bersihkan yang mati
65            teamB.removeIf(c -> !c.isAlive());
66
67            if (isTeamBDefeated()) break;
68
69            // Turn Team B
70            for (Character b : teamB) {
71                if (b.isAlive()) {
72                    Character target = findTarget(b, teamA);
73                    if (target != null) {
74                        b.performTurn(target);
75                    }
76                }
77            }
78            // Bersihkan yang mati
79            teamA.removeIf(c -> !c.isAlive());
80
81            // Angkutan [rule: 35]
82            System.out.println("\n=== RESULT ===");
83            if (isTeamADefeated()) {
84                System.out.println("Team B menang");
85            } else if (isTeamBDefeated()) {
86                System.out.println("Team A menang");
87            } else {
88                System.out.println("Pertempuran berakhir.");
89            }
90
91            System.out.println("\nLalu HP:");
92            teamA.forEach(c -> System.out.println("  " + c));
93            teamB.forEach(c -> System.out.println("  " + c));
94        }
95    }
96 }

```

16. **GameTest.java** (Uji dan Validasi): Kelas utama yang mengatur dan menginisialisasi semua objek (*Player*, *BossMonster*, *Skills*, *Effects*, *Strategies*) dan menjalankan simulasi Battle.

```

1 import java.util.List;
2
3 public class GameTest {
4     public static void main(String[] args) {
5         // --- SETUP KAWANITU ---
6
7         // Player dengan LevelScaledStrategy
8         Player p = new Player("HeroVipkas", 120, 25, 5, new LevelScaledStrategy(2));
9         p.addSkill(new HealSkill(15));
10        p.addSkill(new PiercingStrikeSkill(1.2));
11
12        // BossMonster dengan FixedStrategy
13        BossMonster boss = new BossMonster("Drake", 150, 28, 5, new FixedStrategy());
14
15        // Monster dengan FixedStrategy
16        Monster monster = new Monster("Goblin", 80, 12, 2, new FixedStrategy());
17
18        // --- SETUP IPIS ADAM ---
19        // Tambah efek awal (cite: 27)
20        p.addEffect(new ShieldEffect(10, 3)); // Mengurangi 10 dmg selama 3 turn
21        p.addEffect(new RegenEffect(8, 4)); // Memulihkan 8 HP selama 4 turn
22
23        // --- SETUP TEAM ---
24        List<Character> teamA = List.of(p);
25        List<Character> teamB = List.of(boss, monster);
26
27        System.out.println("=== SETUP ===");
28        System.out.println("Team A:");
29        teamA.forEach(c -> {
30            System.out.println("  " + c);
31            if (c instanceof Player player) {
32                System.out.println("    Skills: " + player.getSkills().stream().map(skill::name).toList());
33            }
34            System.out.println("    Effects: " + c.getEffects().stream().map(effect::name).toList());
35        });
36        System.out.println("Team B:");
37        teamB.forEach(c -> System.out.println("  " + c));
38        System.out.println("-----");
39
40        // --- SALAMAN BATTLE ---
41        Battle battle = new Battle(teamA, teamB);
42        battle.run();
43    }
44 }

```

## E. Verifikasi Output

### 1. Setup

```

=== SETUP ===
Team A:
- Player(name=HeroVipkas, HP=120/120, AP=25, Lv=5)
  Skills: [HealSkill (+15 HP), PiercingStrike (x1.2)]
  Effects: [Shield (-10 dmg, 3 turns), Regen (+8 HP, 4 turns)]
Team B:
- BossMonster(name=Drake, HP=150/150, AP=28, Threat=5)
- Monster(name=Goblin, HP=80/80, AP=12, Threat=2)

```

### 2. Battle Start

```

=== BATTLE START ===

--- TURN 1 ---
[Start Turn] HeroVipkas (HP: 120/120, AP: 25)
HeroVipkas attacks Goblin -> Player attacks (Strategy: 35) + PiercingStrike (x1.2) (x1,2) -> 42 dmg  Goblin HP: 38/80
[Shield] Remaining: 2 turns
[Regen] HeroVipkas heals for 8 HP. HP: 120 -> 128. Remaining: 3 turns
[Start Turn] Drake (HP: 150/150, AP: 28)
Drake attacks HeroVipkas -> Boss attacks (Normal) -> 28 dmg (Shield absorbs 10 dmg)  HeroVipkas HP: 102/120
[Start Turn] Goblin (HP: 38/80, AP: 12)
Goblin attacks HeroVipkas -> Monster attacks (Base: 12, Final: 10 dmg) (Shield absorbs 10 dmg)  HeroVipkas HP: 102/120

--- TURN 2 ---
[Start Turn] HeroVipkas (HP: 102/120, AP: 25)
HeroVipkas attacks Goblin -> Player attacks (Strategy: 35) + PiercingStrike (x1.2) (x1,2) -> 42 dmg  Goblin HP: 0/80
[Shield] Remaining: 1 turns
[Regen] HeroVipkas heals for 8 HP. HP: 102 -> 110. Remaining: 2 turns
[Start Turn] Drake (HP: 150/150, AP: 28)
Drake attacks HeroVipkas -> Boss attacks (Normal) -> 28 dmg (Shield absorbs 10 dmg)  HeroVipkas HP: 92/120

--- TURN 3 ---
[Start Turn] HeroVipkas (HP: 92/120, AP: 25)
HeroVipkas attacks Drake -> Player attacks (Strategy: 35) + PiercingStrike (x1.2) (x1,2) -> 42 dmg  Drake HP: 108/150
[Shield] Remaining: 0 turns
[Effect Expired] HeroVipkas: Shield (-10 dmg, 0 turns)
[Regen] HeroVipkas heals for 8 HP. HP: 92 -> 100. Remaining: 1 turns
[Start Turn] Drake (HP: 108/150, AP: 28)
Drake attacks HeroVipkas -> Boss attacks (RAGE STRIKE: 3rd turn) -> 56 dmg  HeroVipkas HP: 44/120

--- TURN 4 ---
[Start Turn] HeroVipkas (HP: 44/120, AP: 25)
HeroVipkas attacks Drake -> Player uses HealSkill (+15 HP) -> HeroVipkas heals for 15 HP. HP: 44 -> 59
Drake HP: 108/150
[Regen] HeroVipkas heals for 8 HP. HP: 59 -> 67. Remaining: 0 turns
[Effect Expired] HeroVipkas: Regen (+8 HP, 0 turns)
[Start Turn] Drake (HP: 108/150, AP: 28)
Drake attacks HeroVipkas -> Boss attacks (Normal) -> 28 dmg  HeroVipkas HP: 39/120

--- TURN 5 ---
[Start Turn] HeroVipkas (HP: 39/120, AP: 25)
HeroVipkas attacks Drake -> Player uses HealSkill (+15 HP) -> HeroVipkas heals for 15 HP. HP: 39 -> 54
Drake HP: 108/150
[Start Turn] Drake (HP: 108/150, AP: 28)
Drake attacks HeroVipkas -> Boss attacks (Normal) -> 28 dmg  HeroVipkas HP: 26/120

--- TURN 6 ---
[Start Turn] HeroVipkas (HP: 26/120, AP: 25)
HeroVipkas attacks Drake -> Player uses HealSkill (+15 HP) -> HeroVipkas heals for 15 HP. HP: 26 -> 41
Drake HP: 108/150
[Start Turn] Drake (HP: 108/150, AP: 28)
Drake attacks HeroVipkas -> Boss attacks (RAGE STRIKE: 3rd turn) -> 56 dmg  HeroVipkas HP: 0/120

```

### 3. Result

```

=== RESULT ===
Team B menang!

Sisa HP:
- BossMonster(name=Drake, HP=108/150, AP=28, Threat=5)

```

## II. Kesimpulan

Game ini adalah simulasi pertarungan bergantian (turn-based) yang dibangun dengan sangat terstruktur menggunakan aturan **OOP**. Intinya, semua yang ada di game adalah "karakter" yang memiliki aturan dasar yang sama, tetapi dengan perilaku yang sangat berbeda.

### 1. Fondasi yang Sama untuk Semua

Semua unit, dari Player sampai Boss, berasal dari kerangka dasar **Character** (konsep **Abstraction**). Kerangka ini mengatur hal-hal dasar yang tidak bisa diubah: cara giliran dijalankan (**performTurn**), cara mendapatkan status efek, dan perlindungan datanya (seperti HP yang tidak boleh negatif, ini **Encapsulation**).

### 2. Spesialisasi dan Perilaku Unik

Dari kerangka dasar itu, muncullah spesialisasi (**Inheritance**):

- **Player** bisa punya Level dan Skill.
- **Enemy** punya Level Bahaya (threatLevel).

- **BossMonster** menambahkan aturan unik (seperti *Rage Strike* yang diaktifkan jika HP rendah atau setiap 3 giliran) di metode `attack()`-nya. Semua variasi serangan ini adalah **Polymorphism**.

### 3. Logika yang Fleksibel (Interface)

Untuk membuat game fleksibel, logika yang sering berubah dipisahkan menjadi *Interface* (**Abstraction**):

- **AttackStrategy** mengatur bagaimana *damage* dihitung (apakah tetap, atau diskalakan berdasarkan level).
- **StatusEffect** (seperti Shield atau Regen) mengatur kapan efeknya diterapkan (awal atau akhir giliran) dan kapan efeknya hilang. Efek *Shield* bekerja langsung di *hook* kerusakan karakter, menunjukkan kerjasama antar kelas.

### 4. Alur Pertarungan (Battle)

Kelas **Battle** bertindak sebagai wasit. Ia memastikan setiap karakter mendapat giliran, memanggil `performTurn()`, dan menangani pemilihan target secara cerdas:

- Musuh selalu mengincar Player terkuat (HP tertinggi).
- Player selalu mengincar Musuh paling berbahaya (`threatLevel` tertinggi) atau yang paling lemah HP-nya.

Singkatnya, **game ini bekerja dengan memecah setiap perilaku menjadi blok-blok OOP yang terpisah**, memastikan sistem inti (Character) stabil sementara perilaku serangan dan efek (Strategy, Skill, BossMonster) bisa diubah sesuka hati.