LAPORAN PRAKTIKUM
PEMROGRAMAN II
MODUL 6

SOLID PRINCIPLE DAN DESIGN PATTERN

Oleh:

Rizki Adhitiya Maulana **NIM. 2410817110014**

PROGRAM STUDI TEKNOLOGI INFORMASI

FAKULTAS TEKNIK

UNIVERSITAS LAMBUNG MANGKURAT

NOVEMBER 2025

LEMBAR PENGESAHAN
LAPORAN PRAKTIKUM PEMROGRAMAN II
MODUL 6

Laporan Praktikum Pemrograman II Modul 6: SOLID Principle dan Design Pattern ini disusun sebagai syarat lulus mata kuliah Praktikum Pemrograman II. Laporan Praktikum ini dikerjakan oleh:

Nama Praktikan : Rizki Adhitiya Maulana
NIM : 2410817110014

Menyetujui,	Mengetahui,
Asisten Praktikum	Dosen Penanggung Jawab Praktikum
Jovan Gilbert Natamasindah	Irham Maulani Abdul Gani, S.Kom.,
NIM. 2310817310002	M.Kom.
	NIP. 199710312025061009

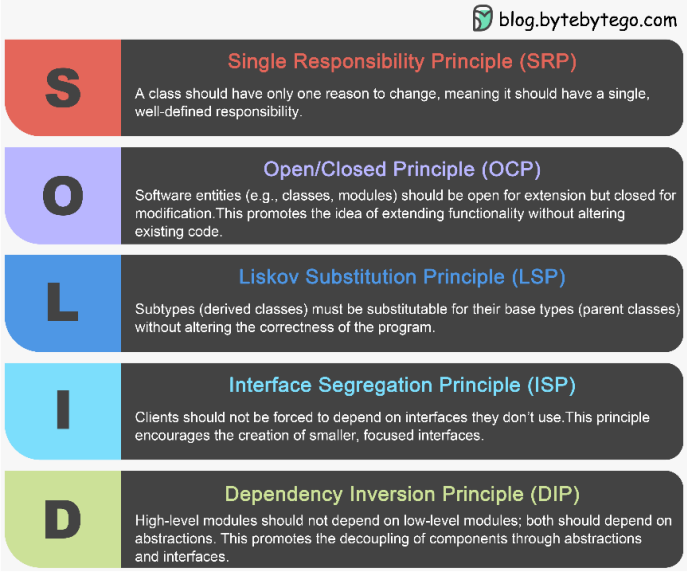
DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

SOAL 1

Berdasarkan tugas CLI template yang telah anda buat masing-masing sebelumnya. Sekarang ubah atau **lakukan refactor terhadap code CLI tersebut untuk menerapkan masing-masing prinsip SOLID**



Gambar 1. SOLID Principle

Kemudian setelah melakukan implementasi dari SOLID, selanjutnya silahkan **pilih masing-masing 1 design pattern dari 3 jenis design pattern untuk diimplementasikan ke kode** tadi dari link ini https://refactoring.guru/design-patterns/catalog (kecuali singleton)

Catatan:

- Untuk format laprak sama seperti yang sebelum-sebelumnya.
- Untuk bagian source code, pisah jadi dua bagian, yaitu berisikan implementasi SOLID dan berisikan implementasi SOLID dengan design pattern yang dipilih.
- Untuk bagian penjelasan, hanya menjelaskan di bagian mana masing-masing bagian SOLID diterapkan dan penjelasan design pattern yang dipilih dengan detail.

A. Source Code

Tabel 1. Source Code - Main.java - Implementasi Single Responsibility Principle

```
1   package DESIGN_PATTERN;
2
3   import DESIGN_PATTERN.core.Game;
4
5   public class Main
6   {
7       public static void main(String[] args)
8       {
9           Game game = new Game();
10          game.start();
11      }
12  }
```

Tabel 2. Source Code - Game.java - Implementasi Single Responsibility Principle

```
1   package DESIGN_PATTERN.core;
2
3   import java.util.Scanner;
4
5   public class Game
6   {
7       private final Scanner scanner = new
    Scanner(System.in);
8       private Player player;
9
10      public void start()
11      {
12          createCharacter();
13          MenuHandler menuHandler = new MenuHandler(this,
    scanner);
14          menuHandler.showMainMenu();
15      }
16
17      private void createCharacter()
18      {
19          System.out.println("======= SELAMAT DATANG DI
    STARDUST VALLEY =======");
20
21          System.out.print("Masukkan nama karakter Anda
    : ");
22          String name = scanner.nextLine();
23
24          System.out.print("Pilih karakter Anda gender
    (L/P)    : ");
25          String genderInput = scanner.nextLine();
26
```

```
27          String gender;
28          if (genderInput.equalsIgnoreCase("L"))
29          {
30              gender = "Laki-laki";
31          }
32          else if (genderInput.equalsIgnoreCase("P"))
33          {
34              gender = "Perempuan";
35          }
36          else
37          {
38              gender = "Tidak diketahui";
39          }
40
41          System.out.print("Masukkan hobi utama karakter
    Anda  : ");
42          String hobby = scanner.nextLine();
43
44          this.player = new Player(name, gender, hobby);
45
46          System.out.println("\nKarakter berhasil
    dibuat!");
47          System.out.println("Nama   : " + name);
48          System.out.println("Gender : " + gender);
49          System.out.println("Hobi   : " + hobby);
50      }
51
52      public Player getPlayer()
53      {
54          return player;
55      }
56  }
```

Tabel 3. Source Code - MenuHandler.java - Implementasi Single Responsibility Principle

```
1   package DESIGN_PATTERN.core;
2
3   import java.util.Scanner;
4   import DESIGN_PATTERN.activities.Activity;
5   import
    DESIGN_PATTERN.activities.LoggingActivityDecorator;
6   import DESIGN_PATTERN.energy.EnergyStrategy;
7   import DESIGN_PATTERN.energy.FixedEnergyStrategy;
8   import DESIGN_PATTERN.factory.ActivityToolFactory;
9
10  public class MenuHandler
11  {
12      private final Game game;
13      private final Scanner scanner;
```

```
14
15      public MenuHandler(Game game, Scanner scanner)
16      {
17          this.game = game;
18          this.scanner = scanner;
19      }
20
21      public void showMainMenu()
22      {
23          while (true)
24          {
25              System.out.println("\n=== STARDUST VALLEY
    CLI ===");
26              System.out.println("1. Lakukan Aktivitas");
27              System.out.println("2. Lihat Status
    Pemain");
28              System.out.println("3. Istirahat");
29              System.out.println("0. Keluar");
30              System.out.print("Pilih menu: ");
31
32              String choice = scanner.nextLine();
33              switch (choice)
34              {
35                  case "1" -> showActivityMenu();
36                  case "2" ->
    game.getPlayer().showStatus();
37                  case "3" ->
    game.getPlayer().restoreEnergy();
38                  case "0" -> {
39                      System.out.println("Terima kasih
    telah bermain!");
40                      return;
41                  }
42                  default -> System.out.println("Pilihan
    tidak valid!");
43              }
44          }
45      }
46
47      private void showActivityMenu()
48      {
49          Player player = game.getPlayer();
50
51          System.out.println("\n=== PILIH AKTIVITAS ===");
52          System.out.println("1. Bertani");
53          System.out.println("2. Menambang");
54          System.out.println("3. Memancing");
55          System.out.println("4. Menebang Pohon");
56          System.out.print("Pilih aktivitas: ");
```

```
57
58          String choice = scanner.nextLine();
59
60          ActivityToolFactory factory = new
    ActivityToolFactory();
61          var result = factory.create(choice);
62
63          if (result == null)
64          {
65              System.out.println("Aktivitas tidak
    dikenal!");
66              return;
67          }
68
69          Activity activity = result.activity();
70
71          Activity loggedActivity = new
    LoggingActivityDecorator(activity);
72          EnergyStrategy strategy = new
    FixedEnergyStrategy();
73          player.performActivity(loggedActivity,
    strategy);
74      }
75  }
```

Tabel 4. Source Code - Player.java - Implementasi Single Responsibility Principle & Single Responsibility Principle

```
1   package DESIGN_PATTERN.core;
2
3   import DESIGN_PATTERN.activities.Activity;
4   import DESIGN_PATTERN.energy.EnergyStrategy;
5
6   public class Player
7   {
8       private final String name;
9       private final String gender;
10      private final String hobby;
11      private int energy;
12
13      public Player(String name, String gender, String
    hobby)
14      {
15          this.name = name;
16          this.gender = gender;
17          this.hobby = hobby;
18          this.energy = 100;
19      }
20
21      public int getEnergy()
```

```
22      {
23          return energy;
24      }
25
26      public void performActivity(Activity activity,
    EnergyStrategy strategy)
27      {
28          if (strategy.getEnergy(this) <= 0) {
29              System.out.println("Kamu sudah kelelahan dan
    tidak bisa melakukan aktivitas!");
30              return;
31          }
32
33          activity.perform(this, strategy);
34      }
35
36      public void reduceEnergy(int amount)
37      {
38          energy -= amount;
39          if (energy < 0) energy = 0;
40      }
41
42      public void restoreEnergy()
43      {
44          energy = 100;
45          System.out.println("Energi telah dikembalikan!
    Energi sekarang: " + energy);
46      }
47
48      public void showStatus()
49      {
50          System.out.println("\n=== STATUS PEMAIN ===");
51          System.out.println("Nama   : " + name);
52          System.out.println("Gender : " + gender);
53          System.out.println("Hobi   : " + hobby);
54          System.out.println("Energi : " + energy);
55      }
56  }
```

Tabel 5. Source Code - Activity.java - Implementasi Open/Closed Principle

```
1   package DESIGN_PATTERN.activities;
2
3   import DESIGN_PATTERN.core.Player;
4   import DESIGN_PATTERN.energy.EnergyStrategy;
5
6   public abstract class Activity
7   {
8       public abstract void perform(Player player,
    EnergyStrategy energyStrategy);
9   }
```

Tabel 6. Source Code - Farming.java - Implementasi Liskov Substitution Principle

```
1   package DESIGN_PATTERN.activities;
2
3   import DESIGN_PATTERN.core.Player;
4   import DESIGN_PATTERN.energy.EnergyStrategy;
5
6   public class Farming extends Activity
7   {
8       @Override
9       public void perform(Player player, EnergyStrategy
    strategy)
10      {
11          System.out.println("Kamu menanam tanaman dan
    menyiram tanaman!");
12          int cost = strategy.calculateEnergyCost(10);
13          player.reduceEnergy(cost);
14      }
15  }
```

Tabel 7. Source Code - Fishing.java - Implementasi Liskov Substitution Principle

```
1   package DESIGN_PATTERN.activities;
2
3   import DESIGN_PATTERN.core.Player;
4   import DESIGN_PATTERN.energy.EnergyStrategy;
5
6   public class Fishing extends Activity
7   {
8       @Override
9       public void perform(Player player, EnergyStrategy
    strategy)
10      {
11          System.out.println("Kamu memancing dan
    mendapatkan ikan!");
12          int cost = strategy.calculateEnergyCost(10);
13          player.reduceEnergy(cost);
```

```
14      }
15  }
```

Tabel 8. Source Code - Mining.java - Implementasi Liskov Substitution Principle

```
1   package DESIGN_PATTERN.activities;
2
3   import DESIGN_PATTERN.core.Player;
4   import DESIGN_PATTERN.energy.EnergyStrategy;
5
6   public class Mining extends Activity
7   {
8       @Override
9       public void perform(Player player, EnergyStrategy
    strategy)
10      {
11          System.out.println("Kamu menambang dan menemukan
    batu!");
12          int cost = strategy.calculateEnergyCost(10);
13          player.reduceEnergy(cost);
14      }
15  }
```

Tabel 9. Source Code - Woodcutting.java - Implementasi Liskov Substitution Principle

```
1   package DESIGN_PATTERN.activities;
2
3   import DESIGN_PATTERN.core.Player;
4   import DESIGN_PATTERN.energy.EnergyStrategy;
5
6   public class Woodcutting extends Activity
7   {
8       @Override
9       public void perform(Player player, EnergyStrategy
    strategy)
10      {
11          System.out.println("Kamu menebang pohon dan
    mendapatkan kayu!");
12          int cost = strategy.calculateEnergyCost(10);
13          player.reduceEnergy(cost);
14      }
15  }
```

Tabel 10. Source Code - Tool.java - Implementasi Open/Closed Principle

```
1   package DESIGN_PATTERN.tools;
2
3   public abstract class Tool
4   {
```

```
5       protected String name;
6
7       public Tool(String name)
8       {
9           this.name = name;
10      }
11
12      public String getName()
13      {
14          return name;
15      }
16  }
```

Tabel 11. Source Code - Tool.java – Implementasi Interface Segregation Principle

```
1   package DESIGN_PATTERN.tools;
2
3   public interface Usable
4   {
5       void use();
6   }
```

Tabel 12. Source Code - Axe.java - Implementasi Liskov Substitution Principle

```
1   package DESIGN_PATTERN.tools;
2
3   public class Axe extends Tool implements Usable
4   {
5       public Axe()
6       {
7           super("Axe");
8       }
9
10      @Override
11      public void use()
12      {
13          System.out.println("Kamu menggunakan " +
    getName() + " untuk menebang pohon...");
14      }
15  }
```

Tabel 13. Source Code - FishingRod.java - Implementasi Liskov Substitution Principle

```
1   package DESIGN_PATTERN.tools;
2
3   public class FishingRod extends Tool implements Usable
4   {
5       public FishingRod()
6       {
```

```
7           super("Fishing Rod");
8       }
9
10      @Override
11      public void use()
12      {
13          System.out.println("Kamu menggunakan " +
    getName() + " untuk memancing ikan...");
14      }
15  }
```

Tabel 14. Source Code - Pickaxe.java - Implementasi Liskov Substitution Principle

```
1   package DESIGN_PATTERN.tools;
2
3   public class Pickaxe extends Tool implements Usable
4   {
5       public Pickaxe()
6       {
7           super("Pickaxe");
8       }
9
10      @Override
11      public void use()
12      {
13          System.out.println("Kamu menggunakan " +
    getName() + " untuk memecah batu...");
14      }
15  }
```

Tabel 15. Source Code - WateringCan.java - Implementasi Liskov Substitution Principle

```
1   package DESIGN_PATTERN.tools;
2
3   public class WateringCan extends Tool implements Usable
4   {
5       public WateringCan()
6       {
7           super("Watering Can");
8       }
9
10      @Override
11      public void use()
12      {
13          System.out.println("Kamu menggunakan " +
    getName() + " untuk menyiram tanaman...");
14      }
15  }
```

Tabel 16. Source Code - ActivityDecorator.java - Implementasi Structural patterns - Decorator

```
1   package DESIGN_PATTERN.activities;
2
3   import DESIGN_PATTERN.core.Player;
4   import DESIGN_PATTERN.energy.EnergyStrategy;
5
6   public abstract class ActivityDecorator extends Activity
7   {
8       protected Activity decoratedActivity;
9
10      public ActivityDecorator(Activity activity)
11      {
12          this.decoratedActivity = activity;
13      }
14
15      @Override
16      public void perform(Player player, EnergyStrategy
    strategy)
17      {
18          decoratedActivity.perform(player, strategy);
19      }
20  }
```

Tabel 17. Source Code - LoggingActivityDecorator.java - Implementasi Structural patterns - Decorator

```
1   package DESIGN_PATTERN.activities;
2
3   import DESIGN_PATTERN.core.Player;
4   import DESIGN_PATTERN.energy.EnergyStrategy;
5
6   public class LoggingActivityDecorator extends
    ActivityDecorator
7   {
8       public LoggingActivityDecorator(Activity activity)
9       {
10          super(activity);
11      }
12
13      @Override
14      public void perform(Player player, EnergyStrategy
    strategy)
15      {
16          System.out.println("[LOG] Aktivitas
    dimulai...");
17          super.perform(player, strategy);
18          System.out.println("[LOG] Aktivitas selesai!");
19      }
20  }
```

Tabe 18. Source Code - EnergyStrategy.java - Implementasi Behavioral patterns - Strategy

```
1   package DESIGN_PATTERN.energy;
2
3   import DESIGN_PATTERN.core.Player;
4
5   public interface EnergyStrategy
6   {
7       int getEnergy(Player player);
8
9       int calculateEnergyCost(int baseCost);
10  }
```

Tabel 19. Source Code - FixedEnergyStrategy.java - Implementasi Behavioral patterns - Strategy

```
1   package DESIGN_PATTERN.energy;
2
3   import DESIGN_PATTERN.core.Player;
4
5   public class FixedEnergyStrategy implements
    EnergyStrategy
6   {
7       @Override
8       public int getEnergy(Player player)
9       {
10          return player.getEnergy();
11      }
12
13      @Override
14      public int calculateEnergyCost(int baseCost)
15      {
16          return baseCost;
17      }
18  }
```

Tabel 20. Source Code - ActivityTool.java - Implementasi Creational patterns - Factory

```
1   package DESIGN_PATTERN.factory;
2
3   import DESIGN_PATTERN.activities.Activity;
4   import DESIGN_PATTERN.tools.Usable;
5
6   public record ActivityTool(Activity activity, Usable
    tool) {}
```

Tabel 21 Source Code - ActivityToolFactory.java - Implementasi Creational patterns - Factory

```
1   package DESIGN_PATTERN.factory;
2
3   import DESIGN_PATTERN.activities.*;
```

```
4   import DESIGN_PATTERN.tools.*;
5
6   public class ActivityToolFactory
7   {
8       public ActivityTool create(String choice)
9       {
10          return switch (choice)
11          {
12              case "1" -> new ActivityTool(new Farming(),
    new WateringCan());
13              case "2" -> new ActivityTool(new Mining(),
    new Pickaxe());
14              case "3" -> new ActivityTool(new Fishing(),
    new FishingRod());
15              case "4" -> new ActivityTool(new
    Woodcutting(), new Axe());
16              default -> null;
17          };
18      }
19  }
```

B. Output Program

```
C:\Users\advan\.jdks\openjdk-24.0.2+12-54\bin\java.exe "-javaag
======= SELAMAT DATANG DI STARDUST VALLEY =======
Masukkan nama karakter Anda        : Adid
Pilih karakter Anda gender (L/P)   : L
Masukkan hobi utama karakter Anda  : Makan

Karakter berhasil dibuat!
Nama   : Adid
Gender : Laki-laki
Hobi   : Makan
```

Gambar 2. Screenshot Output Pembuatan Karakter

```
=== STARDUST VALLEY CLI ===
1. Lakukan Aktivitas
2. Lihat Status Pemain
3. Istirahat
0. Keluar
Pilih menu: 1

=== PILIH AKTIVITAS ===
1. Bertani
2. Menambang
3. Memancing
4. Menebang Pohon
Pilih aktivitas: 1
[LOG] Aktivitas dimulai...
Kamu menanam tanaman dan menyiram tanaman!
[LOG] Aktivitas selesai!
```

Gambar 3. Screenshot Output Aktivitas Bertani

```
=== STARDUST VALLEY CLI ===
1. Lakukan Aktivitas
2. Lihat Status Pemain
3. Istirahat
0. Keluar
Pilih menu: 1

=== PILIH AKTIVITAS ===
1. Bertani
2. Menambang
3. Memancing
4. Menebang Pohon
Pilih aktivitas: 2
[LOG] Aktivitas dimulai...
Kamu menambang dan menemukan batu!
[LOG] Aktivitas selesai!
```

Gambar 4. Screenshot Output Aktivitas Menambang

```
=== STARDUST VALLEY CLI ===
1. Lakukan Aktivitas
2. Lihat Status Pemain
3. Istirahat
0. Keluar
Pilih menu: 1

=== PILIH AKTIVITAS ===
1. Bertani
2. Menambang
3. Memancing
4. Menebang Pohon
Pilih aktivitas: 3
[LOG] Aktivitas dimulai...
Kamu memancing dan mendapatkan ikan!
[LOG] Aktivitas selesai!
```

Gambar 5. Screenshot Output Aktivitas Memancing

```
=== STARDUST VALLEY CLI ===
1. Lakukan Aktivitas
2. Lihat Status Pemain
3. Istirahat
0. Keluar
Pilih menu: 1

=== PILIH AKTIVITAS ===
1. Bertani
2. Menambang
3. Memancing
4. Menebang Pohon
Pilih aktivitas: 4
[LOG] Aktivitas dimulai...
Kamu menebang pohon dan mendapatkan kayu!
[LOG] Aktivitas selesai!
```

Gambar 6. Screenshot Output Aktivitas Menebang Pohon

```
=== STARDUST VALLEY CLI ===
1. Lakukan Aktivitas
2. Lihat Status Pemain
3. Istirahat
0. Keluar
Pilih menu: 2

=== STATUS PEMAIN ===
Nama   : Adid
Gender : Laki-laki
Hobi   : Makan
Energi : 60
```

Gambar 7. Screenshot Output Lihat Status Pemain

```
=== STARDUST VALLEY CLI ===
1. Lakukan Aktivitas
2. Lihat Status Pemain
3. Istirahat
0. Keluar
Pilih menu: 3
Energi telah dikembalikan! Energi sekarang: 100
```

Gambar 8. Screenshot Output Istirahat

```
=== STARDUST VALLEY CLI ===
1. Lakukan Aktivitas
2. Lihat Status Pemain
3. Istirahat
0. Keluar
Pilih menu: 0
Terima kasih telah bermain!

Process finished with exit code 0
```

Gambar 9. Screenshot Output Keluar Dari Program

C. Pembahasan

- **Penerapan SOLID**

  Pada program CLI Stardust Valley, terdapat penerapan **Single Responsibility Principle (SRP)** pada beberapa class. Class "`Main`" hanya berfungsi untuk menjalankan program dan memulai game, sehingga tanggung jawabnya terbatas pada eksekusi utama program. Class "`Game`" hanya bertanggung jawab untuk membuat karakter pemain dan memulai alur

permainan, sedangkan MenuHandler hanya menangani interaksi menu, pemilihan aktivitas, dan navigasi antar menu CLI. Class *"Player"* juga menerapkan SRP karena hanya mengelola atribut pemain seperti nama, gender, hobi, dan energi, serta logika terkait energi. Dengan penerapan SRP pada class-class ini, setiap class memiliki tanggung jawab tunggal sehingga memudahkan pemeliharaan dan pengembangan kode.

Penerapan **Open/Closed Principle (OCP)** diterapkan pada class "Activity" dan *"Tool"*. Class *"Activity"* menjadi class dasar untuk seluruh aktivitas (Farming, Fishing, Mining, Woodcutting) sehingga penambahan aktivitas baru dapat dilakukan melalui subclass tanpa mengubah kode *"Activity"*. Begitu pula class *"Tool"* menjadi dasar untuk semua tool (Axe, FishingRod, Pickaxe, WateringCan) sehingga penambahan tool baru dapat dilakukan tanpa memodifikasi class *"Tool"* yang sudah ada.

Penerapan **Liskov Substitution Principle (LSP)** terlihat pada subclass-subclass "aktivitas" dan "tool". *"Farming"*, *"Fishing"*, *"Mining"*, dan *"Woodcutting"* dapat digunakan sebagai *"Activity"* tanpa mengubah perilaku program, sedangkan *"Axe"*, *"FishingRod"*, *"Pickaxe"*, dan *"WateringCan"* dapat digunakan sebagai *"Tool"* atau *"Usable"* tanpa menimbulkan bug atau perilaku yang tidak diinginkan. Dengan demikian, semua subclass dapat menggantikan superclass mereka secara polimorfik.

Penerapan **Interface Segregation Principle (ISP)** diterapkan pada interface *"Usable"* yang hanya memiliki satu metode *use()*. Semua class tool (*"Axe"*, *"FishingRod"*, *"Pickaxe"*, *"WateringCan"*) hanya mengimplementasikan metode yang relevan, sehingga class tidak dipaksa mengimplementasikan metode yang tidak diperlukan.

Penerapan **Dependency Inversion Principle (DIP)** terdapat pada class *"Player"*, di mana class ini bergantung pada abstraksi *"EnergyStrategy"* untuk menghitung dan memanipulasi energi, bukan pada implementasi konkret. Hal ini memungkinkan strategi energi diganti atau

ditambahkan tanpa perlu mengubah class *"Player"*, meningkatkan fleksibilitas dan skalabilitas program.

- **Penerapan Design Pattern**

  Dalam program CLI ini, terdapat beberapa Design Pattern yang telah diterapkan. Class *"ActivityDecorator"* dan *"LoggingActivityDecorator"* menerapkan **Decorator Pattern (Structural)**, di mana fungsionalitas logging ditambahkan pada aktivitas tanpa mengubah class *"Activity"* asli. *"LoggingActivityDecorator"* menambahkan pesan log sebelum dan sesudah aktivitas dijalankan, sehingga fungsionalitas tambahan dapat diterapkan secara fleksibel.

  Kemudian *"EnergyStrategy"* dan *"FixedEnergyStrategy"* menerapkan **Strategy Pattern (Behavioral)**, memisahkan logika perhitungan energi dari class *"Player"*. Dengan pattern ini, strategi energi dapat diganti atau ditambahkan tanpa memodifikasi class *"Player"*, sehingga program lebih fleksibel dan mudah diperluas.

  Kemudian "ActivityTool" dan ActivityToolFactory menerapkan **Factory Pattern (Creational)**, memisahkan proses pembuatan objek *"Activity"* dan *"Tool"* berdasarkan pilihan pengguna di menu. Dengan pattern ini, logika instansiasi objek tidak tersebar di *"MenuHandler"*, sehingga penambahan aktivitas atau tool baru hanya perlu dilakukan di factory tanpa mengubah menu utama.

TAUTAN GIT

https://github.com/Rizki-A-M/PRAKTIKUM-PEMROGRAMAN-II.git