

IF2010 Pemrograman Berorientasi Objek

# TUGAS BESAR



K06 - G04

Nicholas Zefanya L.(18223111)

Maria Vransiska P.C.T.D.P.(18223119)

Leonard Arif S.(18223120)

Anggita Najmi L.(18223122)

Kenzie Raffa A.(18223127)



# STRESSDEW VALLEY



## USER MANUAL

Stressdew Valley merupakan permainan di mana pemain berperan sebagai seorang petani yang baru saja mewarisi sebuah lahan pertanian di desa Stressdew Valley. Pemain dapat melakukan berbagai aktivitas seperti bercocok tanam, memancing, memasak, serta berinteraksi dan menjalin hubungan dengan NPC, seperti menikah. Setiap aktivitas memberikan pengalaman yang berbeda-beda dan membantu pemain untuk mengembangkan lahannya. Permainan ini berlangsung dalam siklus musim dan hari, dengan perubahan cuaca tersebut maka pemain harus mengelola waktu, energi, dan sumber dayanya agar mencapai target sukses pemain. Permainan ini diprogram menggunakan bahasa java dengan antarmuka berbasis Graphical User Interface (GUI).

# STRESSDEW VALLEY



## USER MANUAL

Pemain mengendalikan karakter menggunakan WASD untuk bergerak ke atas, bawah, kiri, dan kanan. Berikut informasi dari penggunaan button (keyboard) untuk actions:

- Key "I" → Toggle Inventory
- Key "V" → Player Info
- Key "T" → Tiling
- Key "P" → Planting
- Key "L" → Watering
- Key "H" → Harvesting
- Key "F" → Fishing
- Key "E" → Eating
- Key "C" → Cooking
- Key "U" → Matching
- Key "Y" → Sleep
- Key "J" → Add Fuel
- Key "G" → Place Furniture
- Key "O" → Toggle Store (khusus di store)
- Key "R" → Recover Land
- Key "B" → Shipping Bin (player harus berada dekat Shipping Bin)
- Dekati NPC + Key "N" → Interact dengan NPC
- ESC → Menu Shortcut
- Arahkan karakter ke ujung map maka akan membuka WorldMap

## HOW TO PLAY

Cara masuk game:

1. Download ZIP file atau clone repository  
<https://github.com/LeonArif/Stressdew-Valley.git>
2. Download Maven untuk building agar dapat menjalankan game.
3. Buka folder di IDE dan jalankan terminal pada root-folder (Stressdew-Valley): mvn clean compile exec:java

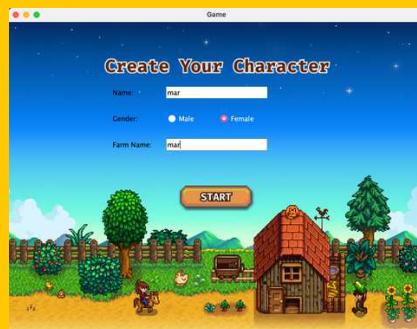
```
PS C:\Users\user\Stressdew-Valley> mvn clean compile exec:java
[INFO] Scanning for projects...
```

4. Jika build sukses, panel GUI akan terbuka dan game siap dimainkan!

# GAMEPLAY



Setelah program dapat dijalankan, player akan melihat tampilan awal dimana player dapat opsi 'PLAY' dan 'EXIT'. Jika klik 'EXIT' maka program akan berhenti berjalan dan menutup game.



Untuk masuk game, player dapat klik 'PLAY' lalu akan mendapat tampilan dimana player harus mengisi nama, gender, dan nama farm untuk membuat karakter baru. Setelah itu klik 'Start Game' maka akan masuk game.

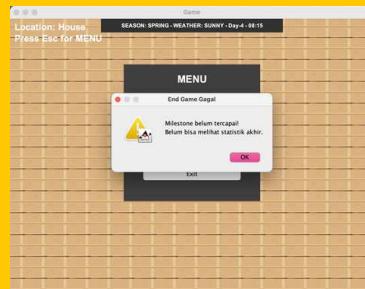
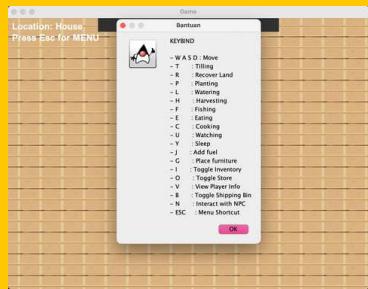
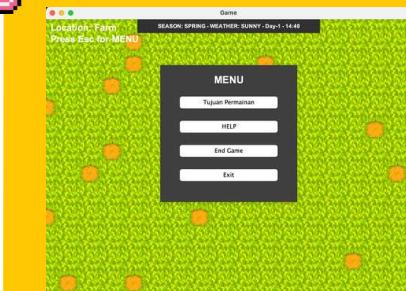


Setelah masuk, akan ada tampilan awal seperti gambar dikiri dimana player dapat bergerak atas, bawah, kanan, dan kiri.

STRESSDEW VALLEY

# GAMEPLAY

## GENERAL



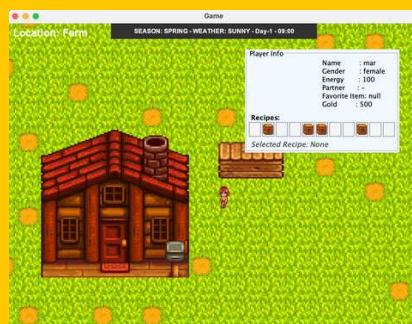
Player dapat membuka menu untuk tujuan dari game, bantuan untuk actions, melihat statistik end game jika milestone sudah tercapai, dan exit untuk keluar dari game.

# GAMEPLAY

## GENERAL



Untuk membuka inventory, player dapat klik "I" pada keyboard dan mengarahkan cursor pada item yang ingin dipilih.



Untuk membuka card dari player info, player dapat klik "V" pada keyboard (lalu tampilan dari player info akan muncul).



Actions memerlukan energy, contoh:  
Tiling → 1 kali action tiling = -5 Energy

STRESSDEW VALLEY

# GAMEPLAY

## MAP

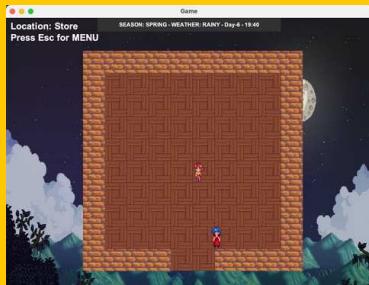
Untuk membuka World Map, player dapat mengarahkan karakter keujung map.



Ocean



Forest River



Store

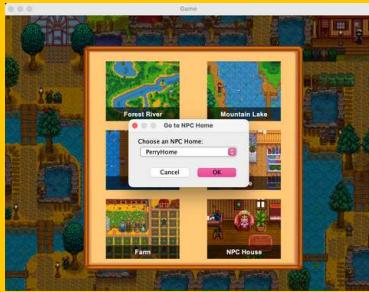


Mountain Lake

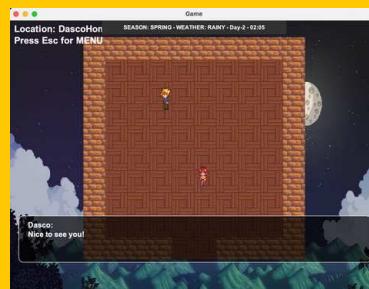
STRESSDEW VALLEY

# GAMEPLAY

## NPC HOME



Pada world map juga terdapat NPC House dimana player dapat masuk ke rumah NPC dan melakukan interaksi dengan NPC (key "N"), seperti chat, marry, propose, dan gift.



# GAMEPLAY

## ACTIONS



Shipping bin adalah action untuk menjual barang yang dimiliki dalam Inventory. Dalam sehari, Player hanya dapat melakukan shipping sekali. Setelah dikirimkan melalui shipping bin, item tidak dapat dikembalikan.



Player dapat memilih item yang ingin dijual melalui inventory sesuai kuantitas yang dimiliki. Maksimal kuantitas yang diinput adalah 16 buah.



Jika kuantitas yang ditambahkan tidak sesuai dengan kuantitas yang dimiliki di Inventory, maka akan mengirimkamini pesan error.

STRESSDEW VALLEY

# GAMEPLAY

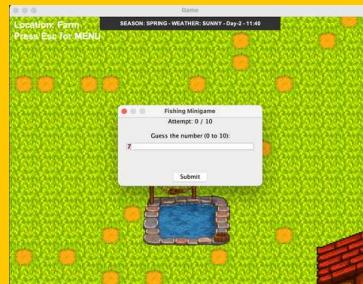
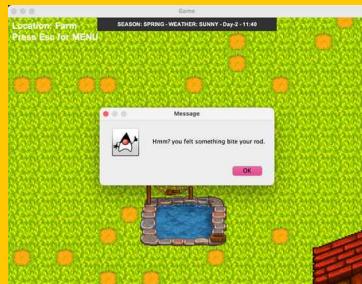
## ACTIONS



# STRESSDEW VALLEY

# GAMEPLAY

## ACTIONS

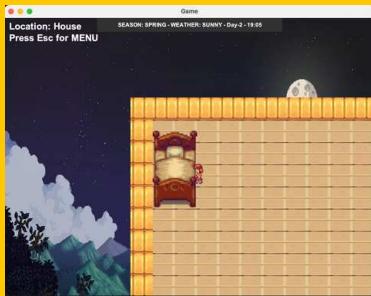


Action selanjutnya ada fishing, untuk melakukan action ini harus terlebih dahulu ke daerah pond. Dengan memakai fishing rod [open inventory] dan lakukan action (key "F") maka player akan melakukan challenge dimana player harus menebak angka antara 1 - 10, dimana jika berhasil menebak maka item akan masuk ke inventory.

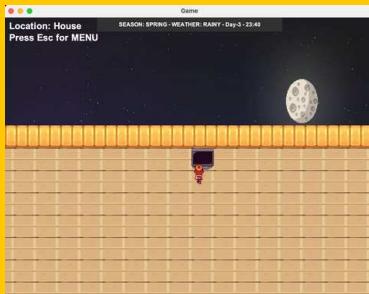
STRESSDEW VALLEY

# GAMEPLAY

## ACTIONS



Ketika energi player habis, player tidak dapat melakukan actions lain. Player dapat tidur dengan mengarahkan player dekat tempat tidur + key "y" agar dapat tidur untuk mengisi kembali energi menjadi full.



Selain itu player juga dapat melakukan action watching (key "U") dan cooking dalam rumah player.

STRESSDEW VALLEY

# GAMEPLAY

## ACTIONS



Selain watching, player juga dapat melakukan action cooking. Cooking dapat dilakukan setelah player mengisi fuel (key "J") pada stove dengan coal (open inventory), setelah fuel stove terisi maka player dapat memilih resep makanan lalu melakukan cooking.

Setelah berhasil melakukan cooking, makanan tadi akan masuk ke inventory player dimana player nantinya dapat memakan makanan dengan melakukan action eating (key "E"), dan action ini akan menambah energi player jika energi player berkurang.

# GAMEPLAY

## ACTIONS



Player dapat melakukan shopping pada store dengan membuka store (key "0"). Ketika player berhasil membeli barang maka kuantitas barang tersebut akan berkurang



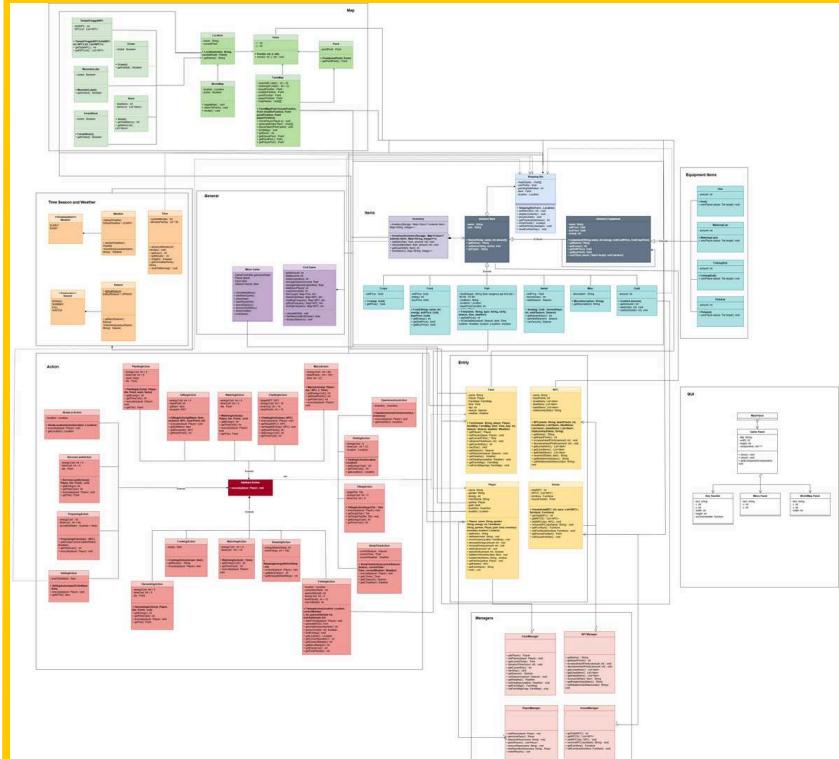
Setelah berhasil membeli, barang tersebut akan masuk ke inventory player.



Contoh player membeli sebuah bed, kembali ke rumah player dan bed itu dapat ditempatkan dimana saja didalam rumah (key "G").

# **STRESSDEW VALLEY**

# CLASS DIAGRAM



[BIT.LY/CLASSDIAGRAMKEL4](http://bit.ly/classdiagramkel4)

# LOG ACTIVITY

**NICHOLAS Z.L.(18223111)**

- Membuat source code TSU: Time, Season, dan Weather
- Membuat source code Items: Seeds
- Membuat GUI Textbox
- Membuat source code entity: House
- Mencari resource pack
- Membuat class diagram

**LEONARD A.S.(18223120)**

- Membuat source code entity: player dan NPC, maps: farmMap, worldMap, ocean, forrest river, mountain lake, store, meu game, actions: proposing, moving, show time, crops,
- Membuat GUI MenuPanel, main, tilechecker, worldmaps: farm, ocean, mountain, forrest river, store, player, shipping bin, fishing, cooking, watching, inventory, dan worldmapPanel.
- Membuat setup build game dan class diagram

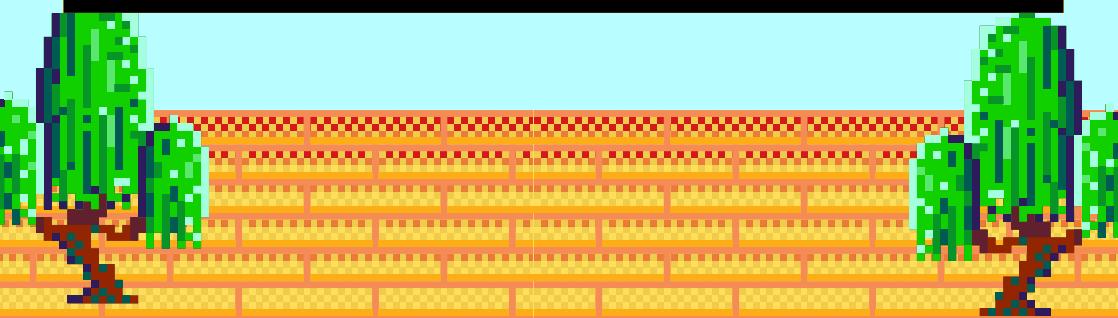
# LOG ACTIVITY

**KENZIE R.A (18223127)**

- Membuat source code actions: Tilling, Recover Land, Planting, Eating, Cooking, Visiting, Open Inventory, dan Show Location
- Membuat source code umum & entity: Items dan Shipping Bin
- Membuat booklet : Design Pattern, Dokumentasi, dan Class Diagram

**ANGGITA N.L. (18223122)**

- Membuat source code map: farm, actions: marry, gifting, planting, item: inventory, gold, fish, dan inheritance NPC serta NPC Home
- Membuat GUI shortcutMenuPanel, EndGamePanel, TWSPanel, map NPChomes, interact NPC: chat, marry, gift, NPCs, dan worldMapPanel
- Mencari resources pack
- Membuat class diagram dan buklet



# LOG ACTIVITY

**MARIA V.P.C.T.D.P.(18223119)**

- Membuat source code actions: Harvesting, Sleeping, Fishing, Watching, items: food
- Membuat source code umum: EndGame
- Membuat Booklet: User Manual, Game Control, How to Play, dan Gameplay,
- Membuat class diagram

# DESIGN PATTERN

## SINGLETON PATTERN

```
1 // File: PlayerManager.java
2 public class PlayerManager {
3     private List<Player> playerList;
4
5     public PlayerManager() {
6         this.playerList = new ArrayList<Player>();
7     }
8     // Methods to manage players
9 }
10
11 // File: NPCManager.java
12 public class NPCManager {
13     private List<NPC> npcList;
14
15     public NPCManager() {
16         this.npcList = new ArrayList<NPC>();
17     }
18     // Methods to manage NPCs
19 }
```

### Definisi:

Pattern yang memastikan sebuah kelas hanya memiliki satu instance dan menyediakan titik akses global ke instance tersebut.

Meskipun tidak mengimplementasikan singleton klasik (dengan instance statis dan private constructor), kedua manager ini berfungsi sebagai single point of access untuk koleksi objek terkait, yang merupakan esensi dari singleton pattern.

# DESIGN PATTERN

## OBSERVER PATTERN

```
1 // File: Time.java
2 public class Time implements Runnable {
3     private int hour = 6;
4     private int minute = 0;
5     private int day = 1;
6     private boolean running = true;
7     private boolean paused = false;
8     Season season;
9     Weather weather;
10
11    @Override
12    public void run() {
13        while (running) {
14            try {
15                synchronized (lock) {
16                    while (paused) {
17                        lock.wait();
18                    }
19                }
20                advanceTime(5);
21                Thread.sleep(1000);
22                changeDay();
23                if (day % 10 == 0) {
24                    season.changeSeasons();
25                }
26            } catch (InterruptedException e) {
27                Thread.currentThread().interrupt();
28            }
29        }
30    }
31
32    public synchronized void changeDay(){
33        if(day > lastDay) {
34            System.out.println("Day-" + day + " has started");
35            weather.nextDayWeather();
36            lastDay = day;
37        }
38    }
39 }
```

# DESIGN PATTERN

## OBSERVER PATTERN

Definisi:

Mendefinisikan ketergantungan one-to-many antar objek sehingga ketika satu objek mengubah state, semua dependennya diberi notifikasi dan diperbarui otomatis.

Di sini, Time bertindak sebagai Subject yang menjalankan thread, dan objek Season dan Weather berperan sebagai Observer yang secara otomatis merespons perubahan state pada objek Time.

# DESIGN PATTERN

## STRATEGY PATTERN

```
1 // File: Equipment.java
2 public abstract class Equipment extends Item {
3     // ...
4     public abstract void use();
5 }
6
7 // File: Hoe.java
8 public class Hoe extends Equipment {
9     // ...
10    public void use() {
11        System.out.println("Hoe something...");
12    }
13 }
14
15 // File: FishingRod.java
16 public class FishingRod extends Equipment {
17     // ...
18     public void use() {
19         System.out.println("Fish something...");
20     }
21 }
22
23 // File: Pickaxe.java
24 public class Pickaxe extends Equipment {
25     // ...
26     public void use() {
27         System.out.println("Mine something...");
28     }
29 }
30
31 // File: WateringCan.java
32 public class WateringCan extends Equipment {
33     // ...
34     public void use() {
35         System.out.println("Water something...");
36     }
37 }
```

# DESIGN PATTERN

## STRATEGY PATTERN

Definisi:

Mendefinisikan serangkaian algoritma, mengenkapsulasi masing-masing, dan membuatnya dapat dipertukarkan. Strategy memungkinkan algoritma bervariasi secara independen dari klien yang menggunakannya.

Setiap subclass Equipment mengimplementasikan metode use() dengan cara yang berbeda, memungkinkan pertukaran strategi "penggunaan" saat runtime.

# DESIGN PATTERN

## COMMAND PATTERN

```
1 // File: Action.java
2 public interface Action {
3     public boolean execute(Player p);
4     /*
5      *   kalo actionnya berhasil return true
6      *   kalo gagal return false
7     */
8 }
```

### Definisi:

Mengenapsulasi permintaan sebagai objek, memungkinkan parametrisasi klien dengan permintaan yang berbeda, antrian atau pencatatan permintaan, dan mendukung operasi yang dapat dibatalkan.

Interface Action menunjukkan implementasi Command Pattern, di mana berbagai perintah game dienkapsulasi sebagai objek yang dapat dijalankan dengan parameter Player.

# DESIGN PATTERN

## COMPOSITE PATTERN

```
1 // File: Item.java
2 public class Item {
3     private String itemName;
4     private String itemType;
5     private boolean isSellable;
6     private Gold sellPrice;
7     // Methods...
8 }
9
10 // Berbagai subclass Item
11 // File: Crop.java
12 public class Crop extends Item { /*...*/ }
13 // File: Fish.java
14 public class Fish extends Item { /*...*/ }
15 // File: Equipment.java
16 public abstract class Equipment extends Item { /*...*/ }
17 // File: Food.java
18 public class Food extends Item { /*...*/ }
19 // File: Misc.java
20 public class Misc extends Item { /*...*/ }
21 // File: Seed.java
22 public class Seed extends Item { /*...*/ }
23 // File: Inventory.java
24 public class Inventory {
25     private Map<Class<?>, Map<String, Integer>> inventoryStorage;
26
27     // Metode untuk mengelola berbagai jenis item secara seragam
28     public boolean additem(Item addedItem, int amount) { /*...*/ }
29     public boolean removeitem(Item removedItem, int amount) { /*...*/ }
30     public int getItemAmount(Item items) { /*...*/ }
31
32 }
```

# DESIGN PATTERN

## COMPOSITE PATTERN

Definisi:

Menyusun objek ke dalam struktur pohon untuk mewakili hierarki bagian-keseluruhan. Composite memungkinkan klien memperlakukan objek individual dan komposisi objek secara seragam.

Struktur ini memungkinkan Inventory memperlakukan semua jenis Item secara seragam, meskipun mereka memiliki implementasi dan perilaku yang berbeda.

# DESIGN PATTERN

## STATE PATTERN

```
1 // File: Season.java
2 public class Season {
3     public enum Seasons {
4         SUMMER, AUTUMN, WINTER, SPRING;
5     }
6
7     public Seasons currentSeason;
8
9     public Season() {
10         this.currentSeason = Seasons.SPRING;
11     }
12
13     public void changeSeasons(){
14         switch (currentSeason) {
15             case SUMMER:
16                 currentSeason = Seasons.AUTUMN;
17                 break;
18             case AUTUMN:
19                 currentSeason = Seasons.WINTER;
20                 break;
21             case WINTER:
22                 currentSeason = Seasons.SPRING;
23                 break;
24             case SPRING:
25                 currentSeason = Seasons.SUMMER;
26                 break;
27         }
28     }
29 }
30 // File: Weather.java
31 public class Weather {
32     public enum WeatherCondition{
33         SUNNY, RAINY;
34     }
35
36     private WeatherCondition currentWeather;
37
38     public void changeWeatherRandomly() {
39         if (generateRandom() == 0 || generateRandom() == 1) {
40             setCurrentWeather(WeatherCondition.SUNNY);
41         } else {
42             setCurrentWeather(WeatherCondition.RAINY);
43         }
44     }
45
46 }
```

# DESIGN PATTERN

## STATE PATTERN

Definisi:

Memungkinkan objek mengubah perilakunya ketika state internalnya berubah. Objek akan terlihat seperti mengubah kelasnya.

Dalam kode ini, perubahan musim dan cuaca mengubah state sistem, yang kemudian mempengaruhi perilaku game (pertumbuhan tanaman, energi pemain, dll).

# DESIGN PATTERN

## TEMPLATE METHOD PATTERN

```
1 // File: Furniture.java
2 public abstract class Furniture {
3     // Atribut dan konstruktur...
4
5     // Template method yang sama untuk semua furnitur
6     public void rotateFurniture(){
7         int temp = furnitureSizeX;
8         furnitureSizeX = furnitureSizeY;
9         furnitureSizeY = temp;
10    }
11
12    // Metode yang harus diimplementasikan oleh subclass
13    public abstract void useFurniture(Player p);
14 }
15
16 // File: Bed.java
17 public class Bed extends Furniture {
18     // Implementasi spesifik
19     public void useFurniture(Player p){
20         //nanti sleep ini kl dh ada actionny
21     }
22 }
23 // File: Stove.java
24 public class Stove extends Furniture{
25     public void useFurniture(Player p) {
26         //nanti cooking disini kl dah jadi actionny
27     }
28 }
29
30
31 // File: TV.java
32 public class TV extends Furniture{
33     public void useFurniture(Player p) {
34         //nanti tv ini kl dah jadi actionny
35     }
36 }
```

# DESIGN PATTERN

## TEMPLATE METHOD PATTERN

Definisi:

Mendefinisikan kerangka algoritma dalam sebuah metode, menunda beberapa langkah ke subclass. Template Method memungkinkan subclass mendefinisikan ulang langkah-langkah tertentu dari algoritma tanpa mengubah struktur algoritma.

Kelas Furniture mendefinisikan metode umum seperti `rotateFurniture()`, tetapi membiarkan implementasi spesifik `useFurniture()` untuk subclass.

# DESIGN PATTERN

## FACTORY METHOD PATTERN

```
1 // File: HouseMaptes.java (contoh penggunaan)
2 // Pembuatan objek sederhana yang bisa dikembangkan menjadi factory pattern
3 Bed bed = new Bed("bed1", "King Bed", "A comfortable king size bed", 2, 3, 2);
4 Stove stove = new Stove();
5
6 // Potensi implementasi factory:
7 // public class FurnitureFactory {
8 //     public static Furniture createFurniture(String type, String id, String name, String desc, int sizeX, int sizeY) {
9 //         switch(type) {
10 //             case "Bed": return new Bed(id, name, desc, sizeX, sizeY, 2);
11 //             case "Stove": return new Stove();
12 //             case "TV": return new TV(desc, sizeX, sizeY);
13 //             default: throw new IllegalArgumentException("Unknown furniture type");
14 //         }
15 //     }
16 // }
```

Definisi:

Mendefinisikan antarmuka untuk membuat objek, tetapi membiarkan subclass memutuskan kelas mana yang akan diinstansiasi. Factory Method membiarkan kelas menunda instantiasi ke subclass.

Meskipun tidak diimplementasikan secara eksplisit, pola pembuatan objek furnitur menunjukkan kesempatan untuk menggunakan Factory Method Pattern.

# DESIGN PATTERN

## BUILDER PATTERN

```
1  public class CreatePlayerPanel extends JPanel {
2      MainPanel mainPanel;
3
4      public CreatePlayerPanel(JFrame frame, MainPanel mainPanel) {
5          this.mainPanel = mainPanel;
6
7          // Setup UI components...
8
9          startButton.addActionListener(e -> {
10              String name = nameField.getText();
11              String gender = maleButton.isSelected() ? "male" : (femaleButton.isSelected() ? "female" : "");
12              String fName = fNameField.getText();
13
14              if (name.isEmpty() || gender.isEmpty() || fName.isEmpty()) {
15                  JOptionPane.showMessageDialog(frame, "Please fill all fields.");
16                  return;
17              }
18
19              mainPanel.startGame(name, gender, fName);
20          });
21      }
22  }
```

CreatePlayerPanel menerapkan pola Builder sederhana dengan mengumpulkan atribut yang diperlukan (nama, gender, nama pertanian) secara bertahap sebelum membuat objek Player yang lengkap melalui method startGame(). Ini memisahkan proses pengumpulan parameter dari konstruksi objek sebenarnya.

# DESIGN PATTERN

## EVENT NOTIFICATION PATTERN

```
1 public boolean execute(Player player) {  
2     // Logika aksi...  
3     // Memberitahu tentang hasil aksi  
4     System.out.println("You have eaten a " + itemToEat.getItemName() + ". Your current energy: " + player.getEnergy() + ".");  
5     //...  
6     return true;  
7 }  
8 }
```

Ketika aksi dieksekusi, mereka memberitahu pemain tentang hasilnya melalui pesan konsol atau pembaruan UI. Ini merupakan bentuk sederhana dari pola Event Notification.

# DESIGN PATTERN

## PROXY PATTERN

```
1  public class ShippingBin {
2      private int MAX_UNIQUE_ITEMS = 16;
3      private Map<Item, Integer> shippingBinStorage;
4      private int lastSoldDay = 0;
5
6      public int sellShippingBin(Time time, Player player) {
7          if (cooldown(time)) {
8              System.out.println("You can only sell once per day.");
9              return 0;
10         }
11         int totalValue = 0;
12         for (Item item : shippingBinStorage.keySet()) {
13             totalValue += item.getSellPrice().getGold() * shippingBinStorage.get(item);
14         }
15         player.totalIncome += totalValue;
16         player.getPlayerGold().addGold(totalValue);
17         shippingBinStorage.clear();
18         lastSoldDay = time.getDay();
19         System.out.println("Items sold! You earned: " + totalValue + " gold.");
20         return totalValue;
21     }
22
23     public boolean cooldown(Time time){
24         return lastSoldDay == time.getDay();
25     }
26 }
```

ShippingBin bertindak sebagai proxy yang mengontrol akses ke fungsionalitas penjualan. Ini mengimplementasikan mekanisme cooldown yang mencegah penjualan lebih dari sekali per hari, berfungsi sebagai protection proxy untuk mengontrol akses ke fungsionalitas penjualan.

# DESIGN PATTERN

## ITERATOR PATTERN

```
1 public void printInventory() {
2     boolean hasAnyItem = inventoryStorage.values().stream().anyMatch(map > !map.isEmpty());
3     if (!hasAnyItem) {
4         System.out.println("No items in inventory.");
5         return;
6     }
7     System.out.println("----- INVENTORY STORAGE -----");
8     for (Class<?> cls : typeToClassMap.values()) {
9         Map<Item, Integer> map = inventoryStorage.get(cls);
10        if (map != null && !map.isEmpty()) {
11            System.out.println("Items of type: " + cls.getSimpleName());
12            for (Map.Entry<Item, Integer> entry : map.entrySet()) {
13                System.out.println(" " + entry.getKey().getItemName() + ": " + entry.getValue());
14            }
15            System.out.println("-----");
16        }
17    }
18 }
```

Kode ini menggunakan pola Iterator Java melalui loop for-each dan operasi stream saat bekerja dengan koleksi. Kode ini memanfaatkan pola Iterator yang disediakan oleh Java Collection Framework untuk mengakses elemen dalam koleksi secara berurutan.

# DESIGN PATTERN

## PROSES PENGEMBANGAN

Dalam mengerjakan tugas besar ini, kami lebih banyak mengadakan kerja kelompok beberapa kali secara offline, sehingga hal-hal yang didiskusikan lebih banyak dibahas secara lisan.

Untuk mempermudah workflow penggerjaan, kami membagi tugas mengerjakan source code berdasarkan pembagian berikut

### PEMBAGIAN TUGAS

Agar workflow rapi dan ter-track dengan baik, kami menggunakan sistem branch pada git berdasarkan kategori class atau parent class-nya.

Tentu terdapat perubahan terhadap diagram class yang kami rencanakan di awal, seperti misalnya implementasi GUI yang pada akhirnya hampir setiap implementasi parent class memiliki panel GUI-nya masing-masing. Kami juga menambahkan beberapa class manager untuk entitas yang perlu dihitung dan di-manage seperti Player, NPC, dan NPCHome.

STRESSDEW VALLEY

# DOKUMENTASI



STRESSDEW VALLEY

# DOKUMENTASI



# STRESSDEW VALLEY



**SELAMAT BERMAIN**