RESPONSI
PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK
2023/2024

IDENTITAS

Nama	:	*Mohammad Nazhiif Al-Ghoniy*
NIM	:	*L0123084*
Kelas	:	*C*
Judul Program	:	*RPG Abal-Abal*
Deskripsi Program	:	*Program sebuah permainan RPG berbasis teks (text-based RPG) yang memungkinkan pemain untuk memilih karakter, bertarung dengan monster, mengelola inventaris, serta membeli item di toko.*

DOKUMENTASI PROGRAM

(masukkan berbagai penjelasan program yang lebih detail di sini, seperti instruksi pemakaian, fitur-fitur, screenshot program, screenshot source code dan penjelasannya, dan lain-lain)

==== FITUR-FITUR ====

1. Pemain dapat memilih karakter yang ingin dimainkan. Terdapat dua pilihan karakter:
    - Warrior (Prajurit): Memiliki lebih banyak health dan kemampuan untuk menyerang dengan senjata fisik.
    - Mage (Penyihir): Memiliki serangan sihir dengan damage lebih tinggi, namun sedikit lebih lemah dalam hal health dibandingkan Warrior.

    Pemain diminta untuk memilih karakter melalui input dari pengguna, dan karakter yang dipilih akan digunakan sepanjang permainan.
2. Battle Mode (Mode Pertempuran) Pemain dapat memasuki mode pertempuran melawan monster-monster yang muncul secara acak. Beberapa fitur dalam mode pertempuran:
    - Giliran Pemain: Pemain dapat memilih untuk menyerang, bertahan, atau menggunakan item dalam giliran mereka.
    - Giliran Monster: Setelah pemain melakukan tindakan, monster akan menyerang pemain.
    - Kondisi Kemenangan: Pemain akan mendapatkan emas (gold) setelah mengalahkan monster, sesuai dengan level monster yang dikalahkan.
    - Kondisi Kekalahan: Jika health karakter pemain habis (health = 0), karakter akan di-revive dengan health penuh (50 HP) untuk melanjutkan permainan.

3. Inventaris (Inventory) Pemain memiliki inventaris yang dapat diakses kapan saja untuk melihat item yang dimiliki. Setiap karakter dapat mengumpulkan item, dan item dapat digunakan selama pertempuran. Saat ini, item yang tersedia adalah Potion yang dapat digunakan untuk menyembuhkan health.
4. Toko (Shop) Pemain dapat mengunjungi toko untuk membeli potion dengan menggunakan emas yang dimiliki. Ada tiga jenis potion yang dijual di toko:
    - Healing Potion: Menyembuhkan 20 HP.
    - Mana Potion: Menyembuhkan 30 HP.
    - Elixir Potion: Menyembuhkan 40 HP.

    Pemain dapat memilih potion yang ingin dibeli, jika mereka memiliki cukup emas. Setiap potion yang dibeli akan ditambahkan ke dalam inventaris pemain.
5. Karakter & Monster
    - Karakter: Setiap karakter (baik Warrior atau Mage) memiliki atribut seperti health (health), damage (kerusakan serangan), dan emas (gold). Pemain juga dapat melakukan tindakan bertahan yang mengurangi damage yang diterima.
    - Monster: Setiap monster memiliki atribut seperti nama, level, health, dan damage. Monster muncul secara acak di dalam pertempuran, dan akan terus menyerang pemain hingga mati atau pemain kalah.

==== INSTRUKSI PEMAKAIAN ====

1. Mulai Permainan:
    - Ketika pemain pertama kali menjalankan program, mereka akan diminta untuk memilih karakter. Pilih antara 1 untuk Warrior atau 2 untuk Mage.
    - Setelah memilih karakter, permainan akan membawa pemain ke menu utama.

```
Choose your character:
1. Warrior
2. Mage
Enter your choice: 1
```

2. Menu utama:
    - Enter Battle Mode: Memasuki mode pertempuran melawan monster.
    - Show Inventory: Melihat inventaris karakter dan item yang dimiliki.

```
=== Inventory (Albert) ===
1. Potion (Type: Elixir Potion, Qty: 1, Price: 40)

Press Enter key to continue...
```

    - Character Profile: Melihat profil karakter, termasuk nama, health, damage, dan emas.

```
Choose an option: 3

Name: Albert
HP: 100
Damage: 40
Press Enter key to continue...
```

    - Visit Shop: Mengunjungi toko untuk membeli item (potions).
    - Exit: Keluar dari permainan.

```
=== Menu ===
1. Enter Battle Mode
2. Show Inventory
3. Character Profile
4. Visit Shop
5. Exit
Choose an option:
```

3. **Mode Pertempuran:**
    - Pemain akan melawan monster secara giliran.

```
Entering Battle Mode...
A wild Monster 1 appears!
Monster 1 has 80 health and 20 damage.

=== Your Turn ===
1. Attack
2. Defend
3. Use Item
Choose an action:
```

```
Monster 1 is attacking Albert.
Albert takes 20 damage!
Albert has 80 health remaining.

Press Enter key to continue...
```

- Pemain memilih antara menyerang, bertahan, atau menggunakan item (seperti potion) untuk meningkatkan peluang bertahan hidup.

```
Choose an action: 1

Albert slashes with a sword.
Monster 1 takes 40 damage

Monster 1 has 40 health remaining.

Press Enter key to continue...
```

```
Choose an action: 2
Albert is defending and will take reduced damage.

Press Enter key to continue...
```

```
=== Your Turn ===
1. Attack
2. Defend
3. Use Item
Choose an action: 3

=== Choose an Item to Use ===
1. Potion (Type: Elixir Potion, Qty: 1, Price: 40)
Enter the number of the item you want to use: 1
Using Elixir Potion potion. Healing for 40 HP.
You used a Elixir Potion and restored 40 health.
You have 0 Elixir Potion left.
```

- Setiap monster memiliki level, health, dan damage yang berbeda. Ketika monster kalah, pemain akan mendapatkan emas.

4. **Menggunakan potion**

- Potion yang dimiliki akan muncul dalam inventaris. Pemain dapat menggunakan potion untuk menyembuhkan HP.
- Terdapat beberapa jenis potion yang dapat dipilih untuk digunakan, seperti Healing Potion, Mana Potion, dan Elixir Potion.

```
=== Choose an Item to Use ===
1. Potion (Type: Healing Potion, Qty: 1, Price: 20)
2. Potion (Type: Mana Potion, Qty: 1, Price: 30)
3. Potion (Type: Elixir Potion, Qty: 1, Price: 40)
Enter the number of the item you want to use:
```

5. **Toko**

- Pemain dapat mengunjungi toko dan membeli potion dengan emas yang dimiliki.
- Setiap potion memiliki harga yang berbeda dan dapat membantu pemain untuk menyembuhkan HP selama pertempuran.

```
=== Welcome to the Shop ===
You have 150 gold.

Items for Sale:
1. Healing Potion +20 health (Price: 20 gold)
2. Mana Potion +30 health (Price: 30 gold)
3. Elixir Potion +40 health (Price: 40 gold)
0. Exit Shop
Enter the number of the potion you want to buy:
```

- Pemain dapat membeli potion jika mereka memiliki cukup emas.

```
Enter the number of the potion you want to buy: 3
You have bought a Elixir Potion for 40 gold.

Press Enter key to continue...
```

==== **PENJELASAN DETAIL TENTANG KELAS-KELAS DALAM PROGRAM** ====

1. Main:
    - Merupakan titik masuk (entry point) permainan, di mana pemilihan karakter dilakukan, menu utama ditampilkan, dan game loop dijalankan. Pemain dapat berinteraksi dengan game melalui menu pilihan.
    - Pada mode pertempuran, Main akan menangani logika pertempuran dan kondisi kemenangan atau kekalahan.
2. RPGCharacter Interface:
    - Mendefinisikan tindakan dasar untuk karakter dalam permainan, seperti attack(), defend(), takeDamage(), dan getInfo().
    - Semua kelas karakter (Warrior, Mage, dan lainnya) mengimplementasikan interface ini untuk memastikan semua karakter memiliki tindakan dasar yang sama.
3. Character (Abstract Class):
    - Kelas abstrak yang mewarisi RPGCharacter, yang menyediakan implementasi dasar untuk karakter. Ini mencakup atribut seperti nama, health, damage, inventaris, dan emas.
    - Ada metode untuk menyerang (attack()), bertahan (defend()), menerima damage (takeDamage()), melihat inventaris (showInventory()), dan menggunakan item (useItem()).
4. Warrior dan Mage:
    - Kelas-kelas turunan dari Character yang memiliki implementasi khusus untuk menyerang dan bertahan. Warrior menggunakan serangan fisik, sedangkan Mage menggunakan serangan sihir.
5. Monster:
    - Menyimpan atribut monster seperti nama, level, health, dan damage. Monster dapat menyerang karakter dan menerima damage dari serangan karakter.
6. Item dan Potion:
    - Item adalah kelas abstrak untuk item yang dapat dimiliki oleh karakter.
    - Potion adalah jenis item yang dapat digunakan untuk menyembuhkan health. Terdapat beberapa jenis potion dengan efek penyembuhan yang berbeda.
7. Shop:
    - Mengelola item yang dijual di toko (hanya potion saat ini). Pemain dapat membeli item dengan emas mereka dan item tersebut ditambahkan ke inventaris.

==== CARA BERMAIN ====

1. Pilih karakter (Warrior atau Mage).
2. Pilih opsi dari menu utama untuk bertarung melawan monster, melihat inventaris, memeriksa profil karakter, atau mengunjungi toko.
3. Saat bertarung, pilih tindakan yang ingin diambil: menyerang, bertahan, atau menggunakan item.
4. Kumpulkan emas dari mengalahkan monster dan gunakan untuk membeli potion di toko.
5. Lanjutkan petualangan dan kalahkan monster lebih kuat, tingkatkan inventaris, dan capai kemenangan.

Main.java

```
package com.game;

import com.game.character.Character;
import com.game.character.Warrior;
import com.game.character.Mage;
import com.game.monster.Monster;
import com.game.shop.Shop;

import java.util.*;

public class Main {
    private static final Random random = new Random();
    public static final Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        System.out.println("Choose your character:");
        System.out.println("1. Warrior");
        System.out.println("2. Mage");
        System.out.print("Enter your choice: ");
        int characterChoice = scanner.nextInt();

        Character player;

        if (characterChoice == 1) {
            player = new Warrior("Albert", 100, 40, 150);
```

```
        } else {
            player = new Mage("Gandalf", 80, 60, 150);
        }

        Shop shop = new Shop();

        while (true) {
            showMenu();
            int choice = getUserInput();

            switch (choice) {
                case 1 -> {
                    enterBattleMode(player);
                    pressAny();
                }
                case 2 -> {
                    player.showInventory();
                    pressAny();
                }
                case 3 -> {
                    player.getInfo();
                    pressAny();
                }
                case 4 -> {
                    shop.showShopMenu(player);  // Open the shop to buy potions
                    pressAny();
                }
                case 5 -> {
                    System.out.println("Exiting the game. Goodbye!");
                    scanner.close();
                    return;
                }
                default -> System.out.println("Invalid option. Please try
again.");
            }
        }
    }
```

```
    private static void showMenu() {
        clearScreen();
        System.out.println("\n=== Menu ===");
        System.out.println("1. Enter Battle Mode");
        System.out.println("2. Show Inventory");
        System.out.println("3. Character Profile");
        System.out.println("4. Visit Shop");
        System.out.println("5. Exit");
        System.out.print("Choose an option: ");
    }

    private static int getUserInput() {
        while (true) {
            try {
                return scanner.nextInt();
            } catch (InputMismatchException e) {
                System.out.println("Invalid input. Please enter a number.");
                scanner.nextLine(); // clear the invalid input
            }
        }
    }

    private static void enterBattleMode(Character character) {
        clearScreen();
        System.out.println("\nEntering Battle Mode...");
        battleMode(character);
        if (character.getHealth() == 0) {
            character.setHealth(50);
        }
    }

    private static void battleMode(Character character) {
        int numberOfMonsters = random.nextInt(3) + 1;
        List<Monster> monsterList = generateMonsters(numberOfMonsters);

        for (Monster monster : monsterList) {
            System.out.printf("A wild %s appears!%n", monster.getName());
```

```
            System.out.printf("%s has %d health and %d damage.%n",
monster.getName(), monster.getHealth(), monster.getDamage());

            while (monster.isAlive() && character.isAlive()) {
                try {
                    characterTurn(character, monster);
                } catch (InputMismatchException e) {
                    System.out.println("Invalid action choice, try again.");
                    scanner.nextLine();  // clear the invalid input
                    continue;
                }
                if (monster.isAlive()) {
                    monsterTurn(character, monster);
                }
            }

            if (!monster.isAlive()) {
                int goldReward = monster.getLevel() * 10;
                System.out.printf("You defeated %s! You earned %d gold.%n",
monster.getName(), goldReward);
                character.addGold(goldReward);
            }
        }
    }

    private static List<Monster> generateMonsters(int numberOfMonsters) {
        List<Monster> monsterList = new ArrayList<>();
        for (int i = 0; i < numberOfMonsters; i++) {
            String monsterName = String.format("Monster %d", i + 1);
            int level = random.nextInt(5) + 1;
            int health = level * 20;
            int damage = level * 5;
            monsterList.add(new Monster(monsterName, level, health, damage));
        }
        return monsterList;
    }

    private static void characterTurn(Character character, Monster monster) {
```

```
        System.out.println("\n=== Your Turn ===");
        System.out.println("1. Attack");
        System.out.println("2. Defend");
        System.out.println("3. Use Item");
        System.out.print("Choose an action: ");

        int action = scanner.nextInt();

        switch (action) {
            case 1 -> {
                character.attack();
                monster.takeDamage(character.getDamage());
                System.out.printf("%s takes %d damage%n", monster.getName(),
character.getDamage());
                System.out.printf("%n%s has %d health remaining.%n",
monster.getName(), monster.getHealth());
                pressAny();
            }
            case 2 -> {
                character.defend();
                pressAny();
            }
            case 3 -> {
                try {
                    character.useItem();
                } catch (IndexOutOfBoundsException e) {
                    System.out.println("Invalid item choice. Please try again.");
                }
                pressAny();
            }
            default -> System.out.println("Invalid action. Please try again.");
        }
    }

    private static void monsterTurn(Character character, Monster monster) {
        try {
            int damage = monster.getDamage();
            character.takeDamage(damage);
```

```
            clearScreen();
            System.out.printf("%s is attacking %s.%n", monster.getName(),
character.getName());
            System.out.printf("%s takes %d damage!%n", character.getName(),
damage);
            System.out.printf("%s has %d health remaining.%n",
character.getName(), character.getHealth());

            if (!character.isAlive()) {
                System.out.println("You have been defeated!");
                System.out.println("You will be revived with 50 health");
                pressAny();
                return;
            }

        } catch (Exception e) {
            System.out.println("An error occurred during the monster's turn.");
//              e.printStackTrace();
        }
        pressAny();
    }

    private static void clearScreen() {
        for (int i = 0; i < 50; i++) {
            System.out.println();
        }
    }

    private static void pressAny()
    {
        System.out.print("\nPress Enter key to continue... ");
        Scanner s = new Scanner(System.in);
        s.nextLine();
    }
}
```

==== FOLDER CHARACTER ====

RPGCharacter.java

```java
package com.game.character;

public interface RPGCharacter {
    void attack();
    void defend();
    void takeDamage(int damage);
    void getInfo();
}
```

Character.java

```java
package com.game.character;

import com.game.item.Item;
import com.game.item.Potion;

import java.util.ArrayList;
import java.util.InputMismatchException;
import java.util.List;
import java.util.Scanner;

public abstract class Character implements RPGCharacter {
    private final String name;
    private int health;
    private final int damage;
    private final List<Item> inventory;
    protected boolean isDefending;
    private int gold;

    public Character(String name, int health, int damage, int gold) {
        this.name = name;
        this.health = health;
        this.damage = damage;
        this.inventory = new ArrayList<>();
        this.isDefending = false;
        this.gold = gold;
    }
```

```
    public String getName() {
        return name;
    }

    public int getHealth() {
        return health;
    }

    public void setHealth(int health) {
        this.health += health;
    }

    public int getDamage() {
        return damage;
    }

    public int getGold() {
        return gold;
    }

    public void addGold(int amount) {
        gold += amount;
    }

    public void deductGold(int amount) {
        if (gold >= amount) {
            gold -= amount;
        } else {
            System.out.print("You don't have enough gold!\n");
        }
    }

    public void takeDamage(int damage) {
        if (isDefending) {
            damage /= 2;
            isDefending = false;
        }
```

```
        health = Math.max(health - damage, 0);
    }

    public boolean isAlive() {
        return health > 0;
    }

    public void showInventory() {
        System.out.printf("%n=== Inventory (%s) ===%n", getName());
        for (int i = 0; i < inventory.size(); i++) {
            System.out.printf("%d. %s%n", i + 1, inventory.get(i).getInfo());
        }
    }

    public void addItem(Item item) {
        inventory.add(item);
    }

    public void useItem() {
        if (inventory.isEmpty()) {
            System.out.println("You have no items in your inventory.");
            return;
        }

        System.out.println("\n=== Choose an Item to Use ===");
        // Display the inventory for the user to choose an item
        for (int i = 0; i < inventory.size(); i++) {
            System.out.printf("%d. %s%n", i + 1, inventory.get(i).getInfo());
        }

        System.out.print("Enter the number of the item you want to use: ");
        try {
            Scanner scanner = new Scanner(System.in);
            int choice = scanner.nextInt() - 1;

            if (choice < 0 || choice >= inventory.size()) {
                System.out.println("Invalid choice. Please try again.");
                return;
```

```
            }

            Item item = inventory.get(choice);
            if (item != null) {
                Potion potion = (Potion) item;

                if (potion.getQuantity() <= 0) {
                    System.out.println("This potion is out of stock.");
                } else {
                    // Use the potion
                    potion.use();

                    int healingAmount = potion.getHealingAmount(); // Potion
restores 50 health
                    health = Math.min(100, health + healingAmount); // Max health
is 100
                    System.out.printf("You used a %s and restored %d health.%n",
potion.getType(), healingAmount);

                    // Display remaining potions
                    System.out.printf("You have %d %s left.%n",
potion.getQuantity(), potion.getType());
                }
            } else {
                System.out.println("This item cannot be used.");
            }
        } catch (InputMismatchException e) {
            System.out.println("Invalid input. Please enter a valid item
number.");
        } catch (IndexOutOfBoundsException e) {
            System.out.println("Invalid item index. Please select a valid
item.");
        }
    }

    public void getInfo() {
        System.out.printf("%nName: %s%nHP: %d%nDamage: %d%nGold: %d", getName(),
getHealth(), getDamage(), getGold());
```

```
    }
}
```

Warrior.java

```
package com.game.character;

public class Warrior extends Character {
    public Warrior(String name, int health, int damage, int gold) {
        super(name, health, damage, gold);
    }

    @Override
    public void attack() {
        System.out.printf("\n%s slashes with a sword.%n", getName());
    }

    @Override
    public void defend() {
        isDefending = true;
        System.out.printf("%s is defending and will take reduced damage.%n",
getName());
    }
}
```

Mage.java

```
package com.game.character;

public class Mage extends Character {
    public Mage(String name, int health, int damage, int gold) {
        super(name, health, damage, gold);
    }

    @Override
    public void attack() {
        System.out.printf("\n%s casts a fireball.%n", getName());
    }

    @Override
```

```
    public void defend() {
        isDefending = true;
        System.out.printf("%s shields with magic, reducing damage.%n",
getName());
    }
}
```

==== FOLDER MONSTER ====

Monster.java

```
package com.game.monster;

public class Monster {
    private final String name;
    private final int level;
    private int health;
    private final int damage;

    public Monster(String name, int level, int health, int damage) {
        this.name = name;
        this.level = level;
        this.health = health;
        this.damage = damage;
    }

    public String getName() {
        return name;
    }

    public int getLevel() {
        return level;
    }

    public int getDamage() {
        return damage;
    }

    public int getHealth() {
        return health;
```

```
    }

    public boolean isAlive() {
        return health > 0;
    }

    public void takeDamage(int damage) {
        health = Math.max(health - damage, 0);
    }
}
```

==== FOLDER ITEM ====

Item.java

```
package com.game.item;

public abstract class Item {
    private final int price;
    private int quantity;

    public Item(int price, int quantity) {
        this.price = price;
        this.quantity = quantity;
    }

    public int getPrice() {
        return price;
    }

    public int getQuantity() {
        return quantity;
    }
    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
    public abstract String getInfo();
    public abstract void use();
}
```

Potion.java

```java
package com.game.item;

public class Potion extends Item {
    private final String type;
    private final int healingAmount;

    public Potion(String type, int quantity, int price, int healingAmount) {
        super(price, quantity);
        this.type = type;
        this.healingAmount = healingAmount;
    }

    public String getType() {
        return type;
    }

    public int getHealingAmount() {
        return healingAmount;
    }

    @Override
    public void use() {
        if (getQuantity() > 0) {
            setQuantity(getQuantity() - 1);
            System.out.printf("Using %s potion. Healing for %d HP.%n", type,
healingAmount);
        } else {
            System.out.println("No more potions left.");
        }
    }

    @Override
    public String getInfo() {
        return String.format("Potion (Type: %s, Qty: %d, Price: %d)", getType(),
getQuantity(), getPrice());
    }
}
```

==== FOLDER SHOP ====

Shop.java

```
package com.game.shop;

import com.game.character.Character;
import com.game.item.Potion;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Shop {
    private final List<Potion> potionsForSale;

    public Shop() {
        potionsForSale = new ArrayList<>();
        potionsForSale.add(new Potion("Healing Potion", 50, 20, 20));
        potionsForSale.add(new Potion("Mana Potion", 50, 30, 30));
        potionsForSale.add(new Potion("Elixir Potion", 50, 40, 40));
    }

    public void showShopMenu(Character character) {
        System.out.println("\n=== Welcome to the Shop ===");
        System.out.printf("You have %d gold.%n", character.getGold());
        System.out.println("\nItems for Sale:");

        for (int i = 0; i < potionsForSale.size(); i++) {
            Potion potion = potionsForSale.get(i);
            System.out.printf("%d. %s +%d health (Price: %d gold)%n", i + 1,
potion.getType(), potion.getHealingAmount(), potion.getPrice());
        }
        System.out.println("0. Exit Shop");

        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of the potion you want to buy: ");
        int choice = scanner.nextInt() - 1;

        if (choice >= 0 && choice < potionsForSale.size()) {
```

```
            Potion potion = potionsForSale.get(choice);
            if (character.getGold() >= potion.getPrice()) {
                character.deductGold(potion.getPrice());
                character.addItem(new Potion(potion.getType(), 1,
potion.getPrice(), potion.getHealingAmount())); // Add potion to character's
inventory
                System.out.printf("You have bought a %s for %d gold.%n",
potion.getType(), potion.getPrice());
            } else {
                System.out.println("You don't have enough gold to buy this
item.");
            }
        } else if (choice == -1){
            System.out.println("Quitting Shop.");
        } else {
            System.out.println("Invalid choice, please try again.");
        }
    }
}
```

Tabel penggunaan materi:

NO	MATERI	PERAN
1.	Class, Object, Data Type and Constructor	**Class** Class adalah cetak biru (template) untuk membuat objek. Program ini terdiri dari beberapa kelas seperti Character, Warrior, Mage, Monster, Item, Potion, Shop, dan Main. • Character: Kelas dasar untuk karakter dalam permainan yang mendefinisikan atribut seperti nama, kesehatan, damage, dan inventaris. • Warrior dan Mage: Kelas turunan dari Character, masing-masing mengimplementasikan aksi yang spesifik untuk karakter tersebut (misalnya, serangan fisik untuk Warrior dan serangan sihir untuk Mage).

		• Monster: Kelas yang merepresentasikan musuh dalam pertempuran. • Potion: Kelas yang mewakili item potions yang dapat digunakan oleh karakter untuk menyembuhkan HP. **Object** Objek adalah instance dari kelas. Dalam program ini, ketika pemain memilih karakter, objek Warrior atau Mage dibuat berdasarkan pilihan pemain. Character player = new Warrior("Albert", 100, 40, 150); Objek player adalah instance dari kelas Warrior yang memiliki atribut seperti nama, kesehatan, damage, dan emas. **Data Type** Setiap variabel dalam kelas memiliki tipe data tertentu. Misalnya, int health di kelas Character adalah tipe data integer yang menyimpan nilai kesehatan karakter. Data type juga digunakan untuk mendefinisikan jenis objek seperti Potion yang merupakan objek dari kelas Potion, dan List<Item> yang digunakan untuk menyimpan item dalam inventaris. **Constructor** Constructor adalah metode khusus yang digunakan untuk menginisialisasi objek saat pertama kali dibuat. Misalnya, di kelas Character ada constructor yang digunakan untuk menginisialisasi nama, kesehatan, damage, dan emas ketika objek karakter dibuat:

2.	Inheritance	Inheritance memungkinkan satu kelas untuk mewarisi atribut dan metode dari kelas lain. Dalam program ini, prinsip inheritance diterapkan untuk mengurangi pengulangan kode dan memungkinkan perbedaan perilaku antara jenis karakter. • Warrior dan Mage mewarisi kelas Character. Mereka semua berbagi atribut seperti name, health, damage, dan gold dari kelas induk Character, namun masing-masing mengimplementasikan metode attack() dan defend() secara berbeda. • Mage dan Warrior tidak perlu menulis ulang kode untuk pengelolaan inventaris atau atribut umum lainnya, karena sudah disediakan oleh kelas Character. `public class Warrior extends Character { public Warrior(String name, int health, int damage, int gold) { super(name, health, damage, gold); } @Override public void attack() { System.out.printf("\n%s slashes with a sword.%n", getName()); } @Override public void defend() { isDefending = true; System.out.printf("%s is defending and will take reduced damage.%n", getName()); } }`

3	Encapsulation	Encapsulation adalah prinsip OOP di mana data (atribut) disembunyikan dan hanya dapat diakses atau dimodifikasi melalui metode tertentu, bukan langsung oleh pengguna.
		
		Atribut seperti health, damage, dan gold di kelas Character bersifat private atau dilindungi, sehingga tidak bisa langsung diubah dari luar kelas. Sebagai gantinya, akses ke atribut-atribut ini dilakukan melalui metode publik seperti getHealth(), setHealth(), getGold(), addGold(), dll.
		
		`private final String name;`
		`private int health;`
		`private final int damage;`
		`private final List<Item> inventory;`
		`protected boolean isDefending;`
		`private int gold;`
		
		Ini mencegah akses langsung dan memungkinkan validasi data, seperti memastikan health tidak lebih dari 100 atau tidak negatif.
		
		`public String getName() {`
		`return name;`
		`}`
		
		`public int getHealth() {`
		`return health;`
		`}`
		
		`public void setHealth(int health) {`
		`this.health += health;`
		`}`

<table>
<tr>
<td>4</td>
<td>Polymorphism</td>
<td>Polymorphism memungkinkan objek dari kelas yang berbeda untuk diperlakukan sebagai objek dari kelas induk yang sama, dan menyediakan implementasi yang berbeda untuk metode yang sama.

Dalam program ini, baik Warrior maupun Mage mengimplementasikan metode attack() dan defend() dengan cara yang berbeda, meskipun keduanya merupakan jenis dari kelas Character.


```
    @Override
    public void attack() {
        System.out.printf("\n%s slashes with a
sword.%n", getName());
    }
```



```
    @Override
    public void attack() {
        System.out.printf("\n%s casts a fireball.%n",
getName());
    }
```


Ketika program memanggil attack(), metode yang sesuai dengan jenis objek yang dimiliki oleh karakter yang sedang aktif akan dipanggil (baik itu Warrior atau Mage), meskipun pemanggilan attack() dilakukan pada referensi bertipe Character.</td>
</tr>
<tr>
<td>5</td>
<td>Exception</td>
<td>Exception handling digunakan untuk menangani situasi tak terduga atau kesalahan yang terjadi selama program berjalan. Di program ini, beberapa bagian menggunakan penanganan exception untuk menghindari kesalahan dan memberikan pengalaman pengguna yang lebih baik.

Saat memilih item dari inventaris atau memilih tindakan dalam pertempuran, pengguna mungkin memasukkan input yang tidak</td>
</tr>
</table>

		valid. Program menangani kesalahan ini menggunakan try-catch:
		
		`try { character.useItem(); } catch (IndexOutOfBoundsException e) { System.out.println("Invalid item choice. Please try again."); }`
		
		`try { return scanner.nextInt(); } catch (InputMismatchException e) { System.out.println("Invalid input. Please enter a number."); scanner.nextLine(); // clear the invalid input }`

6	Abstract class & Interface	**Abstract** Abstract class adalah kelas yang tidak bisa langsung diinstansiasi dan sering digunakan sebagai kelas dasar. Kelas ini dapat memiliki metode yang diimplementasikan sebagian (partial implementation) dan juga metode yang harus diimplementasikan oleh kelas turunannya. Kelas Character adalah kelas abstrak yang tidak bisa dibuat secara langsung. Ia menyediakan implementasi umum untuk atribut dan metode (seperti takeDamage(), addGold()), namun metode attack() dan defend() didefinisikan sebagai abstrak dan harus diimplementasikan oleh kelas turunannya (Warrior dan Mage). `public abstract class Character implements RPGCharacter`

		Interface
		Interface mendefinisikan kontrak atau spesifikasi yang harus diikuti oleh kelas yang mengimplementasikannya. Sebuah interface hanya berisi deklarasi metode tanpa implementasi.
		
		Interface RPGCharacter mendefinisikan metode umum yang harus diimplementasikan oleh semua kelas karakter (attack(), defend(), takeDamage(), dan getInfo()).
		
		`public interface RPGCharacter {`
		`    void attack();`
		`    void defend();`
		`    void takeDamage(int damage);`
		`    void getInfo();`
		`}`