

**UNIwersYTET RZESZOWSKI**  
**WYDZIAŁ NAUK ŚCISŁYCH I TECHNICZNYCH**  
**INSTYTUT INFORMATYKI**



*Gabriel Węglarz*  
137140

*Informatyka i ekonometria*

*Dokumentacja projektu prostej gry konsolowej w języku Java.*

Praca projektowa

Praca wykonana pod kierunkiem  
mgr. inż. Ewa Żesławska

Rzeszów 2025



## Spis treści

<b>1. Wprowadzenie</b>	6
1.1. Cele pracy	6
1.2. Zawartość pracy	6
<b>2. Struktura projektu</b>	7
2.1. Opis struktury projektu	7
2.2. Main oraz główne klasy programu	8
2.3. Klasa AItem oraz pochodne jej klasy	8
2.4. Klasa Entity oraz pochodne jej klasy	9
2.5. Typy wyliczeniowe i wyjątki	9
<b>3. Przykłady działania kodu oraz metod z wyjaśnieniem</b>	10
3.1. Kod metod oraz klas	10
3.2. Dane	13
<b>4. Założenia projektu oraz harmonogram jego realizacji</b>	14
4.1. Wymagania funkcjonalne	14
4.2. Wymagania нефункционалне	14
4.3. Harmonogram realizacji projektu	15
<b>5. Prezentacja warstwy użytkowej projektu</b>	16
5.1. Wyłapywanie błędów	17
5.2. Działanie systemu ekwipunku i przedmiotów	17
<b>6. Podsumowanie</b>	18
6.1. Dalsze możliwe kierunki rozwojowe	18
<b>Spis tabel</b>	19
<b>Spis rysunków</b>	20
<b>Spis listingów</b>	21

# 1. Wprowadzenie

Java to wysokopoziomowy, obiektowy język programowania opracowany przez firmę Oracle. Java jest językiem bezpiecznym oraz silnie typowanym, który kompiluje się do tzw. Bajtkodu. Posiada własny garbage collector oraz rozbudowane biblioteki standardowe.

## 1.1. Cele pracy

Celem pracy jest stworzenie prostej gry konsolowej w stylu RPG w języku Java. Praca projektowa pozwalać ma użytkownikowi na tworzenie, modyfikowanie, wyświetlanie oraz usuwanie danych, w tym przypadku postaci.

Celem pracy jest także demonstracja zapisu oraz odczytywaniu stanu obiektów stworzonych przez użytkownika. Stan wszystkich obiektów, stworzonych przez użytkownika i nie tylko, jest przechowywany w odrębnym pakiecie `data`.

Podczas realizacji projektu zostały zachowane podstawowe zasady programowania obiektowego przy projektowaniu struktury gry.

## 1.2. Zawartość pracy

Cała struktura pracy jest oparta na 20 klasach, w tym:

- trzy klasy abstrakcyjne,
- jeden interfejs,
- jeden wyjątek,
- dwa typy wyliczeniowe

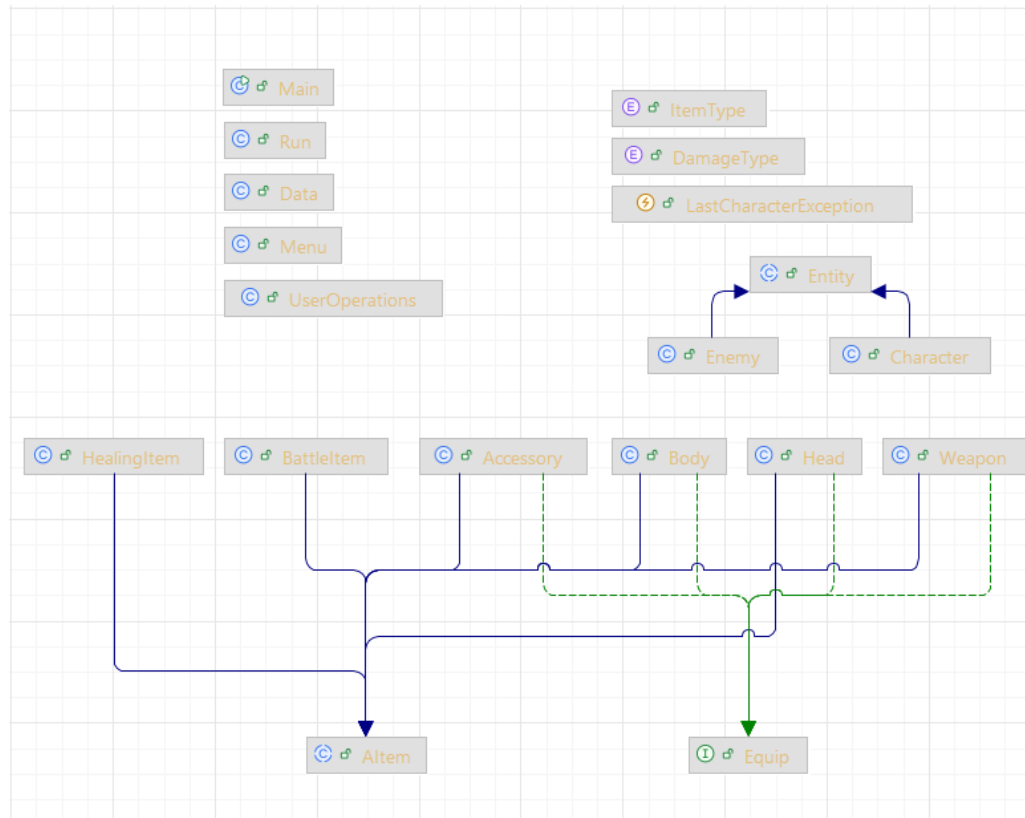
Projekt pozwala na tworzenie i przechowywanie informacji o stworzonych postaciach oraz ich klasach, statystykach. Pakiet `data` pełni rolę bazy danych, która przechowuje informacje wprowadzone przez użytkownika jak i dane dotyczące obiektów klas: `Enemy` oraz klas dziedziczących po klasie abstrakcyjnej `EquippableItem`.

Zawartość pracy obejmuje także:

- projektowanie i implementacje klas zgodnie z zasadami paradygmatu programowania obiektowego,
- obsługa własnego wyjątku dziedziczącego po klasie `Exception`,
- odczyt i zapis z plików tekstowych,
- dokumentację wykonaną w  $\text{\LaTeX}$ .

## 2. Struktura projektu

W rozdziale tym przedstawiono strukturę projektu w Javie. Omówione szczegółowo zostały metody i pola klas oraz pakiet przechowujący dane. Poniżej przedstawiono diagram klas stworzonego projektu.



Rys. 2.1. Diagram klas projektu.

### 2.1. Opis struktury projektu

Cała struktura pracy jest oparta na 20 klasach, w tym:

- trzy klasy abstrakcyjne,
- jeden interfejs,
- jeden wyjątek,
- dwa typy wyliczeniowe.

Projekt pozwala na tworzenie i przechowywanie informacji o stworzonych postaciach oraz ich klasach i statystykach. Pakiet data pełni rolę bazy danych, która przechowuje informacje wprowadzone przez użytkownika oraz dane dotyczące obiektów klas: Enemy oraz klas dziedziczących po klasie abstrakcyjnej EquipableItem.

Zawartość pracy obejmuje także:

- projektowanie i implementację klas zgodnie z zasadami paradygmatu programowania obiektowego,

- obsługę własnego wyjątku dziedziczącego po klasie `Exception`,
- odczyt i zapis z plików tekstowych,
- dokumentację wykonaną w `LaTeX`.

## 2.2. Main oraz główne klasy programu

Logika działania programu skupia się głównie na metodach klas: `Run`, `Data`, `UserOperations`, `Menu`. Metody te są składowymi metody `run`, która jest później uruchamiana w klasie `Main`.

Zadaniem klasy `Main` jest uruchomienie całego programu. Cała klasa składa się praktycznie z wywołania pojedynczej metody klasy `Run`. Dzięki prostocie klasy `Main` program jest bardziej przejrzysty, a logika działania menu użytkownika oraz reszty funkcji jest oddzielona do pozostałych klas.

Klasa `Run`, podobnie jak klasa `Main`, jest uboga w kod. Wywołuje ona metody klas `Data` i `Menu`, które stanowią składowe jedynej metody w tej klasie - `runMain()`. Ciało tej metody jest podzielone na dwie części: wczytanie danych z plików oraz uruchomienie menu użytkownika.

Program najpierw wczytuje informacje z plików tekstowych, a następnie uruchamia menu użytkownika.

Klasa `Menu` zawiera całą logikę funkcjonowania menu użytkownika. Składa się z metod, z których każda wyświetla inną sekcję menu użytkownika. Metody `startMenu()` oraz `mainMenu()` zawierają łącznie całą funkcjonalność interfejsu użytkownika.

Klasa `Data` przechowuje pola i metody.

Pola te przechowują dane o przedmiotach, przeciwnikach, postaciach, ekwipunku oraz pieniądzach. Dane o przedmiotach i przeciwnikach są wczytywane na początku programu z plików tekstowych umieszczonych w pakiecie `data`. Dane o postaciach są pobierane albo od użytkownika po rozpoczęciu nowej gry, albo przy wczytywaniu stanu gry z pliku tekstowego.

Klasa `UserOperations` zawiera głównie metody CRUD-owskie, odpowiedzialne za kolejno:

- tworzenie postaci,
- wyświetlanie danych postaci,
- aktualizowanie danych postaci,
- usuwanie postaci.

Metody te są niezbędne w funkcjonowaniu operacji wejścia/wyjścia użytkownika.

## 2.3. Klasa `Item` oraz pochodne jej klasy

Klasa `Item` to klasa abstrakcyjna będąca podstawą w dziedziczeniu dla klas `HealingItem`, `BattleItem` oraz `Weapon`, `Head`, `Body` i `Accessory`. Jest dość złożona, posiadając pola, konstruktory, gettery, settery oraz metodę `toString()`.

Klasami dziedziczącymi po niej są: `HealingItem`, `BattleItem`, `Weapon`, `Body`, `Head` oraz `Accessory`.

Można te klasy pogrupować na dwie grupy:

- Wyposażenie (`Weapon`, `Head`, `Body`, `Accessory`)
- Przedmioty (`HealingItem`, `BattleItem`)

Gdzie klasy odpowiadające jako wyposażenie, służą głównie do tego by program rozróżnił obiekty różnych klas. Przykładowo klasa `Character` ma cztery pola odpowiadające za wyposażenie i każde jest innego typu. Tak aby łatwiej można było przedzielać przedmioty do postaci. Wszystkie te klasy implementują interfejs `Equip`.

## 2.4. Klasa `Entity` oraz pochodne jej klasy

Klasa `Entity` jest klasą abstrakcyjną, która służy jako szablon dla klas `Enemy` oraz `Character`. Posiada wiele pól, głównie typu całkowitego, gettery i settery oraz metodę przysłoniętą `toString`. Jest to klasa abstrakcyjna, więc stworzenie jej obiektów jest niemożliwe.

Klasy dziedziczące po niej to:

- `Enemy`
- `Character`

Gdzie klasa `Character` przechowuje dane o postaci którą stworzy użytkownik, przechowuje obiekty klas `Item` oraz posiada metody modyfikujące wyposażenie. Posiada również metody służące do zapisywania danych postaci do pliku. Najpierw dane są przetworzone na ciąg znaków (`String`), a następnie zapisane do pliku tekstowego (`save.txt`).

Klasa `Enemy` natomiast ma służyć do przechowania wartości o przeciwnikach.

## 2.5. Typy wyliczeniowe i wyjątki

Klasa `DamageType` jest klasą, która służy jako typ wyliczeniowy. Celem tej klasy jest kategoryzowanie obiektów dziedziczących po `Item`, aby łatwiej było je rozróżniać i wczytywać z plików tekstowych przez program.

Typ wyliczeniowy `DamageType` może przyjmować jedną z trzech dostępnych wartości:

- `ONE`,
- `ALL`,
- `SPLASH`.

Klasa `ItemType` jest klasą, która służy jako typ wyliczeniowy. Celem tej klasy jest kategoryzowanie obiektów dziedziczących po `Item`, aby łatwiej było je rozróżniać i wczytywać z plików tekstowych przez program.

Typ wyliczeniowy `ItemType` może przyjmować jedną z sześciu dostępnych wartości:

- `HEALING`,
- `BATTLE`,
- `WEAPON`,
- `HEAD`,
- `BODY`,
- `ACCESSORY`.

Wyjątek `LastCharacterException` dziedziczy po klasie `Exception`, co oznacza, że jest ona klasą obsługującą własny wyjątek, tj. wyjątek w przypadku gdy użytkownik będzie chciał usunąć postać, mimo że została tylko jedna postać w drużynie.

Wyjątek ten ma za zadanie uniemożliwić sytuację, w której nie ma żadnej postaci w drużynie.

## 3. Przykłady działania kodu oraz metod z wyjaśnieniem

### 3.1. Kod metod oraz klas

Oto objaśnienie niektórych metod oraz klas stworzonych na potrzeby projektu w Javie.

**Listing 3.1.** Metoda newGame() w klasie Menu

```
1 // Metoda rozpoczynająca nową grę
2 public static void newGame() {
3     System.out.println("Stwórz 4 postaci:");
4     Data.characters[0] = UserOperations.createCharacter();
5     Data.characters[1] = UserOperations.createCharacter();
6     Data.characters[2] = UserOperations.createCharacter();
7     Data.characters[3] = UserOperations.createCharacter();
8     for (Character x : Data.characters) {
9         System.out.println(x.getName());
10    }
11 }
```

Metoda prosi użytkownika o stworzenie czterech postaci, wykorzystując metodę createCharacter(). Następnie wykonywana jest pętla for each, która wyświetla nazwy postaci nadane przez użytkownika.

**Listing 3.2.** Pola klasy Data

```
1 // Listy i tablica przechowujące dane o przedmiotach, przeciwnikach,
2 // drużynie oraz ewkipunku
3 static Character[] characters = new Character[4];
4 static ArrayList<AItem> itemList = new ArrayList<>();
5 static ArrayList<Enemy> enemyList = new ArrayList<>();
6 static ArrayList<AItem> inventory = new ArrayList<>();
7 static int money = 100;
8
9 // Statyczne pola do wczytywania danych od użytkownika
10 static Scanner readStringInput = new Scanner(System.in);
11 static Scanner readIntInput = new Scanner(System.in);
```

Pola przechowują dane wprowadzone przez użytkownika (jak w przypadku tablicy characters) lub odczytane z plików tekstowych znajdujących się w pakiecie data.

Pola typu Scanner służą do odczytywania danych wprowadzanych przez użytkownika.

**Listing 3.3.** Klasa Run

```
1 import java.io.IOException;
2 public class Run {
3     // Metoda rozpoczynająca grę
4     public static void runMain() throws IOException {
5         // Wczytanie danych z plików
6         Data.readEnemies();
7         Data.readItems();
8
9         // Menu startowe - Nowa gra, kontynuacja, pomoc, itd.
10        Menu.startMenu();
11    }
```



```

11
12     // Menu główne gry
13     Menu.mainMenu();
14 }
15 }

```

Klasa Run jest dość uboga i zawiera tylko jedną metodę – `runMain()`, składającą się z czterech innych metod klas `Data` oraz `Menu`. Metoda ta jest następnie wywoływana w klasie `Main`.

Klasa ta zawiera wyłącznie metody CRUD-owskie. Wśród nich znajdują się:

**Listing 3.4.** Metoda `createCharacter()`

```

1 // Metoda do tworzenia nowej postaci z poziomu użytkownika
2 public static Character createCharacter() {
3     System.out.println("Podać imię postaci");
4     characterName = Data.readStringInput.nextLine();
5     do{
6         System.out.println("Wybierz klasę postaci");
7         System.out.println("1. Rycerz\n2. Czarny Mag\n3. Biały Mag\n4. Złodziej");
8         switch (Data.readIntInput.nextInt()) {
9             case 1:
10                 return new Character(characterName, 200, 20, 15, 20, 5, 15, 10, 0);
11             case 2:
12                 return new Character(characterName, 100, 50, 5, 10, 25, 20, 15, 0);
13             case 3:
14                 return new Character(characterName, 125, 40, 5, 15, 20, 25, 12, 0);
15             case 4:
16                 return new Character(characterName, 150, 30, 15, 15, 10, 15, 20, 0);
17             default:
18                 System.out.println("Niepoprawna odpowiedź, wprowadź dane jeszcze raz
19                 .");
20         }
21     } while (true);
22 }

```

Metoda ta umożliwia tworzenie obiektów klasy `Character` bezpośrednio z poziomu konsoli. Użytkownik wprowadza imię postaci, a następnie program przechodzi do pętli, w której pyta o wybór klasy dla tej postaci. Jest to istotny krok, ponieważ wybór klasy determinuje wartości pól postaci. Pętla wykonuje się do momentu, gdy użytkownik poda jedną z akceptowalnych wartości.

**Listing 3.5.** Metoda `deleteCharacter()`

```

1 // Metoda usuwająca postać
2 public static void deleteCharacter() throws LastCharacterException{
3     try {
4         int nullCounter = 0;
5         for (Character character : Data.characters) {
6             if (character == null) nullCounter++;
7         }
8         if(nullCounter >= 3) throw new LastCharacterException("Nie można usunąć
ostatniej postaci!");
9         System.out.println("Wybierz postać do usunięcia:");
10        for (int i = 0; i < Data.characters.length; i++) {
11            if (Data.characters[i] == null) continue;
12            else {
13                System.out.println((i + 1) + ". " + Data.characters[i].getName());

```

```

14         }
15     }
16     switch (Data.readIntInput.nextInt()) {
17         case 1:
18             Data.characters[0] = null;
19             break;
20         case 2:
21             Data.characters[1] = null;
22             break;
23         case 3:
24             Data.characters[2] = null;
25             break;
26         case 4:
27             Data.characters[3] = null;
28             break;
29         default:
30             break;
31     }
32 }
33 catch (LastCharacterException e) {
34     e.printStackTrace();
35 }
36 }

```

Ta metoda usuwa postać z tablicy postaci. Działa w sposób interaktywny, ponieważ nie przyjmuje argumentów - zamiast tego użytkownik jest proszony o podanie indeksu postaci, którą chce usunąć. Program wyświetla nazwy wszystkich postaci wraz z ich indeksami, aby użytkownik mógł dokonać świadomego wyboru.

Metoda generuje również wyjątek własnej klasy `LastCharacterException` w sytuacji, gdy w tablicy pozostałe tylko jedna postać, a pozostałe elementy mają wartość `null`.

**Listing 3.6.** Typ wyliczeniowy `DamageType`

```

1 public enum DamageType {
2     ONE,
3     ALL,
4     SPLASH;
5
6     public static DamageType parseString(String s) {
7         return switch (s) {
8             case "ONE" -> DamageType.ONE;
9             case "ALL" -> DamageType.ALL;
10            case "SPLASH" -> DamageType.SPLASH;
11            default -> throw new IllegalArgumentException("Błędne dane: " + s);
12        };
13    }
14 }

```

**Listing 3.7.** Typ wyliczeniowy `ItemType`

```

1 // Typ wyliczeniowy ItemType używany do łatwiejszego rozróżniania przedmiotów
2 // do wczytywania z plików tekstowych
3 public enum ItemType {
4     HEALING,
5     BATTLE,
6     WEAPON,
7     HEAD,

```

```

8     BODY,
9     ACCESSORY;
10
11     public static ItemType parseString(String s) {
12         return switch (s) {
13             case "HEALING" -> ItemType.HEALING;
14             case "BATTLE" -> ItemType.BATTLE;
15             case "WEAPON" -> ItemType.WEAPON;
16             case "HEAD" -> ItemType.HEAD;
17             case "BODY" -> ItemType.BODY;
18             case "ACCESSORY" -> ItemType.ACCESSORY;
19             default -> throw new IllegalArgumentException("Błędne dane: " + s);
20         };
21     }
22 }

```

## 3.2. Dane

Poniżej znajdują się przykładowe dane obiektów przechowywane w plikach `items.txt` oraz `enemies.txt`. Nazwy pól są umieszczone na górze tabeli.

**Tabela 3.1.** Tabela przedmiotów - obiektów dziedziczących po `Item`.

itemName	value	sellValue	ItemType
Terra-ostrze	90	1000	WEAPON
Maska Cieni	14	300	HEAD
Pancerz Szlachecki	36	620	BODY
Amulet Królestwa	9	500	ACCESSORY
Eliksir	500	300	HEALING
Dynamit	150	180	BATTLE

**Tabela 3.2.** Tabela przeciwników

Nazwa	HP	Atak	Szybkość	Exp	\$
Zwinięty Wąż	20	5	2	1	4
Wampir	150	30	10	120	75
Złe Oko	130	22	8	65	35
Starman	400	55	10	250	120

## 4. Założenia projektu oraz harmonogram jego realizacji

Celem projektu jest stworzenie systemu gry konsolowej umożliwiającej użytkownikowi stworzenie, modyfikowanie, usuwanie oraz wyświetlanie postaci stworzonych przez użytkownika. Cały system ma na celu zautomatyzowanie procesu tworzenia postaci oraz modyfikowania jej statystyk. Zabezpieczenie w programie powodują że jest on odporny na błędy człowieka.

Program rozwiązuje problem dotyczący błędu człowieka podczas zapisywania danych, który zazwyczaj odbywa się w sposób ręczny w takiego typu grach. Problem ten pojawia się często w trakcie gier planszowych gdzie statystyki postaci często muszą być zapisywane ręcznie oraz ciągle aktualizowane podczas trwania gry.

W celu rozwiązania tego problemu zaprojektowałem program który sprawnie zapisuje oraz wyświetla statystyki postaci wraz z przedmiotami które posiadają, przy czym nie ma potrzeby na ręczne zapisywanie danych, gdyż są one przechowywane w oddzielnym pliku.

### 4.1. Wymagania funkcjonalne

- Tworzenie i edytowanie - Aplikacja umożliwia tworzenie nowych postaci z użyciem danych wprowadzonych od użytkownika oraz ich późniejszą modyfikację. W przypadku postaci można modyfikować tylko ich nazwę.
- Usuwanie - użytkownik po stworzeniu postaci może zdecydować o jej usunięciu później z opcji menu drużyny. Usunięte postacie nie mogą być przywrócone i zmniejszy to ilość członków w drużynie.
- Wyświetlanie danych - dane poszczególnych postaci, takie jak statystyki i ich wyposażenie są dostępne z poziomu użytkownika za pomocą menu drużyny. Dane te są aktualizowane po zmianie wyposażenia postaci.
- Zapisywanie i wczytywanie - Program ma funkcje zapisu stanu gry, tj. statystyk postaci ich ekwipunku oraz przedmiotów które zakupił użytkownik z menu sklepu. Dane te są konwertowane na ciąg znaków a następnie umieszczane w osobnym pliku w pakiecie `data`. Po uruchomieniu programu użytkownik może wybrać opcję wczytania stanu poprzedniej gry, jeżeli wcześniej ją zapisał.

### 4.2. Wymagania niefunkcjonalne

- Łatwość obsługi - program jest intuicyjny, interfejs użytkownika zawsze pokazuje użytkownikowi możliwe opcje do wyboru. Żadna opcja której użytkownik nie może wybrać nie jest ukryta.
- Zabezpieczenia - program jest zabezpieczony przez obsługiwane wyjątków oraz petlę które się wykonują bądź wyrzucą przy błędzie użytkownika. Po podaniu złej wartości lub wartości z poza zakresu program w odpowiednich miejscach albo wybierze wartość domyślną, np. wyjście z menu albo wyświetli komunikat o podaniu złej wartości i po prostu użytkownika o ponownym podaniu danych
- Czytelność - kod programu jest bardzo czytelny, klasy nie posiadają zbyt dużej ilości kodu tak aby w razie potencjalnego rozwoju programu można byłoby go łatwo edytować i modyfikować. Metody klas zostały specjalnie wydzielone do innych klas tak aby wszystkie metody nie skupiały się tylko w jednej klasie. Projekt zawiera duże ilości komentarzy które wyjaśniają działanie i przeznaczenie poszczególnych funkcji, metod, pól.

### 4.3. Harmonogram realizacji projektu

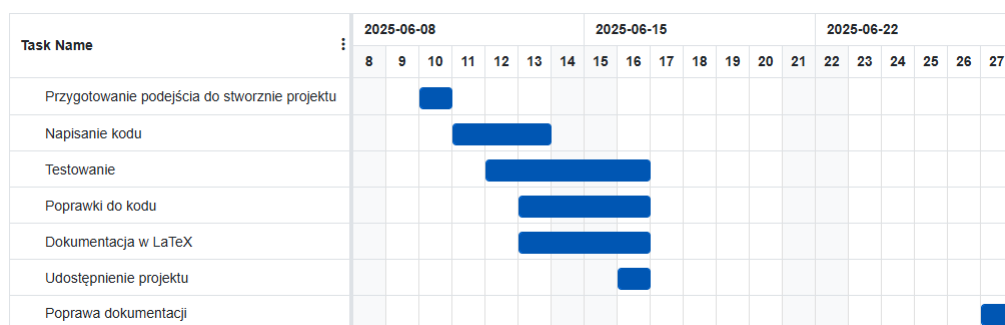
Harmonogram realizacji projektu został przedstawiony na diagramie ganta poniżej (Rys. 4.1).

Każde z zadań pokazanych na diagramie jest rozłożone w czasie od początku realizacji projektu do jego końca.

Główne zadania które zostały wykonane podczas projektu to przygotowanie samego pomysłu oraz podejścia do niego. Pomysł musiał być zgodny z zasadami programowania obiektowego w języku Java oraz musiał opierać się na klasach pochodnych, interfejsach oraz wyjątkach.

Kolejnym ważnym zadaniem była sama realizacja kodu w języku Java, która była kluczowym elementem tego projektu oraz testowanie kodu napisanego na bieżąco z pisaniem nowego.

Dużo czasu zostało też poświęcone na dokumentację która zdecydowanie okazała się najtrudniejszym oraz najbardziej czasochłonnym zadaniem.

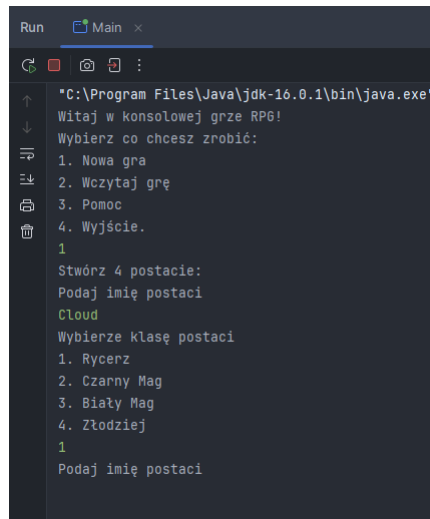


Rys. 4.1. Diagram Gantta dotyczący realizacji projektu

Diagram Ganta został wykonany za pomocą strony internetowej <https://www.onlinegantt.com/>

## 5. Prezentacja warstwy użytkowej projektu

W tym rozdziale jest przedstawiona przykładowe działanie programu. Program jest uruchomiony gdy uruchomiona zostanie klasa Main. Program przyjmuje dane od użytkownika z klawiatury, po wyborze złej opcji lub wpisaniu danych błędnego typu program się zapętli i będzie prosił użytkownika jeszcze raz o podanie danych. Nowa gra i tworzenie postaci wraz z wyborem klasy jest przedstawione na Rys. 5.1.



```
Run Main x
"C:\Program Files\Java\jdk-16.0.1\bin\java.exe"
Witaj w konsolowej grze RPG!
Wybierz co chcesz zrobić:
1. Nowa gra
2. Wczytaj grę
3. Pomoc
4. Wyjście.
1
Stwórz 4 postaci:
Podaj imię postaci
Cloud
Wybierz klasę postaci
1. Rycerz
2. Czarny Mag
3. Biały Mag
4. Złodziej
1
Podaj imię postaci
```

Rys. 5.1. Interfejs menu użytkownika.

Poniżej przedstawiony jest system działania sklepu w programie. Przedmioty te są generowane losowo, jednak ich wartości i nazwy są wczytywane z pliku `data.txt`.



```
Wybierz jedną z opcji:
1. Walka
2. Drużyna
3. Sklep
4. Zapis
5. Wyjście
3
Witaj w sklepie!
Mam do zaoferowania dziś takie oto przedmioty!
1. Proca, $60
2. Błyskawiczna Bomba, $210
3. Skórzana Czapa, $20
4. Proca, $60
5. Amulet Królestwa, $650
6. Szata Astralna, $520
3
Dziękuję, zapraszam ponownie!
Wybierz jedną z opcji:
1. Walka
2. Drużyna
3. Sklep
4. Zapis
5. Wyjście
|
```

```
Wybierz członka drużyny:
1. Cloud
2. Barret
3. Alice
4. Zack
1
Cloud
=====
HP: 200
MP: 20
Atak: 15
Obrona: 20
Magiczny Atak: 5
Magiczna Obrona: 15
Szybkość: 10
Doświadczenie: 0
Wybierz co chcesz zrobić:
1. Wyposażenie
2. Edycja
3. Usuń
4. Wyjście
1
1. Skórzana Czapa
Wybierz przedmiot:
1
Wyposażono przedmiot!
```

(a)

(b)

Rys. 5.2. Mechanizm sklepu (a) oraz mechanizm ekwipunku wraz z statystykami postaci (b)

## 5.1. Wyłapywanie błędów

Program jest zabezpieczony przez wpisywaniem błędnych wartości przez użytkownika. Jest zaprogramowany by wykrywać błędne typy tych wartości lub wartości z poza zakresu.

```
Wybierz klasę postaci
1. Rycerz
2. Czarny Mag
3. Biały Mag
4. Złodziej
X
Niepoprawne dane!
Wybierz klasę postaci
1. Rycerz
2. Czarny Mag
3. Biały Mag
4. Złodziej
|
```

(a)

```
Witaj w sklepie!
Mam do zaoferowania dziś takie oto przedmioty!
1. Elik sir, $300
2. Hełm Stalowy, $150
3. Mega Eter, $350
4. Zbroja Ciężka, $780
5. Ostrze Czarnej Materii, $500
6. Szata Starożytna, $550
9
Nie mam takiego przedmiotu!
Wybierz jedną z opcji:
1. Walka
2. Drużyna
3. Sklep
4. Zapis
5. Wyjście
|
```

(b)

**Rys. 5.3.** Mechanizm obsługi błędów użytkownika. W przypadku (a) program się zapętlą, a w przypadku (b) program wyjdzie z menu sklepu.

## 5.2. Działanie systemu ekwipunku i przedmiotów

Ekwipunek funkcjonuje jako lista przechowująca przedmioty, które zostają zakupione ze sklepu. Użytkownik może wyposażać postaci w te przedmioty za pomocą systemu wyposażenia. Po wyposażeniu danego przedmiotu statystyki postaci się zmieniają, jak i zamiast –Brak– mamy wpisaną nazwę danego przedmiotu.

```
Kamil
=====
HP: 100
MP: 50
Atak: 5
Obrona: 10
Magiczny atak: 25
Magiczna obrona: 20
Szybkość: 15
Doświadczenie: 0
Broń: --Brak--
Nakrycie głowy: --Brak--
Pancerz: --Brak--
Akcesorium: --Brak--
Wybierz co chcesz zrobić:
1. Wyposażenie
2. Edycja
3. Usuń
4. Wyjście
```

(a)

```
Kamil
=====
HP: 100
MP: 50
Atak: 95
Obrona: 10
Magiczny atak: 25
Magiczna obrona: 20
Szybkość: 15
Doświadczenie: 0
Broń: Terra-ostre
Nakrycie głowy: --Brak--
Pancerz: --Brak--
Akcesorium: --Brak--
Wybierz co chcesz zrobić:
1. Wyposażenie
2. Edycja
3. Usuń
4. Wyjście
```

(b)

**Rys. 5.4.** Statystyki postaci wraz z ekwipunkiem. Warto zwrócić uwagę na różnicę Ataku oraz Broni w obu przypadkach.

## 6. Podsumowanie

Projekt prostej gry konsolowej został w całości zaimplementowany w języku Java. System umożliwia wczytywanie danych od użytkownika i tworzenie na ich podstawie obiektów różnych klas. Obsługuje również odczyt danych z plików tekstowych oraz zapisywanie postępu użytkownika.

Projekt został zrealizowany w środowisku IntelliJ IDEA.

### 6.1. Dalsze możliwe kierunki rozwojowe

Dalszy rozwój projektu mógłby obejmować rozbudowę mechaniki dotyczącej postaci i przeciwników oraz poszerzenie funkcjonalności przedmiotów. Warto też wymienić możliwości:

- System walki z przeciwnikami
- Rozbudowany rozwój postaci z umiejętnościami
- Fabuła gry oraz system dialogu



# Spis tabel

3.1	Tabela przedmiotów - obiektów dziedziczących po AItem. . . . .	13
3.2	Tabela przeciwników . . . . .	13

# Spis rysunków

2.1	Diagram klas projektu. . . . .	7
4.1	Diagram Gantta dotyczący realizacji projektu . . . . .	15
5.1	Interfejs menu użytkownika. . . . .	16
5.2	Mechanizm sklepu (a) oraz mechanizm ekwipunku wraz z statystykami postaci (b) . . . .	16
5.3	Mechanizm obsługi błędów użytkownika. W przypadku (a) program się zapętli, a w przypadku (b) program wyjdzie z menu sklepu. . . . .	17
5.4	Statystyki postaci wraz z ekwipunkiem. Warto zwrócić uwagę na różnicę <code>Ataku</code> oraz <code>Broni</code> w obu przypadkach. . . . .	17

## Spis listingów

3.1	Metoda newGame() w klasie Menu . . . . .	10
3.2	Pola klasy Data . . . . .	10
3.3	Klasa Run . . . . .	10
3.4	Metoda createCharacter() . . . . .	11
3.5	Metoda deleteCharacter() . . . . .	11
3.6	Typ wyliczeniowy DamageType . . . . .	12
3.7	Typ wyliczeniowy ItemType . . . . .	12

Załącznik nr 2 do Zarządzenia nr 228/2021 Rektora Uniwersytetu Rzeszowskiego z dnia 1 grudnia 2021 roku w sprawie ustalenia procedury antyplagiatowej w Uniwersytecie Rzeszowskim

### OŚWIADCZENIE STUDENTA O SAMODZIELNOŚCI PRACY

..... Gabriel Węglarz .....  
Imię (imiona) i nazwisko studenta

Wydział Nauk Ścisłych i Technicznych

..... Informatyka i ekonometria .....  
Nazwa kierunku

..... 137140 .....  
Numer albumu

1. Oświadczam, że moja praca projektowa pt.: Dokumentacja projektu prostej gry konsolowej w języku Java, przygotowana w  $\text{\LaTeX}$ 
  - 1) została przygotowana przeze mnie samodzielnie\*,
  - 2) nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2021 r., poz. 1062) oraz dóbr osobistych chronionych prawem cywilnym,
  - 3) nie zawiera danych i informacji, które uzyskałem/am w sposób niedozwolony,
  - 4) nie była podstawą otrzymania oceny z innego przedmiotu na uczelni wyższej ani mnie, ani innej osobie.
2. Jednocześnie wyrażam zgodę/~~nie wyrażam zgody~~\*\* na udostępnienie mojej pracy projektowej do celów naukowo-badawczych z poszanowaniem przepisów ustawy o prawie autorskim i prawach pokrewnych.

Rzeszów 15.06.2025  
(miejscowość, data)

Gabriel Węglarz  
(czytelny podpis studenta)

\* Uwzględniając merytoryczny wkład prowadzącego przedmiot

\*\* – niepotrzebne skreślić