**Projekt zaliczeniowy**



Programowanie obiektowe

Rok akademicki 2024/2025

**Autorzy:**

Autor1: Julia Garlacz

Autor2: Piotr Filipiak

Autor3: Jakub Morawski

**System zarządzania wypożyczalnią gier**

Aplikacja do zarządzania wypożyczalnią gier, w której można dodać rezerwację gry, zwracać ją w określonych terminach oraz umożliwia zarządzanie bazą klientów i gier. System śledzi dostępność gier, rezerwacje oraz status klientów. W projekcie zaimplementowano klasy do reprezentowania gier, klientów (oraz osoby), rezerwacji i wypożyczalni jako całości.

Aplikacja posiada GUI, które umożliwia użytkownikowi:

* Dodawanie nowych klientów, gier oraz rezerwacji.
* Usuwanie istniejących klientów, gier i zwracanie rezerwacji.
* Przeglądanie wszystkich dodanych gier, klientów oraz rezerwacji w sposób przejrzysty.
* Zarządzanie dostępnością gier oraz historią wypożyczeń i zwrotów.

Program umożliwia zapis i odczyt danych do plików w formatach XML i JSON, dzięki czemu dane mogą być łatwo przechowywane i wczytywane w przyszłości.

**Podział ról**

Autor1: Interfejs graficzny (GUI) + Interfejs IComparable + klasa Gra + raport + zapis do XML oraz odczytanie

Autor2: Testy jednostkowe + dokumentacja + Interfejsy ICloneable + Interfejs IEquatable + klasa Klient + instrukcja działania programu GUI

Autor 3: Wykres UML + klasa Wypożyczalnia + klasa Rezerwacja + Klasa Program + klasa Osoba + wyjątki własne + zapis i odczytanie do json

1. **Klasa Gra**

Opis ogólny:

Klasa Gra reprezentuje grę dostępną w wypożyczalni. Zawiera podstawowe informacje o grze, takie jak tytuł, gatunek oraz liczba dostępnych egzemplarzy. Jest odpowiedzialna za przechowywanie danych dotyczących gier, które można wypożyczyć, i udostępnia metody pozwalające na porównanie gier oraz ich klonowanie.

**Uzasadnienie potrzeby stworzenia klasy:**

1. **Reprezentacja gier**: Klasa **Gra** pozwala na przechowywanie informacji o grze, takich jak tytuł, gatunek i dostępność, co jest niezbędne do funkcjonowania wypożyczalni.
2. **Zarządzanie dostępnością**: Dzięki polu Dostepnosc, system może śledzić liczbę dostępnych egzemplarzy gry i odpowiednio reagować na próby rezerwacji.
3. **Porównanie gier**: Implementacja interfejsu IEquatable<Gra> pozwala na łatwe porównywanie gier, co jest przydatne np. przy wyszukiwaniach i porównaniach gier w systemie.
4. **Klonowanie gier**: Zastosowanie interfejsu ICloneable umożliwia tworzenie kopii gier, co może być przydatne w sytuacjach, w których chcemy mieć identyczne obiekty, ale w różnych częściach programu.

**Modyfikatory dostępu:**

1. **Pola**:
   * tytul, gatunek, dostepnosc są prywatne (private), ponieważ nie chcemy, aby były bezpośrednio modyfikowane z zewnątrz klasy, a jedynie przez publiczne właściwości. Zabezpiecza to integralność danych.
2. **Właściwości**:
   * Tytul, Gatunek, Dostepnosc mają publiczne właściwości get i set, aby umożliwić dostęp do tych pól z zewnątrz, ale przy jednoczesnej kontroli nad tym, jak te dane są przechowywane i modyfikowane.
3. **Konstruktor**:
   * Konstruktor bezparametrowy jest publiczny, aby umożliwić tworzenie instancji klasy **Gra** bez konieczności ustawiania początkowych wartości. Konieczny jest, aby zapisywać pliki w xml.
   * Konstruktor z parametrami jest publiczny, pozwala na szybkie utworzenie instancji klasy **Gra** z konkretnymi danymi (tytuł, gatunek, dostępność). Dzięki temu użytkownik może tworzyć obiekty klasy **Gra** na podstawie zewnętrznych danych.
4. **Metody**:
   * **ShowInfo()**: Jest metodą publiczną, ponieważ jej celem jest wyświetlenie informacji o grze w konsoli. Można ją wywołać, aby użytkownik mógł zobaczyć szczegóły gry.
   * **Equals(Gra?)**: Metoda publiczna, pozwala na porównanie dwóch gier. Jest to istotne, gdy chcemy sprawdzić, czy dwie gry są takie same (np. przy wyszukiwaniach gier w systemie).
   * **Clone()**: Publiczna metoda umożliwiająca stworzenie kopii obiektu gry. Jest przydatna, gdy musimy stworzyć nowy obiekt, który jest identyczny z istniejącym.
5. **Klasa Osoba**

Opis ogólny:

Klasa Osoba jest klasą abstrakcyjną, która reprezentuje osobę posiadającą imię i nazwisko. Jest to klasa bazowa, która może być dziedziczona przez inne klasy, takie jak Klient. Zawiera podstawowe właściwości pozwalające na przechowywanie imienia i nazwiska osoby.

**Uzasadnienie potrzeby stworzenia klasy:**

1. **Reużywalność kodu:** Klasa Osoba pozwala na zorganizowanie wspólnych danych (imienia i nazwiska) w jednym miejscu, co zmniejsza duplikację kodu w klasach dziedziczących.
2. **Zarządzanie danymi:** Przechowywanie podstawowych informacji o osobach (takich jak imię i nazwisko) pozwala na łatwe rozszerzenie tej klasy o bardziej specyficzne właściwości w przyszłości.

**Modyfikatory dostępu:**

1. **Pola:**

* imie oraz nazwisko są prywatne, aby uniemożliwić bezpośrednią manipulację tymi danymi z zewnątrz klasy.

1. **Właściwości:**

* ImieNazwisko: Publiczna właściwość tylko do odczytu, zwraca pełne imię i nazwisko w formacie "Imię Nazwisko", co umożliwia dostęp do tych danych w jednym ciągu.
* Imie i Nazwisko: Publiczne właściwości z getterem i setterem, umożliwiające dostęp i modyfikację tych pól.

1. **Konstruktor:**

* Konstruktor domyślny jest publiczny, umożliwiając utworzenie instancji klasy Osoba bez konieczności podawania danych w momencie tworzenia obiektu.
* Konstruktor z parametrami jest publiczny, pozwala na inicjalizowanie obiektu klasy Osoba z określonymi wartościami imienia i nazwiska.

1. **Metoda:**

* ShowInfo(): Metoda wirtualna, publiczna, wyświetlająca informacje o osobie. Może być nadpisana w klasach pochodnych, aby dostosować sposób wyświetlania.

1. **Klasa Klient**

Opis ogólny:

Klasa Klient dziedziczy po klasie Osoba i reprezentuje klienta wypożyczalni gier. Zawiera dodatkowe informacje specyficzne dla klienta, takie jak unikalne ID, liczba wypożyczeń oraz status aktywności. Implementuje również interfejs IComparable<Klient>, umożliwiając porównywanie klientów na podstawie ich danych.

**Uzasadnienie potrzeby stworzenia klasy:**

1. **Reprezentacja klienta:** Klasa Klient umożliwia przechowywanie informacji o klientach, takich jak ich dane osobowe (przechwycone z klasy Osoba), ID, liczba wypożyczeń i status aktywności.
2. **Generowanie unikalnych ID:** Klasa automatycznie generuje unikalne ID dla każdego klienta, co jest kluczowe w kontekście identyfikacji użytkowników systemu.
3. **Porównywanie klientów:** Dzięki implementacji interfejsu IComparable<Klient> klasa pozwala na porównywanie klientów na podstawie nazwiska, a w razie potrzeby także na podstawie imienia. To przydatne przy sortowaniu listy klientów.

**Modyfikatory dostępu:**

1. **Pola:**

* nextId: Statyczne pole prywatne, odpowiedzialne za generowanie unikalnych ID dla klientów. Jest prywatne, ponieważ powinno być modyfikowane tylko w ramach klasy Klient.
* liczbaWypozyczen: Publiczne pole, aby umożliwić dostęp do liczby wypożyczeń z zewnątrz.
* IdKlienta: Publiczna właściwość, aby umożliwić dostęp do ID klienta, ale tylko do odczytu.
* IsActive: Publiczna właściwość, która pozwala na sprawdzenie, czy klient ma aktywne wypożyczenie.

1. **Właściwości:**

* Imie, Nazwisko: Dziedziczone z klasy Osoba. Publiczne właściwości, umożliwiające dostęp do imienia i nazwiska.

1. **Konstruktor:**

* Konstruktor domyślny jest publiczny, tworzy klienta z unikalnym ID i ustawia domyślne wartości dla liczby wypożyczeń oraz statusu aktywności.
* Konstruktor z parametrami inicjalizuje dane klienta (imię, nazwisko) oraz przypisuje unikalne ID.

1. **Metody:**

* ShowInfo(): Nadpisana metoda z klasy Osoba, wyświetlająca szczegóły klienta, w tym liczbę wypożyczeń oraz status aktywności.
* CompareTo(Klient): Metoda implementująca interfejs IComparable<Klient>, pozwalająca na porównanie klientów na podstawie nazwiska, a w razie potrzeby także imienia.

1. **Klasa Rezerwacja**

Opis ogólny:

Klasa Rezerwacja reprezentuje rezerwację gry przez klienta w wypożyczalni. Zawiera informacje o samej grze, kliencie dokonującym rezerwacji, dacie rezerwacji oraz opcjonalnej dacie zwrotu gry. Umożliwia również generowanie unikalnych ID dla rezerwacji.

**Uzasadnienie potrzeby stworzenia klasy:**

1. **Zarządzanie rezerwacjami**: Klasa **Rezerwacja** pozwala na przechowywanie danych o rezerwacjach, umożliwiając łatwe zarządzanie tymi informacjami w systemie wypożyczalni.
2. **Łączenie danych**: Dzięki tej klasie możliwe jest powiązanie gry i klienta w kontekście jednej rezerwacji. Ułatwia to przechowywanie i przetwarzanie danych w systemie.
3. **Generowanie unikalnych ID**: Klasa posiada statyczny licznik ID, który automatycznie przypisuje unikalne identyfikatory dla każdej nowej rezerwacji, co ułatwia późniejsze zarządzanie danymi.

**Modyfikatory dostępu:**

* **Pola**:
  + GraR: Publiczna właściwość, która przechowuje obiekt klasy **Gra** reprezentujący zarezerwowaną grę.
  + KlientR: Publiczna właściwość przechowująca obiekt klasy **Klient**, reprezentujący klienta dokonującego rezerwacji.
  + DataR: Publiczna właściwość przechowująca datę dokonania rezerwacji.
  + DataZ: Publiczna właściwość, która przechowuje opcjonalną datę zwrotu gry. Może być pusta (null), jeśli data zwrotu nie została jeszcze określona.
  + IdRezerwacji: Publiczna właściwość zawierająca unikalne ID rezerwacji. Jest przypisywana automatycznie przy tworzeniu rezerwacji.
* **Konstruktor**:
  + Konstruktor klasy **Rezerwacja** jest publiczny i przyjmuje cztery argumenty: obiekt klasy **Gra**, obiekt klasy **Klient**, datę rezerwacji oraz opcjonalną datę zwrotu gry.
* **Metody**:
  + ShowInfo(): Publiczna metoda, która wyświetla szczegóły rezerwacji, w tym ID, tytuł gry, imię i nazwisko klienta oraz daty rezerwacji i ewentualnie datę zwrotu. Metoda ta pozwala na łatwe prezentowanie informacji o danej rezerwacji w systemie.

**Uwagi:**

* **Statyczny licznik nextId**: Wartość tego licznika jest zwiększana za każdym razem, gdy tworzona jest nowa rezerwacja. Zastosowanie licznika umożliwia generowanie unikalnych identyfikatorów dla każdej nowej rezerwacji, co jest istotne z punktu widzenia zarządzania bazą danych rezerwacji.

1. **Klasa Wypozyczalnia**

Opis ogólny:

Klasa **Wypozyczalnia** reprezentuje wypożyczalnię gier, zarządzającą grami, klientami oraz rezerwacjami. Odpowiada za dodawanie, usuwanie, wyszukiwanie oraz zwracanie gier, a także zarządzanie rezerwacjami. Dodatkowo, umożliwia zapis i odczyt danych do i z plików w formacie XML oraz JSON.

**Uzasadnienie potrzeby stworzenia klasy:**

1. **Centralizacja logiki biznesowej**: Klasa zarządza wszystkimi operacjami związanymi z wypożyczaniem gier, co pozwala na centralizację logiki biznesowej w jednym miejscu.
2. **Przechowywanie i manipulacja danymi**: Klasa przechowuje listy gier, klientów i rezerwacji, umożliwiając dodawanie, usuwanie oraz modyfikowanie tych danych.
3. **Obsługa wyjątków**: Klasa zawiera odpowiednie mechanizmy obsługi wyjątków, takie jak brak dostępnych egzemplarzy gry, brak obiektów do usunięcia lub zakończenia rezerwacji, zapewniając bardziej bezpieczne i kontrolowane działanie systemu.

**Modyfikatory dostępu:**

* **Pola**:
  + gry: Prywatna lista przechowująca dostępne gry w wypożyczalni.
  + klienci: Prywatna lista przechowująca zarejestrowanych klientów.
  + rezerwacje: Prywatna lista przechowująca wszystkie rezerwacje.
* **Właściwości**:
  + Klienci: Publiczna właściwość umożliwiająca dostęp do listy klientów.
  + Gry: Publiczna właściwość umożliwiająca dostęp do listy gier.
  + Rezerwacje: Publiczna właściwość umożliwiająca dostęp do listy rezerwacji.
* **Wyjątki**:
  + **BrakEgzemplarzyException**: Rzucany, gdy nie ma dostępnych egzemplarzy danej gry.
  + **ObiektNieznalezionyException**: Rzucany, gdy obiekt (np. klient czy gra) nie zostanie znaleziony w systemie.
  + **NiedostepnaAkcjaException**: Rzucany, gdy próba wykonania akcji jest niedozwolona (np. zwrot gry, która już została oddana).
* **Metody**:
  + **DodajGre(Gra gra)**: Dodaje grę do listy gier w wypożyczalni.
  + **DodajKlienta(Klient klient)**: Dodaje klienta do listy klientów.
  + **DodajRezerwacje(Rezerwacja rezerwacja)**: Dodaje rezerwację, zmniejszając dostępność gry o 1.
  + **ZwrocGre(int id)**: Zwraca grę, zwiększając jej dostępność o 1.
  + **UsunKlienta(Klient klient)**: Usuwa klienta z listy, jeśli nie ma aktywnych rezerwacji.
  + **UsunGre(Gra gra)**: Usuwa grę z listy gier.
  + **UsunRezerwacje(Rezerwacja rezerwacja)**: Usuwa rezerwację z listy rezerwacji.
  + **ZnajdzKlienta(int idKlienta)**: Wyszukuje klienta po ID.
  + **ZnajdzGre(string tytul)**: Wyszukuje grę po tytule.
  + **WyswietlGry()**: Wyświetla dostępne gry.
  + **WyswietlKlientow()**: Wyświetla zarejestrowanych klientów.
  + **WyswietlRezerwacje()**: Wyświetla listę wszystkich rezerwacji.
  + **ZapiszXML(string nazwa)**: Zapisuje dane wypożyczalni do pliku XML.
  + **OdczytajXML(string nazwa)**: Odczytuje dane wypożyczalni z pliku XML.
  + **ZapiszDane(string? sciezka)**: Zapisuje dane wypożyczalni do pliku JSON.
  + **WczytajDane(string? sciezka)**: Wczytuje dane wypożyczalni z pliku JSON.

1. **Klasa Program**

Opis ogólny:  
Klasa Program jest głównym punktem wejścia do aplikacji wypożyczalni gier planszowych. Odpowiada za uruchomienie aplikacji, tworzenie instancji systemu wypożyczalni oraz zarządzanie interakcjami użytkownika poprzez interfejs tekstowy. Umożliwia dodawanie gier, klientów, rezerwacji, a także zarządzanie zapisem i odczytem danych z plików.

**Uzasadnienie potrzeby stworzenia klasy**:

1. **Punkt wejścia do aplikacji**: Klasa Program jest odpowiedzialna za rozpoczęcie działania całej aplikacji. To w tej klasie inicjowane są wszystkie operacje na systemie wypożyczalni, takie jak dodawanie gier, klientów, rezerwacji oraz zarządzanie zapisami i odczytami danych.
2. **Zarządzanie interakcją z użytkownikiem**: Klasa zawiera interfejs tekstowy, który umożliwia użytkownikowi wybór operacji do wykonania. Dzięki temu użytkownik ma pełną kontrolę nad systemem, a każda opcja jest intuicyjnie dostępna.
3. **Testowanie i przykłady danych**: Klasa umożliwia testowanie funkcjonalności systemu poprzez dodawanie przykładowych danych (gier, klientów, rezerwacji) oraz zapis i odczyt danych z plików, co pozwala na łatwe testowanie aplikacji w różnych scenariuszach.

**Modyfikatory dostępu**:

* **public**:
  + Main: Funkcja Main jest publiczna, ponieważ stanowi punkt wejścia do aplikacji i musi być dostępna do uruchomienia programu.
* **internal**:
  + Cała klasa Program jest oznaczona jako internal, co oznacza, że jest dostępna tylko w obrębie tego samego projektu i nie może być używana z innych projektów.

**Metody**:

* **Main**: Główna metoda, która uruchamia aplikację, tworzy instancję wypożyczalni, dodaje dane testowe i wyświetla interfejs użytkownika.
* **TextGUI**: Metoda odpowiedzialna za wyświetlanie menu interaktywnego, w którym użytkownik może wybierać dostępne operacje na systemie wypożyczalni.
* **TestGier**: Metoda do dodawania przykładowych danych (gier, klientów, rezerwacji), umożliwiająca testowanie działania aplikacji.

**Opis działania**:

* Program inicjalizuje obiekt Wypozyczalnia, który zarządza grami, klientami i rezerwacjami.
* Użytkownik może wybrać operacje takie jak dodawanie gier, rezerwowanie gier, usuwanie klientów, zapis i odczyt danych z plików.
* Program umożliwia interakcję z użytkownikiem poprzez prosty interfejs tekstowy, który pomaga zarządzać całą aplikacją wypożyczalni gier planszowych.

**Diagram klas**

Obraz zawierający tekst, diagram, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

**Opis funkcjonalności**

W projekcie zaimplementowane zostały podstawowe funkcjonalności, które umożliwiają pełne zarządzanie systemem wypożyczalni gier planszowych. Każda z funkcji została przypisana odpowiednim klasom, które realizują różne aspekty działania systemu.

Główne funkcjonowanie programu zostało przedstawione w instrukcji i skupia się na interfejsie użytkownika, który został stworzony przy użyciu technologii WPF (Windows Presentation Foundation). Nasza aplikacja WPF została nazwana WpfApp2. Aby połączyć ją z kodem głównym, do projektu WPF dodano osobny projekt o nazwie WypozyczalniaGier, a następnie ustawiono odniesienie do tego projektu w aplikacji WPF.

Dzięki temu rozwiązaniu, w pliku MainWindow.xaml.cs, udało się dodać odniesienie do jednej z klas projektu bazowego. Nie było więc koniecznym, aby pisać oddzielne funkcje   
w GUI. Zamiast tego, w interfejsie użytkownika po prostu odwoływaliśmy się do funkcji znajdujących się w klasach projektu WypozyczalniaGier, co pozwoliło na uproszczenie kodu i zwiększenie spójności między warstwą graficzną a logiką aplikacji.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Skupimy się teraz na tym, jak kod bazowy wykorzystany jest w GUI.

W części wizualnej mamy klasę Gry, gdzie można dodawać grę do systemu:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

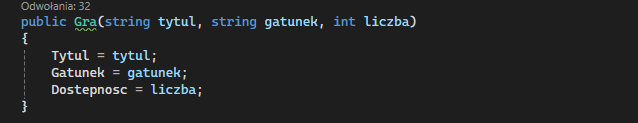
Opis wygenerowany automatycznie

Korzystamy z funkcji DodajGre z klasy wypożyczalnia.

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

Wykorzystujemy także konstruktor z klasy Gra, aby utworzyć w formularzu dodawania gier, nową grę:



Możemy także usunąć grę z systemu, jeżeli wybierzemy jakąś pozycję z listy dodanych gier.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

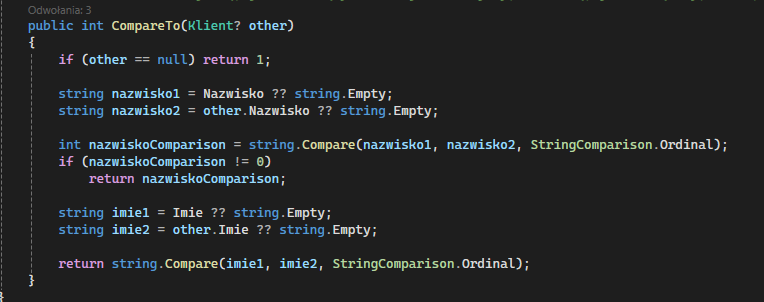
Opis wygenerowany automatycznie

Korzystamy z funkcji UsunGre z wypożyczalni



Analogicznie zachodzą funkcję w innych zakładkach, dlatego nie zamieszczamy do nich zrzutów ekranu kodu.

Zakładka klient umożliwia dodawanie klienta, dzięki funkcji z klasy Wypozyczlania – dodajKlienta oraz konstruktorów z klas Osoba i Klient. Można także usunąć zaznaczonego na liście klienta, za pomocą funkcji UsunKlienta z klasy Wypozyczalnia. Dodatkowo można posortować klientów po nazwiskach dzięki interfejsowi IComparable<Klient> za pomocą funkcji.



Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Dzięki temu, gdy w GUI wykorzystujemy funkcję Sort() klienci na liście sortują się.

Rezerwację możemy dodać za pomocą kodu z klasy Wypozyczalnia oraz konstruktorów z klasy Rezerwacje.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Można też ją zwrócić, dzięki czemu dostępność gry zwiększy się. Funkcja ZwrocGre z klasy Wypozyczalnia odpowiada funkcją usuwania.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie