

laboratoria
Miniprojekt Biblioteka
wersja 1.9

Uwaga: To jest przykład koncepcji rozwiązania zadań laboratoryjnych. Zadania laboratoryjne mogą nakazywać wykonanie czegoś większego i bardziej złożonego oraz czegoś, czego tu nie ma.

Uwaga: Ten dokument co jakiś czas się zmienia. Dochodzą rzeczy z kolejnych etapów prac, a wcześniej zamieszczone rzeczy są poprawiane i dostosowywane do nowych. Proszę zawsze korzystać z najnowszej wersji.

Uwaga: Treść szaro-zielonych ramek, prowadzące z nich strzałki, kolorowe otoczenia fragmentu powtózonego diagramu, zrzuty ekranu programu *Visual Paradigm* – to dodatkowe wyjaśnienia, a nie przykład wykonania zadań laboratoryjnych.

Spis treści

Modelowanie wymagań i przypadków użycia.....	2
Miejsce wdrożenia.....	2
Wymagania.....	2
Diagram przypadków użycia.....	3
Słowny opis przypadków użycia.....	4
Modelowanie realizacji przypadków użycia.....	11
Diagramy czynności.....	11
Modelowanie obiektowej 3-warstwowej architektury.....	14
Diagram komponentów.....	14
Diagram klas komponentu <i>Kontroler</i>	14
Diagram klas komponentu <i>Model</i>	16
Kod klas komponentów <i>Kontroler</i> i <i>Model</i>	19
Modelowanie interakcji między klasami i obiekta mi.....	26
Diagramy sekwencji i kod opisanych nimi operacji.....	26
Wynik działania programu.....	32
Kod klas.....	35
Modelowanie zmienności warstwy widoku.....	59

Modelowanie wymagań i przypadków użycia

Miejsce wdrożenia

dokument tekstowy z zadania 3

Nazwa projektu:

System Zarządzania Biblioteką, zwany dalej systemem.

Klient:

Miejska Biblioteka Publiczna w Miasteczku, zwana dalej biblioteką.

Cel projektu:

Wykonanie systemu informatycznego wspierającego zadania pracowników biblioteki w zakresie przetwarzania danych o wypożyczanych przez nią książkach i o jej czytelnikach.

Zasoby ludzkie:

Użytkownikami systemu są pracownicy biblioteki:

- 1) *Bibliotekarz* – w tej roli występuje każdy pracownik biblioteki. Zajmuje się wprowadzaniem do i usuwaniem książek z inwentarza oraz wypożyczaniem i przyjmowaniem zwrotów książek od czytelników.
- 2) *Kierownik Biblioteki* – w tej roli występuje tylko jedna osoba, kierująca biblioteką. Zajmuje się zarządzaniem danymi czytelników biblioteki oraz zapisywaniem i wypisywaniem ich z biblioteki.

Przepisy i strategie:

- 1) Biblioteka działa na podstawie jej statutu i regulaminu oraz odpowiednich ustaw.
- 2) Użytkowników systemu obowiązuje przestrzeganie art. 13 ust. 1 i ust. 2 Rozporządzenia Parlamentu Europejskiego i Rady (UE) 2016/679 z 27 kwietnia 2016 r. w sprawie ochrony osób fizycznych w związku z przetwarzaniem danych osobowych (tzw. RODO).
- 3) Osobą zarządzającą i administratorem danych osobowych czytelników biblioteki jest Kierownik Biblioteki.
- 4) Wszystkie informacje o pracy biblioteki wcześniej prowadzone były w postaci znormalizowanej papierowej dokumentacji. System przejmuje to zadanie, zapewniając odpowiednie zachowanie stosowanych w tej dokumentacji norm.
- 5) System częściowo lub całkowicie (jeśli możliwe) automatyzuje operacje wykonywane przez jego użytkowników.
- 6) Bibliotekarz jest odpowiedzialny za bezwłoczną i poprawną rejestrację w systemie każdego zdarzenia związanego z książkami biblioteki, w szczególności dotyczącymi zmiany ich stanu i miejsca.
- 7) Kierownik Biblioteki jest odpowiedzialny za bezwłoczną i poprawną rejestrację w systemie każdego zdarzenia związanego z czytelnikami biblioteki, w tym ich zapisanie do i wypisanie z biblioteki.
- 8) Warunkiem wypisania czytelnika z biblioteki jest nieposiadanie przez niego aktywnych wypożyczeń książek.

Dane techniczne:

- 1) Użytkownik systemu korzysta z niego za pomocą komputera stacjonarnego z systemem operacyjnym Ubuntu 24.04 LTS z graficznym interfejsem GNOME.
- 2) Jednocześnie z systemu korzysta tylko jedna osoba.
- 3) Przetwarzanie i przechowywanie danych przez system odbywa się lokalnie, bez dostępu do sieci LAN lub internetu.
- 4) System zapewnia przechowywanie i przetwarzanie danych maksymalnie 10 000 książek (fizycznych bytów) i maksymalnie 20 000 czytelników.

Wymagania

dokument tekstowy z zadania 4

Wymagania funkcjonalne:

- F01) System przechowuje i przetwarza dane o książkach (w tym o ich stanie i miejscu), czytelnikach i wypożyczeniach książek.
- F02) Bibliotekarz wypożycza książkę czytelnikowi.
- F03) Bibliotekarz kończy wypożyczenie książki przez przyjęcie jej zwrotu od czytelnika lub (sporadycznie) przez usunięcie wypożyczonej, ale zgubionej lub zniszczonej książki z inwentarza biblioteki.
- F04) Bibliotekarz wprowadza książkę do inwentarza biblioteki.
- F05) Bibliotekarz usuwa zgubioną lub zniszczoną książkę z inwentarza biblioteki.

F06) Kierownik Biblioteki zarządza danymi osobowymi czytelnika biblioteki.

F07) Kierownik Biblioteki zapisuje czytelnika do biblioteki.

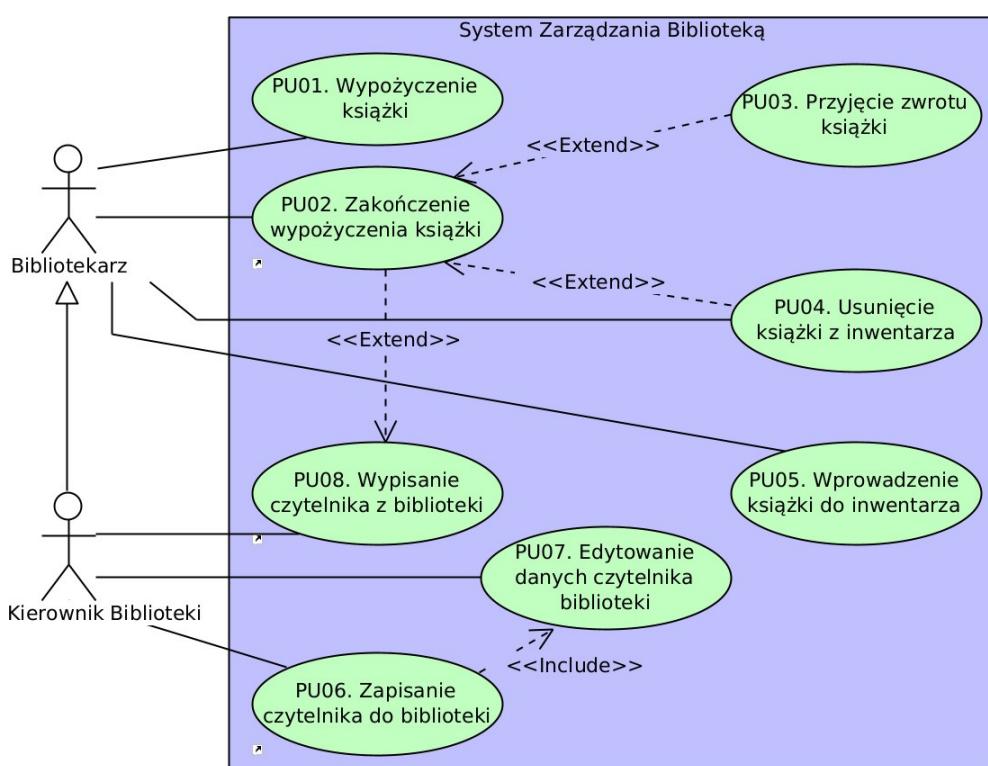
F08) Kierownik Biblioteki wypisuje czytelnika z biblioteki.

Wymagania niefunkcjonalne:

- N01) Działania użytkownika systemu na książkach i na czytelnikach są obsługiwane i automatyzowane (jeśli to możliwe) przez system.
- N02) Działania użytkownika systemu na książkach i na czytelnikach są trwale rejestrowane w systemie.
- N03) System zapewnia przechowywanie i przetwarzanie danych maksymalnie 10 000 książek (fizycznych bytów) i maksymalnie 20 000 czytelników.
- N04) Przetwarzanie i przechowywanie danych przez system odbywa się lokalnie, bez dostępu do sieci LAN lub internetu.
- N05) Wszystkie informacje o pracy biblioteki są rejestrowane w systemie w postaci znormalizowanej.
- N06) Wszystkie działania Bibliotekarza w systemie może też wykonać Kierownik Biblioteki.
- N07) Tylko Kierownik Biblioteki zarządza danymi osobowymi czytelnika biblioteki.
- N08) Warunkiem wypisania czytelnika z biblioteki jest nieposiadanie przez niego aktywnych wypożyczeń książek.

Diagram przypadków użycia

diagram z zadania 5



Rys. 1. Diagram przypadków użycia dla Systemu Zarządzania Biblioteką.

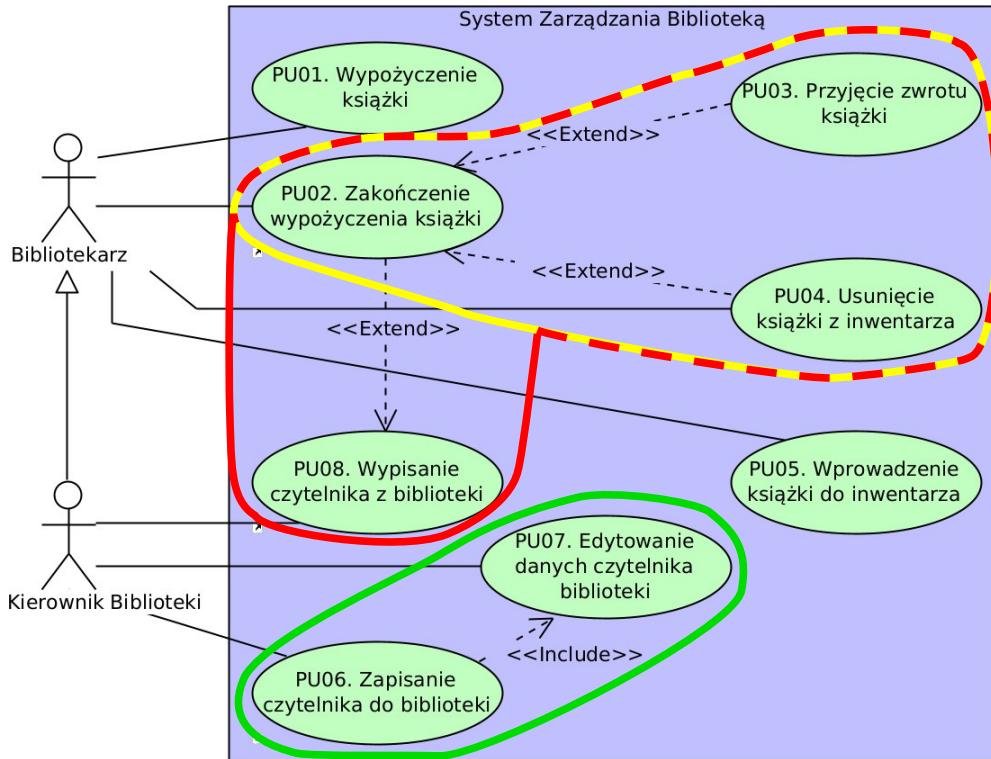
Na podstawie pojedynczego wymagania funkcjonalnego powstał dokładnie jeden prosty lub złożony przypadek użycia. To dobra sytuacja. Ten przypadek użycia jest w asocjacji z aktorem, ponieważ jest lub może być inicjowany przez tego aktora, aby spełnić odpowiadające mu wymaganie funkcjonalne. Pozostałe przypadki użycia wynikają z analizy wspólności i zmienności przypadków użycia.

Które przypadki użycia są lub mogą być inicjowane przez aktora?

- są: PU01, PU05, PU06, PU08.
- mogą być: PU02, PU04, PU07.
- pozostałe są inicjowane tylko przez inne PU.

Które przypadki użycia są złożone?

- PU02 może być rozszerzony albo o PU03, albo o PU04 (alternatywa).
- PU08 może być rozszerzony o PU02 (opcja).
- PU06 zawiera PU07.



Rys. 2. Diagram przypadków użycia dla Systemu Zarządzania Biblioteką z zaznaczeniem złożonych przypadków PU02 (na żółto), PU06 (na zielono) i PU08 (na czerwono).

Słowny opis przypadków użycia dokument tekstowy z zadania 6

PU01. Wypożyczenie książki:

Cel: Wypożyczenie książki czytelnikowi.

Warunki wstępne:

- Inicjacja przez Bibliotekarza:
 - Bibliotekarz zna nr książki i nr czytelnika.

Warunki końcowe:

- Zarejestrowano wypożyczenie książki i wyłączono jej dostępność do wypożyczenia.
- Spełniono wymaganie *F02: Bibliotekarz wypożycza książkę czytelnikowi.*

Scenariusz:

1. Bibliotekarz wprowadza nr książki.
2. Bibliotekarz wprowadza nr czytelnika.
3. Jeśli książka jest dostępna do wypożyczenia i czytelnik jest zarejestrowany w bibliotece:
 - 3.1. Bibliotekarz zatwierdza wypożyczenie książki.
 - 3.2. System rejestruje wypożyczenie książki przez czytelnika.
 - 3.3. System wyłącza dostępność książki do wypożyczenia.
 - 3.4. System rejestruje zdarzenie wypożyczenia książki.
4. W przeciwnym razie:
 - 4.1. System informuje o niemożności wypożyczenia książki czytelnikowi.

PU02. Zakończenie wypożyczenia książki:

Cel: Zakończenie wypożyczenia książki czytelnikowi przez jej zwrot lub usunięcie z inwentarza.

Warunki wstępne:

- Inicjacja przez Bibliotekarza:
 - Bibliotekarz zna nr książki i nr czytelnika.
- Inicjacja przez PU08. *Wypisanie czytelnika z biblioteki:*
 - PU08 przekazuje nr książki.

Warunki końcowe:

- Wykonano PU03. *Przyjęcie zwrotu książki* lub PU04. *Usunięcie książki z inwentarza.*

- Spełniono wymaganie *F03: Bibliotekarz kończy wypożyczenie książki przez przyjęcie jej zwrotu od czytelnika lub (sporadycznie) przez usunięcie wypożyczonej, ale zgubionej lub zniszczonej książki z inwentarza biblioteki.*

Scenariusz:

1. Jeśli inicjacja przez Bibliotekarza:

- 1.1. Bibliotekarz wprowadza nr książki.

2. Bibliotekarz lub Kierownik Biblioteki wybiera opcję.

3. Jeśli opcją jest przyjęcie zwrotu książki:

- 3.1. Inicjacja **PU03. Przyjęcie zwrotu książki** z podaniem numeru książki.

4. A jeśli opcją jest usunięcie książki z inwentarza:

- 4.1. Inicjacja **PU04. Usunięcie książki z inwentarza** z podaniem numeru książki.

W tych punktach następuje rozszerzenie opisywanego przypadku użycia przez PU03 i PU04.

PU03. Przyjęcie zwrotu książki

Cel: Przyjęcie zwrotu książki od czytelnika.

Warunki wstępne:

- Inicjacja przez *PU02. Zakończenie wypożyczenia książki*:

- PU02 przekazuje nr książki.

Warunki końcowe:

- Zarejestrowano przyjęcie zwrotu książki i włączono jej dostępność do wypożyczenia.

Scenariusz:

1. Jeśli książka jest wypożyczona:

- 1.1. Bibliotekarz zatwierdza zwrot książki.

- 1.2. System włącza dostępność książki do wypożyczenia.

- 1.3. System rejestruje zdarzenie zwrotu książki.

2. W przeciwnym razie:

- 2.1. System informuje o niemożności zwrotu książki.

PU04. Usunięcie książki z inwentarza

Cel: Usunięcie książki z inwentarza.

Warunki wstępne:

- Inicjacja przez Bibliotekarza:

- Bibliotekarz zna nr książki.

- Inicjacja przez *PU02. Zakończenie wypożyczenia książki*:

- PU02 przekazuje nr książki.

Warunki końcowe:

- Usunięto książkę z inwentarza.

- Spełniono wymaganie *F05: Bibliotekarz usuwa zgubioną lub zniszczoną książkę z inwentarza biblioteki.*

Scenariusz:

1. Jeśli Inicjacja przez Bibliotekarza:

- 1.1. Bibliotekarz wprowadza nr książki.

2. Bibliotekarz zatwierdza usunięcie książki.

3. Książka jest usuwana z inwentarza.

4. System rejestruje zdarzenie usunięcia książki.

PU05. Wprowadzenie książki do inwentarza

Cel: Wprowadzenie nowej książki do inwentarza.

Warunki wstępne:

- Inicjacja przez Bibliotekarza:

- Bibliotekarz zna wszystkie wymagane dane opisujące książkę.

Warunki końcowe:

- Wprowadzono książkę do inwentarza.

- Spełniono wymaganie *F04: Bibliotekarz wprowadza książkę do inwentarza biblioteki.*

Scenariusz:

1. Bibliotekarz wprowadza dane książki (tytuł, autor, wydawca, rok wydania, nry klasyfikacji tematycznej, nr ISBN, ...).
2. Bibliotekarz zatwierdza wprowadzenie książki do inwentarza.

3. System nadaje książce unikalny numer.
4. System informuje o numerze książki.
5. System wprowadza książkę do inwentarza.
6. System rejestruje zdarzenie wprowadzenia książki do inwentarza.

PU06. Zapisanie czytelnika do biblioteki

Cel: Zapisanie nowego czytelnika do biblioteki.

Warunki wstępne:

- Inicjacja przez Kierownika Biblioteki:
 - Kierownik Biblioteki zna wszystkie wymagane dane osobowe czytelnika.

Warunki końcowe:

- Zapisano czytelnika do biblioteki.
- Spełniono wymaganie *F07: Kierownik Biblioteki zapisuje czytelnika do biblioteki.*

Scenariusz:

1. Inicjacja **PU07. Edytowanie danych czytelnika biblioteki.**
2. Kierownik Biblioteki zatwierdza zapisanie czytelnika do biblioteki.
3. Jeśli osoba o podanych danych jest już czytelnikiem biblioteki:
 - 3.1. System informuje o niemożności zapisania czytelnika.
4. W przeciwnym razie:
 - 4.1. System nadaje czytelnikowi unikalny numer.
 - 4.2. System informuje o numerze czytelnika.
 - 4.3. System zapisuje czytelnika do biblioteki.
 - 4.4. System rejestruje zdarzenie zapisania czytelnika do biblioteki.

W tym punkcie następuje
włączenie PU07
do opisywanego
przypadku użycia.



PU07. Edytowanie danych czytelnika biblioteki

Cel: Edycja lub wprowadzenie danych osobowych czytelnika.

Warunki wstępne:

- Inicjacja przez Kierownika Biblioteki:
 - Kierownik Biblioteki zna nr czytelnika.
- Inicjacja przez *PU06. Zapisanie czytelnika do biblioteki.*

Warunki końcowe:

- Dane czytelnika są poprawne i kompletne.
- Spełniono wymaganie *F06: Kierownik Biblioteki zarządza danymi osobowymi czytelnika biblioteki.*

Scenariusz:

1. Jeśli inicjacja przez Kierownika Biblioteki:
 - 1.1. Kierownik Biblioteki wprowadza nr czytelnika.
 - 1.2. System udostępnia do edycji wszystkie dane osobowe czytelnika.
2. Kierownik Biblioteki może edytować dane osobowe czytelnika (z wyjątkiem numeru czytelnika: imię, nazwisko, nr PESEL, adres, ...).
3. Kierownik Biblioteki zatwierdza edytowanie danych.
4. Jeśli dane osobowe czytelnika są poprawne i kompletne:
 - 4.1. Jeśli inicjacja przez Kierownika Biblioteki:
 - 4.1.1. System aktualizuje dane osobowe czytelnika.
 - 4.1.2. System rejestruje zdarzenie edycji danych osobowych czytelnika.
 - 4.2. W przeciwnym razie:
 - 4.2.1. PU kończy się zwroceniem zatwierdzonych danych osobowych czytelnika.
5. W przeciwnym razie:
 - 5.1. System informuje o niekompletności lub niepoprawności danych osobowych czytelnika.
 - 5.2. Przejdź do 2. Kierownik Biblioteki...

PU08. Wypisanie czytelnika z biblioteki

Cel: Wypisanie czytelnika z biblioteki.

Warunki wstępne:

- Inicjacja przez Kierownika Biblioteki:

- Kierownik Biblioteki zna nr czytelnika.

Warunki końcowe:

- Wypisano czytelnika z biblioteki.
- Wykonano PU02. *Zakończenie wypożyczenia książki dla wszystkich książek wypożyczonych przez czytelnika.*
- Spełniono wymaganie F08: *Kierownik Biblioteki wypisuje czytelnika z biblioteki.*
- Spełniono wymaganie N08: *Warunkiem wypisania czytelnika z biblioteki jest nieposiadanie przez niego aktywnych wypożyczeń książek.*

Scenariusz:

1. Kierownik Biblioteki zatwierdza wypisanie czytelnika.
2. Dla każdej książki wypożyczonej przez czytelnika:
 - 2.1. Inicjacja **PU02. Zakończenie wypożyczenia książki** z podaniem numeru książki.
3. Jeśli w inwentarzu brak książek wypożyczonych przez czytelnika:
 - 3.1. Czytelnik jest wypisywany z biblioteki.
 - 3.2. Dane osobowe czytelnika są usuwane (z wyjątkiem numeru czytelnika).
 - 3.3. System rejestruje zdarzenie wypisania czytelnika.
4. W przeciwnym razie:
 - 4.1. System informuje o niemożności wypisania czytelnika.

PU02 jest rozszerzeniem opisywanego przypadku użycia («extend» na diagramie), ponieważ czytelnik może nie mieć wypożyczonych książek.

Niżej znajduje się ten sam opis przypadków użycia w programie *Visual Paradigm*.

Gdzie w *Visual Paradigm* jest opis przypadku użycia? W oknie *Use Case Details*:

- Cel: zakładka *Info* w polu *Justification*.
- Warunki wstępne: zakładka *Details* w polu *Preconditions*.
- Warunki końcowe: zakładka *Details* w polu *Post-conditions*.
- Scenariusz: zakładka *Flow of events*.

Jak szybko skopiować scenariusz z okna tego programu do sprawozdania?

Kliknąć treść scenariusza, zaznaczyć ją (1x lub 2x Ctrl+A), skopiować (Ctrl+C) i wkleić bez formatowania (Ctrl+Shift+V). Pozostaje już tylko poprawić wygląd tekstu i język na polski na wzór powyższych przykładów.

The screenshot displays three tabs of the *Use Case Details* window for use case PU01. The tabs are: Info, Use Case Notes, Flow of Events, Details, Requirements, Diagrams, Test Plan, and References.

Info Tab: Shows the rank as Unspecified, ID as PU01, status as Identify, and justification as "Wypożyczenie książki czytelnikowi." Primary actors listed are Bibliotekarz.

Flow of Events Tab: Shows a scenario titled "1.Bibliotekarz wprowadza nr książki. 2.Bibliotekarz...". The steps are:

1. Bibliotekarz wprowadza nr książki.
2. Bibliotekarz wprowadza nr czytelnika.
3. if Książka jest dostępna do wypożyczenia i czytelnik jest zarejestrowany w bibliotece.
 - 3.1. Bibliotekarz zatwierdza wypożyczenie książki.
 - 3.2. System rejestruje wypożyczenie książki przez czytelnika.
 - 3.3. System wyłącza dostępność książki do wypożyczenia.
 - 3.4. System rejestruje zdarzenie wypożyczenia książki.
4. else
 - 4.1. System informuje o niemożności wypożyczenia książki czytelnikowi.

Details Tab: Shows the following sections:

- Level: [empty]
- Complexity: [empty]
- Use Case Status: [empty]
- Implementation Status: [empty]
- Preconditions: "Inicjacja przez Bibliotekarza: Bibliotekarz zna nr książki i nr czytelnika."
- Post-conditions: "Zarejestrowano wypożyczenie książki i wyłączono jej dostępność do wypożyczenia. Spełniono wymaganie F02: Bibliotekarz wypożycza książkę czytelnikowi."
- Author: [empty]
- Assumptions: [empty]

Rys. 3. Opis przypadku użycia PU01 kolejno w trzech zakładkach okna *Use Case Details*.

PU02. Zakończenie wypożyczenia książki

Rank: Unspecified

ID: PU02

Status: Identify

Justification: Zakończenie wypożyczenia książki czytelnikowi przez jej zwrot lub usunięcie z inwentarza.

Primary Actors: Bibliotekarz

Supporting Actors:

Level:

Complexity:

Use Case Status:

Implementation Status:

Preconditions: Inicjacja przez Bibliotekarza:
Bibliotekarz zna nr książki i nr czytelnika.
Inicjacja przez PU08. Wyipisanie czytelnika z biblioteki:
PU08 przekazuje nr książki.

Post-conditions: Wykonano PU03. Przyjęcie zwrotu książki lub PU04. Usunięcie książki z inwentarza.
Spłniono wymaganie P03: Bibliotekarz kończy wypożyczenie książki przez przyjęcie jej zwrotu od czytelnika lub (sporadycznie) przez usunięcie wypożyczonej, ale zgubionej lub zniszczonej książki z inwentarza biblioteki.

Author:

Assumptions:

Rys. 4. Opis przypadku użycia PU02 kolejno w trzech zakładkach okna *Use Case Details*.

PU03. Przyjęcie zwrotu książki

Rank: Unspecified

ID: PU03

Status: Identify

Justification: Przyjęcie zwrotu książki od czytelnika.

Primary Actors:

Supporting Actors:

Level:

Complexity:

Use Case Status:

Implementation Status:

Preconditions: Inicjacja przez PU02. Zakończenie wypożyczenia książki:
PU02 przekazuje nr książki.

Post-conditions: Zarejestrowano przyjęcie zwrotu książki i włączono jej dostępność do wypożyczenia.

Author:

Assumptions:

Rys. 5. Opis przypadku użycia PU03 kolejno w trzech zakładkach okna *Use Case Details*.

PU04. Usunięcie książki z inwentarza

Rank: Unspecified

ID: PU04

Status: Identify

Justification: Usunięcie książki z inwentarza.

Primary Actors: Bibliotekarz

Supporting Actors:

Level:

Complexity:

Use Case Status:

Implementation Status:

Preconditions: Inicjacja przez Bibliotekarza:
Bibliotekarz zna nr książki.
Inicjacja przez PU02. Zakończenie wypożyczenia książki:
PU02 przekazuje nr książki.

Post-conditions: Usunięto książkę z inwentarza.
Spłniono wymaganie P05: Bibliotekarz usuwa zgubioną lub zniszczoną książkę z inwentarza biblioteki.

Author:

Assumptions:

Rys. 6. Opis przypadku użycia PU04 kolejno w trzech zakładkach okna *Use Case Details*.

PU05. Wprowadzenie książki do inwentarza

Rank: Unspecified

ID: PU05

Status: Identify

Justification: Wprowadzenie nowej książki do inwentarza.

Primary Actors: Bibliotekarz

Supporting Actors:

Level:

Complexity:

Use Case Status:

Implementation Status:

Preconditions: Inicjacja przez Bibliotekarza:
Bibliotekarz zna wszystkie wymagane dane opisujące książkę.

Post-conditions: Wprowadzono książkę do inwentarza.
Spełniono wymaganie F04: Bibliotekarz wprowadza książkę do inwentarza biblioteki.

Author:

Assumptions:

Rys. 7. Opis przypadku użycia PU05 kolejno w trzech zakładkach okna *Use Case Details*.

PU06. Zapisanie czytelnika do biblioteki

Rank: Unspecified

ID: PU06

Status: Identify

Justification: Zapisanie nowego czytelnika do biblioteki.

Primary Actors: Kierownik Biblioteki

Supporting Actors:

Level:

Complexity:

Use Case Status:

Implementation Status:

Preconditions: Inicjacja przez Kierownika Biblioteki:
Kierownik Biblioteki zna wszystkie wymagane dane osobowe czytelnika.

Post-conditions: Zapisano czytelnika do biblioteki.
Spełniono wymaganie F07: Kierownik Biblioteki zapisuje czytelnika do biblioteki.

Author:

Assumptions:

Rys. 8. Opis przypadku użycia PU06 kolejno w trzech zakładkach okna *Use Case Details*.

PU07. Edytowanie danych czytelnika biblioteki

Rank: Unspecified

ID: PU07

Status: Identify

Justification: Edycja lub wprowadzenie danych osobowych czytelnika.

Primary Actors: Kierownik Biblioteki

Supporting Actors:

Level:

Complexity:

Use Case Status:

Implementation Status:

Preconditions: Inicjacja przez Kierownika Biblioteki:
Kierownik Biblioteki zna dane osobowe czytelnika.
Inicjacja przez Kierownika Biblioteki:
Kierownik Biblioteki zna dane osobowe czytelnika.

Post-conditions: Dane czytelnika są poprawne i kompletne.
Spełniono wymaganie F06: Kierownik Biblioteki zarządza danymi osobowymi czytelnika biblioteki.

Author:

Assumptions:

Rys. 9. Opis przypadku użycia PU07 kolejno w trzech zakładkach okna *Use Case Details*.

PU08. Wypisanie czytelnika z biblioteki

Info Tab:

- Rank: Unspecified
- ID: PU08
- Status: Identify
- Justification: Wypisanie czytelnika z biblioteki.
- Primary Actors: Kierownik Biblioteki
- Supporting Actors: [empty]

Flow of Events Tab:

Scenario: 1. Bibliotekarz zatwierdza wypisanie czytelnika. 2. for...
 1. Kierownik Biblioteki zatwierdza wypisanie czytelnika.
 2. for each Książka wypożyczona przez czytelnika.
 2.1. Inicjacja PU02_Zakończenie wypożyczenia książki z podaniem numeru książki.
 end for each
 3. if W inwentarzu brak książek wypożyczonych przez czytelnika.
 3.1. Czytelnik jest wypisywany z biblioteki.
 3.2. Dane osobowe czytelnika są usuwane (z wyjątkiem numeru czytelnika).
 3.3. System rejestruje zdarzenie wypisania czytelnika.
 4. else
 4.1. System informuje o niemożności wypisania czytelnika.
 end if

Requirements Tab:

- Level: [empty]
- Complexity: [empty]
- Use Case Status: [empty]
- Implementation Status: [empty]
- Preconditions: Inicjacja przez Kierownika Biblioteki:
Kierownik Biblioteki zna nr czytelnika.
- Post-conditions: Wypisano czytelnika z biblioteki.
Wykonano PU02_Zakończenie wypożyczenia książki dla wszystkich książek wypożyczonych przez czytelnika.
Spełniono wymaganie F08: Kierownik Biblioteki wypisuje czytelnika z biblioteki.
Spełniono wymaganie N08: Warunkiem wypisania czytelnika z biblioteki jest nieposiadanie przez niego aktywnych wypożyczeń książek.
- Author: [empty]
- Assumptions: [empty]

Rys. 10. Opis przypadku użycia PU08 kolejno w trzech zakładkach okna *Use Case Details*.

Modelowanie realizacji przypadków użycia

Diagramy czynności

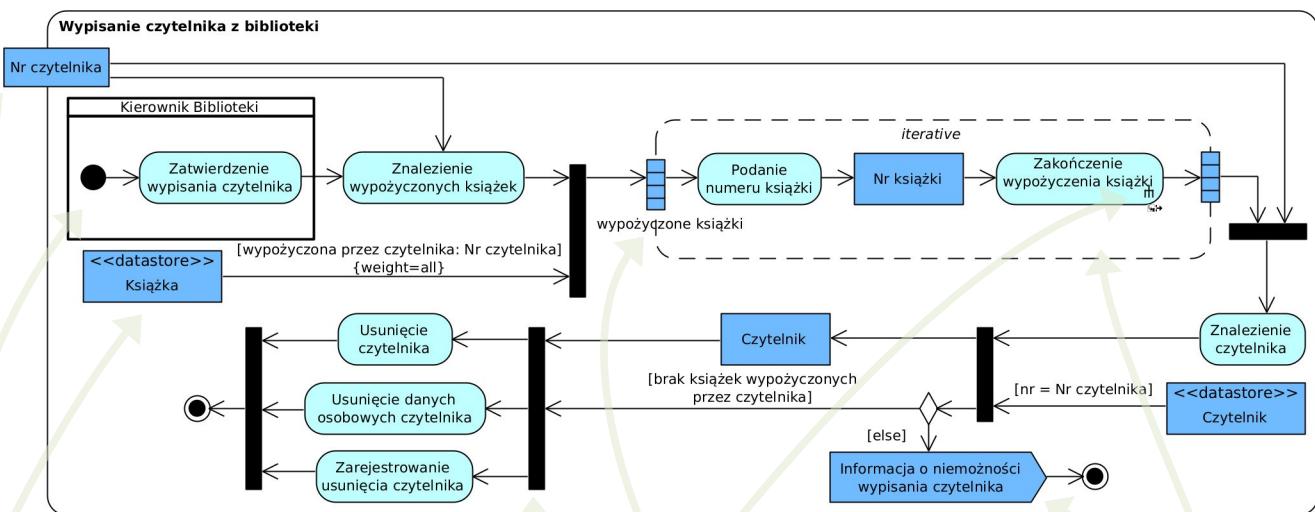
diagramy z zadań 1 i 2

W *Visual Paradigm* przed utworzeniem czynności, która wywołuje jakaś akcję, należy utworzyć tę akcję. Więc należy zacząć od diagramu, a w nim czynności modelującej realizację przypadku użycia, który jest korzeniem złożonego przypadku użycia (nie jest częścią innego przypadku użycia).

Jak w *Visual Paradigm* utworzyć akcję wywołania operacji – czynności? Kolejno:

- 1) Umieścić na diagramie akcję i nazwać ją tak, jak będzie się nazywać wywoływana przez nią czynność.
- 2) Wybrać w menu kontekstowym tej akcji: *Behavior* → *Create Activity*.
- 3) Czynność o nazwie tej akcji zostanie automatycznie utworzona i umieszczona w nowym diagramie czynności, a akcja otrzyma do niej i jej diagramu powiązanie i ikonę „wield”.

Czynność *Wypisanie czytelnika z biblioteki*:



Rys. 11. Diagram czynności *Wypisanie czytelnika z biblioteki*.

Na podstawie scenariusza przypadku użycia PU08 powstała czynność *Wypisanie czytelnika z biblioteki*:

Danymi wejściowymi czynności (activity parameter node) jest *Nr czytelnika*. Trafia on do akcji *Znalezienie wypożyczonych książek* i do akcji *Znalezienie czytelnika*.

Partycja *Kierownik Biblioteki* zawiera tylko akcję *Zatwierdzenie wypisania czytelnika*, ponieważ jedynie ta akcja wykonywana jest przy bezpośrednim udziale aktora *Kierownik Biblioteki*. Pozostałe akcje systemu wykonuje autonomicznie.

Wynikiem wykonania akcji *Znalezienie wypożyczonych książek* jest kolekcja *wypożyczone książki*, która pochodzi z magazynu danych (*datastore*) *Książka* (gromadzącego dane o książkach) i spełnia dozór *wypożyczona przez czytelnika*.

Ta kolekcja trafia do szeregowego (*iterative*) przetworzenia wszystkich jej książek w obszarze rozszerzenia.

Akcja *Zakończenie wypożyczenia książki* jest akcją wywołania operacji – czynności o tej samej nazwie.

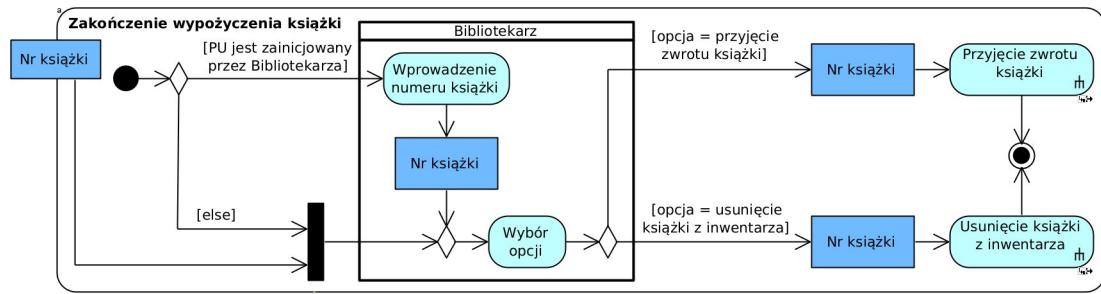
Wynikiem wykonania akcji *Znalezienie czytelnika* jest obiekt *Czytelnik*, który pochodzi z magazynu danych (*datastore*) *Czytelnik* (gromadzącego dane o czytelnikach) i spełnia dozór *nr = Nr czytelnika* (numer znalezionego czytelnika to wartość obiektu *Nr czytelnika*).

Ten obiekt trafia do akcji *Usunięcie czytelnika*, *Usunięcie danych osobowych czytelnika* i *Zarejestrowanie usunięcia czytelnika*.

Te trzy akcje wykonywane są niezależnie od siebie, więc są na współbieżnych przepływach sterowania. Po ich wykonaniu czynność kończy się.

Wykonanie tych trzech akcji zależy od spełnienia dozoru *brak książek wypożyczonych przez czytelnika*. W przeciwnym razie system przekazuje *Kierownikowi Biblioteki* sygnał *Informacja o niemożności wypisania czytelnika*. Po jego przekazaniu czynność kończy się.

Czynność Zakończenie wypożyczenia książki:



Rys. 12. Diagram czynności Zakończenie wypożyczenia książki.

Na podstawie scenariusza przypadku użycia PU02 powstała czynność Zakończenie wypożyczenia książki:

Danymi wejściowymi czynności (activity parameter node) może być **Nº książka**. Trafia on do akcji **Wybór opcji**, jeśli dozór **PU jest zainicjowany przez Bibliotekarza** nie jest spełniony.

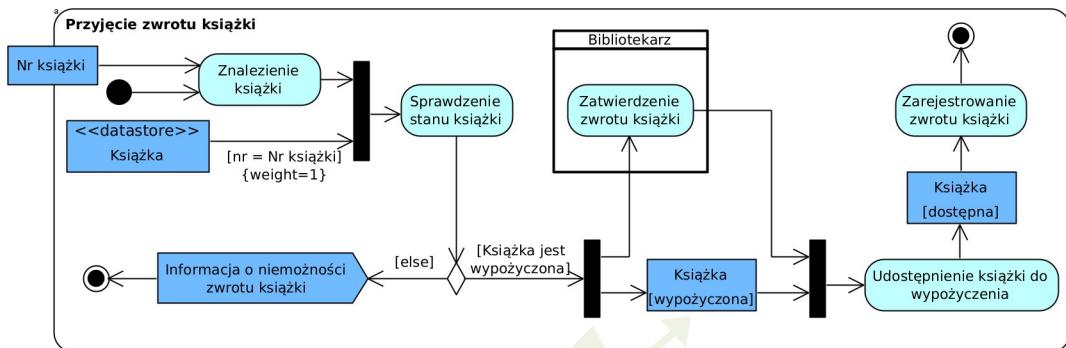
W przeciwnym razie obiekt **Nº książka** jest tworzony przez akcję **Wprowadzenie numeru książki** i przekazywany do akcji **Wybór opcji**.

Partycja **Bibliotekarz** zawiera tylko akcje **Wprowadzenie numeru książki** i **Wybór opcji**, ponieważ jedynie te akcje wykonywane są przy bezpośrednim udziale aktora **Bibliotekarz**. Pozostałe akcje systemu wykonuje autonomicznie.

Akcja **Wybór opcji** przekazuje obiekt **Nº książka** do akcji **Przyjęcie zwrotu książki** lub do akcji **Usunięcie książki z inwentarza**, zależnie od spełnienia dozoru **opcja = ...** (zależnie od wybranej opcji).

Akcje **Przyjęcie zwrotu książki** i **Usunięcie książki z inwentarza** są akcjami wywołania operacji – czynności o tych samych nazwach. Po ich wykonaniu czynność kończy się.

Czynność Przyjęcie zwrotu książki:



Na podstawie scenariusza przypadku użycia PU03 powstała czynność Przyjęcie zwrotu książki:

Danymi wejściowymi czynności (activity parameter node) jest **Nº książka**. Trafia on do akcji **Znalezienie książki**.

Partycja **Bibliotekarz** zawiera tylko akcję **Sprawdzenie stanu książki**, ponieważ jedynie ta akcja wykonywana jest przy bezpośrednim udziale aktora **Bibliotekarz**. Pozostałe akcje systemu wykonuje autonomicznie.

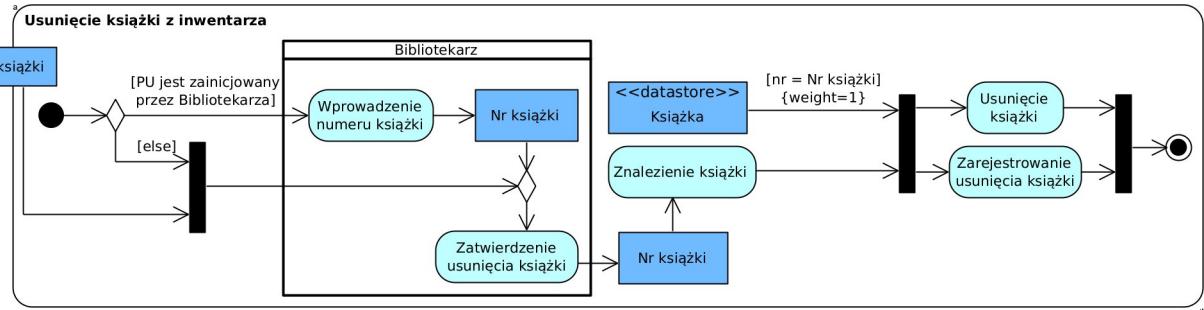
Wynikiem wykonania akcji **Znalezienie książki** jest obiekt **Książka**, który pochodzi z magazynu danych (**datastore**) **Książka** (gromadzącego dane o książkach) i spełnia dozór **nr = Nº książka** (numer znalezionej książki to wartość obiektu **Nº książka**). Ten obiekt trafia do akcji **Sprawdzenie stanu książki**.

Jeśli spełniony jest dozór **Książka jest wypożyczona**, to akcja **Sprawdzenie stanu książki** przekazuje obiekt **Książka** (w stanie **wypożyczona**) do akcji **Udostępnienie książki do wypożyczenia**, mimo że między tymi akcjami wykonywana jest jeszcze akcja **Zatwierdzenie zwrotu książki**.

W przeciwnym razie system przekazuje **Bibliotekarzowi** sygnał **Informacja o niemożności zwrotu książki**. Po jego przekazaniu czynność kończy się.

Akcja **Udostępnienie książki do wypożyczenia** zmienia stan obiektu **Książka** i przekazuje obiekt **Książka** (w stanie **dostępna**) do akcji **Zarejestrowanie zwrotu książki**. Po jej wykonaniu czynność kończy się.

Czynność Usunięcie książki z inwentarza:



Rys. 14. Diagram czynności *Usunięcie książki z inwentarza*.

Na podstawie scenariusza przypadku użycia PU04 powstała czynność *Usunięcie książki z inwentarza*:

Danymi wejściowymi czynności (activity parameter node) może być *Nr książki*. Trafia on do akcji *Zatwierdzenie usunięcia książki*, jeśli dozór *PU jest zainicjowany przez Bibliotekarza* nie jest spełniony.

W przeciwnym razie obiekt *Nr książki* jest tworzony przez akcję *Wprowadzenie numeru książki* i przekazywany do akcji *Zatwierdzenie usunięcia książki*.

Partycja *Bibliotekarz* zawiera tylko akcje *Wprowadzenie numeru książki* i *Zatwierdzenie usunięcia książki*, ponieważ jedynie te akcje wykonywane są przy bezpośrednim udziale aktora *Bibliotekarz*. Pozostałe akcje system wykonuje autonomicznie.

Wynikiem wykonania akcji *Znalezienie książki* jest obiekt *Książka*, który pochodzi z magazynu danych (*datastore*) *Książka* (gromadzącego dane o książkach) i spełnia dozór *nr = Nr książki* (numer znalezionej książki to wartość obiektu *Nr książki*). Ten obiekt trafia do akcji *Usunięcie książki* i do akcji *Zarejestrowanie usunięcia książki*.

Te dwie akcje wykonywane są niezależnie od siebie, więc są na wspólnie sterowanych przepływie sterowania. Po ich wykonaniu czynność się kończy.

Modelowanie obiektowej 3-warstwowej architektury

Diagram komponentów diagram z zadania 1



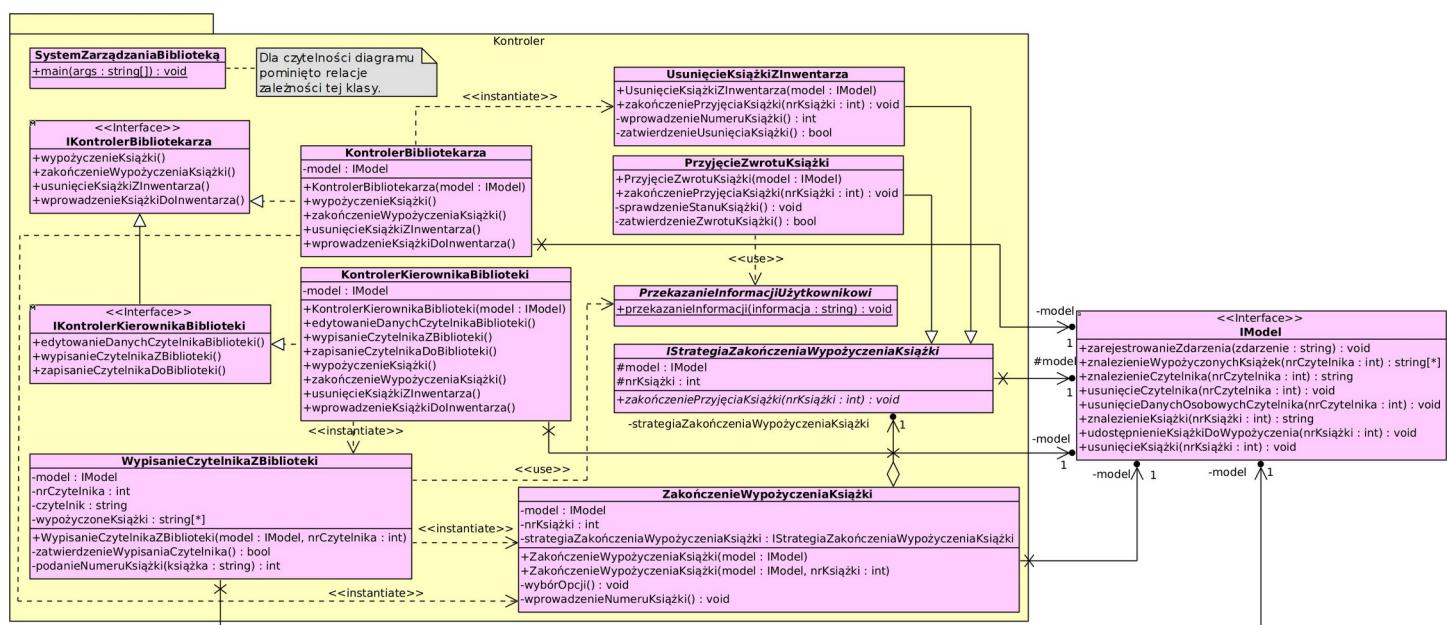
Rys. 15. Diagram komponentów.

Powyższy diagram komponentów określa warstwową architekturę oprogramowania, składającą się z warstwy prezentacji (komponent *Widok*), warstwy kontroli (komponent *Kontroler*) i warstwy encji (komponent *Model*). Łączą je interfejsy *IKontrolerBibliotekarza*, *IKontrolerKierownikaBiblioteki* i *IModel*.

Klasa fasadowa komponentu *Model* realizuje interfejs *IModel*. Używają go klasy komponentu *Kontroler*.

Klasy fasadowe komponentu *Kontroler* realizują interfejsy *IKontrolerBibliotekarza* i *IKontrolerKierownikaBiblioteki*. Używają ich klasy komponentu *Widok*.

Diagram klas komponentu *Kontroler* diagram z zadania 2

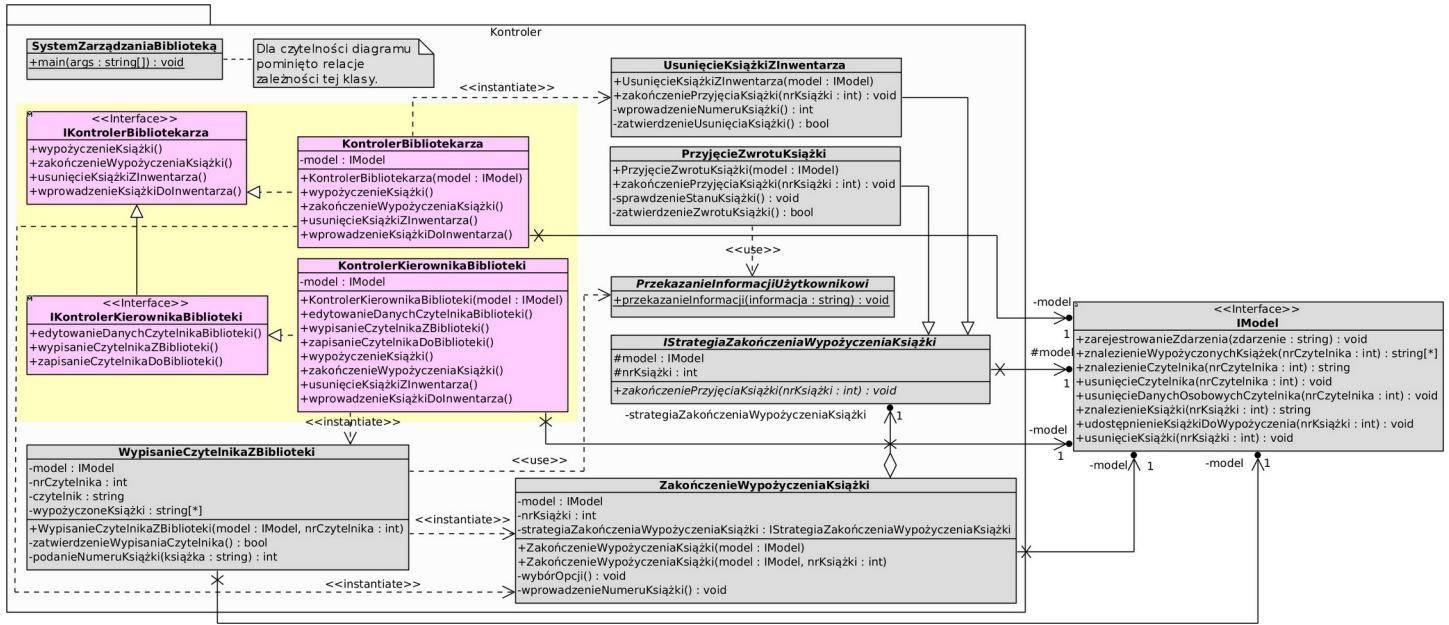


Rys. 16. Diagram klas komponentu *Kontroler*.

Powyższy diagram modeluje klasy komponentu *Kontroler*, zawarte w pakiecie *Kontroler*, oraz używany przez nie interfejs *IModel* z komponentem *Model*.

Założenia projektowe:

- 1) Klasa *SystemZarządzaniaBiblioteką*, zawierająca główną operację *main()*, znajduje się w pakiecie *Kontroler*. Dla lepszej czytelności diagram nie pokazuje relacji zależności (typu *instance* lub *use*) tej klasy od innych klas. W szczególności byłyby to klasy: *IKontrolerBibliotekarza*, *IKontrolerKierownikaBiblioteki* i *IModel*.
- 2) Każda operacja klas fasadowych *KontrolerBibliotekarza* i *KontrolerKierownikaBiblioteki* jest realizacją przypadku użycia bezpośrednio inicjowanego przez *Bibliotekarza* lub *Kierownika Biblioteki* i ma jego nazwę.
- 3) Oprócz klas fasadowych diagram pokazuje tylko klasy potrzebne do realizacji przypadku użycia *Wypisanie Czytelnika z Biblioteki* oraz klasy wynikłe z zastosowania wzorców projektowych.
- 4) Każda klasa potrzebna do realizacji tego złożonego przypadku użycia ma jedną odpowiedzialność – realizację jego składowego pojedynczego przypadku użycia, zgodnie z powiązaną z nim czynnością (wg diagramu czynności). Jej operacje odpowiadają akcjom tej czynności, o ile są wykonywane przez komponent *Kontroler*. Pozostały akcjom tej czynności zwykle odpowiadają operacje zdefiniowane w interfejsie *IModel*, czyli operacje przetwarzania danych.
- 5) Nazwa klasy, atrybutu lub operacji dokładnie, jeśli to możliwe, definiuje odpowiedzialność.
- 6) Realizacja przypadku użycia, czyli wykonanie modelującą go czynność, polega na utworzeniu instancji klasy odpowiedzialnej za tę realizację. Akcje tej czynności, zgodnie z jej diagramem czynności, wykonywane są w ciele konstruktora tej klasy. Alternatywnie można by do tego celu utworzyć specjalną operację w tej klasie.

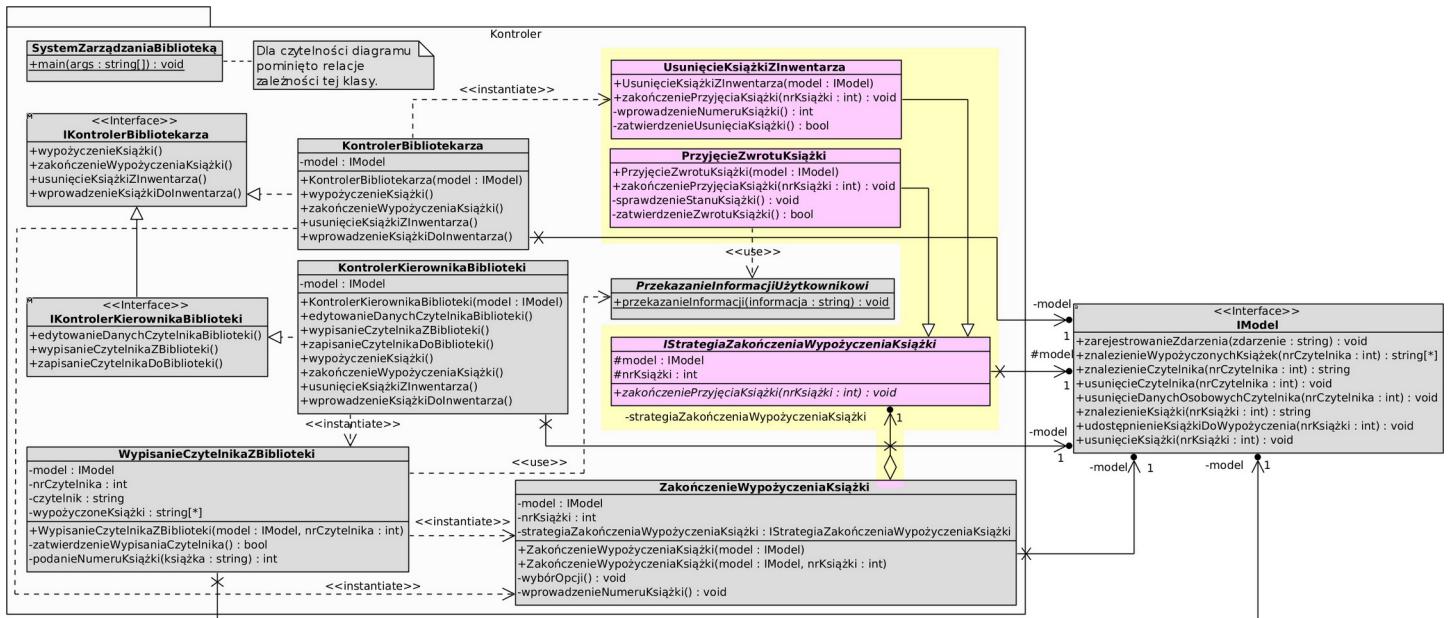


Rys. 17. Zastosowanie wzorca Fasada (kolorowy fragment) na diagramie klas komponentu Kontroler.

Zastosowano wzorzec projektowy *Fasada* w klasach: *IKontrolerBibliotekarza*, *KontrolerBibliotekarza*, *IKontrolerKierownikaBiblioteki* i *KontrolerKierownikaBiblioteki*.

Aktor *Bibliotekarz* kontroluje oprogramowanie poprzez klasę *KontrolerBibliotekarza*, której operacje określają interfejs *IKontrolerBibliotekarza*.

Aktor *Kierownik Biblioteki* kontroluje oprogramowanie poprzez klasę *KontrolerKierownikaBiblioteki*, której operacje określają interfejsy *IKontrolerBibliotekarza* i *IKontrolerKierownikaBiblioteki*.

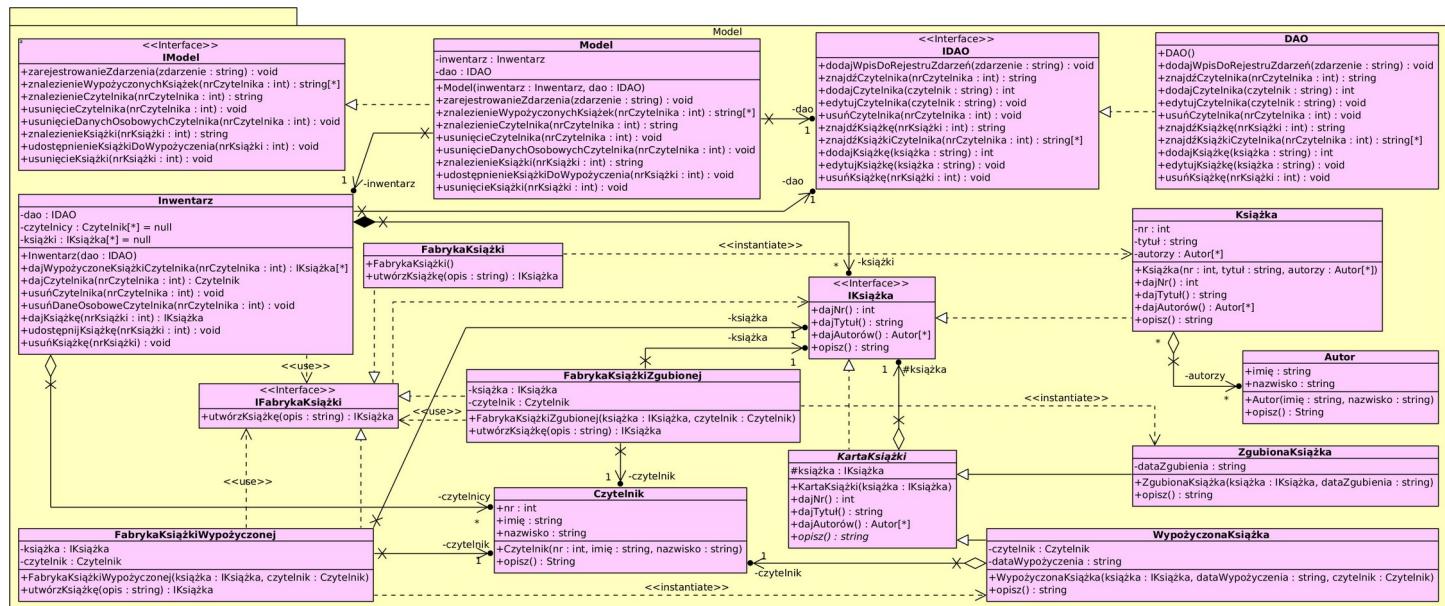


Rys. 18. Zastosowanie wzorca Strategia (kolorowy fragment) na diagramie klas komponentu Kontroler.

Zastosowano wzorzec projektowy *Strategia* w klasach: *ZakończenieWypożyczeniaKsiążki*, *IStrategiaZakończeniaWypożyczeniaKsiążki*, *PrzyjęcieZwrotuKsiążki* i *UsunięcieKsiążkiZInwentarza*.

Klasa *ZakończenieWypożyczeniaKsiążki* w operacji *wybórOpcji()* wybiera strategię realizacji swojego zadania. Jej strategie to różne implementacje operacji *zakończeniePrzyjęciaKsiążki()* w klasach *PrzyjęcieZwrotuKsiążki* i *UsunięcieKsiążkiZInwentarza*, określonej przez klasę abstrakcyjną *IStrategiaZakończeniaWypożyczeniaKsiążki*.

Diagram klas komponentu Model diagram z zadania 3

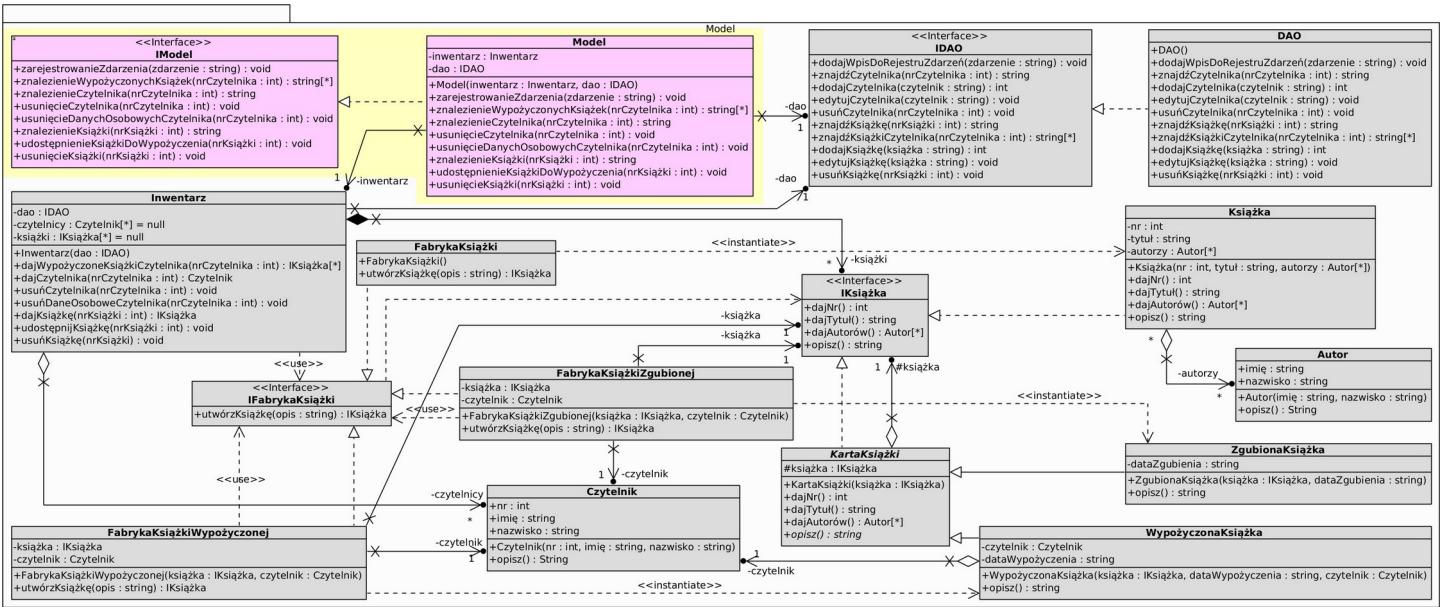


Rys. 19. Diagram klas komponentu Model.

Powyższy diagram modeluje klasy komponentu *Model*, zawarte w pakiecie *Model*.

Założenia projektowe:

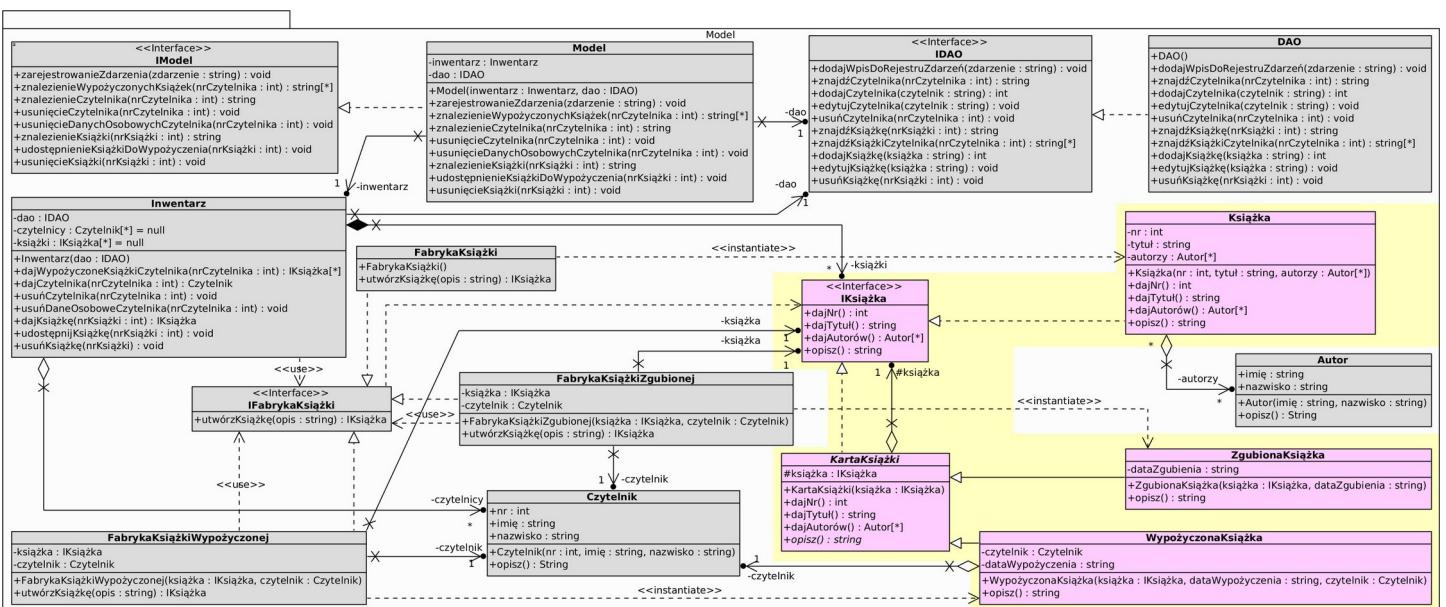
- 7) Każda operacja klasy fasadowej *Model* odpowiada pojedynczej akcji przetwarzania danych (wg diagramu czynności ją zawierającej).
- 8) Operacje klasy fasadowej *Model*, które zwracają komponentowi *Kontroler* przetwarzane dane, przekazują je w formie tekstuowej (np. csv, json, xml), więc ich wyjściowy parametr to *string*.
- 9) Oprócz klas fasadowych diagram pokazuje tylko klasy potrzebne do realizacji przypadku użycia *Wypisanie Czytelnika z Biblioteki* oraz klasy wynikłe z zastosowania wzorców projektowych.
- 10) Każda klasa potrzebna do przetwarzania danych ma jedną odpowiedzialność:
 - klasy *Czytelnik*, *Książka* i *Autor* modelują podstawowe przetwarzane dane;
 - klasa *DAO*, realizująca interfejs *IDAO*, symuluje wywołanie operacji CRUD w warstwie zasobów, gdzie trwale przechowywane są przetwarzane dane (diagram komponentów jej nie pokazuje);
 - klasa *Inwentarz* zarządza obiektową reprezentacją przetwarzanych danych na żądanie operacji klasy *Model* oraz czasowo przechowuje obiekty przetwarzanych danych, aby ograniczyć korzystanie z warstwy zasobów;
 - klasy *FabrykaKsiążki*, *FabrykaWypożyczonejKsiążki* i *FabrykaZgubionejKsiążki*, realizujące interfejs *IFabrykaKsiążki*, tworzą obiektową reprezentację książki, która opcjonalnie może być wypożyczona i / lub zgubiona;
 - klasy *WypożyczonaKsiążka* i *ZgubionaKsiążka*, uszczególniające klasę *KartaKsiążki*, dekorują (opakowują) obiekt klasy *Książka* o stan wypożyczenia i zgubienia książki (odpowiednio); wszystkie realizują interfejs *IKsiążka*.
- 11) Nazwa klasy, atrybutu lub operacji dokładnie, jeśli to możliwe, definiuje odpowiedzialność.
- 12) Operacja *zarejestrowanieZdarzenia()* klasy *Model* wywołuje operację *dodajWPisDoRejestruZdarzeń()* klasy *DAO*. Pozostałe jej operacje używają tylko operacji klasy *Inwentarz*.



Rys. 20. Zastosowanie wzorca Fasada (kolorowy fragment) na diagramie klas komponentu Model.

Zastosowano wzorzec projektowy Fasada w klasach: *IModel* i *Model*.

Klasy komponentu Kontroler korzystają z usług komponentu Model poprzez klasę Model, której operacje określają interfejs *IModel*.



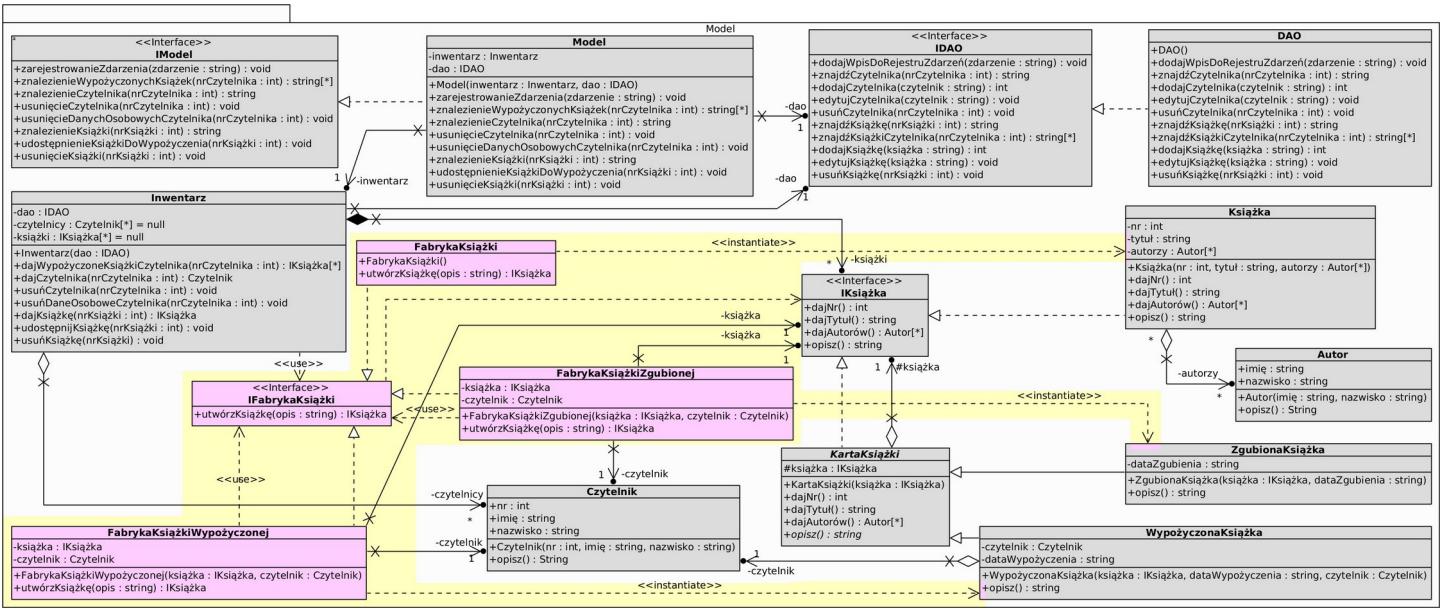
Rys. 21. Zastosowanie wzorca Dekorator (kolorowy fragment) na diagramie klas komponentu Model.

Zastosowano wzorzec projektowy Dekorator w klasach: *IKsiążka*, *Książka*, *KartaKsiążki*, *ZgubionaKsiążka* i *WypożyczonaKsiążka*.

Obiekt klasy *Książka* można opakować dekoratorami – obiektami klas *ZgubionaKsiążka* (dodanie do książki atrybutu *dataZgubienia*) i *WypożyczonaKsiążka* (dodanie do książki atrybutów *dataWypożyczenia* i *czytelnik*), które uszczegóławiają klasę *KartaKsiążki*, będącą abstrakcją dekoratora książki. Przykład opakowania: *ZgubionaKsiążka[WypożyczonaKsiążka[Książka]]*.

Wszystkie te klasy relizują interfejs *IKsiążka*, aby klasa *Inwentarz* mogła używać dowolnej klasy realizującej ten interfejs, czyli książki nieopakowanej lub dowolnie opakowanej w jej wypożyczenie i / lub zgubienie.

Interfejs *IKsiążka* definiuje operację *opisz()*, która zwraca tekstowy opis stanu (np. zawartość atrybutów) obiektu klasy *Książka* z nadbudową tego stanu zawartą w opakowujących go dekoratorach. Ta operacja wywoływana jest w najbardziej zewnętrznym dekoratorze. Następnie wywołuje ona tę samą operację rekurencyjnie w głąb aż do samego obiektu klasy *Książka* i dopisuje do jej wyniku opis swojej części stanu książki.



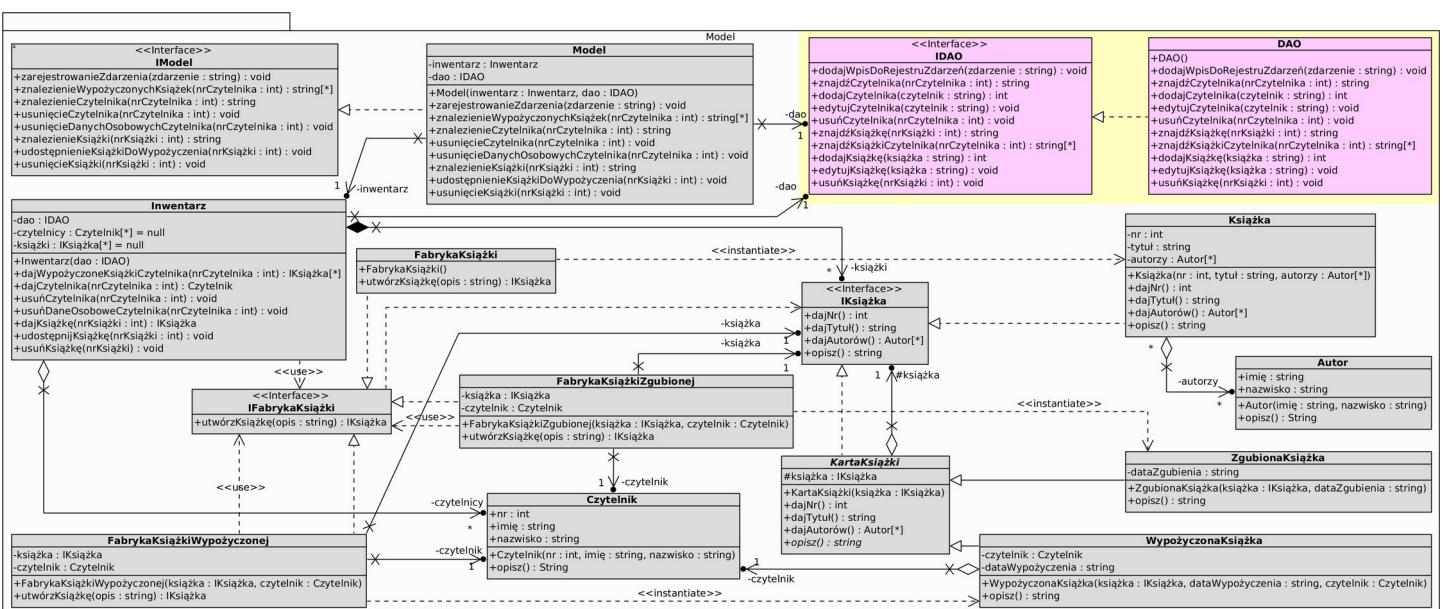
Rys. 22. Zastosowanie wzorca Metoda wytwórcza (kolorowy fragment) na diagramie klas komponentu Model.

Zastosowano wzorzec projektowy *Metoda wytwórcza* w klasach: *IFabrykaKsiążki*, *FabrykaKsiążki*, *FabrykaKsiążki-Zgubionej* i *FabrykaKsiążkiWypożyczonej*.

Klasy *FabrykaKsiążki*, *FabrykaKsiążkiZgubionej* i *FabrykaKsiążkiWypożyczonej* tworzą obiekt realizujący interfejs *Iksiążka*:

- *FabrykaKsiążki* tworzy nieopakowaną książkę (obiekt klasy *Książka*);
- *FabrykaWypożyczonejKsiążki* tworzy wypożyczoną książkę, czyli obiekt klasy *Książka* opakowany dekoratorem *WypożyczonaKsiążka*;
- *FabrykaZgubionejKsiążki* tworzy zgubioną książkę, czyli obiekt klasy *Książka* opakowany dekoratorem *ZgubionaKsiążka* lub zgubioną wypożyczoną książkę, czyli obiekt klasy *Książka* opakowany dekoratorem *WypożyczonaKsiążka* (przy pomocy klasy *FabrykaWypożyczonejKsiążki*), a następnie opakowany dekoratorem *ZgubionaKsiążka*.

Klasa *Inwentarz* wybiera jedną z tych klas i wywołuje jej operację *utwórzKsiążkę()*.



Rys. 23. Zastosowanie wzorca Adapter (kolorowy fragment) na diagramie klas komponentu Model.

Zastosowano wzorzec projektowy *Adapter* w klasach: *IDAO* i *DAO*.

Klasa *DAO*, realizująca interfejs *DAO*, symuluje wywoływanie operacji CRUD w warstwie zasobów, gdzie trwale przechowywane są przetwarzane dane.

Klasy komponentu *Model* korzystają z usług warstwy zasobów tylko poprzez klasę *DAO*.

Kod klas komponentów *Kontroler* i *Model*

kod* z zadania 4

* Kod wygenerowany przez Visual Paradigm. Wymaga poprawienia i uzupełnienia zgodnie z treścią zadania 4.

Kontroler.IKontrolerBibliotekarza.java

```
1. package Kontroler;
2.
3. public interface IKontrolerBibliotekarza {
4.
5.     public void wypożyczenieKsiążki();
6.     public void zakończenieWypożyczeniaKsiążki();
7.     public void usunięcieKsiążkiZInwentarza();
8.     public void wprowadzenieKsiążkiDoInwentarza();
9. }
```

Kontroler.IKontrolerKierownikaBiblioteki.java

```
1. package Kontroler;
2.
3. public interface IKontrolerKierownikaBiblioteki extends IKontrolerBibliotekarza {
4.
5.     public void edytowanieDanychCzytelnikaBiblioteki();
6.     public void wypisanieCzytelnikaZBiblioteki();
7.     public void zapisanieCzytelnikaDoBiblioteki();
8. }
```

Kontroler.IStrategiaZakończeniaWypożyczeniaKsiążki.java

```
1. package Kontroler;
2. import Model.ILogger;
3.
4. public abstract class IStrategiaZakończeniaWypożyczeniaKsiążki {
5.     protected ILogger model;
6.     protected int nrKsiążki;
7.
8.     public abstract void zakończeniePrzyjęciaKsiążki(int nrKsiążki);
9. }
```

Kontroler.KontrolerBibliotekarza.java

```
1. package Kontroler;
2. import Model.ILogger;
3.
4. public class KontrolerBibliotekarza implements IKontrolerBibliotekarza {
5.     private ILogger model;
6.
7.     public KontrolerBibliotekarza(ILogger model) {
8.         throw new UnsupportedOperationException();
9.     }
10.
11.    public void wypożyczenieKsiążki() {
12.        throw new UnsupportedOperationException();
13.    }
14.
15.    public void zakończenieWypożyczeniaKsiążki() {
16.        throw new UnsupportedOperationException();
17.    }
18.
19.    public void usunięcieKsiążkiZInwentarza() {
20.        throw new UnsupportedOperationException();
21.    }
22.
23.    public void wprowadzenieKsiążkiDoInwentarza() {
24.        throw new UnsupportedOperationException();
25.    }
26. }
```

Kontroler.KontrolerKierownikaBiblioteki.java

```
1. package Kontroler;
2. import Model.ILogger;
3.
4. public class KontrolerKierownikaBiblioteki implements IKontrolerKierownikaBiblioteki {
5.     private ILogger model;
6.
7.     public KontrolerKierownikaBiblioteki(ILogger model) {
8.         throw new UnsupportedOperationException();
9.     }
10.
11.    public void edytowanieDanychCzytelnikaBiblioteki() {
12.        throw new UnsupportedOperationException();
13.    }
14.
15.    public void wypisanieCzytelnikaZBiblioteki() {
16.        throw new UnsupportedOperationException();
17.    }
18. }
```

```

19.     public void zapisanieCzytelnikaDoBiblioteki() {
20.         throw new UnsupportedOperationException();
21.     }
22.
23.     public void wypożyczenieKsiążki() {
24.         throw new UnsupportedOperationException();
25.     }
26.
27.     public void zakończenieWypożyczeniaKsiążki() {
28.         throw new UnsupportedOperationException();
29.     }
30.
31.     public void usunięcieKsiążkiZInwentarza() {
32.         throw new UnsupportedOperationException();
33.     }
34.
35.     public void wprowadzenieKsiążkiDolnwentarza() {
36.         throw new UnsupportedOperationException();
37.     }
38. }
```

Kontroler.PrzekazanieInformacjiUżytkownikowi.java

```

1. package Kontroler;
2.
3. public abstract class PrzekazanieInformacjiUżytkownikowi {
4.
5.     public static void przekazanieInformacji(String informacja) {
6.         throw new UnsupportedOperationException();
7.     }
8. }
```

Kontroler.PrzyjęcieZwrotuKsiążki.java

```

1. package Kontroler;
2. import Model.IModel;
3.
4. public class PrzyjęcieZwrotuKsiążki extends IStrategiaZakończeniaWypożyczeniaKsiążki {
5.
6.     public PrzyjęcieZwrotuKsiążki(IModel model) {
7.         throw new UnsupportedOperationException();
8.     }
9.
10.    public void zakończeniePrzyjęciaKsiążki(int nrKsiążki) {
11.        throw new UnsupportedOperationException();
12.    }
13.
14.    private void sprawdzenieStanuKsiążki() {
15.        throw new UnsupportedOperationException();
16.    }
17.
18.    private boolean zatwierdzenieZwrotuKsiążki() {
19.        throw new UnsupportedOperationException();
20.    }
21. }
```

Kontroler.SystemZarządzaniaBiblioteką.java

```

1. package Kontroler;
2.
3. public class SystemZarządzaniaBiblioteką {
4.
5.     public static void main(String[] args) {
6.         throw new UnsupportedOperationException();
7.     }
8. }
```

Kontroler.UsunięcieKsiążkiZInwentarza.java

```

1. package Kontroler;
2. import Model.IModel;
3.
4. public class UsunięcieKsiążkiZInwentarza extends IStrategiaZakończeniaWypożyczeniaKsiążki {
5.
6.     public UsunięcieKsiążkiZInwentarza(IModel model) {
7.         throw new UnsupportedOperationException();
8.     }
9.
10.    public void zakończeniePrzyjęciaKsiążki(int nrKsiążki) {
11.        throw new UnsupportedOperationException();
12.    }
13.
14.    private int wprowadzenieNumeruKsiążki() {
15.        throw new UnsupportedOperationException();
16.    }
17.
18.    private boolean zatwierdzenieUsunięciaKsiążki() {
19.        throw new UnsupportedOperationException();
20.    }
21. }
```

Kontroler.WypisanieCzytelnikaZBiblioteki.java

```
1. package Kontroler;
2. import Model.IModel;
3.
4. public class WypisanieCzytelnikaZBiblioteki {
5.     private IMODEL model;
6.     private int nrCzytelnika;
7.     private String czytelnik;
8.     private String[] wypożyczoneKsiążki;
9.
10.    public WypisanieCzytelnikaZBiblioteki(IMODEL model, int nrCzytelnika) {
11.        throw new UnsupportedOperationException();
12.    }
13.
14.    private boolean zatwierdzenieWypisaniaCzytelnika() {
15.        throw new UnsupportedOperationException();
16.    }
17.
18.    private int podanieNumeruKsiążki(String książka) {
19.        throw new UnsupportedOperationException();
20.    }
21. }
```

Kontroler.ZakończenieWypożyczeniaKsiążki.java

```
1. package Kontroler;
2. import Model.IMODEL;
3.
4. public class ZakończenieWypożyczeniaKsiążki {
5.     private IMODEL model;
6.     private int nrKsiążki;
7.     private IStrategiaZakończeniaWypożyczeniaKsiążki strategiaZakończeniaWypożyczeniaKsiążki;
8.
9.    public ZakończenieWypożyczeniaKsiążki(IMODEL model) {
10.        throw new UnsupportedOperationException();
11.    }
12.
13.    public ZakończenieWypożyczeniaKsiążki(IMODEL model, int nrKsiążki) {
14.        throw new UnsupportedOperationException();
15.    }
16.
17.    private void wybórOpcji() {
18.        throw new UnsupportedOperationException();
19.    }
20.
21.    private void wprowadzenieNumeruKsiążki() {
22.        throw new UnsupportedOperationException();
23.    }
24. }
```

Model.Autor.java

```
1. package Model;
2.
3. public class Autor {
4.     public String imię;
5.     public String nazwisko;
6.
7.     public Autor(String imię, String nazwisko) {
8.         throw new UnsupportedOperationException();
9.     }
10.
11.    public String opisz() {
12.        throw new UnsupportedOperationException();
13.    }
14. }
```

Model.Czytelnik.java

```
1. package Model;
2.
3. public class Czytelnik {
4.     public int nr;
5.     public String imię;
6.     public String nazwisko;
7.
8.     public Czytelnik(int nr, String imię, String nazwisko) {
9.         throw new UnsupportedOperationException();
10.    }
11.
12.    public String opisz() {
13.        throw new UnsupportedOperationException();
14.    }
15. }
```

Model.DAO.java

```
1. package Model;
2.
3. public class DAO implements IDAO {
4.
5.     public DAO() {
6.         throw new UnsupportedOperationException();
6.     }
6. }
```

```

7.    }
8.
9.    public void dodajWpisDoRejestruZdarzeń(String zdarzenie) {
10.        throw new UnsupportedOperationException();
11.    }
12.
13.    public String znajdźCzytelnika(int nrCzytelnika) {
14.        throw new UnsupportedOperationException();
15.    }
16.
17.    public int dodajCzytelnika(String czytelnik) {
18.        throw new UnsupportedOperationException();
19.    }
20.
21.    public void edytujCzytelnika(String czytelnik) {
22.        throw new UnsupportedOperationException();
23.    }
24.
25.    public void usuńCzytelnika(int nrCzytelnika) {
26.        throw new UnsupportedOperationException();
27.    }
28.
29.    public String znajdźKsiążkę(int nrKsiążki) {
30.        throw new UnsupportedOperationException();
31.    }
32.
33.    public String[] znajdźKsiążkiCzytelnika(int nrCzytelnika) {
34.        throw new UnsupportedOperationException();
35.    }
36.
37.    public int dodajKsiążkę(String książka) {
38.        throw new UnsupportedOperationException();
39.    }
40.
41.    public void edytujKsiążkę(String książka) {
42.        throw new UnsupportedOperationException();
43.    }
44.
45.    public void usuńKsiążkę(int nrKsiążki) {
46.        throw new UnsupportedOperationException();
47.    }
48. }
```

Model.FabrykaKsiążki.java

```

1. package Model;
2.
3. public class FabrykaKsiążki implements IFabrykaKsiążki {
4.
5.     public FabrykaKsiążki() {
6.         throw new UnsupportedOperationException();
7.     }
8.
9.     public IKsiążka utwórzKsiążkę(String opis) {
10.        throw new UnsupportedOperationException();
11.    }
12. }
```

Model.FabrykaKsiążkiWypożyczonej.java

```

1. package Model;
2.
3. public class FabrykaKsiążkiWypożyczonej implements IFabrykaKsiążki {
4.     private IKsiążka książka;
5.     private Czytelnik czytelnik;
6.
7.     public FabrykaKsiążkiWypożyczonej(IKsiążka książka, Czytelnik czytelnik) {
8.         throw new UnsupportedOperationException();
9.     }
10.
11.    public IKsiążka utwórzKsiążkę(String opis) {
12.        throw new UnsupportedOperationException();
13.    }
14. }
```

Model.FabrykaKsiążkiZgubionej.java

```

1. package Model;
2.
3. public class FabrykaKsiążkiZgubionej implements IFabrykaKsiążki {
4.     private IKsiążka książka;
5.     private Czytelnik czytelnik;
6.
7.     public FabrykaKsiążkiZgubionej(IKsiążka książka, Czytelnik czytelnik) {
8.         throw new UnsupportedOperationException();
9.     }
10.
11.    public IKsiążka utwórzKsiążkę(String opis) {
12.        throw new UnsupportedOperationException();
13.    }
14. }
```

Model.IDAO.java

```

1. package Model;
2. public interface IDAO {
3.
4.     public void dodajWpisDoRejestruZdarzeń(String zdarzenie);
5.     public String znajdźCzytelnika(int nrCzytelnika);
6.     public int dodaCzytelnika(String czytelnik);
7.     public void edytujCzytelnika(String czytelnik);
8.     public void usuńCzytelnika(int nrCzytelnika);
9.     public String znajdźKsiążkę(int nrKsiążki);
10.    public String[] znajdźKsiążkiCzytelnika(int nrCzytelnika);
11.    public int dodaKsiążkę(String książka);
12.    public void edytujKsiążkę(String książka);
13.    public void usuńKsiążkę(int nrKsiążki);
14. }
```

Model.IFabrykaKsiążki.java

```

1. package Model;
2.
3. public interface IFabrykaKsiążki {
4.     public IKsiążka utwórzKsiążkę(String opis);
5. }
```

Model.IKsiążka.java

```

1. package Model;
2.
3. public interface IKsiążka {
4.
5.     public int dajNr();
6.     public String dajTytuł();
7.     public Autor[] dajAutorów();
8.     public String opisz();
9. }
```

Model.IModel.java

```

1. package Model;
2.
3. public interface IModel {
4.
5.     public void zarejestrowanieZdarzenia(String zdarzenie);
6.     public String[] znalezienieWypożyczonychKsiążek(int nrCzytelnika);
7.     public String znalezienieCzytelnika(int nrCzytelnika);
8.     public void usunięcieCzytelnika(int nrCzytelnika);
9.     public void usunięcieDanychOsobowychCzytelnika(int nrCzytelnika);
10.    public String znalezienieKsiążki(int nrKsiążki);
11.    public void udostępnienieKsiążkiDoWypożyczenia(int nrKsiążki);
12.    public void usunięcieKsiążki(int nrKsiążki);
13. }
```

Model.Inwentarz.java

```

1. package Model;
2.
3. public class Inwentarz {
4.     private IDAO dao;
5.     private Czytelnik[] czytelnicy = null;
6.     private IKsiążka[] książki = null;
7.
8.     public Inwentarz(IDAO dao) {
9.         throw new UnsupportedOperationException();
10.    }
11.
12.    public IKsiążka[] dajWypożyczoneKsiążkiCzytelnika(int nrCzytelnika) {
13.        throw new UnsupportedOperationException();
14.    }
15.
16.    public Czytelnik dajCzytelnika(int nrCzytelnika) {
17.        throw new UnsupportedOperationException();
18.    }
19.
20.    public void usuńCzytelnika(int nrCzytelnika) {
21.        throw new UnsupportedOperationException();
22.    }
23.
24.    public void usuńDaneOsoboweCzytelnika(int nrCzytelnika) {
25.        throw new UnsupportedOperationException();
26.    }
27.
28.    public IKsiążka dajKsiążkę(int nrKsiążki) {
29.        throw new UnsupportedOperationException();
30.    }
31.
32.    public void udostępnijKsiążkę(int nrKsiążki) {
33.        throw new UnsupportedOperationException();
34.    }
35.
36.    public void usuńKsiążkę(Object nrKsiążki) {
37.        throw new UnsupportedOperationException();
38.    }
39. }
```

Model.KartaKsiążki.java

```
1. package Model;
2.
3. public abstract class KartaKsiążki implements IKsiążka {
4.     protected IKsiążka książka;
5.
6.     public KartaKsiążki(IKsiążka książka) {
7.         throw new UnsupportedOperationException();
8.     }
9.
10.    public int dajNr() {
11.        throw new UnsupportedOperationException();
12.    }
13.
14.    public String dajTytuł() {
15.        throw new UnsupportedOperationException();
16.    }
17.
18.    public Autor[] dajAutorów() {
19.        throw new UnsupportedOperationException();
20.    }
21.
22.    public abstract String opisz();
23. }
```

Model.Książka.java

```
1. package Model;
2.
3. public class Książka implements IKsiążka {
4.     private int nr;
5.     private String tytuł;
6.     private Autor[] autorzy;
7.
8.     public Książka(int nr, String tytuł, Autor[] autorzy) {
9.         throw new UnsupportedOperationException();
10.    }
11.
12.    public int dajNr() {
13.        throw new UnsupportedOperationException();
14.    }
15.
16.    public String dajTytuł() {
17.        throw new UnsupportedOperationException();
18.    }
19.
20.    public Autor[] dajAutorów() {
21.        throw new UnsupportedOperationException();
22.    }
23.
24.    public String opisz() {
25.        throw new UnsupportedOperationException();
26.    }
27. }
```

Model.Model.java

```
1. package Model;
2.
3. public class Model implements IModel {
4.     private Inwentarz inwentarz;
5.     private IDAO dao;
6.
7.     public Model(Inwentarz inwentarz, IDAO dao) {
8.         throw new UnsupportedOperationException();
9.     }
10.
11.    public void zarejestrowanieZdarzenia(String zdarzenie) {
12.        throw new UnsupportedOperationException();
13.    }
14.
15.    public String[] znalezienieWypożyczonychKsiążek(int nrCzytelnika) {
16.        throw new UnsupportedOperationException();
17.    }
18.
19.    public String znalezienieCzytelnika(int nrCzytelnika) {
20.        throw new UnsupportedOperationException();
21.    }
22.
23.    public void usunięcieCzytelnika(int nrCzytelnika) {
24.        throw new UnsupportedOperationException();
25.    }
26.
27.    public void usunięcieDanychOsobowychCzytelnika(int nrCzytelnika) {
28.        throw new UnsupportedOperationException();
29.    }
30.
31.    public String znalezienieKsiążki(int nrKsiążki) {
32.        throw new UnsupportedOperationException();
33.    }
34.
35.    public void udostępnienieKsiążkiDoWypożyczenia(int nrKsiążki) {
36.        throw new UnsupportedOperationException();
37.    }
38.
```

```
37.     }
38.
39.     public void usunięcieKsiążki(int nrKsiążki) {
40.         throw new UnsupportedOperationException();
41.     }
42. }
```

Model.WypożyczonaKsiążka.java

```
1. package Model;
2.
3. public class WypożyczonaKsiążka extends KartaKsiążki {
4.     private Czytelnik czytelnik;
5.     private String dataWypożyczenia;
6.
7.     public WypożyczonaKsiążka(IKsiążka książka, String dataWypożyczenia, Czytelnik czytelnik) {
8.         throw new UnsupportedOperationException();
9.     }
10.
11.    public String opisz() {
12.        throw new UnsupportedOperationException();
13.    }
14.
15.    public int dajNr() {
16.        throw new UnsupportedOperationException();
17.    }
18.
19.    public String dajTytuł() {
20.        throw new UnsupportedOperationException();
21.    }
22.
23.    public Autor[] dajAutorów() {
24.        throw new UnsupportedOperationException();
25.    }
26. }
```

Model.ZgubionaKsiążka.java

```
1. package Model;
2.
3. public class ZgubionaKsiążka extends KartaKsiążki {
4.     private String dataZgubienia;
5.
6.     public ZgubionaKsiążka(IKsiążka książka, String dataZgubienia) {
7.         throw new UnsupportedOperationException();
8.     }
9.
10.    public String opisz() {
11.        throw new UnsupportedOperationException();
12.    }
13.
14.    public int dajNr() {
15.        throw new UnsupportedOperationException();
16.    }
17.
18.    public String dajTytuł() {
19.        throw new UnsupportedOperationException();
20.    }
21.
22.    public Autor[] dajAutorów() {
23.        throw new UnsupportedOperationException();
24.    }
25. }
```

Modelowanie interakcji między klasami i obiektami

Diagramy sekwencji i kod opisanych nimi operacji diagramy z zadania 1 i fragmenty kodu z zadania 2

Poniższe diagramy sekwencji modelują interakcje klas i obiektów w realizacji przypadku użycia PU08. Wypisanie czytelnika z biblioteki, zaczynając od operacji `Kontroler.KontrolerKierownikaBiblioteki.wypisanieCzytelnikaZBiblioteki()`.

Uwzględniono tylko operacje klas komponentu `Kontroler` i klasy fasadowej komponentu `Model`, a pominieto operacje związane z symulacją działania warstwy prezentacji (klasy pakietu `Komunikacja`).

Przyjęto następujące zasady:

- 1) Nazwa linii życia obiektu, którego operację modeluje diagram (wiadomość 1) to : `NazwaKlasy`.
- 2) Nazwa linii życia klasy, której statyczną operację lub konstruktor wywołuje wiadomość to `NazwaKlasy : NazwaKlasy`.
- 3) Nazwa linii życia tworzonego obiektu to `nazwaTworzonegoObiektu : NazwaKlasy`.
- 4) Nazwa linii życia pozostałych obiektów to „`nazwaObiektu : NazwaKlasy`”.
- 5) Operacja w dozorze (`[...]`) fragmentu (`alt, opt, loop, ...`) jest też pierwszą wiadomością w jego pierwszym operandzie.
- 6) Pętlę typu `for each` modeluje fragment `loop` o stereotypie «`forEach`» z nagłówkiem „`loop (element : kolekcja)`”.
- 7) Kolekcję danych oznaczają nawiasy `[]` lub `[*]`.
- 8) Brak wiadomości zwrotnej, czyli powrotu z wykonania operacji, jeśli ta operacja niczego nie zwraca (jest typu `void`).
- 9) Opis wiadomości zwrotnej, gdy nie wiadomo, jaki jest wynik operacji:

`miejsceZapisaniaWynikuOperacji = nazwaOperacji(wartośćParametruOperacji) : typWynikuOperacji`

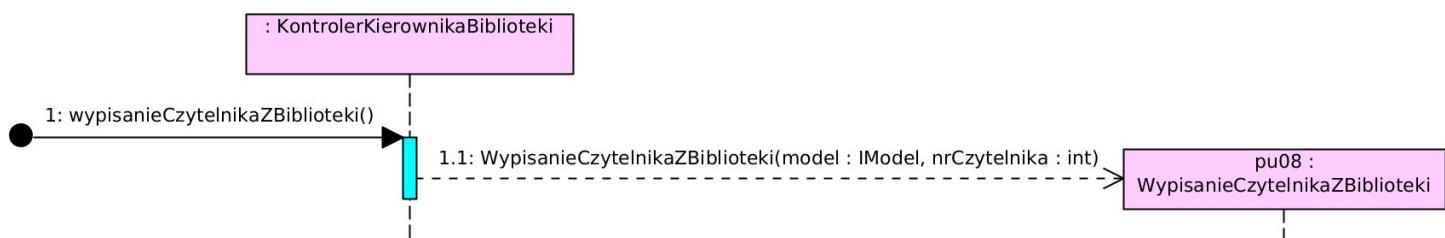
- 10) Opis wiadomości zwrotnej, gdy wiadomo, jaki jest wynik operacji:

`miejsceZapisaniaWynikuOperacji = nazwaOperacji(wartośćParametruOperacji) : wartośćWynikuOperacji`

Przy każdym diagramie umieszczono skrót kodu (pełny kod znajduje się w dalszej części tego dokumentu) odpowiadającego modelowanej na nim operacji:

- pominieto rzeczy mało istotne i rzeczy nie modelowane na diagramie,
- na prawym marginesie powiązano daną linię kodu z numerem odpowiedniej wiadomości lub fragmentu diagramu (tego zadanie laboratoryjne nie wymaga).

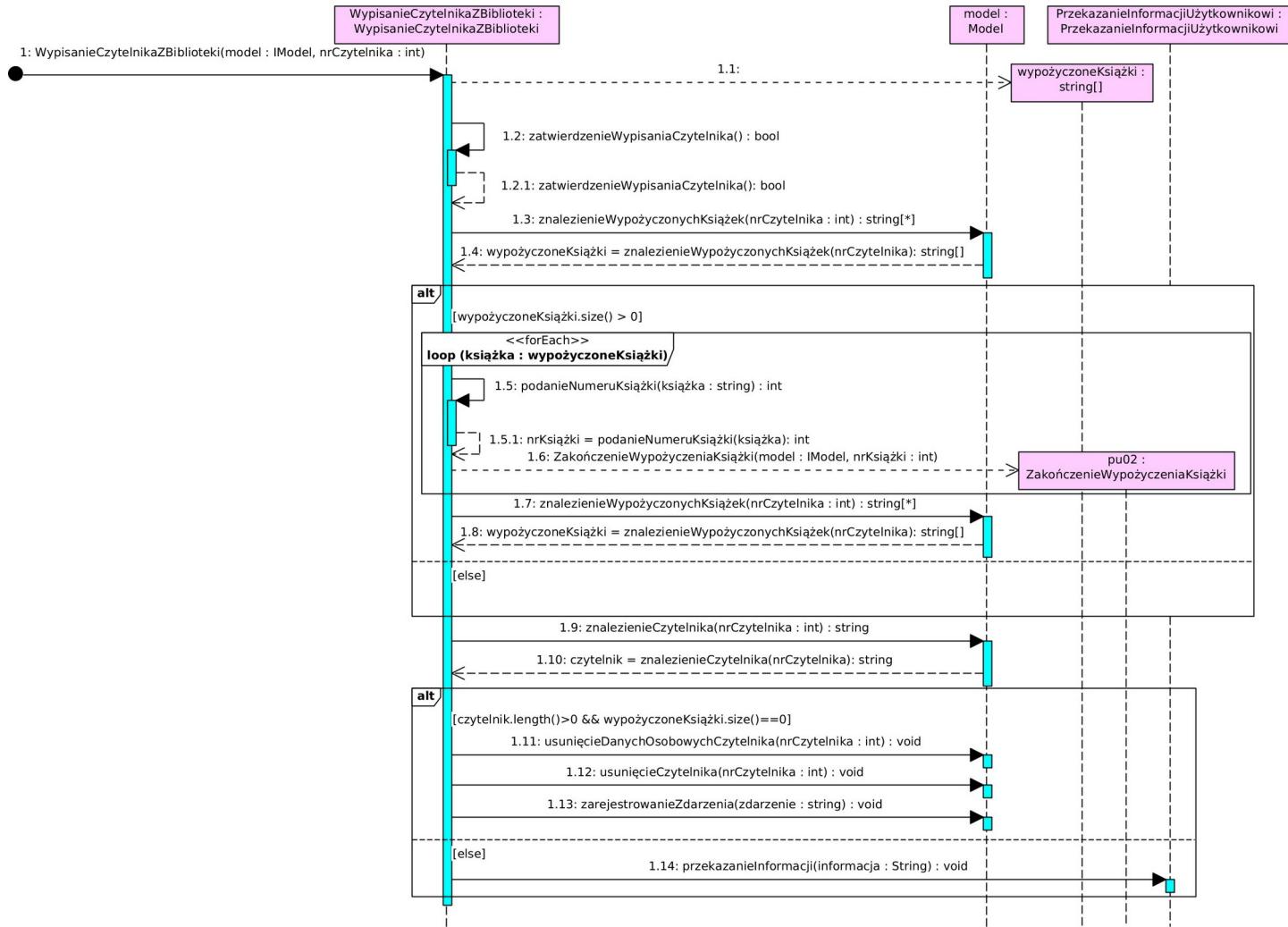
Operacja `Kontroler.KontrolerKierownikaBiblioteki.wypisanieCzytelnikaZBiblioteki()`



Rys. 24. Diagram sekwencji dla operacji `Kontroler.KontrolerKierownikaBiblioteki.wypisanieCzytelnikaZBiblioteki()`.

1. `public void wypisanieCzytelnikaZBiblioteki() {` 1
2. `int nrCzytelnika = 3;`
3. `WypisanieCzytelnikaZBiblioteki pu08 = new WypisanieCzytelnikaZBiblioteki(this.model, nrCzytelnika);` 1.1
4. `}`

Operacja Kontroler.WypisanieCzytelnikaZBiblioteki.WypisanieCzytelnikaZBiblioteki()



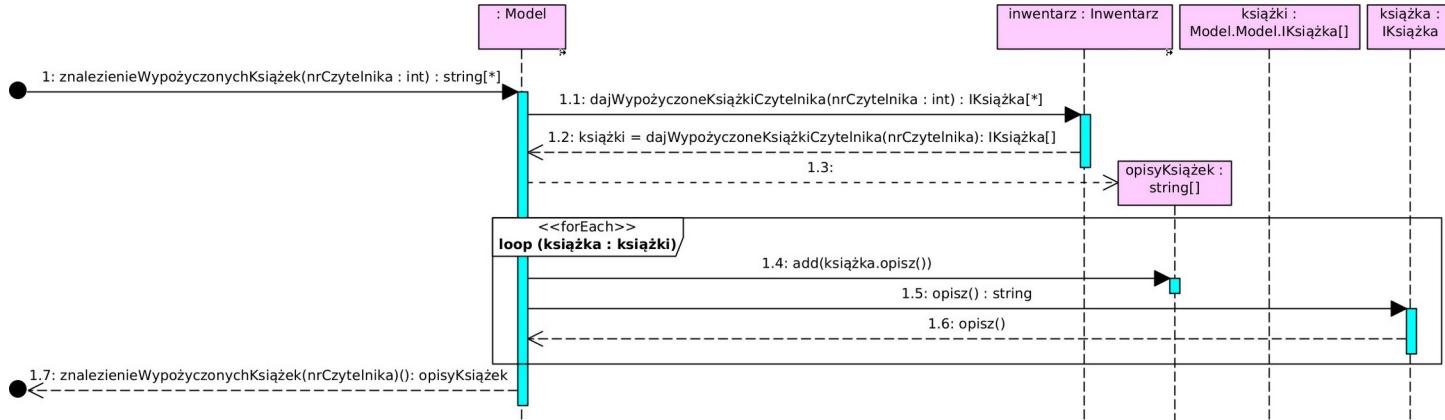
Rys. 25. Diagram sekwencji dla operacji Kontroler.WypisanieCzytelnikaZBiblioteki.WypisanieCzytelnikaZBiblioteki().

```

1. public WypisanieCzytelnikaZBiblioteki(IModel model, int nrCzytelnika) { ..... 1
2.     this.model = model;
3.     this.nrCzytelnika = nrCzytelnika;
4.     this.czytelnik = "";
5.     this.wypożyczoneKsiążki = new ArrayList<String>(); ..... 1.1
6.     this.zatwierdzenieWypisaniaCzytelnika(); ..... 1.2
7.     this.wypożyczoneKsiążki = this.model.znalezienieWypożyczonychKsiążek(this.nrCzytelnika); ..... 1.3
8.     if (this.wypożyczoneKsiążki.size()>0) { ..... 1.4
9.         for (String książka: this.wypożyczoneKsiążki) { ..... 1.5
10.             int nrKsiążki = this.podanieNumeruKsiążki(książka); ..... 1.5
11.             ZakończenieWypożyczeniaKsiążki pu02 = new ZakończenieWypożyczeniaKsiążki(this.model, nrKsiążki); ..... 1.6
12.         } ..... 1.7
13.         this.wypożyczoneKsiążki = this.model.znalezienieWypożyczonychKsiążek(this.nrCzytelnika); ..... 1.7
14.     }
15.     else
16.     ;
17.     this.czytelnik = this.model.znalezienieCzytelnika(this.nrCzytelnika); ..... 1.9
18.     if (this.czytelnik.length()>0 && this.wypożyczoneKsiążki.size()==0) { ..... 1.10
19.         this.model.usunięcieDanychOsobowychCzytelnika(this.nrCzytelnika); ..... 1.11
20.         this.model.usunięcieCzytelnika(this.nrCzytelnika); ..... 1.12
21.         this.model.zarejestrowanieZdarzenia("Wypisano czytelnika nr " + this.nrCzytelnika + "."); ..... 1.13
22.     }
23.     else {
24.         PrzekazanieInformacjiUzytkownikowi.przekazanieInformacji("X" + "Nie wypisano czytelnika: " + this.nrCzytelnika + "."); ..... 1.14
25.     }

```

Operacja Model.Model.znalezienieWypożyczonychKsiążek()



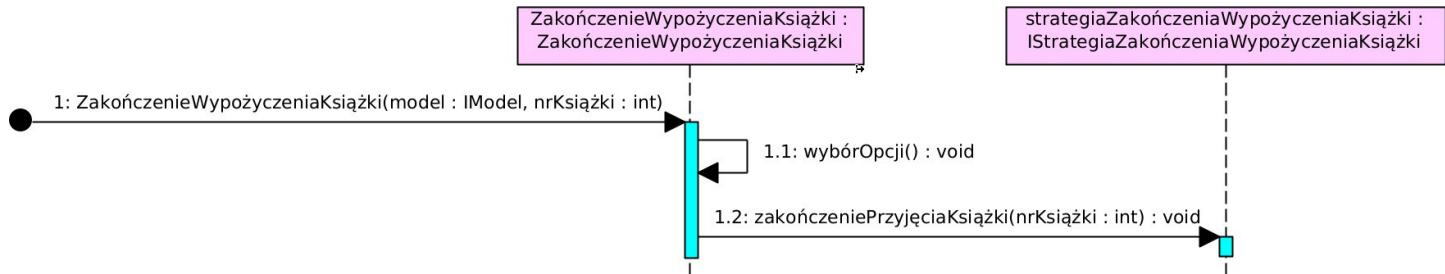
Rys. 26. Diagram sekwencji dla operacji *Model.Model.znalezienieWypożyczonychKsiążek()*.

```

1. public ArrayList<String> znalezienieWypożyczonychKsiążek(int nrCzytelnika) { ..... 1
2.     ArrayList<IKsiążka> książki = this.inwentarz.dajWypożyczoneKsiążkiCzytelnika(nrCzytelnika); ..... 1.1
3.     ArrayList<String> opisyKsiążek = new ArrayList<String>(); ..... 1.3
4.     for (IKsiążka książka: książki) { ..... loop ..... 1.5
5.         opisyKsiążek.add(książka.opisz()); ..... 1.5
6.     }
7.     return opisyKsiążek;
8. }

```

Operacja Kontroler.ZakończenieWypożyczeniaKsiążki.ZakończenieWypożyczeniaKsiążki()



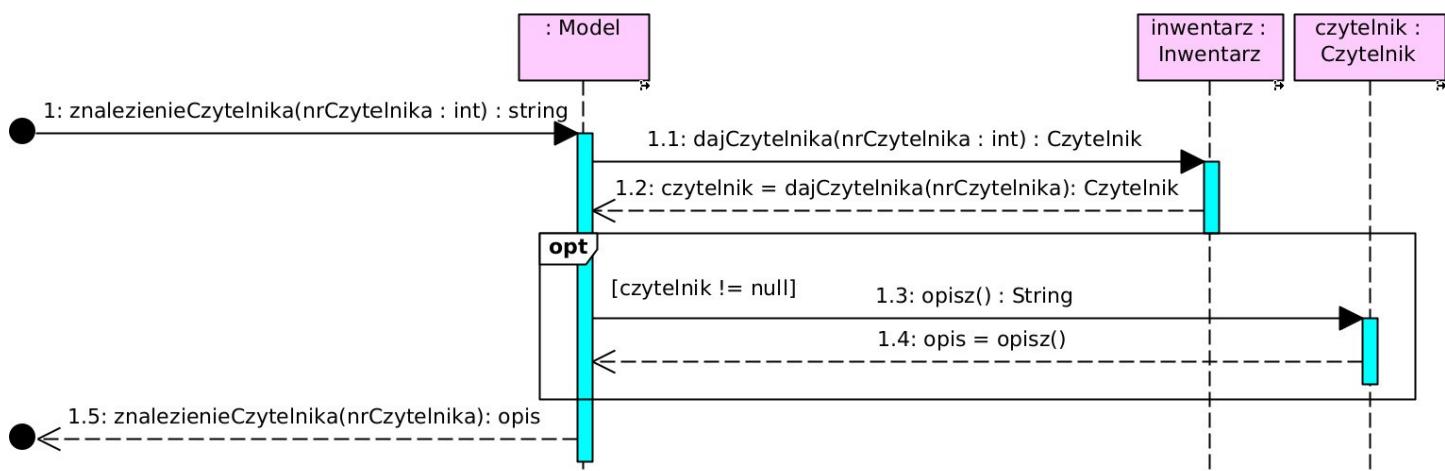
Rys. 27. Diagram sekwencji dla operacji *Kontroler.ZakończenieWypożyczeniaKsiążki.ZakończenieWypożyczeniaKsiążki()*.

```

1. public ZakończenieWypożyczeniaKsiążki(IModel model, int nrKsiążki) { ..... 1
2.     this.model = model;
3.     this.nrKsiążki = nrKsiążki;
4.     this.wybórOpcji(); ..... 1.1
5.     this.strategiaZakończeniaWypożyczeniaKsiążki.zakończeniePrzyjęciaKsiążki(this.nrKsiążki); ..... 1.2
6. }

```

Operacja Model.Model.znalezienieCzytelnika()



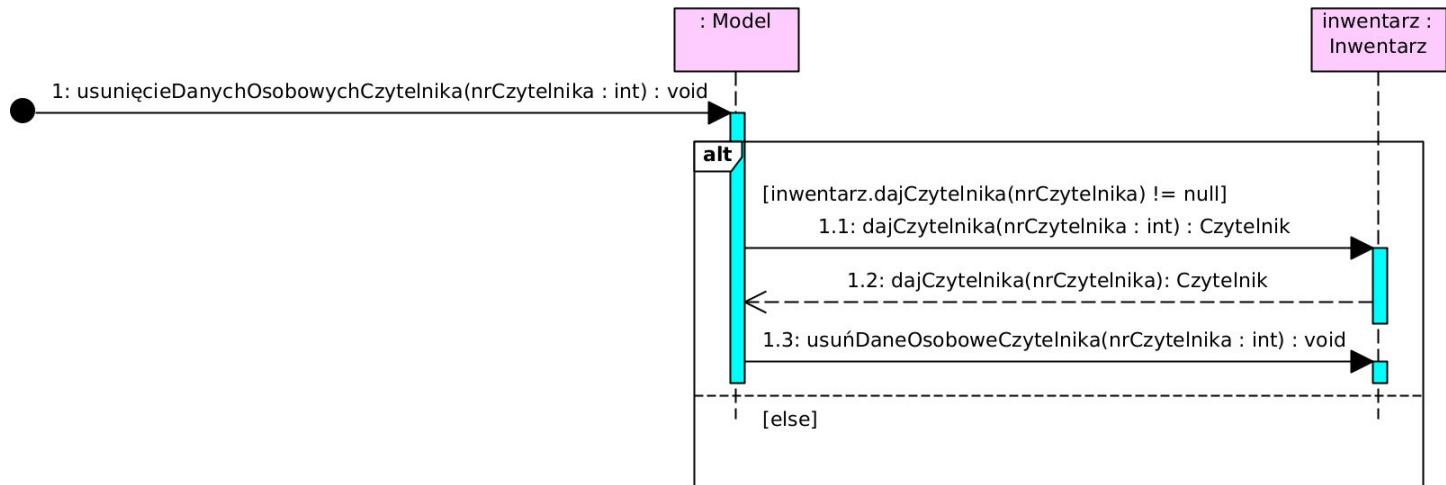
Rys. 28. Diagram sekwencji dla operacji *Model.Model.znalezienieCzytelnika()*.

```

1. public String znalezienieCzytelnika(int nrCzytelnika) { ..... 1
2.     String opis = "";
3.     Czytelnik czytelnik = this.inwentarz.dajCzytelnika(nrCzytelnika); ..... 1.1
4.     if (czytelnik != null) ..... opt ..... 1.3
5.         opis = czytelnik.opisz(); ..... 1.3
6.     return opis;
7. }

```

Operacja Model.Model.usunięcieDanychOsobowychCzytelnika()



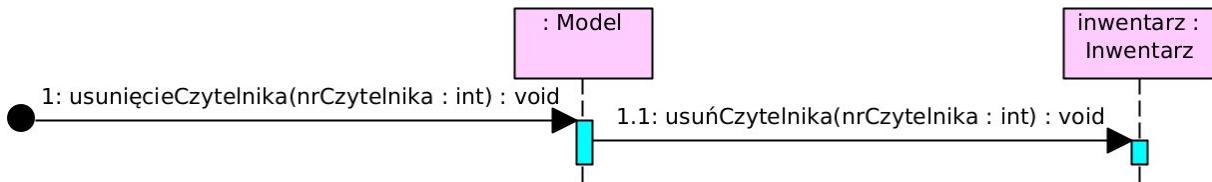
Rys. 29. Diagram sekwencji dla operacji *Model.Model.usunięcieDanychOsobowychCzytelnika()*.

```

1. public void usunięcieDanychOsobowychCzytelnika(int nrCzytelnika) { ..... 1
2.     if (this.inwentarz.dajCzytelnika(nrCzytelnika) != null) { ..... alt, 1.1
3.         this.inwentarz.usuńDaneOsoboweCzytelnika(nrCzytelnika); ..... 1.3
4.     }
5. } else
6. ;
7. }

```

Operacja Model.Model.usunięcieCzytelnika()



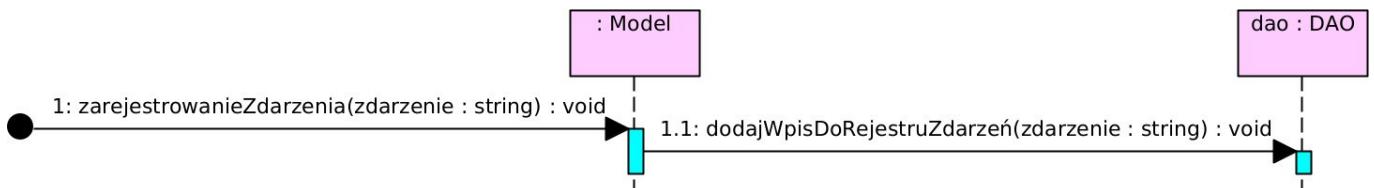
Rys. 30. Diagram sekwencji dla operacji *Model.Model.usunięcieCzytelnika()*.

```

1. public void usunięcieCzytelnika(int nrCzytelnika) { ..... 1
2.     this.inwentarz.usuńCzytelnika(nrCzytelnika); ..... 1.1
3. }

```

Operacja Model.Model.zarejestrowanieZdarzenia()



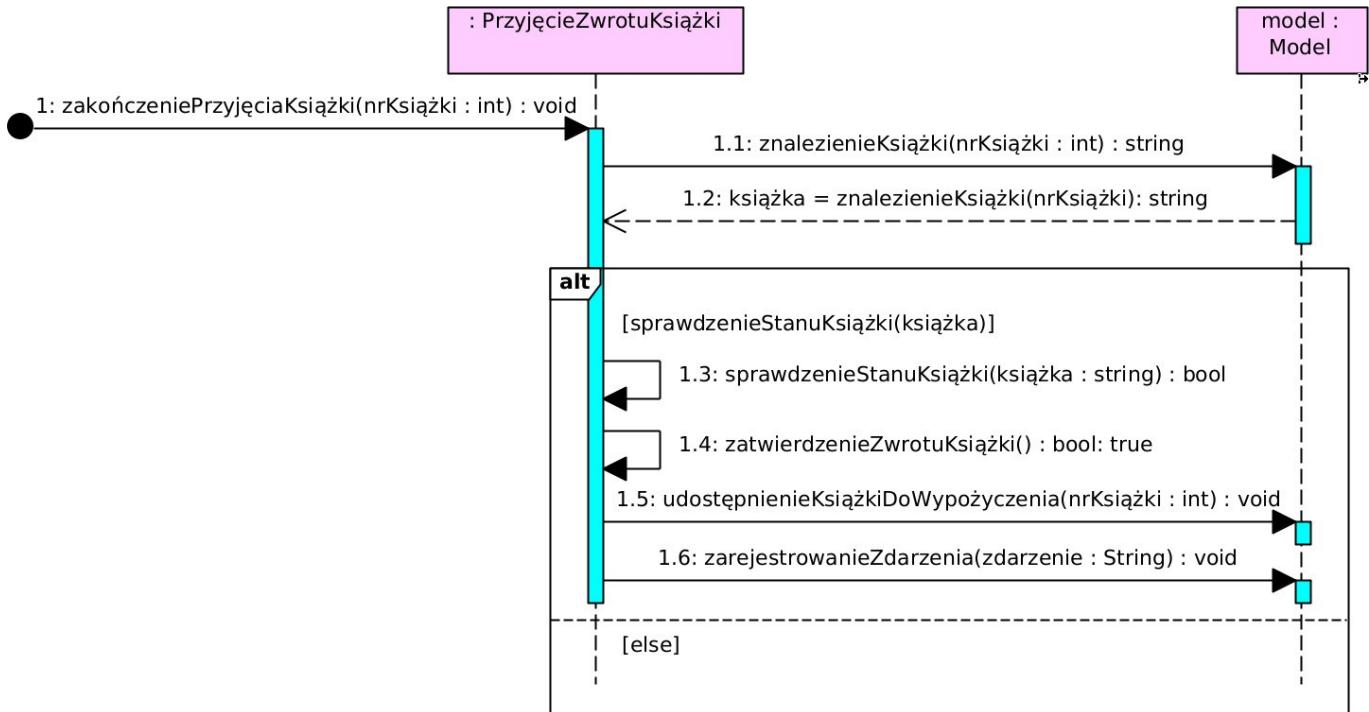
Rys. 31. Diagram sekwencji dla operacji *Model.Model.zarejestrowanieZdarzenia()*.

```

1. public void zarejestrowanieZdarzenia(String zdarzenie) { ..... 1
2.     this.dao.dodajWpisDoRejestruZdarzeń(zdarzenie); ..... 1.1
3. }

```

Operacja Kontroler.PrzyjęcieZwrotuKsiążki.zakończeniePrzyjęciaKsiążki()

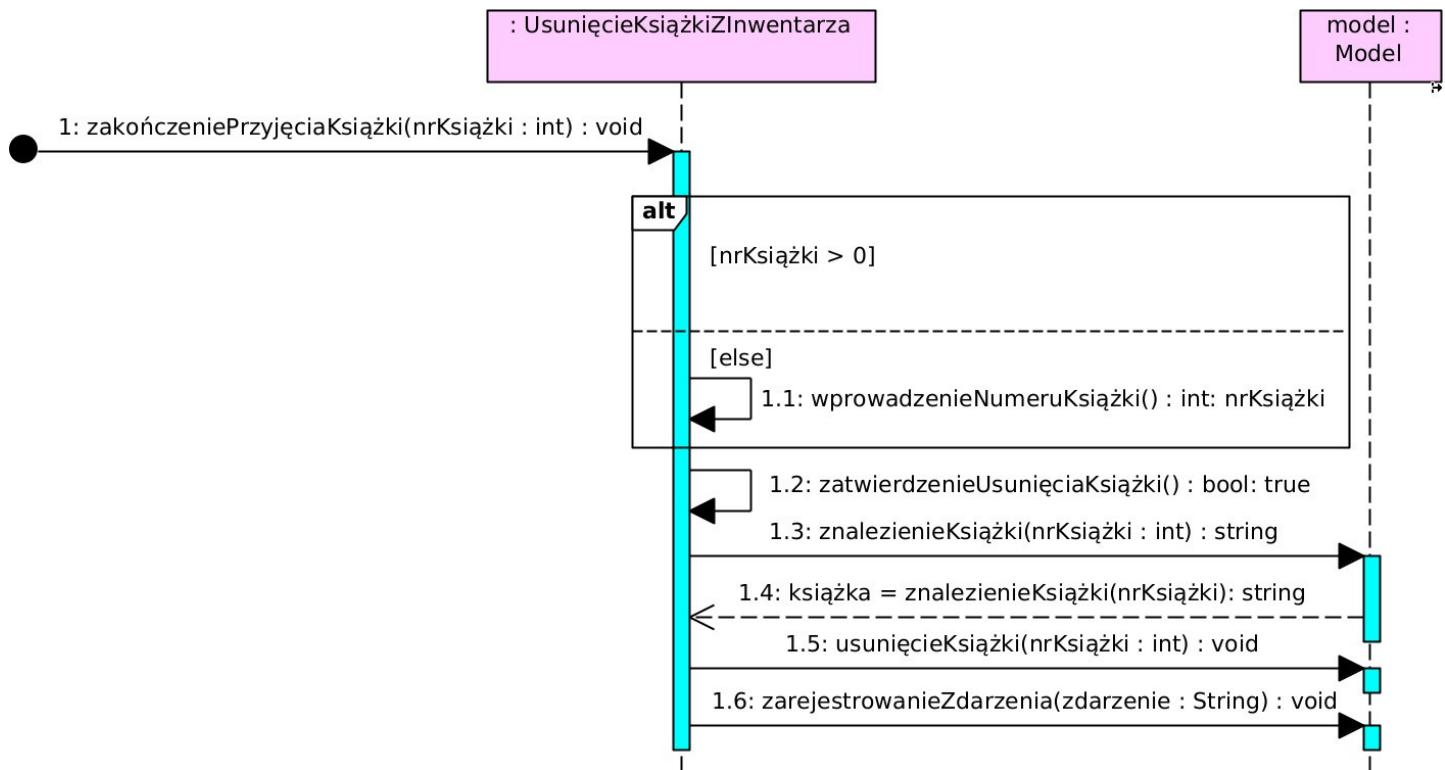


Rys. 32. Diagram sekwencji dla operacji Kontroler.PrzyjęcieZwrotuKsiążki.zakończeniePrzyjęciaKsiążki().

```

1. public void zakończeniePrzyjęciaKsiążki(int nrKsiążki) { ..... 1
2.     this.nrKsiążki = nrKsiążki;
3.     String książka = this.model.znalezienieKsiążki(nrKsiążki); ..... 1.1
4.     if (this.sprawdzenieStanuKsiążki(książka)) { ..... alt, 1.3
5.         this.zatwierdzenieZwrotuKsiążki(); ..... 1.4
6.         this.model.udostępnienieKsiążkiDoWypożyczenia(nrKsiążki); ..... 1.5
7.         this.model.zarejestrowanieZdarzenia("Przyjęto zwrot książki nr " + this.nrKsiążki + "."); ..... 1.6
8.     }
9.     else {
10.     };
11. }
12. 
```

Operacja Kontroler.UsunięcieKsiążkiZInwentarza.zakończeniePrzyjęciaKsiążki()



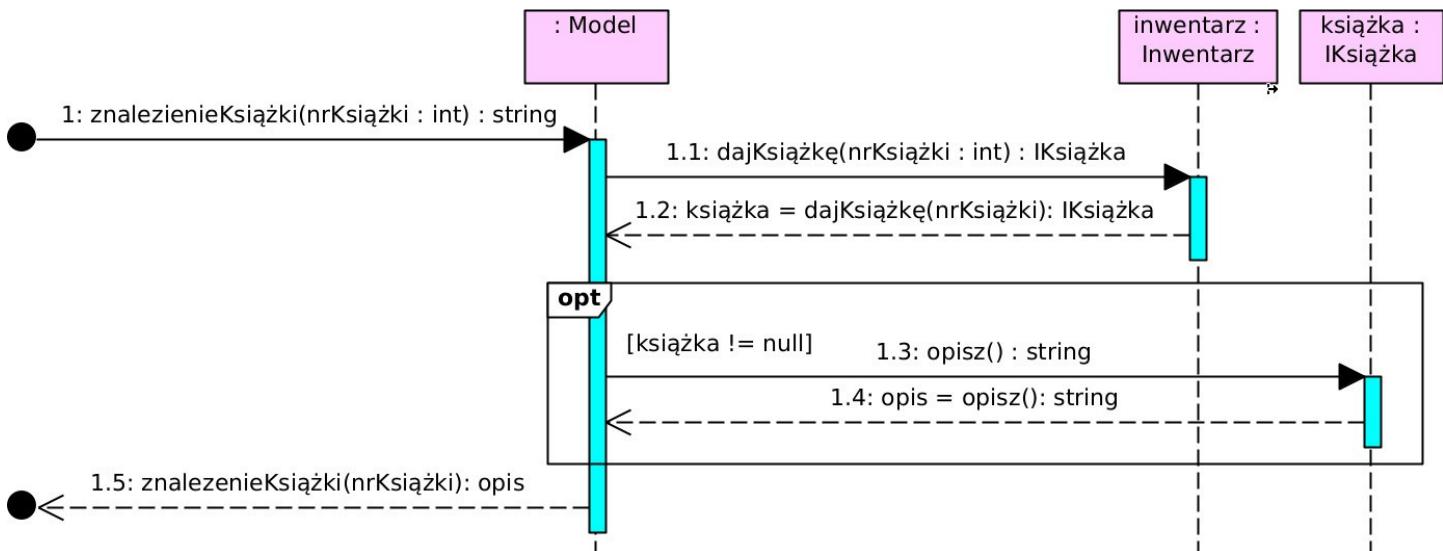
Rys. 33. Diagram sekwencji dla operacji Kontroler.UsunięcieKsiążkiZInwentarza.zakończeniePrzyjęciaKsiążki().

```

1. public void zakończeniePrzyjęciaKsiążki(int nrKsiążki) { ..... 1
2.     if (nrKsiążki > 0) {
3.         this.nrKsiążki = nrKsiążki;
4.     }
5.     else {
6.         this.nrKsiążki = this.wprowadzenieNumeruKsiążki(); ..... 1.1
7.     }
8.     this.zatwierdzenieUsunięciaKsiążki(); ..... 1.2
9.     String książka = this.model.znalezienieKsiążki(nrKsiążki); ..... 1.3
10.    this.model.usunięcieKsiążki(nrKsiążki); ..... 1.5
11.    this.model.zarejestrowanieZdarzenia("Usunięto książkę nr " + this.nrKsiążki + " z inwentarza."); ..... 1.6
12. }

```

Operacja Model.Model.znalezienieKsiążki()



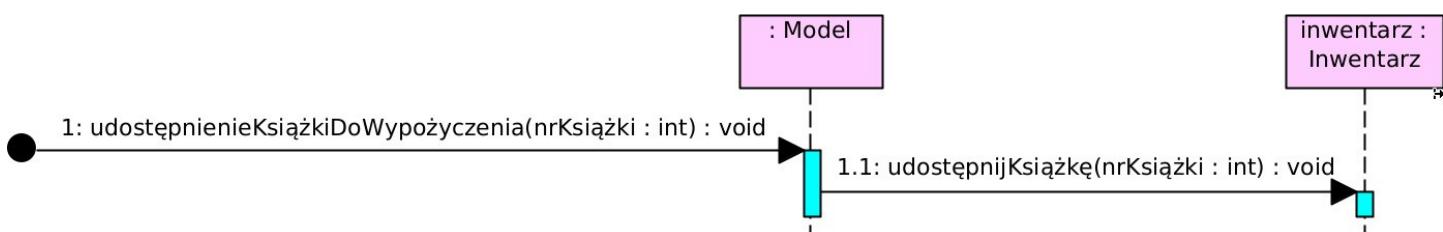
Rys. 34. Diagram sekwencji dla operacji *Model.Model.znalezienieKsiążki()*.

```

1. public String znalezienieKsiążki(int nrKsiążki) { ..... 1
2.     String opis = "";
3.     IKsiążka książka = this.inwentarz.dajKsiążkę(nrKsiążki); ..... 1.1
4.     if (książka != null) ..... 1.2
5.         opis = książka.opisz(); ..... 1.3
6.     return opis;
7. }

```

Operacja Model.Model.udostępnienieKsiążkiDoWypożyczenia()



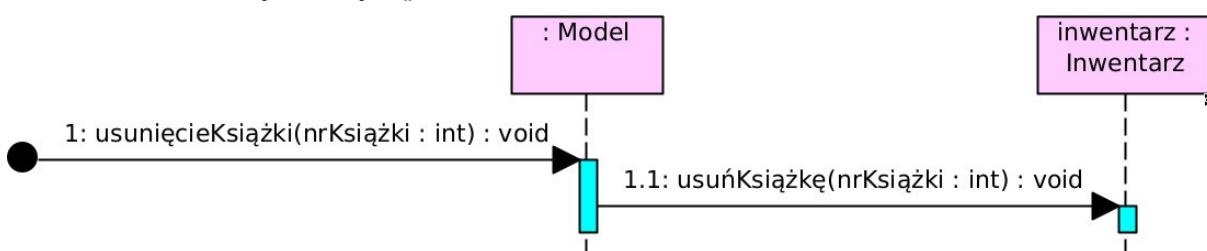
Rys. 35. Diagram sekwencji dla operacji *Model.Model.udostępnienieKsiążkiDoWypożyczenia()*.

```

1. public void udostępnienieKsiążkiDoWypożyczenia(int nrKsiążki) { ..... 1
2.     this.inwentarz.udostępnijKsiążkę(nrKsiążki); ..... 1.1
3. }

```

Operacja Model.Model.usunięcieKsiążki()



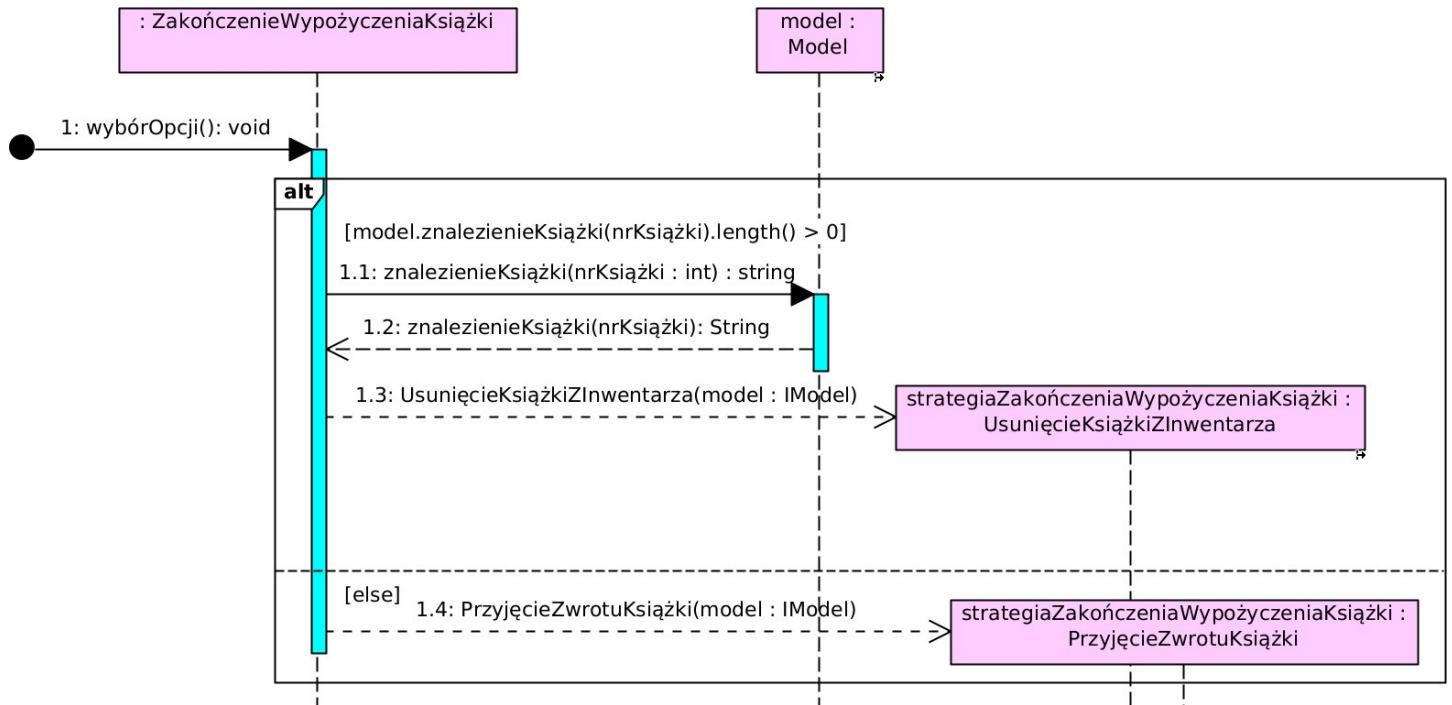
Rys. 36. Diagram sekwencji dla operacji *Model.Model.usunięcieKsiążki()*.

```

1. public void usunięcieKsiążki(int nrKsiążki) { ..... 1
2.     this.inwentarz.usuńKsiążkę(nrKsiążki); ..... 1.1
3. }

```

Operacja Kontroler.ZakończenieWypożyczeniaKsiążki.wybórOpcji()



Rys. 37. Diagram sekwencji dla operacji `Kontroler.ZakończenieWypożyczeniaKsiążki.wybórOpcji()`.

```

1. private void wybórOpcji() { ..... 1
2.     if (this.model.znalezienieKsiążki(this.nrKsiążki).length() > 0) { ..... alt, 1.1
3.         this.strategiaZakończeniaWypożyczeniaKsiążki = new UsunięcieKsiążkiZInwentarza(this.model); 1.3
4.     }
5.     else {
6.         this.strategiaZakończeniaWypożyczeniaKsiążki = new PrzyjęcieZwrotuKsiążki(this.model); 1.4
7.     }
8. }

```

Wynik działania programu w trybie testowym

```

1. Kontroler.SystemZarządzaniaBiblioteką
2. ✓ main():
3. --- System Zarządzania Biblioteką rozpoczął działanie.
4.
5. Model.DAO
6. ✓ pokażBazęCzytelników():
7. --- czytelnik nr 1 - Adam Orzeszko
8. --- czytelnik nr 2 - Stefania Michalska
9. --- czytelnik nr 3 - Agnieszka Nowacka
10. --- czytelnik nr 4 - Jan Jankowski
11. --- czytelnik nr 5 - Marek Sielski
12. --- czytelnik nr 6 - Barbara Głodna
13.
14. Model.DAO
15. ✓ pokażBazęKsiążek():
16. --- książka nr 1 - 'Niezwyciężony'; autor: Stanisław Lem
17. --- książka nr 2 - 'Ida sierpniowa'; autor: Małgorzata Musierowicz; wypożyczona dnia 2025-02-15 przez czytelnika nr 3
18. --- książka nr 3 - 'Nad Niemnem'; autor: Eliza Orzeszkowa; wypożyczona dnia 2021-06-29 przez czytelnika nr 3; zgubiona dnia 2021-12-01
19. --- książka nr 4 - 'Popioły'; autor: Stefan Żeromski
20. --- książka nr 5 - 'Paradyża'; autor: Janusz Zajdel; wypożyczona dnia 2025-11-06 przez czytelnika nr 2
21. --- książka nr 6 - 'Ben Hur'; autor: Lewis Wallace; zgubiona dnia 2011-12-31
22. --- książka nr 7 - 'Teatrzyk Zielona Gęś'; autor: Konstanty Ildefons Gałczyński
23.
24. Kontroler.KontrolerKierownikaBiblioteki
25. ✓ wypisanieCzytelnikaZBiblioteki():
26. --- Rozpoczęto realizację PU Wypisanie czytelnika z biblioteki.
27.
28. Kontroler.KontrolerKierownikaBiblioteki
29. ✓ wypisanieCzytelnikaZBiblioteki():
30. --- Wybrano do wypisania czytelnika nr 3.
31.
32. Kontroler.WypisanieCzytelnikaZBiblioteki
33. ✓ zatwierdzenieWypisaniaCzytelnika():
34. --- Kierownik biblioteki zatwierdził wypisanie czytelnika nr 3 z biblioteki.
35.
36. Model.Model
37. ✓ znalezienieWypożyczonychKsiążek():
38. --- Rozpoczęto szukanie książek wypożyczonych przez czytelnika nr 3.
39.
40. Model.Inwentarz
41. ✓ dajCzytelnika():
42. --- Znaleziono i pobrano: czytelnik nr 3 - Agnieszka Nowacka.
43.

```

44. Model.Inwentarz
✓ dajKsiążkę():
46. --- Znaleziono i pobrano: książka nr 2 - 'Ida sierpniowa'; autor: Małgorzata Musierowicz; wypożyczona dnia 2025-02-15 przez czytelnika nr 3.
47.
48. Model.Inwentarz
✓ dajCzytelnika():
50. --- Znaleziono i pobrano: czytelnik nr 3 - Agnieszka Nowacka.
51.
52. Model.Inwentarz
✓ dajKsiążkę():
54. --- Znaleziono i pobrano: książka nr 3 - 'Nad Niemnem'; autor: Eliza Orzeszkowa; wypożyczona dnia 2021-06-29 przez czytelnika nr 3; zgubiona dnia 2021-12-01.
55.
56. Model.Inwentarz
✓ dajWypożyczoneKsiążkiCzytelnika():
58. --- Znaleziono i pobrano książki czytelnika nr 3:
59. --- książka nr 2 - 'Ida sierpniowa'; autor: Małgorzata Musierowicz; wypożyczona dnia 2025-02-15 przez czytelnika nr 3
60. --- książka nr 3 - 'Nad Niemnem'; autor: Eliza Orzeszkowa; wypożyczona dnia 2021-06-29 przez czytelnika nr 3; zgubiona dnia 2021-12-01
61.
62. Model.Model
✓ znalezienieWypożyczonychKsiążek():
64. --- Zakończono szukanie książek wypożyczonych przez czytelnika nr 3.
65.
66. Kontroler.WypisanieCzytelnikaZBiblioteki
✓ WypisanieCzytelnikaZBiblioteki():
68. --- Znaleziono książki wypożyczone przez czytelnika nr 3:
69. --- książka nr 2 - 'Ida sierpniowa'; autor: Małgorzata Musierowicz; wypożyczona dnia 2025-02-15 przez czytelnika nr 3
70. --- książka nr 3 - 'Nad Niemnem'; autor: Eliza Orzeszkowa; wypożyczona dnia 2021-06-29 przez czytelnika nr 3; zgubiona dnia 2021-12-01
71.
72. Model.Inwentarz
✓ dajCzytelnika():
74. --- Znaleziono i pobrano: czytelnik nr 3 - Agnieszka Nowacka.
75.
76. Model.Inwentarz
✓ dajKsiążkę():
78. --- Znaleziono i pobrano: książka nr 2 - 'Ida sierpniowa'; autor: Małgorzata Musierowicz; wypożyczona dnia 2025-02-15 przez czytelnika nr 3.
79.
80. Kontroler.ZakończenieWypożyczeniaKsiążki
✓ wybórOpcji():
82. --- Kierownik biblioteki wybrał przyjęcie zwrotu książki nr 2.
83.
84. Model.Inwentarz
✓ dajCzytelnika():
86. --- Znaleziono i pobrano: czytelnik nr 3 - Agnieszka Nowacka.
87.

88. Model.Inwentarz
✓ dajKsiążkę():
90. --- Znaleziono i pobrano: książka nr 2 - 'Ida sierpniowa'; autor: Małgorzata Musierowicz; wypożyczona dnia 2025-02-15 przez czytelnika nr 3.
91.
92. Kontroler.PrzyjęcieZwrotuKsiążki
✓ zatwierdzenieZwrotuKsiążek():
94. --- Bibliotekarz zatwierdził przyjęcie zwrotu książki nr 2.
95.
96. Model.Model
✓ udostępnienieKsiążkiDoWypożyczenia():
98. --- Rozpoczęto udostępnienie książki nr 2.
99.
100. Model.Inwentarz
✓ dajCzytelnika():
102. --- Znaleziono i pobrano: czytelnik nr 3 - Agnieszka Nowacka.
103.
104. Model.Inwentarz
✓ dajKsiążkę():
106. --- Znaleziono i pobrano: książka nr 2 - 'Ida sierpniowa'; autor: Małgorzata Musierowicz; wypożyczona dnia 2025-02-15 przez czytelnika nr 3.
107.
108. Model.Inwentarz
✓ udostępnijKsiążkę():
110. --- Zmieniono książkę na dostępną: książka nr 2 - 'Ida sierpniowa'; autor: Małgorzata Musierowicz.
111.
112. Model.Model
✓ udostępnienieKsiążkiDoWypożyczenia():
114. --- Zakończono udostępnienie książki nr 2.
115.
116. Kontroler.PrzyjęcieZwrotuKsiążki
✓ zakończeniePrzyjęciaKsiążki():
118. --- Przyjęto zwrot książki nr 2.
119.
120. Model.Inwentarz
✓ dajCzytelnika():
122. --- Znaleziono i pobrano: czytelnik nr 3 - Agnieszka Nowacka.
123.
124. Model.Inwentarz
✓ dajKsiążkę():
126. --- Znaleziono i pobrano: książka nr 3 - 'Nad Niemnem'; autor: Eliza Orzeszkowa; wypożyczona dnia 2021-06-29 przez czytelnika nr 3; zgubiona dnia 2021-12-01.
127.
128. Kontroler.ZakończenieWypożyczeniaKsiążki
✓ wybórOpcji():
130. --- Kierownik biblioteki wybrał usunięcie książki nr 3.
131.
132. Kontroler.UsunięcieKsiążkiZInwentarza
✓ zatwierdzenieUsunięciaKsiążki():
134. --- Bibliotekarz zatwierdził usunięcie książki nr 3.
135.
136. Model.Inwentarz

137. ✓ dajCzytelnika():
138. --- Znaleziono i pobrano: czytelnik nr 3 - Agnieszka Nowacka.
139.
140. Model.Inwentarz
141. ✓ dajKsiążkę():
142. --- Znaleziono i pobrano: książka nr 3 - 'Nad Niemnem'; autor: Eliza Orzeszkowa; wypożyczona dnia 2021-06-29 przez czytelnika nr 3; zgubiona dnia 2021-12-01.
143.
144. Model.Model
145. ✓ usunięcieKsiążki():
146. --- Rozpoczęto usuwanie książki nr 3.
147.
148. Model.Inwentarz
149. ✓ dajCzytelnika():
150. --- Znaleziono i pobrano: czytelnik nr 3 - Agnieszka Nowacka.
151.
152. Model.Inwentarz
153. ✓ dajKsiążkę():
154. --- Znaleziono i pobrano: książka nr 3 - 'Nad Niemnem'; autor: Eliza Orzeszkowa; wypożyczona dnia 2021-06-29 przez czytelnika nr 3; zgubiona dnia 2021-12-01.
155.
156. Model.Inwentarz
157. ✓ usuńKsiążkę():
158. --- Usunięto: książka nr 3 - 'Nad Niemnem'; autor: Eliza Orzeszkowa; wypożyczona dnia 2021-06-29 przez czytelnika nr 3; zgubiona dnia 2021-12-01.
159.
160. Model.Model
161. ✓ usunięcieKsiążki():
162. --- Zakończono usuwanie książki nr 3.
163.
164. Kontroler.UsunięcieKsiążkiZInwentarza
165. ✓ zakończeniePrzyjęciaKsiążki():
166. --- Usunięto książkę nr 3 z inwentarza.
167.
168. Model.Model
169. ✓ znalezienieWypożyczonychKsiążek():
170. --- Rozpoczęto szukanie książek wypożyczonych przez czytelnika nr 3.
171.
172. Model.Inwentarz
173. X dajWypożyczoneKsiążkiCzytelnika():
174. --- Nie znaleziono książek czytelnika nr 3.
175.
176. Model.Model
177. ✓ znalezienieWypożyczonychKsiążek():
178. --- Zakończono szukanie książek wypożyczonych przez czytelnika nr 3.
179.
180. Model.Inwentarz
181. ✓ dajCzytelnika():
182. --- Znaleziono i pobrano: czytelnik nr 3 - Agnieszka Nowacka.
183.
184. Kontroler.WypisanieCzytelnikaZBiblioteki
185. ✓ WypisanieCzytelnikaZBiblioteki():
186. --- Rozpoczęto wypisywanie czytelnika: czytelnik nr 3 - Agnieszka Nowacka.
187.
188. Model.Inwentarz
189. ✓ dajCzytelnika():
190. --- Znaleziono i pobrano: czytelnik nr 3 - Agnieszka Nowacka.
191.
192. Model.Inwentarz
193. ✓ dajCzytelnika():
194. --- Znaleziono i pobrano: czytelnik nr 3 - Agnieszka Nowacka.
195.
196. Model.Model
197. ✓ usunięcieDanychOsobowychCzytelnika():
198. --- Usunięto dane osobowe czytelnika nr 3.
199.
200. Model.Model
201. ✓ usunięcieCzytelnika():
202. --- Rozpoczęto usuwanie czytelnika nr 3.
203.
204. Model.Inwentarz
205. ✓ dajCzytelnika():
206. --- Znaleziono i pobrano: czytelnik nr 3 - usunięto imię usunięto nazwisko.
207.
208. Model.Inwentarz
209. ✓ usuńCzytelnika():
210. --- Usunięto: czytelnik nr 3 - usunięto imię usunięto nazwisko.
211.
212. Model.Model
213. ✓ usunięcieCzytelnika():
214. --- Zakończono usuwanie czytelnika nr 3.
215.
216. Kontroler.WypisanieCzytelnikaZBiblioteki
217. ✓ WypisanieCzytelnikaZBiblioteki():
218. --- Wypisano czytelnika: 3.
219.
220. Kontroler.KontrolerKierownikaBiblioteki
221. ✓ wypisanieCzytelnikaZBiblioteki():
222. --- Zakończono realizację PU Wypisanie czytelnika z biblioteki.
223.
224. Kontroler.SystemZarządzaniaBiblioteką
225. ✓ main():
226. --- System Zarządzania Biblioteką zakończył działanie.
227.
228. Model.DAO
229. ✓ pokażBazęCzytelników():
230. --- czytelnik nr 1 - Adam Orzeszko

```

231. --- czytelnik nr 2 - Stefania Michalska
232. --- czytelnik nr 4 - Jan Jankowski
233. --- czytelnik nr 5 - Marek Sielski
234. --- czytelnik nr 6 - Barbara Głodna
235.
236. Model.DAO
237. ✓ pokażBazęKsiążek():
238. --- książka nr 1 - 'Niezwyciężony'; autor: Stanisław Lem
239. --- książka nr 2 - 'Ida sierpiowa'; autor: Małgorzata Musierowicz
240. --- książka nr 4 - 'Popioly'; autor: Stefan Żeromski
241. --- książka nr 5 - 'Paradyżja'; autor: Janusz Zajdel; wypożyczona dnia 2025-11-06 przez czytelnika nr 2
242. --- książka nr 6 - 'Ben Hur'; autor: Lewis Wallace; zgubiona dnia 2011-12-31
243. --- książka nr 7 - 'Teatrzyk Zielona Gęś'; autor: Konstanty Ildefons Gałczyński

```

Kod klas

kod* wszystkich klas z zadania 2

* Kod wygenerowany przez Visual Paradigm, poprawiony i uzupełniony zgodnie z treścią zadania 2.

Klasy pakietu *Kontroler* zaimplementowano w zakresie umożliwiającym wykonanie przypadku użycia PU08. *Wypisanie czytelnika z biblioteki*.

Akcje użytkownika zostały zastąpione sztywnym kodem symulującym ich wynik.

Kontroler.IKontrolerBibliotekarza.java

```

1. package Kontroler;
2.
3. /**
4.  * Interfejs fasady Kontrolera dla Bibliotekarza.
5. */
6. public interface IKontrolerBibliotekarza {
7.
8.     /**
9.      * Realizacja PU Wypożyczenie Książki (wypożyczenie książki czytelnikowi).
10.     */
11.    public void wypożyczenieKsiążki();
12.
13.    /**
14.     * Realizacja PU Zakończenie wypożyczenia książki (zakończenie wypożyczenia książki czytelnikowi przez jej zwrot lub usunięcie z inwentarza).
15.     */
16.    public void zakończenieWypożyczeniaKsiążki();
17.
18.    /**
19.     * Realizacja PU Usunięcie książki z inwentarza.
20.     */
21.    public void usunięcieKsiążkiZInwentarza();
22.
23.    /**
24.     * Realizacja PU Wprowadzenie książki do inwentarza (wprowadzenie nowej książki do inwentarza).
25.     */
26.    public void wprowadzenieKsiążkiDoInwentarza();
27. }

```

Kontroler.IKontrolerKierownikaBiblioteki.java

```

1. package Kontroler;
2.
3. /**
4.  * Interfejs fasady Kontrolera dla Kierownika Biblioteki.
5. */
6. public interface IKontrolerKierownikaBiblioteki extends IKontrolerBibliotekarza {
7.
8.     /**
9.      * Realizacja PU Edytowanie danych czytelnika biblioteki (edytacja lub wprowadzenie danych osobowych czytelnika).
10.     */
11.    public void edytowanieDanychCzytelnikaBiblioteki();
12.
13.    /**
14.     * Realizacja PU wypisanie czytelnika z biblioteki.
15.     */
16.    public void wypisanieCzytelnikaZBiblioteki();
17.
18.    /**
19.     * Realizacja PU zapisanie czytelnika do biblioteki (zapisanie nowego czytelnika do biblioteki).
20.     */
21.    public void zapisanieCzytelnikaDoBiblioteki();
22. }

```

Kontroler.IStrategiaZakończeniaWypożyczeniaKsiążki.java

```

1. package Kontroler;
2. import Model.IModel;

```

```

3.
4. /**
5. * Abstrakcja strategii zakończenia wypożyczenia książki.
6. */
7. public abstract class IStrategiaZakończeniaWypożyczeniaKsiążki {
8.     protected IModel model;
9.     protected int nrKsiążki;
10.
11.    public abstract void zakończeniePrzyjęciaKsiążki(int nrKsiążki);
12. }
13.

```

Kontroler.KontrolerBibliotekarza.java

```

1. package Kontroler;
2. import Model.IModel;
3.
4. /**
5. * Fasada Kontrolera dla Bibliotekarza.
6. */
7. public class KontrolerBibliotekarza implements IKontrolerBibliotekarza {
8.     private IModel model;
9.
10.    /**
11.     * @param model fasada Modelu
12.     */
13.    public KontrolerBibliotekarza(IModel model) {
14.        this.model = model;
15.    }
16.
17.    public void wypożyczenieKsiążki() {
18.        throw new UnsupportedOperationException();
19.    }
20.
21.    public void zakończenieWypożyczeniaKsiążki() {
22.        throw new UnsupportedOperationException();
23.    }
24.
25.    public void usunięcieKsiążkiZInwentarza() {
26.        throw new UnsupportedOperationException();
27.    }
28.
29.    public void wprowadzenieKsiążkiDoInwentarza() {
30.        throw new UnsupportedOperationException();
31.    }
32. }

```

Kontroler.KontrolerKierownikaBiblioteki.java

```

1. package Kontroler;
2. import Komunikacja.Widok;
3. import Model.IModel;
4.
5. /**
6. * Fasada Kontrolera dla Kierownika Biblioteki.
7. */
8. public class KontrolerKierownikaBiblioteki implements IKontrolerKierownikaBiblioteki {
9.     private IModel model;
10.
11.    /**
12.     * @param model fasada Modelu
13.     */
14.    public KontrolerKierownikaBiblioteki(IModel model) {
15.        this.model = model;
16.    }
17.
18.    public void edytowanieDanychCzytelnikaBiblioteki() {
19.        throw new UnsupportedOperationException();
20.    }
21.
22.    public void wypisanieCzytelnikaZBiblioteki() {
23.        Widok.pokaż(this.getClass().getCanonicalName(), "wypisanieCzytelnikaZBiblioteki", true, "Rozpoczęto realizację PU Wypisanie czytelnika z biblioteki.");
24.        // symulacja działania użytkownika: odpowiedź na pytanie o numeru czytelnika do wypisania
25.        int nrCzytelnika = 3; // czytelnik nr 3 ma 1 wypożyczoną i 1 zgubioną książkę
26.        Widok.pokaż(this.getClass().getCanonicalName(), "wypisanieCzytelnikaZBiblioteki", true, "Wybrano do wypisania czytelnika nr " + nrCzytelnika + ".");
27.        // przekazanie obsługi tego zadania klasie WypisanieCzytelnikaZBiblioteki
28.        WypisanieCzytelnikaZBiblioteki pu08 = new WypisanieCzytelnikaZBiblioteki(this.model, nrCzytelnika);
29.        Widok.pokaż(this.getClass().getCanonicalName(), "wypisanieCzytelnikaZBiblioteki", true, "Zakończono realizację PU Wypisanie czytelnika z biblioteki.");
30.    }
31.
32.    public void zapisanieCzytelnikaDoBiblioteki() {
33.        throw new UnsupportedOperationException();
34.    }
35.
36.    public void wypożyczenieKsiążki() {
37.        throw new UnsupportedOperationException();
38.    }

```

```

39.
40.     public void zakończenieWypożyczeniaKsiążki() {
41.         throw new UnsupportedOperationException();
42.     }
43.
44.     public void usunięcieKsiążkiZInwentarza() {
45.         throw new UnsupportedOperationException();
46.     }
47.
48.     public void wprowadzenieKsiążkiDoInwentarza() {
49.         throw new UnsupportedOperationException();
50.     }
51. }
```

Kontroler.PrzekazanieInformacjiUżytkownikowi.java

```

1. package Kontroler;
2. import Komunikacja.Widok;
3.
4. public abstract class PrzekazanieInformacjiUżytkownikowi {
5.
6.     /**
7.      * Wyświetlenie informacji na ekranie użytkownika.
8.      * @param informacja  treść informacji ((X)?informacja)
9.      */
10.    public static void przekazanieInformacji(String informacja) {
11.        // informacja o porażce zaczyna się od X
12.        Boolean sukces = informacja.substring(0, 1) == "X" ? true : false;
13.        Widok.pokaż("Kontroler.PrzekazanieInformacjiUżytkownikowi", "przekazanieInformacji", sukces, informacja.replace("X", ""));
14.    }
15. }
```

Kontroler.PrzyjęcieZwrotuKsiążki.java

```

1. package Kontroler;
2. import Komunikacja.CSV;
3. import Komunikacja.Widok;
4. import Model.IModel;
5.
6. public class PrzyjęcieZwrotuKsiążki extends IStrategiaZakończeniaWypożyczeniaKsiążki {
7.
8.     /**
9.      * Realizacja PU Przyjęcie zwrotu książki.
10.     * @param model  fasada Modelu
11.     */
12.    public PrzyjęcieZwrotuKsiążki(IModel model) {
13.        this.model = model;
14.    }
15.
16.    /**
17.     * Zakończenie przyjęcia książki w postaci jej zwrotnego.
18.     */
19.    public void zakończeniePrzyjęciaKsiążki(int nrKsiążki) {
20.        this.nrKsiążki = nrKsiążki;
21.        // znalezienie książki
22.        String książka = this.model.znalezienieKsiążki(nrKsiążki);
23.        // sprawdzenie stanu książki, czy jest wypożyczona
24.        if (this.sprawdzenieStanuKsiążki(książka)) {
25.            // zatwierdzenie zwrotu książki
26.            this.zatwierdzenieZwrotuKsiążki();
27.            // udostępnienie książki do wypożyczenia
28.            this.model.udostępnienieKsiążkiDoWypożyczenia(nrKsiążki);
29.            // zarejestrowanie zwrotu książki
30.            this.model.zarejestrowanieZdarzenia("Przyjęto zwrot książki nr " + this.nrKsiążki + ".");
31.            Widok.pokaż(this.getClass().getCanonicalName(), "zakończeniePrzyjęciaKsiążki", true, "Przyjęto zwrot książki nr " + this.nrKsiążki + ".");
32.        } else {
33.            // w przeciwnym razie - informacja o niemożności zwrotu książki
34.            Widok.pokaż(this.getClass().getCanonicalName(), "zakończeniePrzyjęciaKsiążki", false, "Nie przyjęto zwrotu książki nr " + this.nrKsiążki + ".");
35.        }
36.    }
37.
38.    /**
39.     * Sprawdzenie stanu książki.
40.     * @param książka  csv książka
41.     * @return          true, jeśli jest wypożyczona
42.     */
43.    private boolean sprawdzenieStanuKsiążki(String książka) {
44.        CSV csv = new CSV();
45.        if (csv.znajdźDatemWypożyczeniaKsiążki(książka).length() > 0)
46.            return true;
47.        else return false;
48.    }
49.
50.    /**
51.     * Czy zatwierdzić zwrot książki.
52.     * @return          true, jeśli zatwierdzić
53.     */
54.    private boolean zatwierdzenieZwrotuKsiążki() {
```

```

56. // symulacja działania użytkownika: odpowiedź na pytanie o zatwierdzenie zwrotu książki
57. boolean zatwierdzono = true; // zatwierdzono
58. if (zatwierdzono)
59.     Widok.pokaż(this.getClass().getCanonicalName(), "zatwierdzenieZwrotuKsiążki", true, "Bibliotekarz zatwierdził przyjęcie zwrotu książki nr " + this.nrKsiążki+ ".");
60. else
61.     Widok.pokaż(this.getClass().getCanonicalName(), "zatwierdzenieZwrotuKsiążki", false,
62.                 "Bibliotekarz nie zatwierdził przyjęcia zwrotu książki nr " + this.nrKsiążki+ ".");
63. return zatwierdzono;
64. }
65.

```

Kontroler.SystemZarządzaniaBiblioteką.java

```

1. package Kontroler;
2. import Komunikacja.Widok;
3. import Model.*;
4.
5. public class SystemZarządzaniaBiblioteką {
6.
7.     /**
8.      * Główna operacja programu, testująca PU Wypisanie czytelnika z biblioteki.
9.      * @param args
10.     */
11.    public static void main(String[] args) {
12.        // ustawienie poziomu gadatliwości programu
13.        Widok.ustawPoziomGadatliwości(Widok.Gadatliwość.WYSOKA);
14.        // utworzenie obiektów komponentu Model
15.        IDAO dao = new DAO();
16.        Inwentarz inventarz = new Inwentarz(dao);
17.        IModel model = new Model(inwentarz, dao);
18.        Widok.pokaż("Kontroler.SystemZarządzaniaBiblioteką", "main", true, "System Zarządzania Biblioteką rozpoczął działanie.");
19.        // wyświetlenie początkowej zawartości bazy danych (do celów testowych)
20.        dao.pokażBazęCzytelników();
21.        dao.pokażBazęKsiążek();
22.        // utworzenie obiektów komponentu Kontroler
23.        KontrolerKierownikaBiblioteki kontroler = new KontrolerKierownikaBiblioteki(model);
24.        // testowe wykonanie PU Wypisanie czytelnika z biblioteki
25.        kontroler.wypisanieCzytelnikaZBiblioteki();
26.        Widok.pokaż("Kontroler.SystemZarządzaniaBiblioteką", "main", true, "System Zarządzania Biblioteką zakończył działanie.");
27.        // wyświetlenie końcowej zawartości bazy danych (do celów testowych)
28.        dao.pokażBazęCzytelników();
29.        dao.pokażBazęKsiążek();
30.    }
31. }

```

Kontroler.UsuniecieKsiążkiZInwentarza.java

```

1. package Kontroler;
2. import Komunikacja.Widok;
3. import Model.IModel;
4.
5. public class UsunięcieKsiążkiZInwentarza extends IStrategiaZakończeniaWypożyczeniaKsiążki {
6.
7.     /**
8.      * Realizacja PU Usunięcie książki z inwentarza.
9.      * @param model fasada Modelu
10.     */
11.    public UsunięcieKsiążkiZInwentarza(IModel model) {
12.        this.model = model;
13.    }
14.
15.    /**
16.     * Zakończenie przyjęcia książki w postaci jej zwrócenia.
17.     */
18.    public void zakończeniePrzyjęciaKsiążki(int nrKsiążki) {
19.        // jeśli podano numer książki
20.        if (nrKsiążki > 0) {
21.            this.nrKsiążki = nrKsiążki;
22.        }
23.        // jeśli nie podano numeru książki (czyli podano 0)
24.        else {
25.            this.nrKsiążki = this.wprowadzenieNumeruKsiążki();
26.        }
27.        // zatwierdzenie usunięcia książki z inwentarza
28.        this.zatwierdzenieUsunięciaKsiążki();
29.        // znalezienie książki
30.        String książka = this.model.znalezienieKsiążki(nrKsiążki);
31.        // usunięcie książki
32.        this.model.usunięcieKsiążki(nrKsiążki);
33.        // zarejestrowanie usunięcia książki
34.        this.model.zarejestrowanieZdarzenia("Usunięto książkę nr " + this.nrKsiążki + " z inwentarza.");
35.        Widok.pokaż(this.getClass().getCanonicalName(), "zakończeniePrzyjęciaKsiążki", true, "Usunięto książkę nr " + this.nrKsiążki + " z inwentarza.");
36.    }
37.
38.    /**
39.     * Wprowadzenie numeru usuwanej książki,
40.     * jeśli usunięcie książki z inwentarza jest inicjowane przez bibliotekarza, a nie jest częścią Wypisania czytelnika.
41.     * @return numer książki
42.     */
43.    private int wprowadzenieNumeruKsiążki() {

```

```

44. // symulacja działania użytkownika: wybór numeru książki 1
45. int nrKsiążki = 1;
46. return nrKsiążki;
47. }
48.
49. /**
50. * Czy zatwierdzić usunięcie książki.
51. * @return true, jeśli zatwierdzić
52. */
53. private boolean zatwierdzenieUsunięciaKsiążki() {
54.     // symulacja działania użytkownika: odpowiedź na pytanie o zatwierdzenie usunięcia książki
55.     boolean zatwierdzono = true; // zatwierdzono
56.     if (zatwierdzono)
57.         Widok.pokaż(this.getClass().getCanonicalName(), "zatwierdzenieUsunięciaKsiążki", true, "Bibliotekarz zatwierdził usunięcie książki nr " + this.nrKsiążki+ ".");
58.     else
59.         Widok.pokaż(this.getClass().getCanonicalName(), "zatwierdzenieUsunięciaKsiążki", false,
60.                     "Bibliotekarz nie zatwierdził usunięcia książki nr " + this.nrKsiążki+ ".");
61.     return zatwierdzono;
62. }
63.

```

Kontroler.WypisanieCzytelnikaZBiblioteki.java

```

1. package Kontroler;
2. import java.util.ArrayList;
3. import Komunikacja.CSV;
4. import Komunikacja.Widok;
5. import Model.IModel;
6.
7. public class WypisanieCzytelnikaZBiblioteki {
8.     private IMModel model;
9.     private int nrCzytelnika;
10.    private String czytelnik;
11.    private ArrayList<String> wypożyczoneKsiążki;
12.
13.    /**
14.     * Realizacja PU Wypisanie czytelnika z biblioteki.
15.     * @param model fasada Modelu
16.     * @param nrCzytelnika numer czytelnika
17.    */
18.    public WypisanieCzytelnikaZBiblioteki(IMModel model, int nrCzytelnika) {
19.        this.model = model;
20.        this.nrCzytelnika = nrCzytelnika;
21.        this.czytelnik = "";
22.        this.wypożyczoneKsiążki = new ArrayList<String>();
23.        CSV csv = new CSV();
24.        // zatwierdzenie wypisania czytelnika przez Kierownika Biblioteki
25.        this.zatwierdzenieWypisaniaCzytelnika();
26.        // znalezienie książek wypożyczonych przez wypisywanego czytelnika
27.        this.wypożyczoneKsiążki = this.model.znalezienieWypożyczonychKsiążek(this.nrCzytelnika);
28.        if (this.wypożyczoneKsiążki.size()>0) {
29.            ArrayList<String> wiadomość = new ArrayList<String>();
30.            wiadomość.add("Znaleziono książki wypożyczone przez czytelnika nr " + this.nrCzytelnika + ":" );
31.            for (String książka: this.wypożyczoneKsiążki) {
32.                wiadomość.add(csv.opiszKsiążkę(książka));
33.            }
34.            Widok.pokaż(this.getClass().getCanonicalName(), "WypisanieCzytelnikaZBiblioteki", true, wiadomość);
35.            // podanie numeru każdej wypożyczonej książki
36.            for (String książka: this.wypożyczoneKsiążki) {
37.                int nrKsiążki = this.podanieNumeruKsiążki(książka);
38.                // zakończenie wypożyczenia książki - przekazanie obsługi tego zadania klasie ZakończenieWypożyczeniaKsiążki
39.                ZakończenieWypożyczeniaKsiążki pu02 = new ZakończenieWypożyczeniaKsiążki(this.model, nrKsiążki);
40.            }
41.            // znalezienie książek nadal wypożyczonych przez wypisywanego czytelnika
42.            this.wypożyczoneKsiążki = this.model.znalezienieWypożyczonychKsiążek(this.nrCzytelnika);
43.        }
44.        else
45.            Widok.pokaż(this.getClass().getCanonicalName(), "WypisanieCzytelnikaZBiblioteki", false,
46.                        "Nie znaleziono książek wypożyczonych przez czytelnika nr " + this.nrCzytelnika + ":" );
47.        // znalezienie czytelnika
48.        this.czytelnik = this.model.znalezienieCzytelnika(this.nrCzytelnika);
49.        // jeśli czytelnik istnieje i brak książek wypożyczonych przez czytelnika
50.        if (this.czytelnik.length()>0 && this.wypożyczoneKsiążki.size()==0) {
51.            Widok.pokaż(this.getClass().getCanonicalName(), "WypisanieCzytelnikaZBiblioteki", true,
52.                        "Rozpoczęto wypisywanie czytelnika: " + csv.opiszCzytelnika(this.czytelnik) + ".");
53.            // usunięcie danych osobowych czytelnika - przekazanie obsługi tego zadania klasie Model
54.            this.model.usunięcieDanychOsobowychCzytelnika(this.nrCzytelnika);
55.            // usunięcie czytelnika - przekazanie obsługi tego zadania klasie Model
56.            this.model.usunięcieCzytelnika(this.nrCzytelnika);
57.            // zarejestrowanie usunięcia czytelnika - przekazanie obsługi tego zadania klasie Model
58.            this.model.zarejestrowanieZdarzenia("Wypisano czytelnika nr " + this.nrCzytelnika + ".");
59.            Widok.pokaż(this.getClass().getCanonicalName(), "WypisanieCzytelnikaZBiblioteki", true, "Wypisano czytelnika: " + this.nrCzytelnika + ".");
60.        }
61.        // w przeciwnym razie
62.        else {
63.            // poinformowanie o niemożności wypisania czytelnika
64.            Widok.pokaż(this.getClass().getCanonicalName(), "WypisanieCzytelnikaZBiblioteki", false, "Nie wypisano czytelnika: " + this.nrCzytelnika + ".");
65.            PrzekazanieInformacjiUżytkownikowi.przekazanieInformacji("X" + "Nie wypisano czytelnika: " + this.nrCzytelnika + ".");
66.        }

```

```

67.    /**
68.     * Czy zatwierdzić wypisanie czytelnika.
69.     * @return true, jeśli zatwierdzić
70.    */
71.    private boolean zatwierdzenieWypisaniaCzytelnika() {
72.        // symulacja działania użytkownika: odpowiedź na pytanie o zatwierdzenie wypisania czytelnika
73.        boolean zatwierdzono = true; // zatwierdzono
74.        if (zatwierdzono)
75.            Widok.pokaż(this.getClass().getCanonicalName(), "zatwierdzenieWypisaniaCzytelnika", true,
76.                        "Kierownik biblioteki zatwierdził wypisanie czytelnika nr " + this.nrCzytelnika + " z biblioteki.");
77.        else
78.            Widok.pokaż(this.getClass().getCanonicalName(), "zatwierdzenieWypisaniaCzytelnika", false,
79.                        "Kierownik biblioteki nie zatwierdził wypisania czytelnika nr " + this.nrCzytelnika + " z biblioteki.");
80.        return zatwierdzono;
81.    }
82.
83.    /**
84.     * Znalezienie numeru książki na podstawie jej csv.
85.     * @param książka csv książki
86.     * @return numer książki
87.    */
88.    private int podanieNumeruKsiążki(String książka) {
89.        CSV csv = new CSV();
90.        return csv.znajdźNrKsiążki(książka);
91.    }

```

Kontroler.ZakończenieWypożyczeniaKsiążki.java

```

1. package Kontroler;
2. import Komunikacja.CSV;
3. import Komunikacja.Widok;
4. import Model.ILogic;
5.
6. public class ZakończenieWypożyczeniaKsiążki {
7.     private ILogic model;
8.     private int nrKsiążki;
9.     private IStrategiaZakończeniaWypożyczeniaKsiążki strategiaZakończeniaWypożyczeniaKsiążki;
10.
11.    /**
12.     * Realizacja PU Zakończenie wypożyczenia książki, inicjowana przez Bibliotekarza.
13.     * @param model fasada Modelu
14.    */
15.    public ZakończenieWypożyczeniaKsiążki(ILogic model) {
16.        this.model = model;
17.    }
18.
19.    /**
20.     * Realizacja PU Zakończenie wypożyczenia książki, inicjowana przez PU Wypisanie czytelnika z biblioteki.
21.     * @param model fasada Modelu
22.     * @param nrKsiążki numer książki
23.    */
24.    public ZakończenieWypożyczeniaKsiążki(ILogic model, int nrKsiążki) {
25.        this.model = model;
26.        this.nrKsiążki = nrKsiążki;
27.        // wybranie opcji przez Bibliotekarza
28.        this.wybórOpcji();
29.        this.strategiaZakończeniaWypożyczeniaKsiążki.zakończeniePrzyjęciaKsiążki(this.nrKsiążki);
30.    }
31.
32.    /**
33.     * Wybór przez Bibliotekarza, czy przyjąć zwrot książki (niezgubiona), czy usunąć ją z inwentarza (zgubiona).
34.    */
35.    private void wybórOpcji() {
36.        // symulacja działania użytkownika: przyjąć niezgubioną książkę, usunąć zgubioną książkę
37.        CSV csv = new CSV();
38.        // jeśli to zgubiona książka
39.        if (csv.znajdźDatęZgubieniaKsiążki(this.model.znalezienieKsiążki(this.nrKsiążki)).length() > 0) {
40.            // wybór strategii zakończenia wypożyczenia książki - usunięcie książki z inwentarza
41.            this.strategiaZakończeniaWypożyczeniaKsiążki = new UsunięcieKsiążkiZInwentarza(this.model);
42.            Widok.pokaż(this.getClass().getCanonicalName(), "wybórOpcji", true, "Kierownik biblioteki wybrał usunięcie książki nr " + this.nrKsiążki + ".");
43.        }
44.        else {
45.            // wybór strategii zakończenia wypożyczenia książki - przyjęcie zwrotu książki
46.            this.strategiaZakończeniaWypożyczeniaKsiążki = new PrzyjęcieZwrotuKsiążki(this.model);
47.            Widok.pokaż(this.getClass().getCanonicalName(), "wybórOpcji", true, "Kierownik biblioteki wybrał przyjęcie zwrotu książki nr " + this.nrKsiążki + ".");
48.        }
49.    }
50.
51.    /**
52.     * Wprowadzenie numeru książki przez Bibliotekarza.
53.    */
54.    private void wprowadzenieNumeruKsiążki() {
55.        throw new UnsupportedOperationException();
56.    }
57. }

```

Klasy pakietu *Model* zaimplementowano w zakresie umożliwiającym wykonanie przypadku użycia PU08. Wypisanie czytelnika z biblioteki.

Akcje operacji na bazie danych i jej zawartość zostały zastąpione sztywnym kodem symulującym ich wynik.

Model.Autor.java

```
1. package Model;
2. import Komunikacja.CSV;
3.
4. public class Autor {
5.     public String imię;
6.     public String nazwisko;
7.
8.     /**
9.      * Model autora.
10.     * @param imię      imię autora książki
11.     * @param nazwisko  nazwisko autora książki
12.    */
13.    public Autor(String imię, String nazwisko) {
14.        this.imię = imię;
15.        this.nazwisko = nazwisko;
16.    }
17.
18.    /**
19.     * Utworzenie csv autora.
20.     * @return  csv autora
21.    */
22.    public String opisz() {
23.        CSV csv = new CSV();
24.        return csv.utwórzCsvAutora(this.imię, this.nazwisko);
25.    }
26. }
```

Model.Czytelnik.java

```
1. package Model;
2. import Komunikacja.CSV;
3.
4. public class Czytelnik {
5.     public int nr;
6.     public String imię;
7.     public String nazwisko;
8.
9.     /**
10.      * Model czytelnika.
11.      * @param nr      numer czytelnika
12.      * @param imię    imię czytelnika
13.      * @param nazwisko nazwisko czytelnika
14.    */
15.    public Czytelnik(int nr, String imię, String nazwisko) {
16.        this.nr = nr;
17.        this.imię = imię;
18.        this.nazwisko = nazwisko;
19.    }
20.
21.    /**
22.     * Utworzenie csv czytelnika.
23.     * @return  csv czytelnika
24.    */
25.    public String opisz() {
26.        CSV csv = new CSV();
27.        return csv.utwórzCsvCzytelnika(this.nr, this.imię, this.nazwisko);
28.    }
29. }
```

Model.DAO.java

```
1. package Model;
2. import java.util.ArrayList;
3. import java.util.HashMap;
4. import java.util.Map;
5. import Komunikacja.CSV;
6. import Komunikacja.Widok;
7.
8. /**
9.  * Symulacja dostępu do bazy danych, która "trwale" dane przechowuje zserializowane w postaci csv.
10. *
11. * W stanie początkowym jest już 6 czytelników i 7 książek.
12. * Ponowne uruchomienie programu przywraca stan początkowy bazy.
13. */
14. public class DAO implements IDAO {
15.     /**
16.      * Opisy czytelników (nr;imię;nazwisko).
17.      */
18.     private Map<Integer, String> bazaCzytelników = new HashMap<Integer, String>();
19.     /**
20.      * Opisy książek (nrKsiążki;tytuł(;imięAutora,nazwiskoAutora)*(;dataWypożyczenia;nrCzytelnika)?(;dataZgubienia)?).
```

```

21. */
22. private Map<Integer, String> bazaCzytelników = new HashMap<Integer, String>();
23. // inkrementowane indeksy czytelników i książek (niegdy nie zerowane)
24. private int ostatniNrCzytelnika;
25. private int ostatniNrKsiążki;
26.
27. public DAO() {
28.     // załadowanie bazy czytelników przykładowymi danymi
29.     this.bazaCzytelników.put(1, "1;Adam;Orzeszko");
30.     this.bazaCzytelników.put(2, "2;Stefania;Michalska");
31.     this.bazaCzytelników.put(3, "3;Agnieszka;Nowacka");
32.     this.bazaCzytelników.put(4, "4;Jan;Jankowski");
33.     this.bazaCzytelników.put(5, "5;Marek;Sielski");
34.     this.bazaCzytelników.put(6, "6;Barbara;Głodna");
35.     this.ostatniNrCzytelnika = 6;
36.     // załadowanie bazy książek przykładowymi danymi
37.     this.bazaKsiążek.put(1, "1;Niezwycojony;Stanisław,Lem"); // dostępna
38.     this.bazaKsiążek.put(2, "2;Ida sierpiowa;Małgorzata,Musierowicz;20250215;3"); // wypożyczona
39.     this.bazaKsiążek.put(3, "3;Nad Niemnem;Eliza Orzeszkowa;20210629;3;20211201"); // wypożyczona i zgubiona
40.     this.bazaKsiążek.put(4, "4;Popioly;Stefan,Żeromski"); // dostępna
41.     this.bazaKsiążek.put(5, "5;Paradyżja;Janusz,Zajdel;20251106;2"); // wypożyczona
42.     this.bazaKsiążek.put(6, "6;Ben Hur;Lewis,Wallace;20111231"); // zgubiona
43.     this.bazaKsiążek.put(7, "7;Teatrzyk Zielona Gęś;Konstanty Ildefons, Gałczyński"); // dostępna
44.     this.ostatniNrKsiążki = 7;
45. }
46.
47. public void dodajWpisDoRejestruZdarzeń(String zdarzenie) {
48.     Widok.pokaż(this.getClass().getCanonicalName(), "dodajWpisDoRejestruZdarzeń", true, "Do bazy danych dodano zdarzenie: " + zdarzenie);
49. }
50.
51. public String znajdźCzytelnika(int nrCzytelnika) {
52.     if (this.bazaCzytelników.containsKey(nrCzytelnika)) {
53.         String opis = this.bazaCzytelników.get(nrCzytelnika);
54.         CSV csv = new CSV();
55.         Widok.pokaż(this.getClass().getCanonicalName(), "znajdźCzytelnika", true, "W bazie danych znaleziono: " + csv.opiszCzytelnika(opis) + ".");
56.         return opis;
57.     } else {
58.         Widok.pokaż(this.getClass().getCanonicalName(), "znajdźCzytelnika", true, "W bazie danych nie ma czytelnika nr " + nrCzytelnika + ".");
59.         return Integer.toString(nrCzytelnika);
60.     }
61. }
62.
63. public int dodajCzytelnika(String czytelnik) {
64.     CSV csv = new CSV();
65.     String imię = csv.znajdźImięCzytelnika(czytelnik);
66.     String nazwisko = csv.znajdźNazwiskoCzytelnika(czytelnik);
67.     this.ostatniNrCzytelnika += 1;
68.     String opis = csv.utwórzCsvCzytelnika(this.ostatniNrCzytelnika, imię, nazwisko);
69.     this.bazaCzytelników.put(this.ostatniNrCzytelnika, opis);
70.     Widok.pokaż(this.getClass().getCanonicalName(), "dodajCzytelnika", true, "Do bazy danych dodano: " + csv.opiszCzytelnika(opis) + ".");
71.     return this.ostatniNrCzytelnika;
72. }
73.
74. public void edytujCzytelnika(String czytelnik) {
75.     CSV csv = new CSV();
76.     int nr = csv.znajdźNrCzytelnika(czytelnik);
77.     if (this.bazaCzytelników.containsKey(nr)) {
78.         this.bazaCzytelników.replace(nr, czytelnik);
79.         Widok.pokaż(this.getClass().getCanonicalName(), "edytujCzytelnika", true,
80.                     "W bazie danych zmieniono: " + csv.opiszCzytelnika(this.bazaCzytelników.get(nr)) + ".");
81.     } else {
82.         Widok.pokaż(this.getClass().getCanonicalName(), "edytujCzytelnika", false, "W bazie danych nie ma czytelnika nr " + nr + ".");
83.     }
84.
85. public void usuńCzytelnika(int nrCzytelnika) {
86.     if (this.bazaCzytelników.containsKey(nrCzytelnika)) {
87.         CSV csv = new CSV();
88.         String opis = csv.opiszCzytelnika(this.bazaCzytelników.get(nrCzytelnika));
89.         this.bazaCzytelników.remove(nrCzytelnika);
90.         Widok.pokaż(this.getClass().getCanonicalName(), "usuńCzytelnika", true, "Z bazy danych usunięto: " + opis + ".");
91.     } else {
92.         Widok.pokaż(this.getClass().getCanonicalName(), "usuńCzytelnika", false, "W bazie danych nie ma czytelnika nr " + nrCzytelnika + ".");
93.     }
94. }
95.
96. public String znajdźKsiążkę(int nrKsiążki) {
97.     if (this.bazaKsiążek.containsKey(nrKsiążki)) {
98.         String opis = this.bazaKsiążek.get(nrKsiążki);
99.         CSV csv = new CSV();
100.        Widok.pokaż(this.getClass().getCanonicalName(), "znajdźKsiążkę", true, "W bazie danych znaleziono: " + csv.opiszKsiążkę(opis) + ".");
101.        return opis;
102.    } else {
103.        Widok.pokaż(this.getClass().getCanonicalName(), "znajdźKsiążkę", false, "W bazie danych nie ma książki nr " + nrKsiążki + ".");
104.        return Integer.toString(nrKsiążki);
105.    }
106. }
107.
108. public ArrayList<String> znajdźKsiążkiCzytelnika(int nrCzytelnika) {
109.     ArrayList<String> książkiCzytelnika = new ArrayList<String>();
110.     if (this.bazaCzytelników.containsKey(nrCzytelnika)) {

```

```

111. CSV csv = new CSV();
112.     for (String książka : this.bazaKsiążek.values()) {
113.         if (nrCzytelnika == csv.znajdźNrCzytelnikaKsiążki(książka)) {
114.             książkiCzytelnika.add(książka);
115.             Widok.pokaż(this.getClass().getCanonicalName(), "znajdźKsiążkiCzytelnika", true, "W bazie danych znaleziono: " + csv.opiszKsiążkę(książka) + ".");
116.         }
117.     }
118.     if (książkiCzytelnika.isEmpty())
119.         Widok.pokaż(this.getClass().getCanonicalName(), "znajdźKsiążkiCzytelnika", false, "W bazie danych nie ma książek czytelnika nr " + nrCzytelnika + ".");
120.
121. } else {
122.     Widok.pokaż(this.getClass().getCanonicalName(), "znajdźKsiążkiCzytelnika", false, "W bazie danych nie ma czytelnika nr " + nrCzytelnika + " ");
123. }
124. return książkiCzytelnika;
125. }
126.
127. public int dodajKsiążkę(String książka) {
128.     CSV csv = new CSV();
129.     String tytuł = csv.znajdźTytułKsiążki(książka);
130.     ArrayList<String> autorzy = csv.znajdźAutorówKsiążki(książka);
131.     ArrayList<String> imiona = new ArrayList<String>();
132.     ArrayList<String> nazwiska = new ArrayList<String>();
133.     for (String autor: autorzy) {
134.         imiona.add(csv.znajdźImięAutora(autor));
135.         nazwiska.add(csv.znajdźNazwiskoAutora(autor));
136.     }
137.     csv = new CSV();
138.     this.ostatniNrKsiążki += 1;
139.     String opis = csv.utwórzCsvKsiążki(this.ostatniNrKsiążki, tytuł, imiona, nazwiska);
140.     this.bazaKsiążek.put(this.ostatniNrKsiążki, opis);
141.     Widok.pokaż(this.getClass().getCanonicalName(), "dodajKsiążkę", true, "Do bazy danych dodano: " + csv.opiszKsiążkę(opis) + " ");
142.     return this.ostatniNrKsiążki;
143. }
144.
145. public void edytujKsiążkę(String książka) {
146.     CSV csv = new CSV();
147.     int nr = csv.znajdźNrKsiążki(książka);
148.     if (this.bazaKsiążek.containsKey(nr)) {
149.         this.bazaKsiążek.replace(nr, książka);
150.         Widok.pokaż(this.getClass().getCanonicalName(), "edytujKsiążkę", true, "W bazie danych zmieniono: " + csv.opiszKsiążkę(this.bazaKsiążek.get(nr)) + ".");
151.     } else {
152.         Widok.pokaż(this.getClass().getCanonicalName(), "edytujKsiążkę", false, "W bazie danych nie ma książki nr " + nr + ".");
153.     }
154. }
155.
156. public void usuńKsiążkę(int nrKsiążki) {
157.     if (this.bazaKsiążek.containsKey(nrKsiążki)) {
158.         CSV csv = new CSV();
159.         String opis = csv.opiszKsiążkę(this.bazaKsiążek.get(nrKsiążki));
160.         this.bazaKsiążek.remove(nrKsiążki);
161.         Widok.pokaż(this.getClass().getCanonicalName(), "usuńKsiążkę", true, "Z bazy danych usunięto: " + opis + ".");
162.     } else {
163.         Widok.pokaż(this.getClass().getCanonicalName(), "usuńKsiążkę", false, "W bazie danych nie ma książki nr " + nrKsiążki + ".");
164.     }
165. }
166.
167. public void pokażBazęCzytelników() {
168.     CSV csv = new CSV();
169.     ArrayList<String> czytelnicy = new ArrayList<String>();
170.     for (String czytelnik : this.bazaCzytelników.values()) {
171.         czytelnicy.add(csv.opiszCzytelnika(czytelnik));
172.     }
173.     Widok.pokaż(this.getClass().getCanonicalName(), "pokażBazęCzytelników", true, czytelnicy);
174. }
175.
176. public void pokażBazęKsiążek() {
177.     CSV csv = new CSV();
178.     ArrayList<String> książki = new ArrayList<String>();
179.     for (String książka : this.bazaKsiążek.values()) {
180.         książki.add(csv.opiszKsiążkę(książka));
181.     }
182.     Widok.pokaż(this.getClass().getCanonicalName(), "pokażBazęKsiążek", true, książki);
183. }
184. }

```

Model.FabrykaKsiążki.java

```

1. package Model;
2. import java.util.ArrayList;
3. import Komunikacja.CSV;
4. import Komunikacja.Widok;
5.
6. public class FabrykaKsiążki implements IFabrykaKsiążki {
7.
8.     /**
9.      * Fabryka dostępnej książki.
10.     */
11.    public FabrykaKsiążki() {
12.    }
13.

```

```

14. /**
15. * Utworzenie dostępnej książki na podstawie jej opisu (pomijając ewentualne informacje o jej wypożyczeniu i zgubieniu).
16. * @param opisCsv csv książka
17. * @return dostępna książka
18. */
19. public IKsiążka utwórzKsiążkę(String opisCsv) {
20.     CSV csv = new CSV();
21.     ArrayList<Autor> autorzy = new ArrayList<Autor>();
22.     ArrayList<String> opisyAutorów = csv.znajdźAutorówKsiążki(opisCsv);
23.     for (String autor: opisyAutorów)
24.         autorzy.add(new Autor(csv.znajdźImięAutora(autor), csv.znajdźNazwiskoAutora(autor)));
25.     Książka nowaKsiążka = new Książka(csv.znajdźNrKsiążki(opisCsv), csv.znajdźTytułKsiążki(opisCsv), autorzy);
26.     Widok.pokaż(this.getClass().getCanonicalName(), "utwórzKsiążkę", true, "Utworzono: " + csv.opiszKsiążkę(nowaKsiążka.opisz()) + ".");
27.     return nowaKsiążka;
28. }
29. }
```

Model.FabrykaKsiążkiWypożyczonej.java

```

1. package Model;
2. import Komunikacja.CSV;
3. import Komunikacja.Widok;
4.
5. public class FabrykaKsiążkiWypożyczonej implements IFabrykaKsiążki {
6.     private IKsiążka książka;
7.     private Czytelnik czytelnik;
8.
9. /**
10. * Fabryka wypożyczonej książki.
11. * @param książka książka do opakowania wypożyczeniem (jeśli opis nie definiuje książki)
12. * @param czytelnik wypożyczający czytelnik
13. */
14. public FabrykaKsiążkiWypożyczonej(IKsiążka książka, Czytelnik czytelnik) {
15.     this.książka = książka;
16.     this.czytelnik = czytelnik;
17. }
18.
19. /**
20. * Utworzenie wypożyczonej książki lub udekorowanie gotowej książki w jej wypożyczenie na podstawie jej opisu (pomija ewentualne informacje o jej zgubieniu).
21. * Fabryka musi mieć wypożyczającego czytelnika.
22. * @param opisCsv csv książki lub data wypożyczenia
23. * @return wypożyczona książka
24. */
25. public IKsiążka utwórzKsiążkę(String opisCsv) {
26.     CSV csv = new CSV();
27.     IKsiążka nowaKsiążka = null;
28.     // opakowanie gotowej książki w wypożyczenie, jeśli opis zawiera tylko datę wypożyczenia RRRRMMDD
29.     if(csv.czyData(opisCsv) && this.książka != null && this.czytelnik != null) {
30.         nowaKsiążka = new WypożyczonaKsiążka(this.książka, opisCsv, this.czytelnik);
31.         Widok.pokaż(this.getClass().getCanonicalName(), "utwórzKsiążkę", true, "Wypożyczono książkę: " + csv.opiszKsiążkę(nowaKsiążka.opisz()) + ".");
32.     }
33.     // utworzenie książki i opakowanie jej w wypożyczenie
34.     else if (lcsv.czyData(opisCsv) && this.czytelnik != null) {
35.         FabrykaKsiążki fabryka = new FabrykaKsiążki();
36.         nowaKsiążka = new WypożyczonaKsiążka(fabryka.utwórzKsiążkę(opisCsv), csv.znajdźDatęWypożyczeniaKsiążki(opisCsv), this.czytelnik);
37.         Widok.pokaż(this.getClass().getCanonicalName(), "utwórzKsiążkę", true, "Utworzono i wypożyczono: " + csv.opiszKsiążkę(nowaKsiążka.opisz()) + ".");
38.     }
39.     return nowaKsiążka;
40. }
41. }
```

Model.FabrykaKsiążkiZgubionej.java

```

1. package Model;
2. import Komunikacja.CSV;
3. import Komunikacja.Widok;
4.
5. public class FabrykaKsiążkiZgubionej implements IFabrykaKsiążki {
6.     private IKsiążka książka;
7.     private Czytelnik czytelnik;
8.
9. /**
10. * Fabryka zgubionej książki.
11. * @param książka książka do opakowania zgubieniem (jeśli opis nie definiuje książki)
12. * @param czytelnik wypożyczający czytelnik (jeśli książka jest wypożyczona)
13. */
14. public FabrykaKsiążkiZgubionej(IKsiążka książka, Czytelnik czytelnik) {
15.     this.książka = książka;
16.     this.czytelnik = czytelnik;
17. }
18.
19. /**
20. * Utworzenie zgubionej książki lub udekorowanie gotowej książki w jej zgubienie na podstawie jej opisu.
21. * Jeśli tworzona książka jest wypożyczona, to fabryka musi mieć wypożyczającego czytelnika.
22. * @param opisCsv opis książki lub data zgubienia
23. * @return zgubiona książka
24. */
25. public IKsiążka utwórzKsiążkę(String opisCsv) {
26.     CSV csv = new CSV();
27.     IKsiążka nowaKsiążka = null;
28.     // opakowanie gotowej książki w zgubienie, jeśli opis zawiera tylko datę zgubienia RRRRMMDD
```

```

29.     if (csv.czyData(opisCsv) && this.książka != null) {
30.         nowaKsiążka = new ZgubionaKsiążka(this.książka, opisCsv);
31.         Widok.pokaż(this.getClass().getCanonicalName(), "utwórzKsiążkę", true, "Zgubiono książkę: " + csv.opiszKsiążkę(nowaKsiążka.opisz()) + ".");
32.     }
33.     // utworzenie wypożyczonej książki i opakowanie jej w zgubienie
34.     else if (!csv.czyData(opisCsv) && lcsv.znajdźDatęWypożyczeniaKsiążki(opisCsv).isEmpty() && this.czYTELNik != null) {
35.         FabrykaKsiążkiWypożyczonej fabryka = new FabrykaKsiążkiWypożyczonej(this.książka, this.czYTELNik);
36.         nowaKsiążka = new ZgubionaKsiążka(fabryka.utwórzKsiążkę(opisCsv), csv.znajdźDatęZgubieniaKsiążki(opisCsv));
37.         Widok.pokaż(this.getClass().getCanonicalName(), "utwórzKsiążkę", true, "Utworzono i zgubiono: " + csv.opiszKsiążkę(nowaKsiążka.opisz()) + ".");
38.     }
39.     // utworzenie dostępnej książki i opakowanie jej w zgubienie
40.     else if (!lcsv.czyData(opisCsv) && csv.znajdźDatęWypożyczeniaKsiążki(opisCsv).isEmpty()) {
41.         FabrykaKsiążki fabryka = new FabrykaKsiążki();
42.         nowaKsiążka = new ZgubionaKsiążka(fabryka.utwórzKsiążkę(opisCsv), csv.znajdźDatęZgubieniaKsiążki(opisCsv));
43.         Widok.pokaż(this.getClass().getCanonicalName(), "utwórzKsiążkę", true, "Utworzono i zgubiono: " + csv.opiszKsiążkę(nowaKsiążka.opisz()) + ".");
44.     }
45.     return nowaKsiążka;
46. }

```

Model.IDAO.java

```

1. package Model;
2. import java.util.ArrayList;
3.
4. /**
5. * Interfejs obiektu dostępu do danych trwałych.
6. */
7. public interface IDAO {
8.
9.     /**
10.      * Dodanie wpisu o zdarzeniu.
11.      * @param zdarzenie    treść zdarzenia
12.      */
13.     public void dodajWpisDoRejestruZdarzeń(String zdarzenie);
14.
15.    /**
16.     * Pobranie czytelnika.
17.     * @param nrCzytelnika numer szukanego czytelnika
18.     * @return               csv czytelnika lub jego nr, jeśli go nie znaleziono
19.    */
20.    public String znajdźCzytelnika(int nrCzytelnika);
21.
22.    /**
23.     * Dodanie czytelnika.
24.     * @param czytelnik    csv czytelnika
25.     * @return              nr dodanego czytelnika
26.    */
27.    public int dodajCzytelnika(String czytelnik);
28.
29.    /**
30.     * Zmiana czytelnika (bez zmiany numeru).
31.     * @param czytelnik    csv czytelnika
32.    */
33.    public void edytujCzytelnika(String czytelnik);
34.
35.    /**
36.     * Usunięcie czytelnika.
37.     * @param nrCzytelnika numer czytelnika
38.    */
39.    public void usuńCzytelnika(int nrCzytelnika);
40.
41.    /**
42.     * Pobranie książki.
43.     * @param nrKsiążki   numer książki
44.     * @return             csv książki lub jej numer, jeśli jej nie znaleziono
45.    */
46.    public String znajdźKsiążkę(int nrKsiążki);
47.
48.    /**
49.     * Pobranie książek wypożyczonych przez czytelnika.
50.     * @param nrCzytelnika numer czytelnika
51.     * @return              lista csv książek
52.    */
53.    public ArrayList<String> znajdźKsiążkiCzytelnika(int nrCzytelnika);
54.
55.    /**
56.     * Dodanie książki.
57.     * @param książka     csv książki
58.     * @return             numer dodanej książki
59.    */
60.    public int dodajKsiążkę(String książka);
61.
62.    /**
63.     * Zmiana książki (bez zmiany numeru).
64.     * @param książka     csv książki
65.    */
66.    public void edytujKsiążkę(String książka);
67.

```

```

68.    /**
69.     * Usunięcie książki.
70.     * @param nrKsiążki numer książki
71.     */
72.    public void usuńKsiążkę(int nrKsiążki);
73.
74.    /**
75.     * Pokazanie zawartości bazy czytelników.
76.     */
77.    public void pokażBazęCzytelników();
78.
79.    /**
80.     * Pokazanie zawartości bazy książek.
81.     */
82.    public void pokażBazęKsiążek();
83. }

```

Model.IFabrykaKsiążki.java

```

1. package Model;
2.
3. /**
4.  * Interfejs fabryki książek.
5.  */
6. public interface IFabrykaKsiążki {
7.
8.    /**
9.     * Fabryka książki - utworzenie lub udekorowanie książki na podstawie jej csv.
10.    * @param opisCsv csv książki
11.    * @return utworzona lub opakowana książka
12.   */
13.   public IKsiążka utwórzKsiążkę(String opisCsv);
14. }

```

Model.IKsiążka.java

```

1. package Model;
2. import java.util.ArrayList;
3.
4. /**
5.  * Interfejs książki i jej dekoratorów
6.  */
7. public interface IKsiążka {
8.
9.    /**
10.     * @return numer książki
11.    */
12.   public int dajNr();
13.
14.   /**
15.     * @return tytuł książki
16.    */
17.   public String dajTytuł();
18.
19.   /**
20.     * @return autorzy książki
21.    */
22.   public ArrayList<Autor> dajAutorów();
23.
24.   /**
25.     * Utworzenie opisu książki, uwzględniając też jej udekorowanie w wypożyczenie i zgubienie.
26.     * @return opis książki (nrKsiążki;tytuł;(imięAutora,nazwiskoAutora)*(;dataWypożyczenia;nrCzytelnika)?;(dataZgubienia)?)
27.   */
28.   public String opisz();
29. }

```

Model.IModel.java

```

1. package Model;
2. import java.util.ArrayList;
3.
4. /**
5.  * Interfejs fasady modelu.
6.  */
7. public interface IMODEL {
8.
9.    /**
10.     * Zarejestrowanie zdarzenia.
11.     * @param zdarzenie treść zdarzenia
12.    */
13.   public void zarejestrowanieZdarzenia(String zdarzenie);
14.
15.   /**
16.     * Znalezienie książek wypożyczonych przez czytelnika.
17.     * @param nrCzytelnika nr czytelnika
18.     * @return opisy książek (nrKsiążki;tytuł;(imięAutora,nazwiskoAutora)*)
19.   */
20.   public ArrayList<String> znalezienieWypożyczonychKsiążek(int nrCzytelnika);
21.

```

```

22. /**
23. * Znalezienie czytelnika.
24. * @param nrCzytelnika nr czytelnika
25. * @return opis czytelnika (nr;imię;nazwisko)
26. */
27. public String znalezienieCzytelnika(int nrCzytelnika);
28.
29. /**
30. * Trwałe usunięcie czytelnika.
31. * @param nrCzytelnika nr czytelnika
32. */
33. public void usunięcieCzytelnika(int nrCzytelnika);
34.
35. /**
36. * Usunięcie danych osobowych czytelnika.
37. * @param nrCzytelnika nr czytelnika
38. */
39. public void usunięcieDanychOsobowychCzytelnika(int nrCzytelnika);
40.
41. /**
42. * Znalezienie książki.
43. * @param nrKsiążki nr książki
44. * @return opis książki (nrKsiążki;tytuł(;imięAutora,nazwiskoAutora)*(;dataWypożyczenia;nrCzytelnika)?(;dataZgubienia)?)?
45. */
46. public String znalezienieKsiążki(int nrKsiążki);
47.
48. /**
49. * Zdjęcie z książki jej statusu wypożyczenia i / lub zgubienia, czyli udostępnienie jej do wypożyczenia.
50. * @param nrKsiążki nr książki
51. */
52. public void udostępnienieKsiążkiDoWypożyczenia(int nrKsiążki);
53.
54. /**
55. * Trwałe usunięcie książki.
56. * @param nrKsiążki nr książki
57. */
58. public void usunięcieKsiążki(int nrKsiążki);
59. }

```

Model.Inwentarz.java

```

1. package Model;
2. import java.util.ArrayList;
3. import Komunikacja.CSV;
4. import Komunikacja.Widok;
5.
6. public class Inwentarz {
7.     private IDAO dao;
8.     private ArrayList<Czytelnik> czytelnicy = null;
9.     private ArrayList<IKsiążka> książki = null;
10.
11.    /**
12.     * Inwentarz czytelników i książek.
13.     * @param dao adapter DAO
14.    */
15.    public Inwentarz(IDAO dao) {
16.        this.dao = dao;
17.        this.czytelnicy = new ArrayList<Czytelnik>();
18.        this.książki = new ArrayList<IKsiążka>();
19.    }
20.
21.    /**
22.     * Pobranie z lokalnej listy książek wypożyczonych przez czytelnika po jej aktualizacji przy pomocy DAO.
23.     * @param nrCzytelnika numer czytelnika
24.     * @return lista książek lub pusta lista, jeśli nie ma takich książek
25.    */
26.    public ArrayList<IKsiążka> dajWypożyczoneKsiążkiCzytelnika(int nrCzytelnika) {
27.        ArrayList<IKsiążka> książki = new ArrayList<IKsiążka>();
28.        CSV csv = new CSV();
29.        // usunięcie wypożyczonych książek z lokalnej listy
30.        this.książki.removeIf(k -> csv.znajdźNrCzytelnikaKsiążki(k.opisz()) == nrCzytelnika);
31.        // jeśli książki czytelnika są bazie, to dodanie ich do listy i do wyniku tej operacji
32.        ArrayList<String> książkiZBazy = this.dao.znajdźKsiążkiCzytelnika(nrCzytelnika);
33.        if (książkiZBazy.isEmpty())
34.            Widok.pokaż(this.getClass().getCanonicalName(), "dajWypożyczoneKsiążkiCzytelnika", false, "Nie znaleziono książek czytelnika nr " + nrCzytelnika + ".");
35.        else {
36.            ArrayList<String> wiadomość = new ArrayList<String>();
37.            wiadomość.add("Znaleziono i pobrano książki czytelnika nr " + nrCzytelnika + ":");
38.            for (String książkaZBazy: książkiZBazy) {
39.                IKsiążka książka = this.dajKsiążkę(csv.znajdźNrKsiążki(książkaZBazy));
40.                if (książka != null)
41.                    książki.add(książka);
42.                    wiadomość.add(csv.opiszKsiążkę(książka.opisz()));
43.            }
44.        }
45.        Widok.pokaż(this.getClass().getCanonicalName(), "dajWypożyczoneKsiążkiCzytelnika", true, wiadomość);
46.    }
47.    return książki;
48. }

```

```

50. /**
51. * Pobranie z lokalnej listy czytelnika po jej aktualizacji przy pomocy DAO.
52. * @param nrCzytelnika numer czytelnika
53. * @return czytelnik lub null, jeśli nie ma takiego czytelnika
54. */
55. public Czytelnik dajCzytelnika(int nrCzytelnika) {
56.     Czytelnik czytelnik = null;
57.     CSV csv = new CSV();
58.     // usunięcie czytelnika z lokalnej listy
59.     this.czytelnicy.removeIf(c -> c.nr == nrCzytelnika);
60.     // jeśli czytelnik jest bazie, to dodanie go do listy
61.     String czytelnikZBazy = this.dao.znajdzCzytelnika(nrCzytelnika);
62.     if (!csv.czyNumer(czytelnikZBazy)) {
63.         czytelnik = new Czytelnik(nrCzytelnika, csv.znajdzImięCzytelnika(czytelnikZBazy), csv.znajdzNazwiskoCzytelnika(czytelnikZBazy));
64.         this.czytelnicy.add(czytelnik);
65.         Widok.pokaż(this.getClass().getCanonicalName(), "dajCzytelnika", true, "Znaleziono i pobrano: " + csv.opiszCzytelnika(czytelnik.opisz()) + ".");
66.     }
67.     else
68.         Widok.pokaż(this.getClass().getCanonicalName(), "dajCzytelnika", false, "Nie znaleziono czytelnika nr: " + nrCzytelnika + ".");
69.     return czytelnik;
70. }
71.
72. /**
73. * Usunięcie czytelnika z lokalnej listy i z bazy.
74. * @param nrCzytelnika numer czytelnika
75. */
76. public void usuńCzytelnika(int nrCzytelnika) {
77.     // pobranie czytelnika do usunięcia
78.     Czytelnik czytelnik = this.dajCzytelnika(nrCzytelnika);
79.     // jeśli czytelnik istnieje, to jest usuwany
80.     if (czytelnik != null) {
81.         CSV csv = new CSV();
82.         String opis = csv.opiszCzytelnika(czytelnik.opisz());
83.         this.dao.usuńCzytelnika(nrCzytelnika);
84.         this.czytelnicy.remove(czytelnik);
85.         Widok.pokaż(this.getClass().getCanonicalName(), "usuńCzytelnika", true, "Usunięto: " + opis + ".");
86.     }
87.     else
88.         Widok.pokaż(this.getClass().getCanonicalName(), "usuńCzytelnika", false, "Nie znaleziono czytelnika nr: " + nrCzytelnika + ".");
89. }
90.
91. /**
92. * Usunięcie danych osobowych czytelnika z lokalnej listy i z bazy bez usuwania jego numeru.
93. * @param nrCzytelnika numer czytelnika
94. */
95. public void usuńDaneOsoboweCzytelnika(int nrCzytelnika) {
96.     // pobranie czytelnika do edycji
97.     Czytelnik czytelnik = this.dajCzytelnika(nrCzytelnika);
98.     // jeśli czytelnik istnieje
99.     if (czytelnik != null) {
100.        CSV csv = new CSV();
101.        String opis = csv.opiszCzytelnika(czytelnik.opisz());
102.        // usunięcie imienia i nazwiska czytelnika z bazy
103.        String nowyOpis = csv.utwórzCsvCzytelnika(nrCzytelnika, "usunięto imię", "usunięto nazwisko");
104.        this.dao.edytujCzytelnika(nowyOpis);
105.        // zaktualizowanie listy czytelników
106.        this.czytelnicy.remove(czytelnik);
107.        this.czytelnicy.add(new Czytelnik(nrCzytelnika, "usunięto imię", "usunięto nazwisko"));
108.        Widok.pokaż(this.getClass().getCanonicalName(), "usuńDaneOsoboweCzytelnika", true, "Usunięto dane osobowe: " + opis + ".");
109.    }
110.    else
111.        Widok.pokaż(this.getClass().getCanonicalName(), "usuńDaneOsoboweCzytelnika", false, "Nie znaleziono czytelnika nr: " + nrCzytelnika + ".");
112. }
113.
114. /**
115. * Pobranie z lokalnej listy książki po jej aktualizacji przy pomocy DAO.
116. * @param nrKsiążki numer książki
117. * @return książka lub null, jeśli nie ma takiej książki
118. */
119. public IKsiążka dajKsiążkę(int nrKsiążki) {
120.     IKsiążka książka = null;
121.     CSV csv = new CSV();
122.     // usunięcie książki z lokalnej listy
123.     this.książki.removeIf(k -> k.dajNr() == nrKsiążki);
124.     // jeśli książka jest bazie, to dodanie jej do listy
125.     String książkaZBazy = this.dao.znajdzKsiążkę(nrKsiążki);
126.     if (!csv.czyNumer(książkaZBazy)) {
127.         IFabrykaKsiążki fabryka;
128.         int nrCzytelnika = csv.znajdzNrCzytelnikaKsiążki(książkaZBazy);
129.         Czytelnik czytelnik = this.dajCzytelnika(nrCzytelnika);
130.         boolean czyWypożyczona = nrCzytelnika > 0;
131.         boolean czyZgubiona = csv.znajdzDatęZgubieniaKsiążki(książkaZBazy).length() > 0;
132.         // wybór odpowiedniej fabryki książki - jeśli książka jest wypożyczona i zgubiona
133.         if (czyWypożyczona && czyZgubiona)
134.             fabryka = new FabrykaKsiążkiZgubionej(null, czytelnik);
135.         // wybór odpowiedniej fabryki książki - jeśli książka jest wypożyczona
136.         else if (czyWypożyczona)
137.             fabryka = new FabrykaKsiążkiWypożyczonej(null, czytelnik);
138.         // wybór odpowiedniej fabryki książki - jeśli książka jest zgubiona
139.         else if (czyZgubiona)
140.             fabryka = new FabrykaKsiążkiZgubionej(null, null);
141.     }
142. }

```

```

143.     }
144.     // wybór odpowiedniej fabryki książki - jeśli książka jest dostępna
145.     else {
146.         fabryka = new FabrykaKsiążki();
147.     }
148.     // utworzenie nowej książki i dodanie jej do listy
149.     książka = fabryka.utwórzKsiążkę(książkaZBazy);
150.     this.książki.add(książka);
151.     Widok.pokaż(this.getClass().getCanonicalName(), "dajKsiążkę", true, "Znaleziono i pobrano: " + csv.opiszKsiążkę(książka.opisz()) + ".");
152. }
153. else
154.     Widok.pokaż(this.getClass().getCanonicalName(), "dajKsiążkę", false, "Nie znaleziono książki nr: " + nrKsiążki + ".");
155. return książka;
156. }

157. /**
158. * Usunięcie dekoracji książki, czyli jej wypożyczenia i / lub zgubienia.
159. * @param nrKsiążki numer książki
160. */
161.
162. public void udostępnijKsiążkę(int nrKsiążki) {
163.     // pobranie książki do udostępnienia
164.     IKsiążka książka = this.dajKsiążkę(nrKsiążki);
165.     // jeśli książka istnieje
166.     if (książka != null) {
167.         // usunięcie książki
168.         this.książki.remove(książka);
169.         // utworzenie nowej, dostępnej książki na podstawie jej opisu
170.         FabrykaKsiążki fabryka = new FabrykaKsiążki();
171.         CSV csv = new CSV();
172.         książka = fabryka.utwórzKsiążkę(książka.opisz());
173.         this.książki.add(książka);
174.         // zaktualizowanie książki w bazie
175.         this.dao.edytujKsiążkę(książka.opisz());
176.         String opisanie = csv.opiszKsiążkę(książka.opisz());
177.         Widok.pokaż(this.getClass().getCanonicalName(), "udostępnijKsiążkę", true, "Zmieniono książkę na dostępną: " + opisanie + ".");
178.     }
179.     else
180.         Widok.pokaż(this.getClass().getCanonicalName(), "udostępnijKsiążkę", false, "Nie znaleziono książki nr: " + nrKsiążki + ".");
181. }

182. /**
183. * Usunięcie książki z lokalnej listy i z bazy.
184. * @param nrKsiążki numer książki
185. */
186.
187. public void usuńKsiążkę(int nrKsiążki) {
188.     // pobranie książki do usunięcia
189.     IKsiążka książka = this.dajKsiążkę(nrKsiążki);
190.     // jeśli książka istnieje, to jest usuwana
191.     if (książka != null) {
192.         CSV csv = new CSV();
193.         String opis = csv.opiszKsiążkę(książka.opisz());
194.         this.dao.usuńKsiążkę(nrKsiążki);
195.         this.książki.remove(książka);
196.         Widok.pokaż(this.getClass().getCanonicalName(), "usuńKsiążkę", true, "Usunięto: " + opis + ".");
197.     }
198.     else
199.         Widok.pokaż(this.getClass().getCanonicalName(), "usuńKsiążkę", false, "Nie znaleziono książki nr: " + nrKsiążki + ".");
200. }
201. }
202. 
```

Model.KartaKsiążki.java

```

1. package Model;
2. import java.util.ArrayList;
3.
4. /**
5. * Abstrakcja dekoratora książki.
6. */
7. public abstract class KartaKsiążki implements IKsiążka {
8.     protected IKsiążka książka;
9.
10.    /**
11.     * @param książka książka do udekorowania
12.    */
13.    public KartaKsiążki(IKsiążka książka) {
14.        this.książka = książka;
15.    }
16.
17.    public int dajNr() {
18.        return this.książka.dajNr();
19.    }
20.
21.    public String dajTytuł() {
22.        return this.książka.dajTytuł();
23.    }
24.
25.    public ArrayList<Autor> dajAutorów() {
26.        return this.książka.dajAutorów();
27.    }

```

```
28.  
29.     abstract public String opisz();  
30. }
```

Model.Książka.java

```
1. package Model;  
2. import java.util.ArrayList;  
3. import Komunikacja.CSV;  
4.  
5. public class Książka implements IKsiążka {  
6.     private int nr;  
7.     private String tytuł;  
8.     private ArrayList<Autor> autorzy;  
9.  
10.    /**  
11.     * Model książki.  
12.     * @param nr      numer książki  
13.     * @param tytuł   tytuł książki  
14.     * @param autorzy lista autorów książki  
15.    */  
16.    public Książka(int nr, String tytuł, ArrayList<Autor> autorzy) {  
17.        this.nr = nr;  
18.        this. tytuł = tytuł;  
19.        this.autorzy = autorzy;  
20.    }  
21.  
22.    public int dajNr() {  
23.        return this.nr;  
24.    }  
25.  
26.    public String dajTytuł() {  
27.        return this. tytuł;  
28.    }  
29.  
30.    public ArrayList<Autor> dajAutorów() {  
31.        return this.autorzy;  
32.    }  
33.  
34.    /**  
35.     * Utworzenie csv książki.  
36.     * @return  csv książki  
37.    */  
38.    public String opisz() {  
39.        CSV csv = new CSV();  
40.        // utworzenie listy imion autorów i listy nazwisk autorów  
41.        ArrayList<String> imiona = new ArrayList<String>();  
42.        ArrayList<String> nazwiska = new ArrayList<String>();  
43.        for (Autor autor: this.autorzy) {  
44.            imiona.add(autor.imię);  
45.            nazwiska.add(autor.nazwisko);  
46.        }  
47.        // zwrócenie csv książki  
48.        return csv.utwórzCsvKsiążki(this.nr, this. tytuł, imiona, nazwiska);  
49.    }  
50. }
```

Model.Model.java

```
1. package Model;  
2. import java.util.ArrayList;  
3. import Komunikacja.Widok;  
4.  
5. public class Model implements IModel {  
6.     private Inwentarz inwentarz;  
7.     private IDAO dao;  
8.  
9.     /**  
10.      * Fasada Modelu.  
11.      * @param inwentarz  inwentarz czytelników i książek  
12.      * @param dao        adapter DAO  
13.     */  
14.     public Model(Inwentarz inwentarz, IDAO dao) {  
15.         this.inwentarz = inwentarz;  
16.         this.dao = dao;  
17.     }  
18.  
19.     public void zarejestrowanieZdarzenia(String zdarzenie) {  
20.         this.dao.dodajWpisDoRejestruZdarzeń(zdarzenie);  
21.     }  
22.  
23.     public ArrayList<String> znalezienieWypożyczonychKsiążek(int nrCzytelnika) {  
24.         Widok.pokaż(this.getClass().getCanonicalName(), "znalezienieWypożyczonychKsiążek", true,  
25.                     "Rozpoczęto szukanie książek wypożyczonych przez czytelnika nr " + nrCzytelnika + ".");  
26.         // przekazanie obsługi tego zadania klasie Inwentarz  
27.         ArrayList<Książka> książki = this.inwentarz.dajWypożyczoneKsiążkiCzytelnika(nrCzytelnika);  
28.         ArrayList<String> opisyKsiążek = new ArrayList<String>();  
29.         for (IKsiążka książka: książki) {  
30.             opisyKsiążek.add(książka.opisz());
```

```

30.    }
31.    Widok.pokaż(this.getClass().getCanonicalName(), "znalezienieWypożyczonychKsiążek", true,
32.        "Zakończono szukanie książek wypożyczonych przez czytelnika nr " + nrCzytelnika + ".");
33.    return opisyKsiążek;
34. }
35. public String znalezienieCzytelnika(int nrCzytelnika) {
36.     String opis = "";
37.     Czytelnik czytelnik = this.inwentarz.dajCzytelnika(nrCzytelnika);
38.     if (czytelnik != null)
39.         opis = czytelnik.opisz();
40.     return opis;
41. }
42.
43. public void usunięcieCzytelnika(int nrCzytelnika) {
44.     Widok.pokaż(this.getClass().getCanonicalName(), "usunięcieCzytelnika", true, "Rozpoczęto usuwanie czytelnika nr " + nrCzytelnika + ".");
45.     // przekazanie obsługi tego zadania klasie Inwentarz
46.     this.inwentarz.usuńCzytelnika(nrCzytelnika);
47.     Widok.pokaż(this.getClass().getCanonicalName(), "usunięcieCzytelnika", true, "Zakończono usuwanie czytelnika nr " + nrCzytelnika + ".");
48. }
49.
50. public void usunięcieDanychOsobowychCzytelnika(int nrCzytelnika) {
51.     // Jeśli jest taki czytelnik:
52.     if (this.inwentarz.dajCzytelnika(nrCzytelnika) != null) {
53.         this.inwentarz.usuńDaneOsoboweCzytelnika(nrCzytelnika);
54.         Widok.pokaż(this.getClass().getCanonicalName(), "usunięcieDanychOsobowychCzytelnika", true, "Usunięto dane osobowe czytelnika nr " + nrCzytelnika + ".");
55.     }
56.     else
57.         Widok.pokaż(this.getClass().getCanonicalName(), "usunięcieDanychOsobowychCzytelnika", false, "Czytelnik nr " + nrCzytelnika + " nie istnieje.");
58. }
59.
60. public String znalezienieKsiążki(int nrKsiążki) {
61.     String opis = "";
62.     IKsiążka książka = this.inwentarz.dajKsiążkę(nrKsiążki);
63.     if (książka != null)
64.         opis = książka.opisz();
65.     return opis;
66. }
67.
68. public void udostępnienieKsiążkiDoWypożyczenia(int nrKsiążki) {
69.     Widok.pokaż(this.getClass().getCanonicalName(), "udostępnienieKsiążkiDoWypożyczenia", true, "Rozpoczęto udostępnienie książki nr " + nrKsiążki + ".");
70.     // przekazanie obsługi tego zadania klasie Inwentarz
71.     this.inwentarz.udostępnijKsiążkę(nrKsiążki);
72.     Widok.pokaż(this.getClass().getCanonicalName(), "udostępnienieKsiążkiDoWypożyczenia", true, "Zakończono udostępnienie książki nr " + nrKsiążki + ".");
73. }
74.
75. public void usunięcieKsiążki(int nrKsiążki) {
76.     Widok.pokaż(this.getClass().getCanonicalName(), "usunięcieKsiążki", true, "Rozpoczęto usuwanie książki nr " + nrKsiążki + ".");
77.     // przekazanie obsługi tego zadania klasie Inwentarz
78.     this.inwentarz.usuńKsiążkę(nrKsiążki);
79.     Widok.pokaż(this.getClass().getCanonicalName(), "usunięcieKsiążki", true, "Zakończono usuwanie książki nr " + nrKsiążki + ".");
80. }
81. }

```

Model.WypożyczonaKsiążka.java

```

1. package Model;
2. import Komunikacja.CSV;
3.
4. public class WypożyczonaKsiążka extends KartaKsiążki {
5.     private Czytelnik czytelnik;
6.     private String dataWypożyczenia;
7.
8.     /**
9.      * Dekorator wypożyczonej książki.
10.     * @param książka wypożyczona książka
11.     * @param dataWypożyczenia data wypożyczenia
12.     * @param czytelnik wypożyczający czytelnik
13.     */
14.     public WypożyczonaKsiążka(IKsiążka książka, String dataWypożyczenia, Czytelnik czytelnik) {
15.         super(książka);
16.         this.czytelnik = czytelnik;
17.         this.dataWypożyczenia = dataWypożyczenia;
18.     }
19.
20.    /**
21.     * Utworzenie csv książki, uwzględniając też jej opakowanie w wypożyczenie.
22.     * @return csv książka
23.     */
24.     public String opisz() {
25.         CSV csv = new CSV();
26.         return csv.utwórzCsvKsiążkiWypożyczonej(this.książka.opisz(), this.dataWypożyczenia, this.czytelnik.nr);
27.     }
28. }

```

Model.ZgubionaKsiążka.java

```

1. package Model;
2. import Komunikacja.CSV;

```

```

3.
4. public class ZgubionaKsiążka extends KartaKsiążki {
5.     private String dataZgubienia;
6.
7.     /**
8.      * Dekorator zgubionej książki.
9.      * @param książka      zgubiona książka
10.     * @param dataWypożyczenia  data zgubienia
11.    */
12.   public ZgubionaKsiążka(IKsiążka książka, String dataZgubienia) {
13.       super(książka);
14.       this.dataZgubienia = dataZgubienia;
15.   }
16.
17.   /**
18.      * Utworzenie csv książki, uwzględniając też jej opakowanie w zgubienie.
19.      * @return  csv książki
20.   */
21.   public String opisz() {
22.       CSV csv = new CSV();
23.       return csv.utwórzCsvKsiążkiZgubionej(this.książka.opisz(), this.dataZgubienia);
24.   }
25. }
```

Do klas wygenerowanych przez *Visual Paradigm* dodano następujące klasy pomocnicze z pakietu *Komunikacja*:

- klasa *CSV* obsługuje tekstowy zapis danych w formacie csv,
- klasa *Widok* obsługuje pokazywanie na ekranie wyników działania programu.

Komunikacja.CSV.java

```

1. package Komunikacja;
2. import java.util.ArrayList;
3. import java.util.regex.Matcher;
4. import java.util.regex.Pattern;
5.
6. /**
7.  * Wyszukiwanie lub tworzenie żądanego fragmentów opisu danych w formacie csv:
8.  * autor:    (imię,nazwisko)
9.  * czytelnik: (nr;imię;nazwisko)
10. * książka:  (nrKsiążki;tytuł;(imięAutora,nazwiskoAutora)*(;dataWypożyczenia;nrCzytelnika)?;(dataZgubienia)?)
11. */
12. public class CSV {
13.
14.     public CSV() {
15. }
16.
17.     /**
18.      * Znalezienie imienia autora w csv autora.
19.      * @param autor  csv autora (imię,nazwisko)
20.      * @return      imię autora
21.     */
22.     public String znajdźImięAutora(String autor) {
23.         String imię = "";
24.         Pattern wzór = Pattern.compile("^\[\p{L}\ ]+(?=,)");
25.         Matcher dopasowanie = wzór.matcher(autor);
26.         if (dopasowanie.find())
27.             imię = dopasowanie.group();
28.         return imię;
29.     }
30.
31.     /**
32.      * Znalezienie nazwiska autora w csv autora.
33.      * @param autor  csv autora (imię,nazwisko)
34.      * @return      nazwisko autora
35.     */
36.     public String znajdźNazwiskoAutora(String autor) {
37.         String nazwisko = "";
38.         Pattern wzór = Pattern.compile("(?=<=,)[\p{L}\ ]+$");
39.         Matcher dopasowanie = wzór.matcher(autor);
40.         if (dopasowanie.find())
41.             nazwisko = dopasowanie.group();
42.         return nazwisko;
43.     }
44.
45.     /**
46.      * Znalezienie numeru czytelnika w csv czytelnika.
47.      * @param czytelnik  csv czytelnika (nr;imię;nazwisko)
48.      * @return      numer czytelnika
49.     */
50.     public int znajdźNrCzytelnika(String czytelnik) {
51.         int nr = 0;
52.         Pattern wzór = Pattern.compile("^\\d+");
53.         Matcher dopasowanie = wzór.matcher(czytelnik);
54.         if (dopasowanie.find())
55.             nr = Integer.parseInt(dopasowanie.group());
```

```

56.     return nr;
57. }
58.
59. /**
60. * Znalezienie imienia czytelnika w csv czytelnika.
61. * @param czytelnik csv czytelnika (nr;imię;nazwisko)
62. * @return imię czytelnika
63. */
64. public String znajdźImięCzytelnika(String czytelnik) {
65.     String imię = "";
66.     Pattern wzór = Pattern.compile("(?=<;)[\p{L}]*(?=;)"); // regex: (?=<;)[\p{L}]*(?=;)
67.     Matcher dopasowanie = wzór.matcher(czytelnik);
68.     if (dopasowanie.find())
69.         imię = dopasowanie.group();
70.     return imię;
71. }
72.
73. /**
74. * Znalezienie nazwiska czytelnika w csv czytelnika.
75. * @param czytelnik csv czytelnika (nr;imię;nazwisko)
76. * @return nazwisko czytelnika
77. */
78. public String znajdźNazwiskoCzytelnika(String czytelnik) {
79.     String nazwisko = "";
80.     Pattern wzór = Pattern.compile("(?=<;)[\p{L}]*$"); // regex: (?=<;)[\p{L}]*$
81.     Matcher dopasowanie = wzór.matcher(czytelnik);
82.     if (dopasowanie.find())
83.         nazwisko = dopasowanie.group();
84.     return nazwisko;
85. }
86.
87. /**
88. * Znalezienie numeru książki w csv książki.
89. * @param książka csv książki (nrKsiążki;tytuł;imięAutora,nazwiskoAutora)*(;dataWypożyczenia;nrCzytelnika)?(;dataZgubienia)?
90. * @return numer książki
91. */
92. public int znajdźNrKsiążki(String książka) {
93.     int nr = 0;
94.     Pattern wzór = Pattern.compile("^\\d+"); // regex: ^\\d+
95.     Matcher dopasowanie = wzór.matcher(książka);
96.     if (dopasowanie.find())
97.         nr = Integer.parseInt(dopasowanie.group());
98.     return nr;
99. }
100.
101. /**
102. * Znalezienie tytułu książki w csv książki.
103. * @param książka csv książki (nrKsiążki;tytuł;imięAutora,nazwiskoAutora)*(;dataWypożyczenia;nrCzytelnika)?(;dataZgubienia)?
104. * @return numer książki
105. */
106. public String znajdźTytułKsiążki(String książka) {
107.     String tytuł = "";
108.     Pattern wzór = Pattern.compile("(?=<\\d;)[\\p{L}]\\s.(.)*(?=;)"); // regex: (?=<\\d;)[\\p{L}]\\s.(.)*(?=;)
109.     Matcher dopasowanie = wzór.matcher(książka);
110.     if (dopasowanie.find())
111.         tytuł = dopasowanie.group();
112.     return tytuł;
113. }
114.
115. /**
116. * Znalezienie autorów książki w csv książki.
117. * @param książka csv książki (nrKsiążki;tytuł;imięAutora,nazwiskoAutora)*(;dataWypożyczenia;nrCzytelnika)?(;dataZgubienia)?
118. * @return lista autorów książki
119. */
120. public ArrayList<String> znajdźAutorówKsiążki(String książka) {
121.     ArrayList<String> autorzy = new ArrayList<String>();
122.     Pattern wzór = Pattern.compile("(?=<\\D[:] )[\\p{L} ,]+"); // regex: (?=<\\D[:] )[\\p{L} ,] +
123.     Matcher dopasowanie = wzór.matcher(książka);
124.     while (dopasowanie.find())
125.         autorzy.add(dopasowanie.group());
126.     return autorzy;
127. }
128.
129. /**
130. * Znalezienie daty wypożyczenia książki w csv książki.
131. * @param książka csv książki (nrKsiążki;tytuł;imięAutora,nazwiskoAutora)*(;dataWypożyczenia;nrCzytelnika)?(;dataZgubienia)?
132. * @return data wypożyczenia
133. */
134. public String znajdźDatęWypożyczeniaKsiążki(String książka) {
135.     String data = "";
136.     Pattern wzór = Pattern.compile("(?=<\\d)\\d{8}(?=;)"); // regex: (?=<\\d)\\d{8}(?=;)
137.     Matcher dopasowanie = wzór.matcher(książka);
138.     if (dopasowanie.find())
139.         data = dopasowanie.group();
140.     return data;
141. }
142.
143. /**
144. * Znalezienie numeru czytelnika książki w csv książki.
145. * @param książka csv książki (nrKsiążki;tytuł;imięAutora,nazwiskoAutora)*(;dataWypożyczenia;nrCzytelnika)?(;dataZgubienia)?
146. * @return numer czytelnika
147. */

```

```

148. public int znajdzNrCzytelnikaKsiążki(String książka) {
149.     int nr = 0;
150.     Pattern wzór = Pattern.compile("(?=<=\\d{8};)\\d+"); // regex: (?=<=\\d{8};)\\d+
151.     Matcher dopasowanie = wzór.matcher(książka);
152.     if (dopasowanie.find())
153.         nr = Integer.parseInt(dopasowanie.group(0));
154.     return nr;
155. }
156.
157. /**
158. * Znalezienie daty zgubienia książki w csv książce.
159. * @param książka csv książki (nrKsiążki;tytuł;(imięAutora,nazwiskoAutora)*(dataWypożyczenia;nrCzytelnika)?;(dataZgubienia)?)
160. * @return data zgubienia
161. */
162. public String znajdzDatęZgubieniaKsiążki(String książka) {
163.     String data = "";
164.     Pattern wzór = Pattern.compile("(?=<=;)\\d{8}$"); // regex: (?=<=;)\\d{8}$
165.     Matcher dopasowanie = wzór.matcher(książka);
166.     if (dopasowanie.find())
167.         data = dopasowanie.group();
168.     return data;
169. }
170.
171. /**
172. * Sprawdzenie, czy csv jest datą.
173. * @param csv csv daty (RRRRMMDD)
174. * @return true jeśli csv jest datą
175. */
176. public boolean czyData(String csv) {
177.     Pattern wzór = Pattern.compile("^\\d{8}$"); // regex: ^\\d{8}$
178.     Matcher dopasowanie = wzór.matcher(csv);
179.     if (dopasowanie.find())
180.         return true;
181.     else
182.         return false;
183. }
184.
185. /**
186. * Sprawdzenie, czy csv jest numerem.
187. * @param csv csv numeru (1-3 cyfr)
188. * @return true jeśli csv jest numerem
189. */
190. public boolean czyNumer(String csv) {
191.     Pattern wzór = Pattern.compile("^\\d{1,3}$"); // regex: ^\\d{1,3}$
192.     Matcher dopasowanie = wzór.matcher(csv);
193.     if (dopasowanie.find())
194.         return true;
195.     else
196.         return false;
197. }
198.
199. /**
200. * Opisanie autora książki.
201. * @param autor csv autora
202. * @return opisanie autora (Imię Nazwisko)
203. */
204. public String opiszAutora(String autor) {
205.     String opisanie = "";
206.     opisanie = this.znajdzImięAutora(autor) + " " + this.znajdzNazwiskoAutora(autor);
207.     return opisanie;
208. }
209.
210. /**
211. * Opisanie czytelnika.
212. * @param czytelnik csv czytelnika
213. * @return opisanie czytelnika (czytelnik nr CC - Imię Nazwisko)
214. */
215. public String opiszCzytelnika(String czytelnik) {
216.     String opisanie = "";
217.     opisanie = "czytelnik nr " + this.znajdzNrCzytelnika(czytelnik) + " - " + this.znajdzImięCzytelnika(czytelnik) + " " + this.znajdzNazwiskoCzytelnika(czytelnik);
218.     return opisanie;
219. }
220.
221. /**
222. * Opisanie książki.
223. * @param książka csv książki
224. * @return opisanie książki (książka nr CC - 'Tytuł'; ((autor)|(autorzy)): Imię Nazwisko, Imię Nazwisko)*?(); wypożyczona dnia RRRR-MM-DD
225. * przez czytelnika nr CC)?(); zgubiona dnia RRRR-MM-DD)?
226. */
227. public String opiszKsiążkę(String książka) {
228.     String opisanie = "";
229.     opisanie = "książka nr " + this.znajdzNrKsiążki(książka) + " - " + this.znajdzTytułKsiążki(książka) + "";
230.     // opisanie autorów
231.     ArrayList<String> autorzy = this.znajdzAutorówKsiążki(książka);
232.     if (autorzy.size() > 0) {
233.         opisanie += autorzy.size()==1 ? " autor: " : " autorzy: ";
234.         opisanie += this.opiszAutora(autorzy.get(0));
235.         for (int i=1; i<autorzy.size(); i++) {
236.             opisanie += ", " + this.opiszAutora(autorzy.get(i));
237.         }
238.     }
239.     //opisanie wypożyczenia książki

```

```

240. String dataWypożyczenia = this.znajdźDatęWypożyczeniaKsiążki(książka);
241. if (dataWypożyczenia.length()>0) {
242.     dataWypożyczenia = dataWypożyczenia.substring(0, 4) + "-" + dataWypożyczenia.substring(4, 6) + "-" + dataWypożyczenia.substring(6, 8);
243.     opisanie += "; wypożyczona dnia " + dataWypożyczenia + " przez czytelnika nr " + this.znajdźNrCzytelnikaKsiążki(książka);
244. }
245. // opisanie zgubienia książki
246. String dataZgubienia = this.znajdźDatęZgubieniaKsiążki(książka);
247. if (dataZgubienia.length()>0) {
248.     dataZgubienia = dataZgubienia.substring(0, 4) + "-" + dataZgubienia.substring(4, 6) + "-" + dataZgubienia.substring(6, 8);
249.     opisanie += "; zgubiona dnia " + dataZgubienia;
250. }
251. return opisanie;
252. }
253.
254. /**
255. * Utworzenie csv autora książki.
256. * @param imię imię autora
257. * @param nazwisko nazwisko autora
258. * @return csv autora (imię,nazwisko)
259. */
260. public String utwórzCsvAutora(String imię, String nazwisko) {
261.     String csv = imię + "," + nazwisko;
262.     return csv;
263. }
264.
265. /**
266. * Utworzenie csv czytelnika.
267. * @param nr numer czytelnika
268. * @param imię imię czytelnika
269. * @param nazwisko nazwisko czytelnika
270. * @return csv czytelnika (nr;imię;nazwisko)
271. */
272. public String utwórzCsvCzytelnika(int nr, String imię, String nazwisko) {
273.     String csv = Integer.toString(nr) + ":" + imię + ":" + nazwisko;
274.     return csv;
275. }
276.
277. /**
278. * Utworzenie csv dostępnej książki
279. * @param nr numer książki
280. * @param tytuł tytuł książki
281. * @param imionaAutorów imiona autorów książki
282. * @param nazwiskaAutorów nazwiska autorów książki
283. * @return csv książki (nrKsiążki;tytuł;(imięAutora,nazwiskoAutora)*
284. */
285. public String utwórzCsvKsiążki(int nr, String tytuł, ArrayList<String> imionaAutorów, ArrayList<String> nazwiskaAutorów) {
286.     String csv = Integer.toString(nr) + ":" + tytuł;
287.     // przygotowanie i dodanie opisu autorów
288.     for (int i=0; i<imionaAutorów.size(); i++) {
289.         csv += ":" + imionaAutorów.get(i) + ":" + nazwiskaAutorów.get(i);
290.     }
291.     return csv;
292.
293. /**
294. * Utworzenie csv wypożyczonej książki.
295. * @param nr numer książki
296. * @param tytuł tytuł książki
297. * @param imionaAutorów imiona autorów książki
298. * @param nazwiskaAutorów nazwiska autorów książki
299. * @param dataWypożyczenia data wypożyczenia
300. * @param nrCzytelnika numer czytelnika
301. * @return csv książki (nrKsiążki;tytuł;(imięAutora,nazwiskoAutora)*;dataWypożyczenia;nrCzytelnika)
302. */
303. public String utwórzCsvKsiążkiWypożyczonej(int nr, String tytuł, ArrayList<String> imionaAutorów, ArrayList<String> nazwiskaAutorów, String dataWypożyczenia,
304. int nrCzytelnika) {
305.     String csv = this.utwórzCsvKsiążki(nr, tytuł, imionaAutorów, nazwiskaAutorów) + ":" + dataWypożyczenia + ":" + Integer.toString(nrCzytelnika);
306.     return csv;
307. }
308.
309. /**
310. * Utworzenie csv wypożyczonej książki.
311. * @param opisWypożyczanejKsiążki opis książki bez jej wypożyczenia
312. * @param dataWypożyczenia data wypożyczenia
313. * @param nrCzytelnika numer czytelnika
314. * @return csv książki (opisWypożyczanejKsiążki;dataWypożyczenia;nrCzytelnika)
315. */
316. public String utwórzCsvKsiążkiWypożyczonej(String opisWypożyczanejKsiążki, String dataWypożyczenia, int nrCzytelnika) {
317.     String csv = opisWypożyczanejKsiążki + ":" + dataWypożyczenia + ":" + Integer.toString(nrCzytelnika);
318.     return csv;
319.
320. /**
321. * Utworzenie csv wypożyczonej i zgubionej książki.
322. * @param nr numer książki
323. * @param tytuł tytuł książki
324. * @param imionaAutorów imiona autorów książki
325. * @param nazwiskaAutorów nazwiska autorów książki
326. * @param dataWypożyczenia data wypożyczenia
327. * @param nrCzytelnika numer czytelnika
328. * @param dataZgubienia data zgubienia
329. * @return csv książki (nrKsiążki;tytuł;(imięAutora,nazwiskoAutora)*;dataWypożyczenia;nrCzytelnika;dataZgubienia
330. */

```

```

331. public String utwórzCsvKsiążkiWypożyczonejZgubionej(int nr, String tytuł, ArrayList<String> imionaAutorów, ArrayList<String> nazwiskaAutorów,
332.     String dataWypożyczenia, int nrCzytelnika, String dataZgubienia) {
333.     String csv = this.utwórzCsvKsiążkiWypożyczonej(nr, tytuł, imionaAutorów, nazwiskaAutorów, dataWypożyczenia, nrCzytelnika) + ":" + dataZgubienia;
334.     return csv;
335.
336. /**
337. * Utworzenie csv zgubionej książki.
338. * @param nr numer książki
339. * @param tytuł tytuł książki
340. * @param imionaAutorów imiona autorów książki
341. * @param nazwiskaAutorów nazwiska autorów książki
342. * @param dataZgubienia data zgubienia
343. * @return csv książki (nrKsiążki;tytuł;(imięAutora,nazwiskoAutora)*;dataZgubienia
344. */
345. public String utwórzCsvKsiążkiZgubionej(int nr, String tytuł, ArrayList<String> imionaAutorów, ArrayList<String> nazwiskaAutorów, String dataZgubienia) {
346.     String csv = this.utwórzCsvKsiążki(nr, tytuł, imionaAutorów, nazwiskaAutorów) + ":" + dataZgubienia;
347.     return csv;
348. }
349.
350. /**
351. * Utworzenie csv zgubionej książki.
352. * @param opisGubionejKsiążki opis książki bez jej zgubienia
353. * @param dataZgubienia data zgubienia
354. * @return csv książki (opisGubionejKsiążki;dataZgubienia)
355. */
356. public String utwórzCsvKsiążkiZgubionej(String opisGubionejKsiążki, String dataZgubienia) {
357.     String csv = opisGubionejKsiążki + ":" + dataZgubienia;
358.     return csv;
359. }
360. }
361.

```

Komunikacja.Widok.java

```

1. package Komunikacja;
2. import java.util.ArrayList;
3.
4. /**
5. * Pokazanie na ekranie raportu o działaniu programu.
6. */
7. public class Widok {
8.
9. /**
10. * Rodzaje poziomu gadatliwości programu.
11. */
12. public static enum Gadatliwość {
13.     BRAK, // niczego nie pokazać
14.     MAŁA, // pokazać główne raporty klas Kontrolera
15.     SREDNIA, // pokazać wszystkie raporty klas Kontrolera i fasady Modelu
16.     WYSOKA, // pokazać wszystkie raporty klas Kontrolera i fasady oraz inwentarza Modelu
17.     PEŁNA // pokazać wszystko
18. }
19.
20. /**
21. * Poziom gadatliwości programu.
22. */
23. private static Gadatliwość gadatliwość = Gadatliwość.BRAK;
24.
25. /**
26. * Ustawienie poziomu gadatliwości programu.
27. * @param poziom poziom gadatliwości
28. */
29. public static void ustawPoziomGadatliwości(Gadatliwość poziom) {
30.     gadatliwość = poziom;
31. }
32.
33. /**
34. * Sprawdzenie, czy pokazać raport zależnie od jego źródła i poziomu gadatliwości.
35. * @param klasa klasa wykonująca rapportującą operację
36. * @param operacja nazwa operacji
37. * @return true, jeśli wyświetlić raport
38. */
39. private static boolean czyRaportować(String klasa, String operacja) {
40.     // brak gadatliwości
41.     if (gadatliwość == Gadatliwość.BRAK)
42.         return false;
43.     // mała gadatliwość
44.     ArrayList<String> akceptowaneOperacje = new ArrayList<String>();
45.     if (gadatliwość == Gadatliwość.MAŁA) {
46.         akceptowaneOperacje.add("Kontroler.KontrolerKierownikaBiblioteki.wypisanieCzytelnikaZBiblioteki()");
47.         akceptowaneOperacje.add("Kontroler.PrzyjęcieZwrotuKsiążki.zakończeniePrzyjęciaKsiążki()");
48.         akceptowaneOperacje.add("Kontroler.SystemZarządzaniaBiblioteką.main()");
49.         akceptowaneOperacje.add("Kontroler.UsuniecieKsiążkiZInwentarza.zakończeniePrzyjęciaKsiążki()");
50.         akceptowaneOperacje.add("Kontroler.WypisanieCzytelnikaZBiblioteki.WypisanieCzytelnikaZBiblioteki()");
51.         akceptowaneOperacje.add("Model.DAO.pokażBazęCzytelników()");
52.         akceptowaneOperacje.add("Model.DAO.pokażBazęKsiążek()");
53.         if (!akceptowaneOperacje.contains(klasa + "." + operacja + "()"))
54.             return false;
55.     }
56.     // średnia gadatliwość
57.     if (gadatliwość == Gadatliwość.SREDNIA) {

```

```

58. akceptowaneOperacje.add("Kontroler.KontrolerKierownikaBiblioteki.wypisanieCzytelnikaZBiblioteki()");
59. akceptowaneOperacje.add("Kontroler.PrzyjęcieZwrotuKsiążki.zakończeniePrzyjęciaKsiążki()");
60. akceptowaneOperacje.add("Kontroler.SystemZarządzaniaBiblioteką.main()");
61. akceptowaneOperacje.add("Kontroler.UsuniecieKsiążkiZInwentarza.zakończeniePrzyjęciaKsiążki()");
62. akceptowaneOperacje.add("Kontroler.WypisanieCzytelnikaZBiblioteki.WypisanieCzytelnikaZBiblioteki()");
63. akceptowaneOperacje.add("Kontroler.PrzyjęcieZwrotuKsiążki.zatwierdzenieZwrotuKsiążki()");
64. akceptowaneOperacje.add("Kontroler.UsuniecieKsiążkiZInwentarza.zatwierdzenieUsunięciaKsiążki()");
65. akceptowaneOperacje.add("Kontroler.WypisanieCzytelnikaZBiblioteki.zatwierdzenieWypisaniaCzytelnika()");
66. akceptowaneOperacje.add("Kontroler.ZakończenieWypożyczeniaKsiążki.wybórOpcji()");
67. akceptowaneOperacje.add("Model.Model.znalezienieWypożyczonychKsiążek()");
68. akceptowaneOperacje.add("Model.Model.udostępnienieKsiążkiDoWypożyczenia()");
69. akceptowaneOperacje.add("Model.Model.usunięcieKsiążki()");
70. akceptowaneOperacje.add("Model.Model.usunięcieDanychOsobowychCzytelnika()");
71. akceptowaneOperacje.add("Model.Model.usunięcieCzytelnika()");
72. akceptowaneOperacje.add("Model.DAO.pokażBazęCzytelników()");
73. akceptowaneOperacje.add("Model.DAO.pokażBazęKsiążek()");
74. if (akceptowaneOperacje.contains(klasa + " ." + operacja + "()"))
75.     return false;
76. }
77. // wysoka gadatliwość
78. if (gadatliwość == Gadatliwość.WYSOKA) {
79.     akceptowaneOperacje.add("Kontroler.KontrolerKierownikaBiblioteki.wypisanieCzytelnikaZBiblioteki()");
80.     akceptowaneOperacje.add("Kontroler.PrzyjęcieZwrotuKsiążki.zakończeniePrzyjęciaKsiążki()");
81.     akceptowaneOperacje.add("Kontroler.SystemZarządzaniaBiblioteką.main()");
82.     akceptowaneOperacje.add("Kontroler.UsuniecieKsiążkiZInwentarza.zakończeniePrzyjęciaKsiążki()");
83.     akceptowaneOperacje.add("Kontroler.WypisanieCzytelnikaZBiblioteki.WypisanieCzytelnikaZBiblioteki()");
84.     akceptowaneOperacje.add("Kontroler.PrzyjęcieZwrotuKsiążki.zatwierdzenieZwrotuKsiążki()");
85.     akceptowaneOperacje.add("Kontroler.UsuniecieKsiążkiZInwentarza.zatwierdzenieUsunięciaKsiążki()");
86.     akceptowaneOperacje.add("Kontroler.WypisanieCzytelnikaZBiblioteki.zatwierdzenieWypisaniaCzytelnika()");
87.     akceptowaneOperacje.add("Kontroler.ZakończenieWypożyczeniaKsiążki.wybórOpcji()");
88.     akceptowaneOperacje.add("Model.Model.znalezienieWypożyczonychKsiążek()");
89.     akceptowaneOperacje.add("Model.Model.udostępnienieKsiążkiDoWypożyczenia()");
90.     akceptowaneOperacje.add("Model.Model.usunięcieKsiążki()");
91.     akceptowaneOperacje.add("Model.Model.usunięcieDanychOsobowychCzytelnika()");
92.     akceptowaneOperacje.add("Model.Model.usunięcieCzytelnika()");
93.     akceptowaneOperacje.add("Model.Inwentarz.dajKsiążkę()");
94.     akceptowaneOperacje.add("Model.Inwentarz.dajWypożyczoneKsiążkiCzytelnika()");
95.     akceptowaneOperacje.add("Model.Inwentarz.udostępnijKsiążkę()");
96.     akceptowaneOperacje.add("Model.Inwentarz.usuńKsiążkę()");
97.     akceptowaneOperacje.add("Model.Inwentarz.dajCzytelnika()");
98.     akceptowaneOperacje.add("Model.Inwentarz.usuńCzytelnika()");
99.     akceptowaneOperacje.add("Model.DAO.pokażBazęCzytelników()");
100.    akceptowaneOperacje.add("Model.DAO.pokażBazęKsiążek()");
101.    if (akceptowaneOperacje.contains(klasa + " ." + operacja + "()"))
102.        return false;
103. }
104. // pełna gadatliwość
105. return true;
106. }

107. /**
108. * Utworzenie nagłówka raportu operacji.
109. * @param klasa klasa wykonująca raportującą operację
110. * @param operacja nazwa operacji
111. * @param sukces true, jeśli raport z udanego zadania
112. * @return nagłówek raportu
113. */
114.
115. private static String nagłówekRaportu(String klasa, String operacja, Boolean sukces) {
116.     // sukces na niebiesko, porażka na czerwono
117.     String nagłówek = "\n\033[1;30m" + klasa + "\033[0;30m\n";
118.     if (sukces)
119.         nagłówek += "\033[0;34m✓ " + operacja + "():\033[0;30m";
120.     else
121.         nagłówek += "\033[0;31m✗ " + operacja + "():\033[0;30m";
122.     return nagłówek;
123. }

124. /**
125. * Pokazanie raportu operacji.
126. * @param klasa klasa wykonująca raportującą operację
127. * @param operacja nazwa operacji
128. * @param sukces true, jeśli raport z udanego zadania
129. * @param treść 1-liniowa treść do pokazania
130. */
131.
132. public static void pokaż(String klasa, String operacja, Boolean sukces, String treść) {
133.     if (czyRaportować(klasa, operacja)){
134.         String raport = nagłówekRaportu(klasa, operacja, sukces);
135.         raport += "\n--- " + treść;
136.         System.out.println(raport);
137.     }
138. }

139. /**
140. * Pokazanie raportu operacji.
141. * @param klasa klasa wykonująca raportującą operację
142. * @param operacja nazwa operacji
143. * @param sukces true, jeśli raport z udanego zadania
144. * @param treść wielo-liniowa treść do pokazania
145. */
146.
147. public static void pokaż(String klasa, String operacja, Boolean sukces, ArrayList<String> treść) {
148.     if (czyRaportować(klasa, operacja)){
149.         String raport = nagłówekRaportu(klasa, operacja, sukces);
150.         for (String linia: treść)

```

```
151.         report += "\n--- " + linia;
152.         System.out.println(report);
153.     }
154. }
155. }
```

Modelowanie zmienności warstwy widoku

Ciąg dalszy nastąpi...