

Wypożyczalnia i sklep urządzeń komputerowych

1. Dziedzina problemowa:

Projektowany system może znaleźć zastosowanie w dziedzinie handlu elektronicznego oraz w dziedzinie sprzętu komputerowego.

2. Cel:

System ma być pośrednikiem w procesie kupna i wypożyczania sprzętu komputerowego.

3. Zakres odpowiedzialności systemu:

System ma umożliwiać przeglądanie informacji o urządzeniach, które znajdują się w ofercie sklepu/wypożyczalni. Ma umożliwiać kupno oraz wypożyczanie urządzeń. Ma przechowywać informacje o klientach. Ma umożliwiać pracownikom edycję danych o urządzeniach.

4. Użytkownicy systemu:

- a) Gość
- b) Klient
- c) Pracownik serwisu
- d) Menadżer.

5. Wymagania użytkownika:

- a) System powinien przechowywać następujące informacje urządzeniach komputerowych takie jak nazwa, cenę wyjściową urządzenia, cenę z uwzględnionym rabatem, rabat (wspólny dla wszystkich urządzeń, z ograniczeniem, że nie można jednorazowo zmienić rabatu o więcej niż 5 punktów procentowych), opis (opcjonalny), zdjęcia (przynajmniej jedno, ale może być ich więcej), wymiary urządzenia (szerokość, długość i wysokość), nazwę systemu operacyjnego, który wykorzystuje urządzenie oraz informację. Oprócz tych informacji muszą być przechowywane informacje o egzemplarzach danego urządzenia, tj. czy jest przeznaczone do wypożyczenia czy kupna oraz jego aktualny status (dostępny, kupiony, wypożyczony).
- b) Urządzenia dzielą się na dwie kategorie: stacjonarne oraz mobilne. Urządzenie nie może być jednocześnie przenośne i stacjonarne.
- c) Urządzenia stacjonarne zawierają informację o ustawieniu urządzenia (poziome, pionowe) oraz czy istnieje możliwość montażu takiego urządzenia na ścianie.
- d) Urządzenia stacjonarne są podzielone na komputery stacjonarne (cechy charakterystyczne: nazwa obudowy oraz jej rodzaj [mini, midi, big]) oraz konsole, którym przypisana jest liczba kontrolerów załączonych do danego egzemplarza oraz generacja, do jakiej należy konsola. Podział ten jest rozłączny.
- e) Cechy istotne dla urządzeń mobilnych to ich waga, rozdzielcość kamery przedniej oraz ekran (przekątna matrycy w calach, typ matrycy, rozdzielcość matrycy).
- f) Urządzenia przenośne to laptopy (obecność klawiatury numerycznej, czy klawiatura jest podświetlana) i tablety (czy posiada funkcję dzwonienia, rozdzielcość kamery tylnej).
- g) Komputer stacjonarny oraz laptop są komputerami PC, które cechuje wielkość pamięci RAM, nazwa karty graficznej oraz procesora oraz przynajmniej jeden dysk twardy (typ - HDD lub SSD oraz pojemność). Opcjonalnie komputer PC może zawierać jeden lub więcej napędów (rodzaj - DVD, Blu-ray, szybkość pracy).
- h) System powinien pamiętać datę transakcji przeprowadzonych przez klientów. Transakcje dzielą się na kupna i wypożyczenia.
- i) Transakcje kupna powinny przechowywać informacje o płatności (nieopłacone, opłacone), wybranej metodzie płatności oraz sposobie dostawy. Powinna istnieć możliwość dostępu do aktualnego statusu transakcji.
- j) W ramach jednej transakcji kupna użytkownik może kupić wiele urządzeń.

- k) Transakcje wypożyczenia muszą zawierać informacje o dacie zwrotu urządzenia oraz o aktualnym statusie wypożyczenia (wypożyczony, zwrócony). Powinna istnieć możliwość dostępu do aktualnego statusu transakcji.
- l) W ramach jednej transakcji wypożyczenia użytkownik może wypożyczyć tylko jedno urządzenie.
- m) Jedna osoba może wypożyczyć jednocześnie nie więcej niż 3 urządzenia. Maksymalny czas wypożyczenia jednego urządzenia to 180 dni. Powinna być jednak możliwość późniejszej zmiany tych dwóch wartości.
- n) System powinien przechowywać następujące dane pracowników: ich identyfikatory oraz stanowisko, na którym pracują.
- o) Pracownicy sklepu mają mieć możliwość zarządzania urządzeniami.
- p) Przeglądanie całej oferty możliwe jest przez wszystkich użytkowników serwisu. Kupowanie oraz wypożyczanie możliwe jest tylko dla zarejestrowanych użytkowników.
- q) Podczas rejestracji wymagane jest podanie adresu e-mail, danych osobowych (imię [możliwe więcej niż jedno], nazwisko, adres, nr telefonu oraz opcjonalnie datę urodzenia). System powinien być w stanie obliczyć wiek klienta na podstawie jego daty urodzenia.
- r) System powinien pamiętać, ile urządzeń aktualnie wypożycza każdy z klientów.
- s) System powinien umożliwiać takie funkcjonalności jak:
 - i. Przeglądanie urządzeń (tylko dostępnych do kupna, tylko dostępnych do wypożyczenia lub obu tych typów),
 - ii. Wyświetlenie listy urządzeń dostępnych do kupna lub wypożyczenia,
 - iii. Możliwość rejestracji dla osób niemających konta,
 - iv. Możliwość edycji własnych danych osobowych przez zarejestrowanych użytkowników,
 - v. Utworzenie statystyk kupowanych oraz wypożyczanych urządzeń ze względu na cenę oraz rodzaj urządzenia.

6. Opis przyszłej ewolucji systemu:

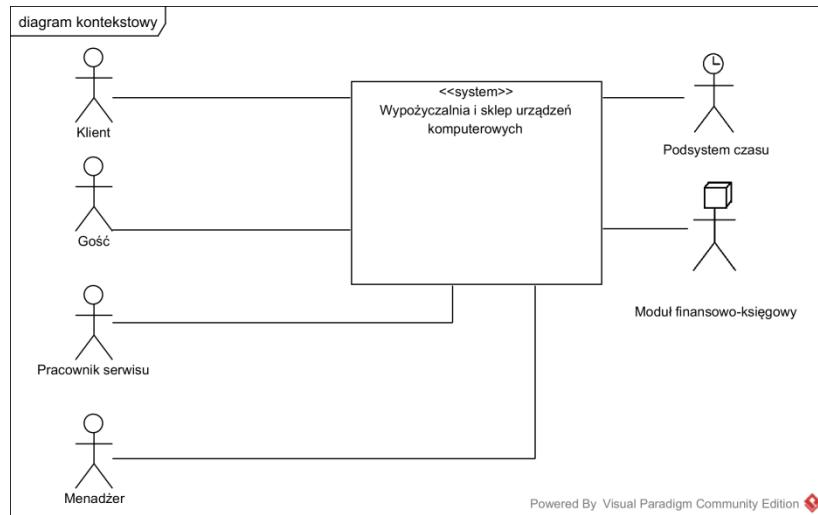
Dodanie nowych rodzajów urządzeń, dodanie dokładniejszych opisów urządzeń.

7. Słownik pojęć:

- a) Gość – osoba nie mająca konta w systemie
- b) Klient – osoba mająca konto w serwisie, która może kupować i wypożyczać urządzenia
- c) Pracownik serwisu – osoba zarządzająca serwisem
- d) Menadżer – osoba zarządzająca pracownikami serwisu

8. Analiza funkcjonalna (model przypadków użycia):

- a) diagram kontekstowy:



- b) lista przypadków użycia:

Nazwa	Kup urządzenie
Nr id	1
Priorytet	Wysoki
Aktorzy	Klient
Warunek początkowy	Co najmniej jedno urządzenie musi być dostępne do kupna
Warunek końcowy	Zarejestrowanie informacji o zakupie
Wymagania niefunkcjonalne	brak
Uwagi dodatkowe	brak

Nazwa	Wypożycz urządzenie
Nr id	2
Priorytet	Wysoki
Aktorzy	Klient
Warunek początkowy	Co najmniej jedno urządzenie musi być dostępne do kupna
Warunek końcowy	Zarejestrowanie informacji o wypożyczeniu
Wymagania niefunkcjonalne	brak
Uwagi dodatkowe	brak

Nazwa	Zarejestruj się
Nr id	3
Priorytet	Wysoki
Aktorzy	Gość
Warunek początkowy	Brak
Warunek końcowy	Zarejestrowanie osoby w serwisie
Wymagania niefunkcjonalne	brak
Uwagi dodatkowe	brak

Nazwa	Dodaj pracownika
Nr id	4
Priorytet	Średni
Aktorzy	Menadżer
Warunek początkowy	Pracownik, który ma być dodany, nie jest zarejestrowany w systemie
Warunek końcowy	Dodanie nowego pracownika do systemu
Wymagania niefunkcjonalne	brak
Uwagi dodatkowe	brak

Nazwa	Edytuj dane pracownika
Nr id	5
Priorytet	Niski
Aktorzy	Menedżer
Warunek początkowy	Brak

Warunek końcowy	Zmiana danych pracownika
Wymagania niefunkcjonalne	brak
Uwagi dodatkowe	brak

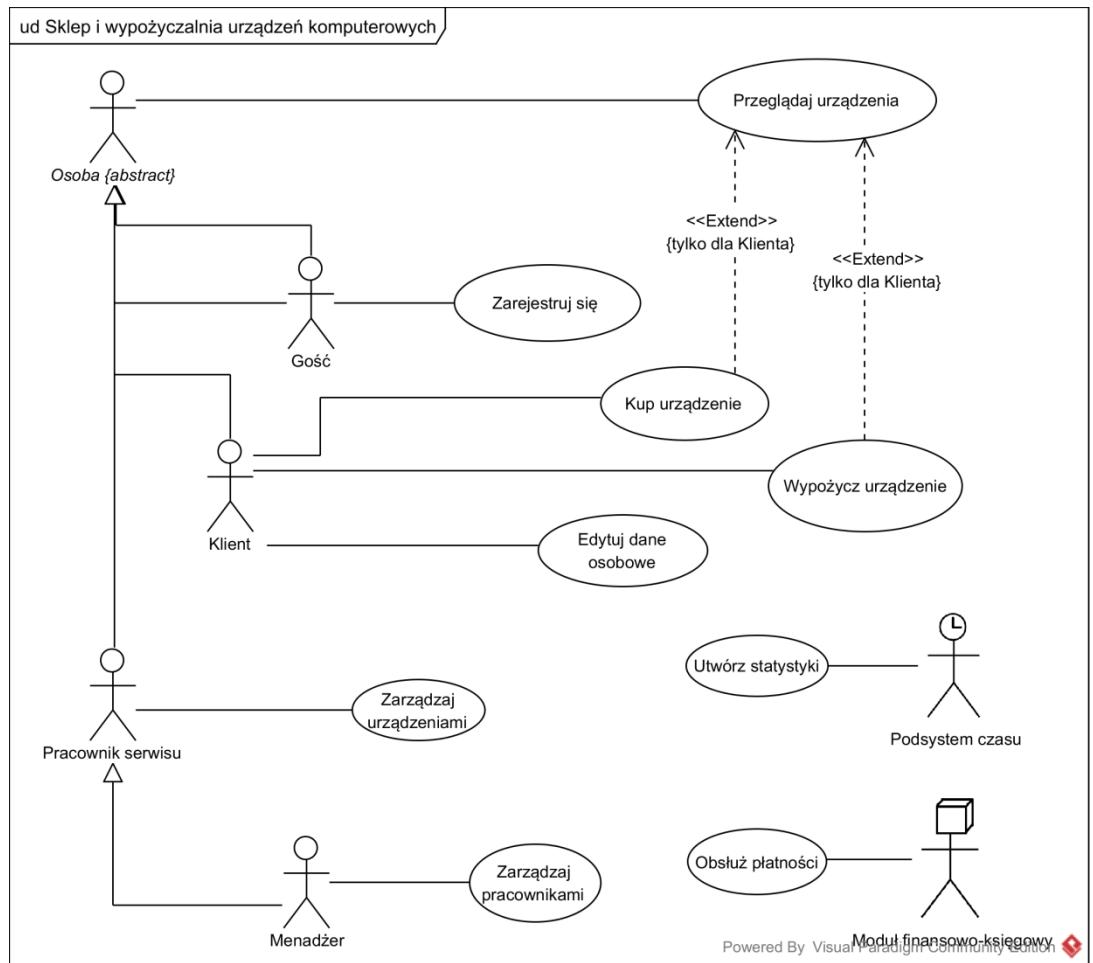
Nazwa	Przeglądaj urządzenia
Nr id	6
Priorytet	Niski
Aktorzy	Osoba
Warunek początkowy	Co najmniej jedno urządzenie w bazie systemu
Warunek końcowy	brak
Wymagania niefunkcjonalne	brak
Uwagi dodatkowe	brak

Nazwa	Edytuj dane urządzenia
Nr id	7
Priorytet	Niski
Aktorzy	Pracownik serwisu
Warunek początkowy	Co najmniej jedno urządzenie w bazie serwisu
Warunek końcowy	Zmiana danych urządzenia
Wymagania niefunkcjonalne	brak
Uwagi dodatkowe	brak

Nazwa	Edytuj dane osobowe
Nr id	8
Priorytet	Niski
Aktorzy	Klient
Warunek początkowy	Osoba edytująca dane musi być zarejestrowana w systemie
Warunek końcowy	Zmiana danych osobowych klienta
Wymagania niefunkcjonalne	brak
Uwagi dodatkowe	brak

Nazwa	Utwórz statystyki
Nr id	9
Priorytet	Niski
Aktorzy	Podsystem czasu
Warunek początkowy	Co najmniej jedna dokonana transakcja
Warunek końcowy	Sporządzenie statystyk
Wymagania niefunkcjonalne	brak
Uwagi dodatkowe	brak

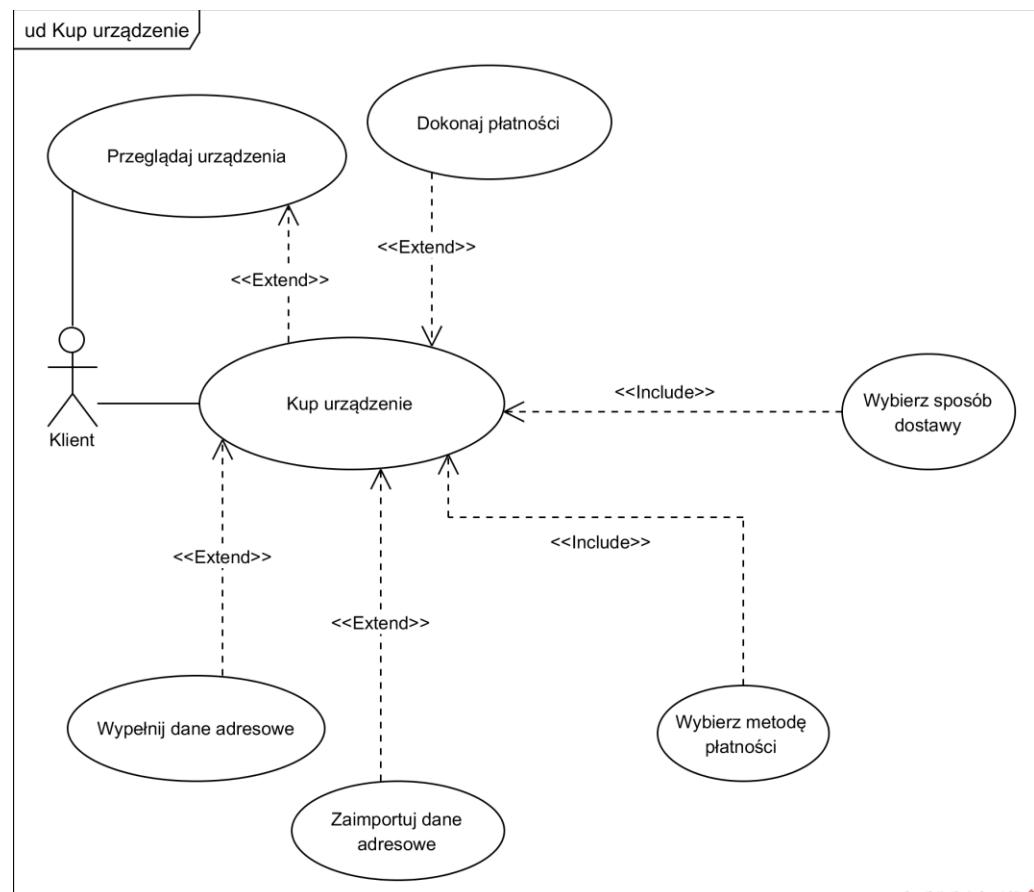
c) diagram przypadków użycia:



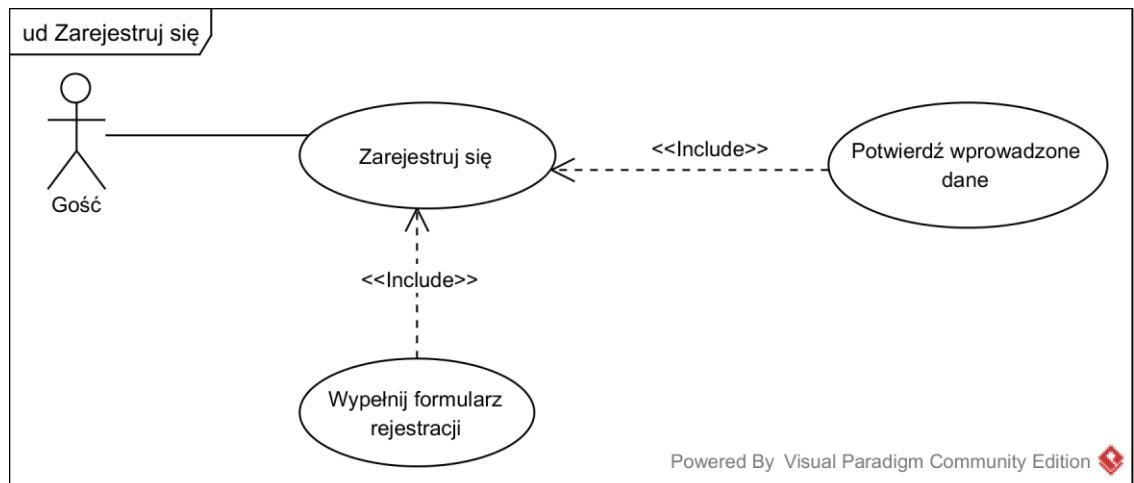
d) dokumentacja przypadków użycia:

Nazwa	Kup urządzenie
Nr id	1
Priorytet	Wysoki
Aktorzy	Klient
Warunek początkowy	Co najmniej jedno urządzenie musi być dostępne do kupna
Warunek końcowy	Zarejestrowanie informacji o kupnie
Główny przepływ zdarzeń	<ol style="list-style-type: none"> Klient uruchamia przypadek użycia "Kup urządzenie" System wyświetla urządzenia, które są możliwe do kupna, Klient wybiera urządzenie, system dodaje je do koszyka System pyta czy Klient chce dodać kolejną rzecz do koszyka, Klient odmawia System wyświetla formularz kupna, Klient wprowadza dane adresowe (imię, nazwisko, adres, e-mail, telefon), równolegle wybiera sposób dostawy, a po wybraniu sposobu dostawy wybiera metodę płatności System weryfikuje, czy wprowadzone dane adresowe są poprawne, weryfikacja pozytywna System wyświetla informację o danych

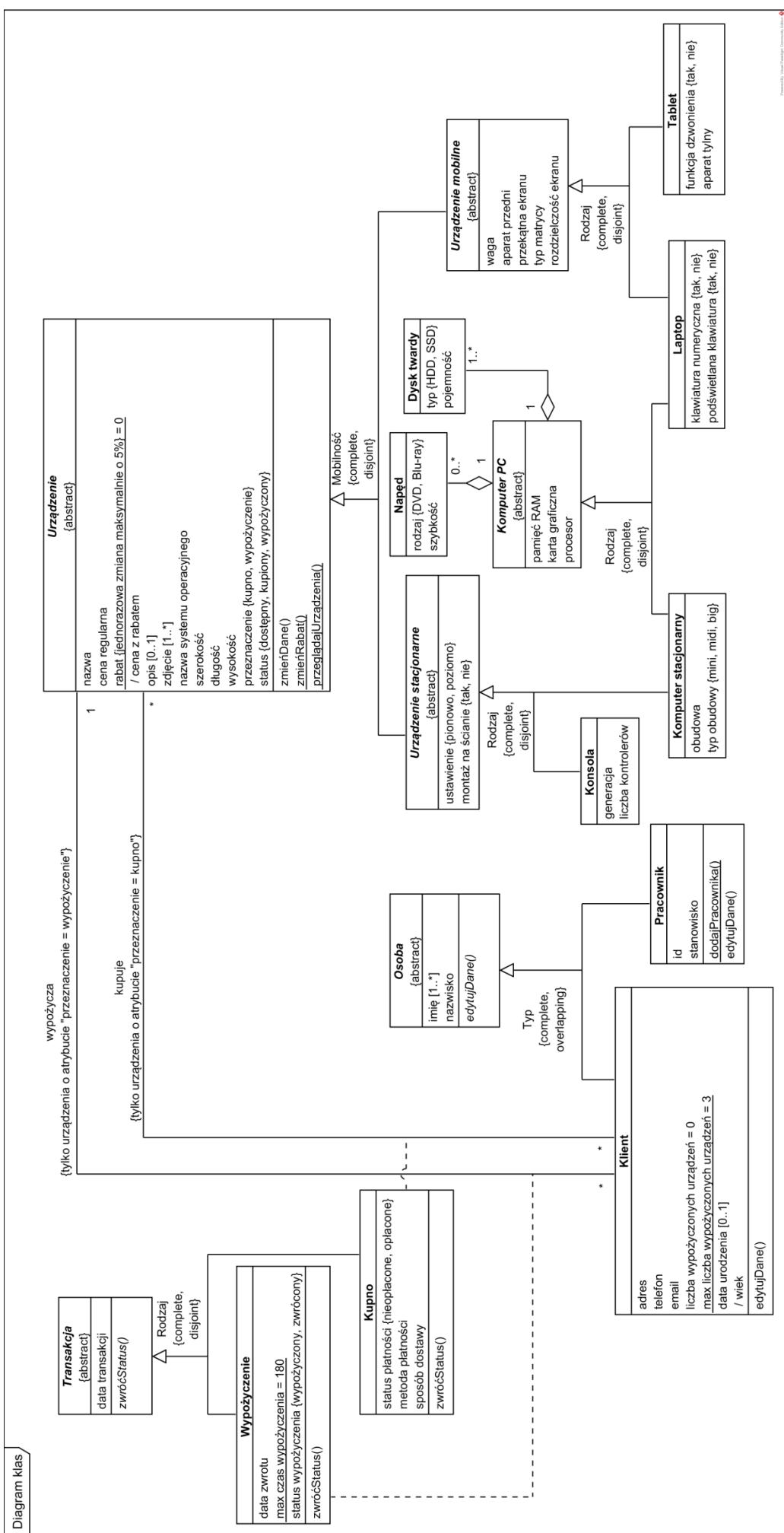
	kupna i prosi o potwierdzenie, Klient potwierdza 7. Klient dokonuje płatności natychmiastowej 8. System informuje o kupnie urządzenia
Alternatywne przepływy zdarzeń	1a. Przypadek użycia uruchamiany w ramach przypadku użycia "Przeglądaj urządzenia", system przechodzi do punktu 3 3a. System pyta czy Klient chce dodać kolejną rzeczkę do koszyka, Klient się zgadza, system przechodzi do punktu 2 4a. System wyświetla formularz kupna, Klient importuje dane adresowe (imię, nazwisko, adres, e-mail, telefon) ze swojego profilu, równolegle wybiera sposób dostawy, a po wybraniu sposobu dostawy wybiera metodę płatności 5a. System weryfikuje, czy wprowadzone dane adresowe są poprawne, weryfikacja negatywna, system przechodzi do punktu 4 7a. Klient nie dokonuje płatności natychmiastowej
Wymagania niefunkcjonalne	brak
Uwagi dodatkowe	brak
Zakończenie	W dowolnym momencie



Nazwa	Zarejestruj się
Nr id	3
Priorytet	Wysoki
Aktorzy	Gość
Warunek początkowy	Brak
Warunek końcowy	Zarejestrowanie osoby w serwisie
Główny przepływ zdarzeń	<p>1. Gość uruchamia przypadek użycia "Zarejestruj się"</p> <p>2. System wyświetla formularz rejestracji, który zawiera pola na wpisanie imion, nazwiska, adresu, nr telefonu adresu e-mail oraz opcjonalnie daty urodzenia; Gość wprowadza dane</p> <p>3. System wyświetla wprowadzone dane i prosi o potwierdzenie, Gość potwierdza</p> <p>4. System informuje o udanej rejestracji</p>
Alternatywne przepływy zdarzeń	2. System informuje o błędach we wprowadzonych danych, system przechodzi do punktu 2
Wymagania niefunkcjonalne	brak
Uwagi dodatkowe	brak
Zakończenie	W dowolnym momencie



9. Analiza strukturalna:



10. Analiza wartości pochodnych, początkowych i granicznych

Wartości pochodne:

Klasa	Atrybut	Typ	Opis
Klient	wiek	pochodna	Atrybut określa wiek klienta, wyliczany jest na podstawie daty urodzenia klienta oraz daty bieżącej.
Urządzenie	cena z rabatem	pochodna	Atrybut jest wyliczany na podstawie ceny regularnej i rabatu - cena regularna pomniejszana jest o wartość rabatu.

Wartości początkowe:

Klient	liczba wypożyczonych urządzeń	początkowa	Atrybut określa aktualną liczbę urządzeń wypożyczonych przez klienta.
Urządzenie	rabat	początkowa	Atrybut określa rabat, wspólny dla wszystkich urządzeń.

Wartości graniczne:

Wypożyczenie	max czas wypożyczenia	graniczna	Atrybut określa maksymalny czas, na jaki klient może wypożyczyć jedno urządzenie. Może ulec zmianie.
Klient	max liczba wypożyczonych urządzeń	graniczna	Atrybut określa maksymalną liczbę urządzeń, jaką jeden klient może mieć wypożyczonych w tym samym czasie. Może ulec zmianie.

Wartości pochodne:

- Atrybut "wiek" z klasy "Klient" zostanie zamieniony na metodę zwracającą wiek klienta.
- Atrybut "cena z rabatem" z klasy "Urządzenie" zostanie zamieniony na metodę zwracającą cenę urządzenia po uwzględnieniu rabatu.

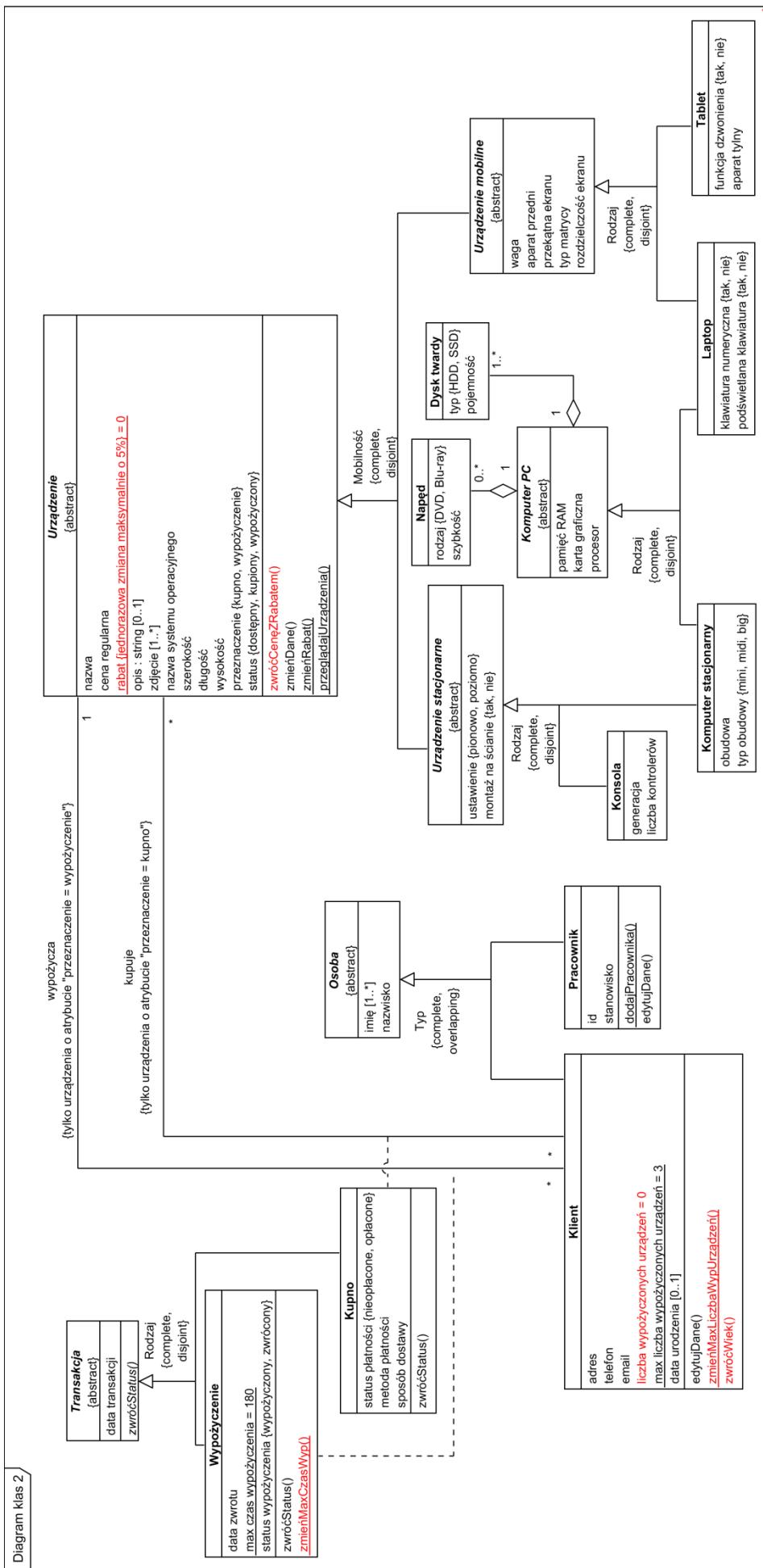
Wartości początkowe:

- Wartość atrybutu "liczba wypożyczonych urządzeń" w momencie tworzenia klasy "Klient" zostanie ustalona na 0.
- Wartość atrybutu "rabat" w momencie tworzenia ekstensji klasy "Urządzenie" zostanie ustalona na 0.

Wartości graniczne:

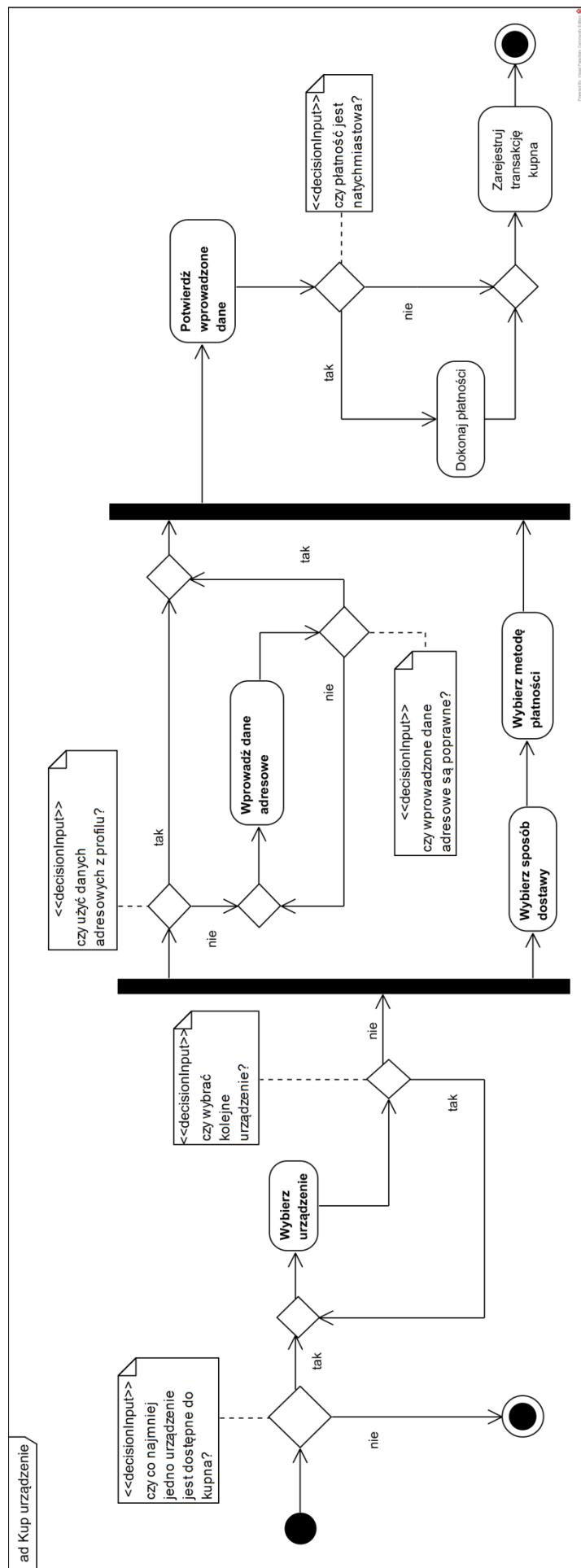
- Dla atrybutu "max czas wypożyczenia" z klasy "Wypożyczenie" dodana zostanie klasowa metoda "zmieńMaxCzasWyp" umożliwiająca zmianę tego atrybutu.
- Dla atrybutu "max liczba wypożyczonych urządzeń" z klasy "Klient" dodana zostanie klasowa metoda "zmieńMaxLiczbaWypUrządzeń" umożliwiająca zmianę tego atrybutu.

Diagram klas po analizie wartości pochodnych, początkowych i granicznych:



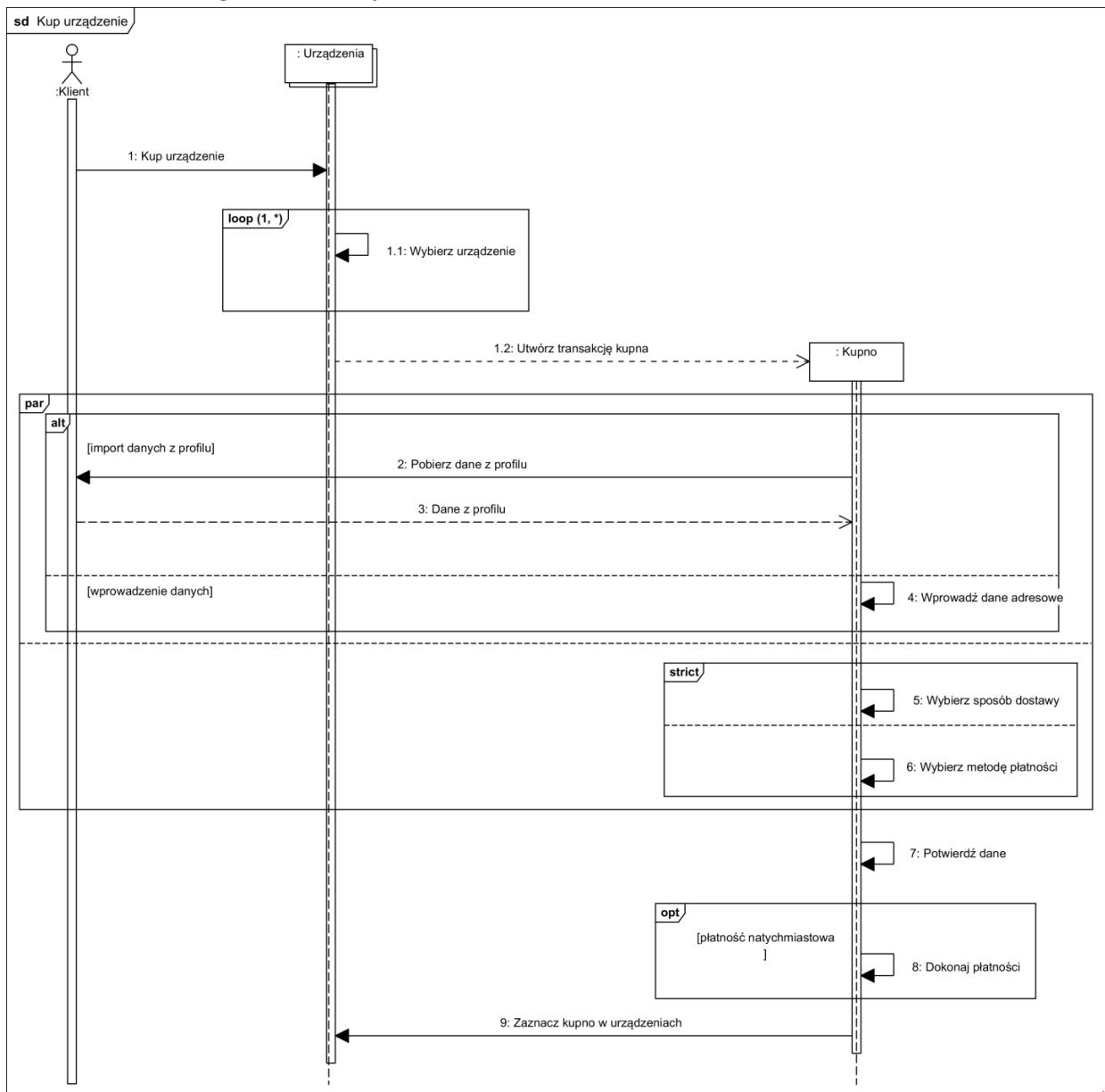
11. Analiza dynamiczna dla wybranego przypadku użycia:

a) diagram aktywności:

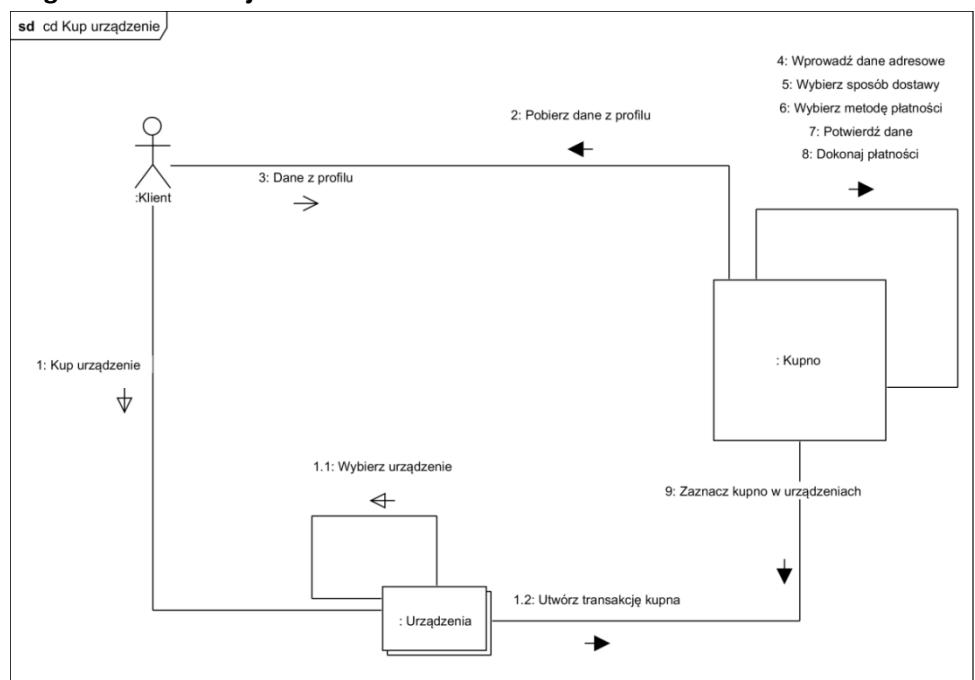


b) diagramy interakcji:

i. diagram sekwencji:



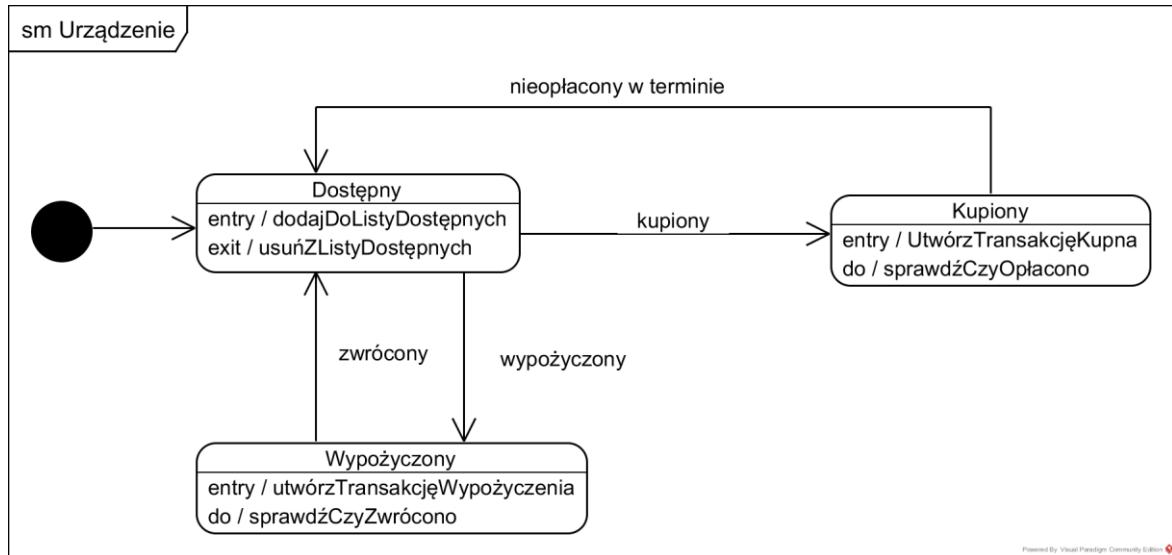
ii. diagram komunikacji:



Na podstawie diagramów aktywności oraz interakcji trzeba będzie uwzględnić następujące elementy:

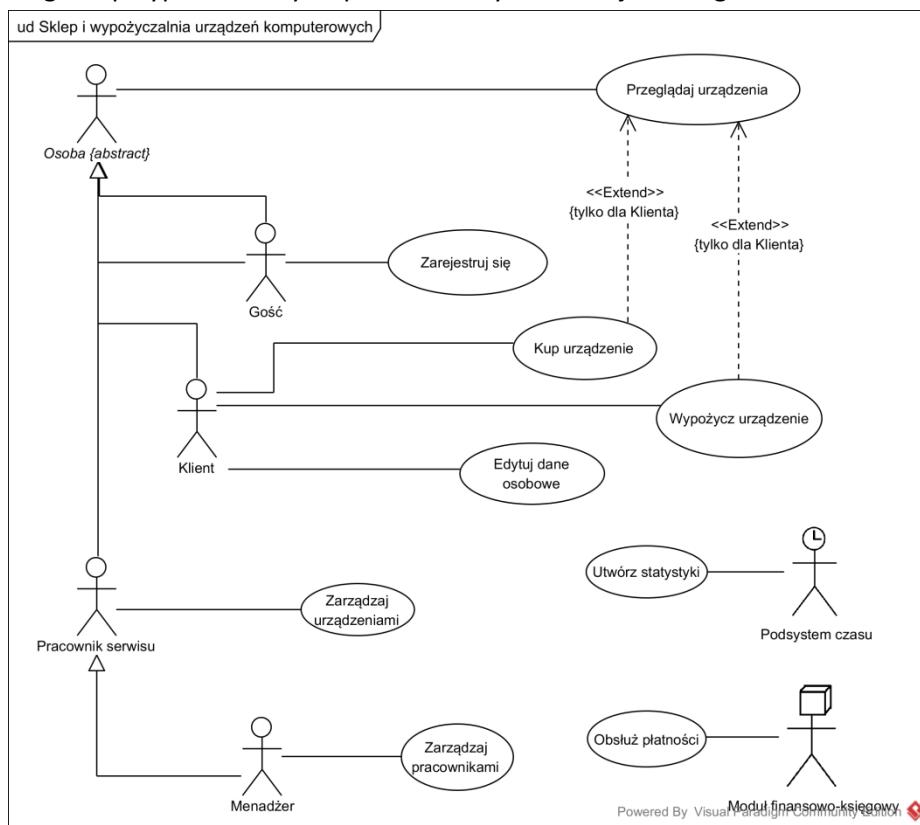
- dodanie atrybutu "dane adresowe" do klasy Kupno
- dodanie metody "WprowadźDaneAdresowe()" do klasy Kupno
- dodanie metody "WybierzSpośobDostawy()" do klasy Kupno
- dodanie metody "WybierzMetodęPłatności()" do klasy Kupno.

12. Analiza dynamiczna dla wybranej klasy obiektów:

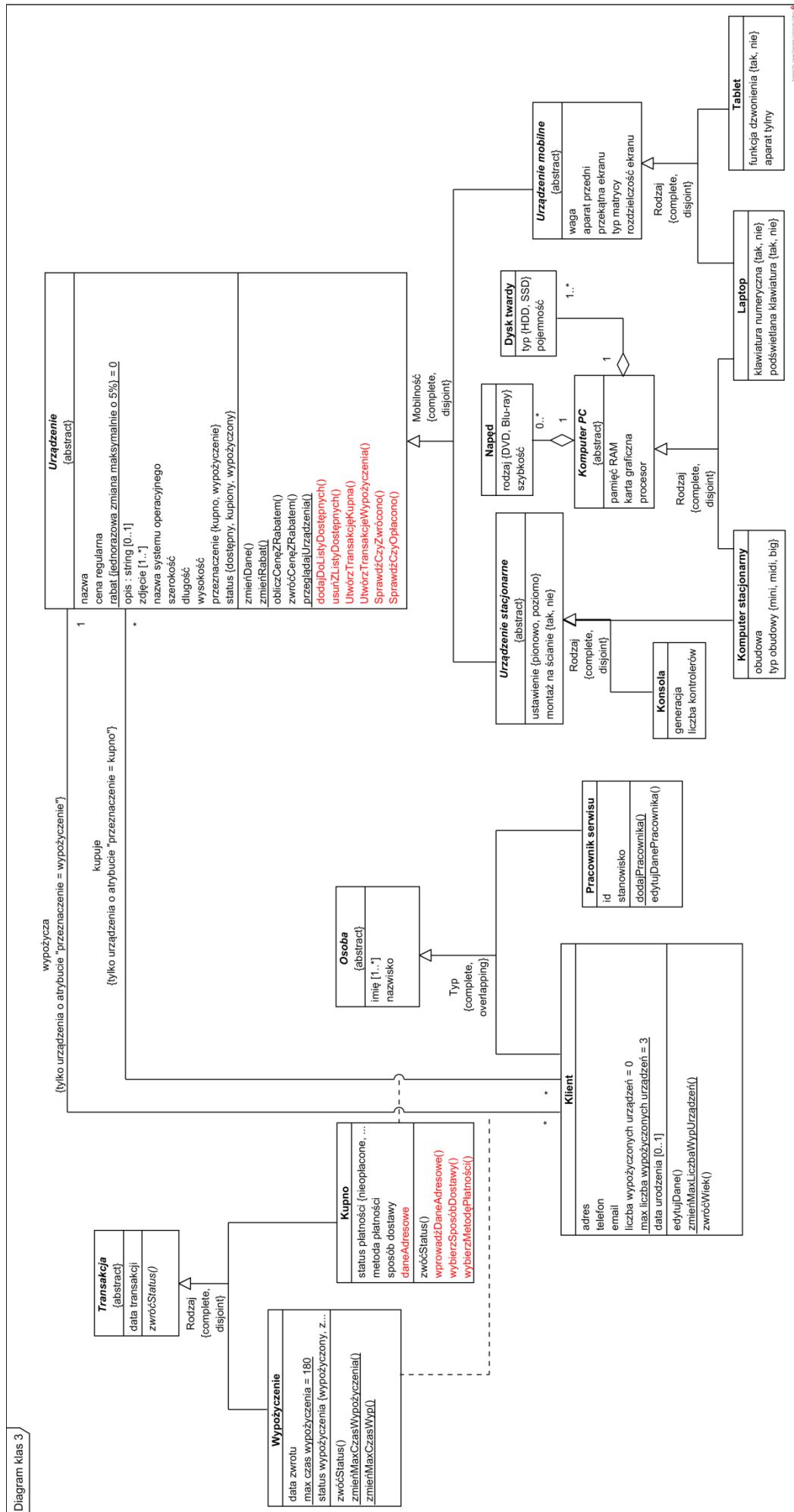


- Do klasy "Urządzenie" dodane zostaną metody "DodajDoListyDostępnych()", "UsuńZListyDostępnych()", "UtwórzTransakcjęKupna()", "UtwórzTransakcjęWypożyczenia()", "SprawdźCzyZwrócono()", "SprawdźCzyOpłacono()".

13. Diagram przypadków użycia po analizie dynamicznej nie uległ zmianie:



14. Schemat pojęciowy po uwzględnieniu analizy dynamicznej:



15. Dokumentacja decyzji projektowych:

Językiem wybranym do realizacji projektu jest C#.

Do reprezentacji atrybutów przechowujących datę zastosowano klasę *DateTime*, będącej częścią biblioteki *System*.

Poniżej został przedstawiony spis problemów, jakie będą oddane decyzjom projektowym:

Klasy	Atrybut/metoda/asocjacja	Problem
Urządzenie	opis	atrybut opcjonalny
Klient	data urodzenia	atrybut opcjonalny
Urządzenie	zdjęcie	atrybut wielokrotny
Klient	imię	atrybut wielokrotny
Urządzenie	rabat	atrybut o zasięgu klasowym
Wypożyczenie	max czas wypożyczenia	atrybut o zasięgu klasowym
Klient	max liczba wypożyczonych urządzeń	atrybut o zasięgu klasowym
Urządzenie	cena z rabatem	atrybut pochodny
Klient	wiek	atrybut pochodny
Osoba, Klient, Pracownik		dziedziczenie nierożłączne
Urządzenie stacjonarne, Urządzenie mobilne, Komputer PC, Komputer stacjonarny, Laptop		dziedziczenie wielokrotne
Urządzenie	PrzeglądajUrządzenia()	metoda klasowa
Urządzenie		ekstensja klasy
Klient		ekstensja klasy
Pracownik		ekstensja klasy
Wypożyczenie		ekstensja klasy
Kupno		ekstensja klasy
Transakcja, Wypożyczenie, Kupno	ZwróćStatus()	operacja polimorficzna
Klient, Wypożyczenie, Urządzenie		asocjacja
Klient, Kupno, Urządzenie		asocjacja
Komputer PC, Napęd		agregacja
Komputer PC, Dysk twardy		agregacja
Urządzenie	rabat	ograniczenie dynamiczne atrybutu
Urządzenie	przeznaczenie	ograniczenie statyczne atrybutu
Urządzenie	status	ograniczenie statyczne atrybutu
Wypożyczenie	status wypożyczenia	ograniczenie statyczne atrybutu
Kupno	status płatności	ograniczenie statyczne atrybutu
Urządzenie stacjonarne	ustawienie	ograniczenie statyczne atrybutu
Urządzenie stacjonarne	montaż na ścianie	ograniczenie statyczne atrybutu
Napęd	rodzaj	ograniczenie statyczne atrybutu
Dysk twardy	typ	ograniczenie statyczne atrybutu
Komputer stacjonarny	typ obudowy	ograniczenie statyczne atrybutu
Laptop	klawiatura numeryczna	ograniczenie statyczne atrybutu
Laptop	podświetlana klawiatura	ograniczenie statyczne atrybutu
Tablet	funkcja dzwonienia	ograniczenie statyczne atrybutu
Klient, Kupno, Urządzenie	kupuje	ograniczenie statyczne asocjacji
Klient, Wypożyczenie, Urządzenie	wypożycza	ograniczenie statyczne asocjacji

Poniżej zostały przedstawione decyzje projektowe, jakie zostały podjęte dla każdego z elementów wymienionych w tabeli powyżej:

- **atrybuty opcjonalne** mają przypisaną wartość `null` w przypadku gdy nie występują:

```
private string opis = null; // atrybut opis z klasy Urządzenie  
  
private DateTime data_urodzenia = null; // atrybut data_urodzenia z klasy Klient
```

- **atrybuty wielokrotne** zostały zaimplementowane w formie kolekcji generycznej `List<T>`, co pozwoli na swobodne dodawanie i usuwanie elementów:

```
private List<string> imię; // atrybut imię z klasy Osoba  
  
private List<string> zdjęcie; // atrybut zdjęcie z klasy Urządzenie
```

- **atrybuty o zasięgu klasowym** zostały zaimplementowane jako atrybuty statyczne z użyciem słowa kluczowego `static`, dzięki czemu dostępne są bez konieczności tworzenia instancji obiektu:

```
private static int rabat = 0; // atrybut rabat z klasy Urządzenie; jego wartość początkowa wynosi 0  
  
private static int max_czas_wypożyczenia = 180; // atrybut max czas wypożyczenia z klasy Wypożyczenie; zgodnie z wymaganiami jego wartość początkowa została ustalona na 180  
  
private static int max_liczba_wypożyczonych_urządzeń = 3; // atrybut max liczba wypożyczonych urządzeń z klasy Klient; zgodnie z wymaganiami wartość początkowa atrybutu została ustalona na 3
```

- **atrybuty pochodne** zostały zastąpione metodami, które obliczają a następnie zwracają wartość pochodną - zamiana została dokonana w punkcie 10 podczas analizy wartości pochodnych, początkowych i granicznych:

```
// metoda ZwróćCenęZRabatem() z klasy Urządzenie  
public double ZwróćCenęZRabatem() {  
    return cena_regularna - rabat * cena_regularna;  
}  
// atrybuty wykorzystywane przez metodę ZwróćCenęZRabatem()  
private double cena_regularna;  
private static int rabat = 0;  
  
////////////////////////////////////////////////////////////////  
  
// metoda ZwróćWiek() z klasy Klient  
public int ZwróćWiek() {  
    if (data_urodzenia == null)  
        return -1;  
    return DateTime.Now.Year - data_urodzenia.Year;  
}  
// atrybuty wykorzystywane przez metodę ZwróćWiek  
private DateTime data_urodzenia = null;
```

W przypadku gdy atrybut opcjonalny `data_urodzenia` nie występuje (ma wartość `null`) metoda `ZwróćWiek()` zwraca wartość -1.

- **ograniczenia statyczne atrybutów** zostaną zaimplementowane na dwa sposoby:
Pierwszy sposób dotyczy atrybutów, które mogą przyjmować tylko wartości tak lub *nie*.
Zostanie nadany im typ *bool*, który może przyjmować wartości *true* lub *false*.

```
// atrybut montaż_na_ścianie z klasy UrządzenieStacjonarne o typie bool
private bool montaż_na_ścianie;

// atrybuty klawiatura_numeryczna oraz podświetlana_klawiatura z klasy Laptop o
// typie bool
private bool klawiatura_numeryczna;
private bool podświetlana_klawiatura;

// atrybut funkcja_dzwonienia z klasy Tablet o typie bool
private bool funkcja_dzwonienia;
```

Drugi sposób dotyczy atrybutów, które mogą przyjmować określona liczbę wartości, którymi nie są tylko wartości *tak* lub *nie*. Te atrybuty zostaną zamienione na atrybuty enumeryczne (typ *enum*).

```
// typy enumeryczne wykorzystywane w klasie Urządzenie
public enum Przeznaczenie { kupno, wypożyczenie };
public enum StatusUrządzenia { dostępny, kupiony, wypożyczony };

// atrybut przeznaczenie z klasy Urządzenie o typie Przeznaczenie
private Przeznaczenie przeznaczenie;
// atrybut status z klasy Urządzenie o typie StatusUrządzenia, atrybut
// przyjmuje wartość początkową równą StatusUrządzenia.dostępny
private StatusUrządzenia status = StatusUrządzenia.dostępny;

///////////////////////////////



// typ enumeryczny wykorzystywany w klasie Wypożyczenie
public enum StatusWypożyczenia { wypożyczony, zwrócony };

// atrybut status_wypożyczenia z klasy Wypożyczenie o typie StatusWypożyczenia
private StatusWypożyczenia status_wypożyczenia;

///////////////////////////////



// typ enumeryczny wykorzystywany w klasie Kupno
public enum StatusPłatności { nieopłacone, opłacone };

// atrybut status_płatności z klasy Kupno o typie StatusPłatności
private StatusPłatności status_płatności;

///////////////////////////////



// typ enumeryczny wykorzystywany w klasie UrządzenieStacjonarne
public enum Ustawienie { pionowo, poziomo };

// atrybut ustawienie z klasy UrządzenieStacjonarne o typie Ustawienie
private Ustawienie ustawienie;

///////////////////////////////
```

```

// typ enumeryczny wykorzystywany w klasie Napęd
public enum RodzajNapędu { DVD, bluray };

// atrybut rodzaj z klasy Napęd o typie RodzajNapędu
private RodzajNapędu rodzaj;

///////////////////////////////
// typ enumeryczny wykorzystywany w klasie DyskTwardy
public enum TypDysku { HDD, SSD };

// atrybut typ z klasy DyskTwardy o typie TypDysku
private TypDysku typ;

/////////////////////////////

```

// typ enumeryczny wykorzystywany w klasie KomputerStacjonarny
 public enum TypObudowy { mini, midi, big };

// atrybut typ_obudowy z klasy KomputerStacjonarny o typie TypObudowy
 private TypObudowy typ_obudowy;

- **ograniczenie dynamiczne** atrybutu rabat z klasy Urządzenie zostało zrealizowane w odpowiedniej metodzie:

```

// metoda ZmieńRabat() ustawiająca wartość atrybutu klasowego Rabat z klasy
Urządzenie; metoda sprawdza, czy nowa wartość rabatu nie różni się więcej
niż o 5% aktualnej wartości rabatu
public static void ZmieńRabat(int nowy_rabat) {
    if(nowy_rabat > this.rabat * 1.05 || nowy_rabat < this.rabat * 0.95)
        throw new Exception("Zmiana rabatu większa niż 5%");
    rabat = nowy_rabat;
}

// atrybut rabat zmieniany przez metodę ZmieńRabat()
private static int rabat = 0;

```

- **operacja polimorficzna:**

```

// metoda abstrakcyjna ZwróćStatus() z klasy Transakcja
public abstract string ZwróćStatus();

```

```

// prześloniona metoda ZwróćStatus() w klasie Wypożyczenie
public override string ZwróćStatus() {
    return status_wypożyczenia.ToString();
}
// zwracany atrybut
private StatusWypożyczenia status_wypożyczenia;

```

```

// prześloniona metoda ZwróćStatus() w klasie Kupno
public override string ZwróćStatus() {
    return status_płatności.ToString();
}
// zwracany atrybut
private StatusPłatności status_płatności;

```

- **metoda klasowa** PrzeglądajUrządzenia() z klasy Urządzenie, działająca na ekstensji klasy Urządzenie została zrealizowana jako publiczna metoda statyczna z użyciem słowa

kluczowego *static*. Jako że wymagane jest, aby można było przeglądać tylko urządzenia do wypożyczenia, tylko do kupna lub oba te typy wykorzystane zostanie przeciążenie tej metody. Wersja nie przyjmująca żadnych argumentów będzie zwracać całą ekstensję klasy Urządzenie, podczas gdy wersja przyjmująca jeden argument (typu *Przeznaczenie*) będzie zwracać albo urządzenia przeznaczone do kupna albo do wypożyczenia.

```
// implementacje metody klasowej PrzeglądajUrządzenia() z klasy Urządzenie

public static List<Urządzenie> PrzeglądajUrządzenia()
{
    List<Urządzenie> l = new List<Urządzenie>();
    foreach (var item in ekstensja)
        l.Add(item.Value);
    return l;
}
public static List<Urządzenie> PrzeglądajUrządzenia(Przeznaczenie
przeznaczenie)
{
    List<Urządzenie> l = new List<Urządzenie>();
    foreach (var item in ekstensja)
    {
        if(item.Value.Przeznaczenie == przeznaczenie)
            l.Add(item.Value);
    }
    return l;
}
```

- **ekstensje klas** zostaną zrealizowane w klasach biznesowych, zostaną zaimplementowane jako atrybuty klasowe z użyciem słowa kluczowego *static* jako mapującą kolekcja generyczna *Dictionary<int, TValue>*. Przy realizacji ekstensji zostaną użyte dodatkowe metody umożliwiające dodanie, usunięcie oraz wyszukanie obiektu w ekstensji. Te metody zostaną zaimplementowane jako metody klasowe z użyciem słowa kluczowego *static*.

Fragmenty kodu realizującego ekstensję dla każdej z klas, które jej wymagają:

```
// implementacja ekstensji klasy Urządzenie
private static Dictionary<int, Urządzenie> ekstensja = new Dictionary<int,
Urządzenie>();

// metody dodająca, usuwająca i wyszukująca elementy w ekstensji klasy
Urządzenie
private static void DodajUrządzenie(int id, Urządzenie urządzenie) {
    ekstensja.Add(id, urządzenie);
}
private static void UsuńUrządzenie(int id) {
    ekstensja.Remove(id);
}
public static Urządzenie ZnajdźUrządzenie(int id)
{
    return ekstensja[id];
}
// konstruktor klasy Urządzenie
public Urządzenie(int id, string nazwa, double cena_regulararna, string opis,
List<string> zdjęcie, string nazwa_systemu_operacyjnego, int szerokość,
int długość, int wysokość, string przeznaczenie, int[] kupnaIds, int[]
wypożyczalniaIds)
{
    this.id = id;
    this.nazwa = nazwa;
    this.cena_regulararna = cena_regulararna;
    this.opis = opis;
    this.zdjęcie = zdjęcie;
    this.nazwa_systemu_operacyjnego = nazwa_systemu_operacyjnego;
```

```

        this.szerokość = szerokość;
        this.długość = długość;
        this.wysokość = wysokość;
        this.przeznaczenie = przeznaczenie;
        this.kupna = kupnaIds;
        this.wypożyczenia = wypożyczeniaIds;
        DodajUrządzenie(this); // dodanie do ekstensji
    }

    /////////////////////////////////



// implementacja ekstensji klasy Osoba - tutaj wykorzystana została kolekcja
generyczna List<T>
private static List<Osoba> ekstensja = new List<Osoba>();

// metody dodająca i usuwająca elementy z ekstensji klasy Osoba
private static void DodajOsobę(Osoba osoba)
{
    ekstensja.Add(osoba);
}
private static void UsuńOsobę(Osoba osoba)
{
    ekstensja.Remove(osoba);
}

// konstruktor klasy Osoba
public Osoba(List<string> imię, string nazwisko) {
    this.imię = imię;
    this.nazwisko = nazwisko;

    DodajOsobę(this);
}

    /////////////////////////////////



// implementacja ekstensji klasy Klient
private static Dictionary<int, Klient> ekstensja = new Dictionary<int, Klient>();

// metody dodająca, usuwająca i wyszukującą elementy w ekstensji klasy
Klient
private static void DodajKlienta(int id, Klient klient)
{
    ekstensja.Add(id, klient);
}
private static void UsuńKlienta(int id)
{
    ekstensja.Remove(id);
}
public static Klient ZnajdźKlienta(int id) {
    return ekstensja[id];
}

// konstruktor klasy Klient
private Klient(Osoba osoba, int id, string adres, int telefon, string email,
              int[] kupnaIds, int[] wypożyczeniaIds)
{
    this.osoba = osoba;
    this.id = id;
    this.adres = adres;
    this.telefon = telefon;
    this.email = email;
    this.kupna = kupnaIds;
    this.wypożyczenia = wypożyczeniaIds;
}

```

```

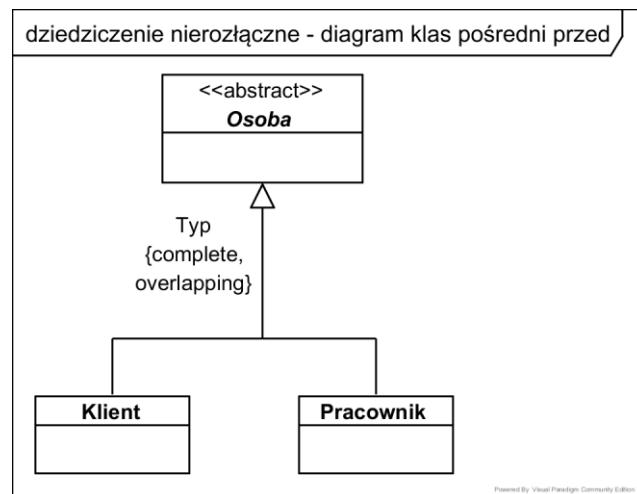
        DodajKlienta(id, this); // dodanie do ekstensji
    }

```

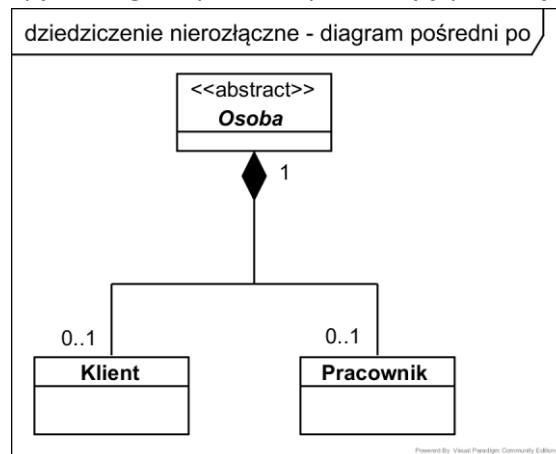
Analogicznie zrealizowano ekstensje w klasach Pracownik, Wypożyczenie oraz Kupno.

- **dziedziczenie nierożłączne (overlapping)** zostało zaimplementowane przy użyciu kompozycji

Poniżej przedstawiony został uproszczony (zawierający jedynie rozpatrywane klasy) diagram klas prezentujący problem:



Następnie przedstawiony jest diagram pośredni prezentujący rozwiązanie problemu:



Kolejnym krokiem jest rozwiązanie problemu, który powstał na diagramie pośrednim, tj. **implementacja kompozycji**. Podczas realizacji zadano o to, aby nie można było tworzyć samodzielnych obiektów części, aby obiekty całości nie współdzieliły obiektów części oraz aby części były usuwane wraz z całością.

```

// realizacja kompozycji w klasie całości Osoba, której częściami są obiekty
// klas Klient oraz Pracownik

private Klient klient;
private Pracownik pracownik;

private static List<Klient> wszyscyKlienci = new List<Klient>();
private static List<Pracownik> wszyscyPracownicy = new List<Pracownik>();

public void DodajKlienta(Klient klient) {
    if(wszyscyKlienci.Contains(klient))
        throw new Exception("Klient należy już do innej osoby");
    this.klient = klient;
}

```

```

        wszyscyKlienci.Add(klient);
    }
    public void DodajPracownika(Pracownik pracownik) {
        if(wszyscyPracownicy.Contains(pracownik))
            throw new Exception("Pracownik należy już do innej osoby");
        this.pracownik = pracownik;
        wszyscyPracownicy.Add(pracownik);
    }

    // realizacja kompozycji w klasie części Klient
    private Osoba osoba;

    private Klient(Osoba osoba, int id, string adres, int telefon, string email,
                  DateTime data_urodzenia)
    {
        this.osoba = osoba;
        this.id = id;
        this.adres = adres;
        this.telefon = telefon;
        this.email = email;
        this.data_urodzenia = data_urodzenia;
        DodajKlienta(this);
    }

    public static Klient UtwórzKlienta(Osoba osoba, int id, string adres, int
                                         telefon, string email, DateTime data_urodzenia)
    {
        if(osoba == null)
            throw new Exception("Osoba nie istnieje");
        Klient k = new Klient(osoba, id, adres, telefon, email, data_urodzenia);
        osoba.DodajKlienta(k);
        return k;
    }

    // realizacja kompozycji w klasie części Pracownik
    private Osoba osoba;

    private Pracownik(Osoba osoba, int id, string stanowisko)
    {
        this.osoba = osoba;
        this.id = id;
        this.stanowisko = stanowisko;
        DodajPracownika(this);
    }

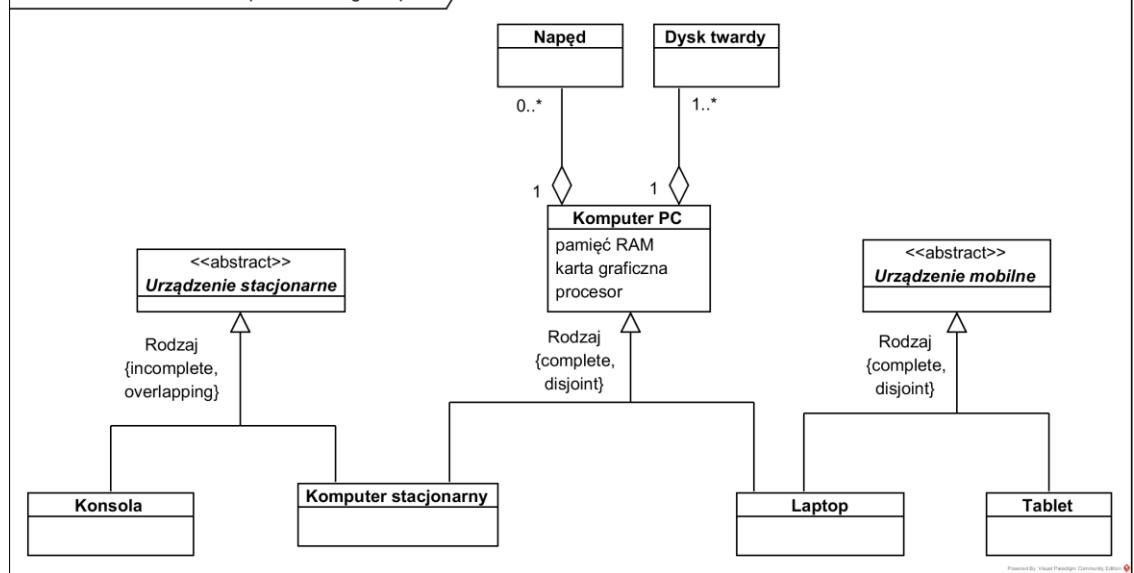
    public static Pracownik UtwórzPracownika(Osoba osoba, int id, string
                                              stanowisko)
    {
        if(osoba == null)
            throw new Exception("Osoba nie istnieje");
        Pracownik p = new Pracownik(osoba, id, stanowisko);
        osoba.DodajPracownika(p);
        return p;
    }
}

```

- **dziedziczenie wielokrotne**

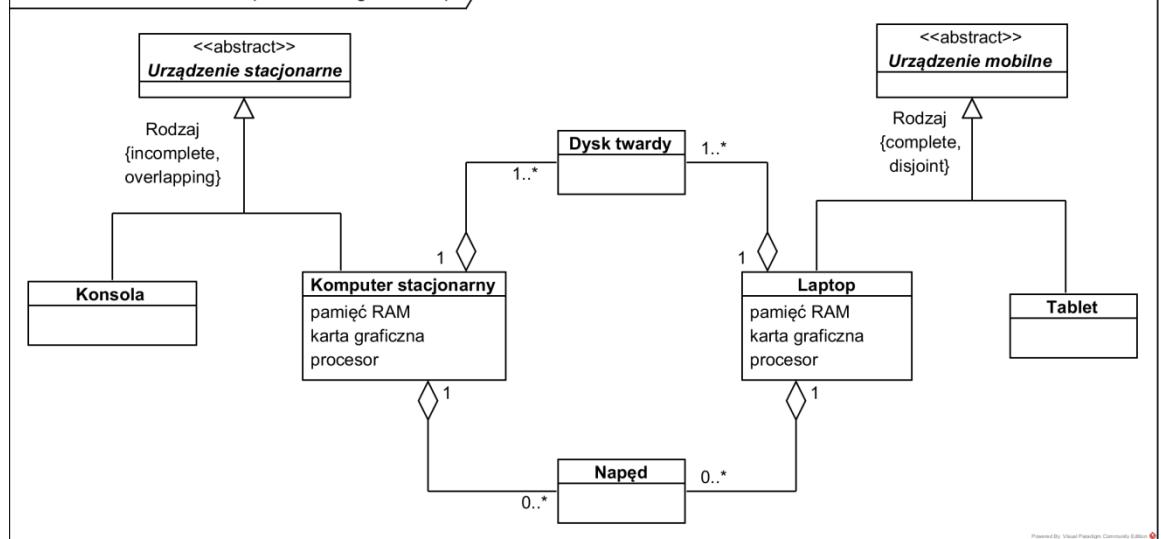
Poniżej przedstawiony został uproszczony (zawierający jedynie rozpatrywane klasy) diagram klas prezentujący problem:

dziedziczenie wielokrotne - pośredni diagram przed



Następnie przedstawiony jest diagram pośredni prezentujący rozwiązanie problemu:

dziedziczenie wielokrotne - pośredni diagram klas po



Klasa Komputer PC została usunięta. Jej atrybuty zostały przeniesione do klas, które po niej dziedziczyły, tj. Komputer stacjonarny i Laptop. Agregacje klasy Komputer PC z klasami Napęd oraz Dysk twardy zostaną zamienione na agregacje klas Napęd i Dysk twardy z klasami Komputer stacjonarny i Laptop. W ten sposób wyeliminowane zostanie dziedziczenie wielokrotne.

• asocjacje

Asocjacje zostały zaimplementowane za pomocą identyfikatorów. Zarówno asocjacja "Klient wypożycza urządzenie" jak i asocjacja "Klient kupuje urządzenie" będą asocjacjami obustronnymi, gdyż chcemy wiedzieć jakie urządzenia kupił i wypożyczył dany klient oraz kto wypożyczył lub kupił dane urządzenie.

Diagram pośredni prezentujący asocjację "Klient wypożycza urządzenie" z klasą asocjacji Wypożyczenie:

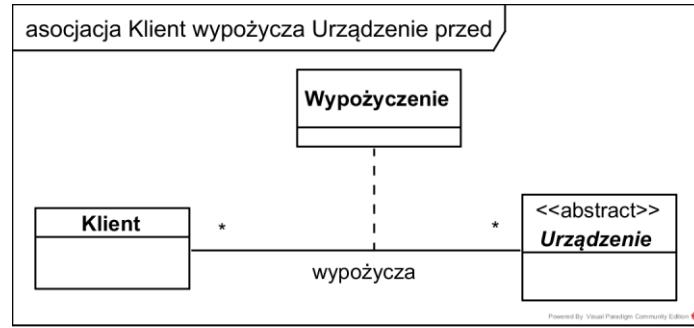
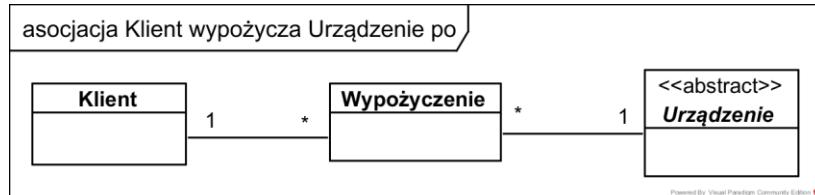


Diagram pośredni prezentujący asocjację "Klient wypożycza urządzenie" po transformacji do diagramu z klasą pośredniczącą Wypożyczenie:



Implementacja asocjacji "Klient wypożycza urządzenie":

```
// implementacja w klasie Klient
private int id;
private int[] wypożyczenia; // klient może mieć wiele wypożyczeń
private static Dictionary<int, Klient> ekstensja = new Dictionary<int, Klient>(); // ekstensja klasy Klient

private Klient(Osoba osoba, int id, string adres, int telefon, string email,
               int[] kupnaIds, int[] wypożyczeniaIds)
{
    this.osoba = osoba;
    this.id = id;
    this.adres = adres;
    this.telefon = telefon;
    this.email = email;
    this.kupna = kupnaIds;
    this.wypożyczenia = wypożyczeniaIds;
    DodajKlienta(id, this); // dodanie do ekstensji
}
public static Klient ZnajdźKlienta(int id) {
    return ekstensja[id];
}

// implementacja w klasie Wypożyczenie
private int id; // w klasie bazowej Transakcja
private int urządzenie; // jedno urządzenie w ramach jednego wypożyczenia
private int klient; // jeden klient w ramach jednego wypożyczenia
private static Dictionary<int, Wypożyczenie> ekstensja = new Dictionary<int, Wypożyczenie>(); // ekstensja klasy Wypożyczenie

public Wypożyczenie(int id, DateTime data_transakcji, string
                     status_wypożyczenia, int urządzenieId, int klientId) : base(id,
                     data_transakcji)
{
    this.status_wypożyczenia = status_wypożyczenia;
    this.urządzenie = urządzenieId;
    this.klient = klientId;
    DodajWypożyczenie(id, this); // dodanie do ekstensji
}
```

```

public static Wypożyczenie ZnajdźWypożyczenie(int id)
{
    return ekstensja[id];
}

// implementacja w klasie Urządzenie
private int id;
private int[] wypożyczenia; // urządzenie może być wypożyczone wiele razy
private static Dictionary<int, Urządzenie> ekstensja = new Dictionary<int,
    Urządzenie>(); // ekstensja klasy Urządzenie

public Urządzenie(int id, string nazwa, double cena_regularna, string opis,
List<string> zdjęcie, string nazwa_systemu_operacyjnego, int szerokość,
int długość, int wysokość, string przeznaczenie, int[] kupnaIds, int[]
wypożyczeniaIds)
{
    this.id = id;
    this.nazwa = nazwa;
    this.cena_regularna = cena_regularna;
    this.opis = opis;
    this.zdjęcie = zdjęcie;
    this.nazwa_systemu_operacyjnego = nazwa_systemu_operacyjnego;
    this.szerokość = szerokość;
    this.długość = długość;
    this.wysokość = wysokość;
    this.przeznaczenie = przeznaczenie;
    this.kupna = kupnaIds;
    this.wypożyczenia = wypożyczeniaIds;
    DodajUrządzenie(this);
}

public static Urządzenie ZnajdźUrządzenie(int id)
{
    return ekstensja[id];
}

```

Diagram pośredni prezentujący asocjację "Klient kupuje urządzenie" z klasą asocjacji Kupno:

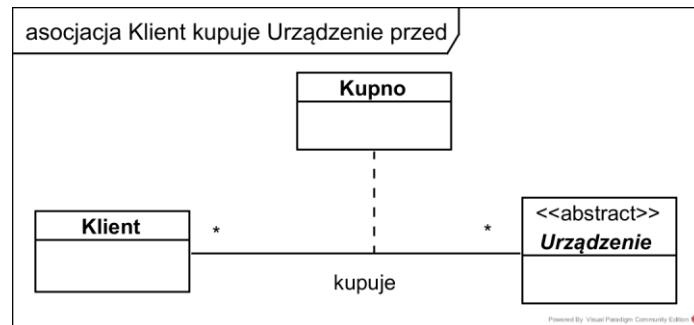
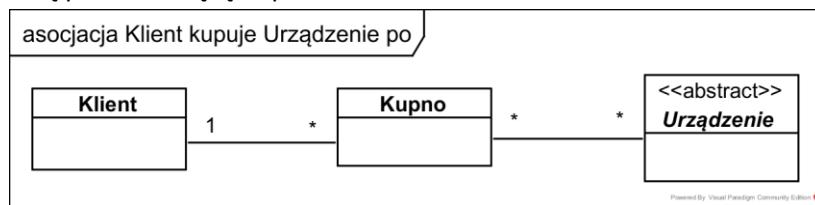


Diagram pośredni prezentujący asocjację "Klient kupuje urządzenie" po transformacji do diagramu z klasą pośredniczącą Kupno:



Implementacja asocjacji "Klient kupuje urządzenie":

```

// implementacja w klasie Klient
private int id;
private int[] kupna; // klient może złożyć wiele transakcji kupna

```

```

private static Dictionary<int, Klient> ekstensja = new Dictionary<int,
    Klient>(); // ekstensja klasy Klient

private Klient(Osoba osoba, int id, string adres, int telefon, string email,
    int[] kupnaIds, int[] wypożyczeniaIds)
{
    this.osoba = osoba;
    this.id = id;
    this.adres = adres;
    this.telefon = telefon;
    this.email = email;
    this.kupna = kupnaIds;
    this.wypożyczenia = wypożyczeniaIds;
    DodajKlienta(id, this); // dodanie do ekstensji
}
public static Klient ZnajdźKlienta(int id) {
    return ekstensja[id];
}

// implementacja w klasie Kupno
private int id; // w klasie bazowej Transakcja
private int[] urządzenia; // wiele urządzeń w ramach jednej transakcji kupna
private int klient; // jeden klient w ramach jednej transakcji kupna
private static Dictionary<int, Kupno> ekstensja = new Dictionary<int, Kupno>();
    // ekstensja klasy Kupno

public Kupno(int id, DateTime data_transakcji, StatusPłatności
    status_płatności, string metoda_płatności, string sposób_dostawy, string
    daneAdresowe, int[] urządzeniaIds, int klientId) : base(id,
    data_transakcji)
{
    this.status_płatności = status_płatności;
    this.metoda_płatności = metoda_płatności;
    this.sposób_dostawy = sposób_dostawy;
    this.daneAdresowe = daneAdresowe;
    this.urządzenia = urządzeniaIds;
    this.klient = klientId;
    DodajKupno(id, this); // dodanie do ekstensji
}
public static Kupno ZnajdźKupno(int id)
{
    return ekstensja[id];
}

// implementacja w klasie Urządzenie
private int id;
private int[] kupna; // urządzenie może być w wielu transakcjach kupna
(zamówienia niezrealizowane)
private static Dictionary<int, Urządzenie> ekstensja = new Dictionary<int,
    Urządzenie>(); // ekstensja klasy Urządzenie

public Urządzenie(int id, string nazwa, double cena_regulararna, string opis,
    List<string> zdjécie, string nazwa_systemu_operacyjnego, int szerokość,
    int długość, int wysokość, string przeznaczenie, int[] kupnaIds, int[]
    wypożyczeniaIds)
{
    this.id = id;
    this.nazwa = nazwa;
    this.cena_regulararna = cena_regulararna;
    this.opis = opis;
    this.zdjęcie = zdjécie;
    this.nazwa_systemu_operacyjnego = nazwa_systemu_operacyjnego;
    this.szerokość = szerokość;
    this.długość = długość;
}

```

```

        this.wysokość = wysokość;
        this.przeznaczenie = przeznaczenie;
        this.kupna = kupnaIds;
        this.wypożyczenia = wypożyczeniaIds;
        DodajUrządzenie(this);
    }

    public static Urządzenie ZnajdźUrządzenie(int id)
    {
        return ekstensja[id];
    }
}

```

- **ograniczenia statyczne asocjacji**

implementacja ograniczenia asocjacji Klient wypożycza Urządzenie {tylko urządzenia o atrybucie przeznaczenie = Przeznaczenie.Wypożyczenie}

```

public Wypożyczenie(int id, DateTime data_transakcji, string
    status_wypożyczenia, int urządzenieId, int klientId) : base(id,
    data_transakcji)
{
    this.status_wypożyczenia = status_wypożyczenia;
    // sprawdzenie czy dodawane urządzenie ma odpowiedni status
    if (Urządzenie.ZnajdźUrządzenie(urządzenieId).Przeznaczenie != Przeznaczenie.Wypożyczenie)
        throw new Exception("Urządzenie nie jest przeznaczone do
            wypożyczenia");
    this.urządzenie = urządzenieId;
    this.klient = klientId;
    DodajWypożyczenie(id, this); // dodanie do ekstensji
}

```

implementacja ograniczenia asocjacji Klient kupuje Urządzenie {tylko urządzenia o atrybucie przeznaczenie = Przeznaczenie.Kupno }

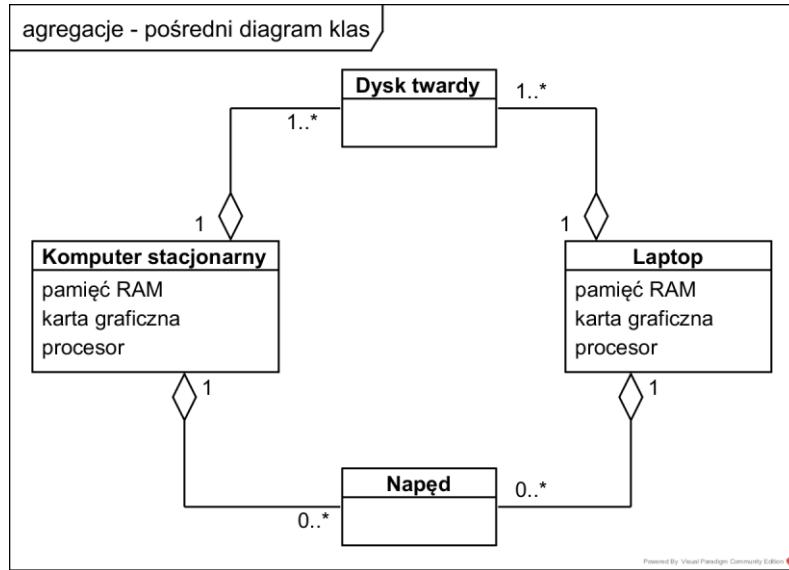
```

public Kupno(int id, DateTime data_transakcji, StatusPłatności
    status_płatności, string metoda_płatności,
    string sposób_dostawy, string daneAdresowe, int[] urządzeniaIds, int
    klientId) : base(id, data_transakcji)
{
    this.status_płatności = status_płatności;
    this.metoda_płatności = metoda_płatności;
    this.sposób_dostawy = sposób_dostawy;
    this.daneAdresowe = daneAdresowe;
    // sprawdzenie czy dodawane urządzenia mają odpowiedni status
    foreach (int item in urządzenia)
        if (Urządzenie.ZnajdźUrządzenie(urządzenieId).Przeznaczenie != Przeznaczenie.Kupno)
            throw new Exception("Niektóre z urządzeń nie są
                przeznaczone do kupna");
    this.urządzenia = urządzeniaIds;
    this.klient = klientId;
    DodajKupno(id, this); // dodanie do ekstensji
}

```

- **agregacje**

Po wyeliminowaniu dziedziczenia wielokrotnego liczba agregacji do zrealizowania zmieniła się z dwóch do czterech:



Przy implementacji agregacji wykorzystany będzie kontener `List<T>` przechowujący informacje o wszystkich częściach powiązanych już z całościami. Potrzebne będą dwa takie kontenery, jeden dla napędów, drugi dla dysków. Jako że w tym wypadku mamy dwie klasy całości, kontenery te zostaną umieszczone tylko w jednej z nich, a konkretnie w klasie Komputer stacjonarny. Klasa Laptop będzie miała dostęp do tych kontenerów za pomocą właściwości `get` zaimplementowanych w klasie Komputer stacjonarny.

```

// realizacja agregacji w klasie części Napęd

private Urządzenie urządzenie;

private Napęd(Urządzenie urządzenie, RodzajNapędu rodzaj, int szybkość) {
    this.urządzenie = urządzenie;
    this.rodzaj = rodzaj;
    this.szybkość = szybkość;
}
public Napęd UtwórzNapęd(Urządzenie urządzenie, RodzajNapędu rodzaj, int szybkość)
{
    if(urządzenie == null)
        throw new Exception("Urządzenie nie istnieje");
    Napęd n = new Napęd(urządzenie, rodzaj, szybkość);
    urządzenie.DodajNapęd(n);
    return n;
}

// realizacja agregacji w klasie części DyskTwardy

private Urządzenie urządzenie;

private DyskTwardy(Urządzenie urządzenie, TypDysku typ, int pojemność) {
    this.urządzenie = urządzenie;
    this.typ = typ;
    this.pojemność = pojemność;
}
public DyskTwardy UtwórzDysk(Urządzenie urządzenie, TypDysku rodzaj, int szybkość)
{
    if(urządzenie == null)
        throw new Exception("Urządzenie nie istnieje");
    DyskTwardy d = new Dysk_twardy(urządzenie, rodzaj, pojemność);
    urządzenie.DodajDysk(d);
    return d;
}
  
```

```

// realizacja agregacji w klasie całości KomputerStacjonarny

private List<DyskTwardy> dyski = new List<DyskTwardy>();
private List<Napęd> napędy = new List<Napęd>();

private static List<DyskTwardy> wszystkieDyski = new List<DyskTwardy>();
private static List<Napęd> wszystkieNapędy = new List<Napęd>();


public static List<DyskTwardy> WszystkieDyski
{
    get
    {
        return wszystkieDyski;
    }
}

public static List<Napęd> WszystkieNapędy
{
    get
    {
        return wszystkieNapędy;
    }
}

public void DodajDysk(DyskTwardy dysk) {
    if (!dyski.Contains(dysk))
    {
        if (wszystkieDyski.Contains(dysk))
            throw new Exception("Ten dysk jest już powiązany z innym
                                  urządzeniem");
        dyski.Add(dysk);
        wszystkieDyski.Add(dysk);
    }
}

public void DodajNapęd(Napęd napęd) {
    if (!napędy.Contains(napęd))
    {
        if (wszystkieNapędy.Contains(napęd))
            throw new Exception("Ten napęd jest już powiązany z innym
                                  urządzeniem");
        napędy.Add(napęd);
        wszystkieNapędy.Add(napęd);
    }
}

// realizacja agregacji w klasie całości KomputerStacjonarny

private List<DyskTwardy> dyski = new List<DyskTwardy>();
private List<Napęd> napędy = new List<Napęd>();

public void DodajDysk(DyskTwardy dysk)
{
    if (!dyski.Contains(dysk))
    {
        if (KomputerStacjonarny.WszystkieDyski.Contains(dysk))
            throw new Exception("Ten dysk jest już powiązany z innym
                                  urządzeniem");
        dyski.Add(dysk);
        KomputerStacjonarny.WszystkieDyski.Add(dysk);
    }
}

public void DodajNapęd(Napęd napęd)
{
    if (!napędy.Contains(napęd))
    {

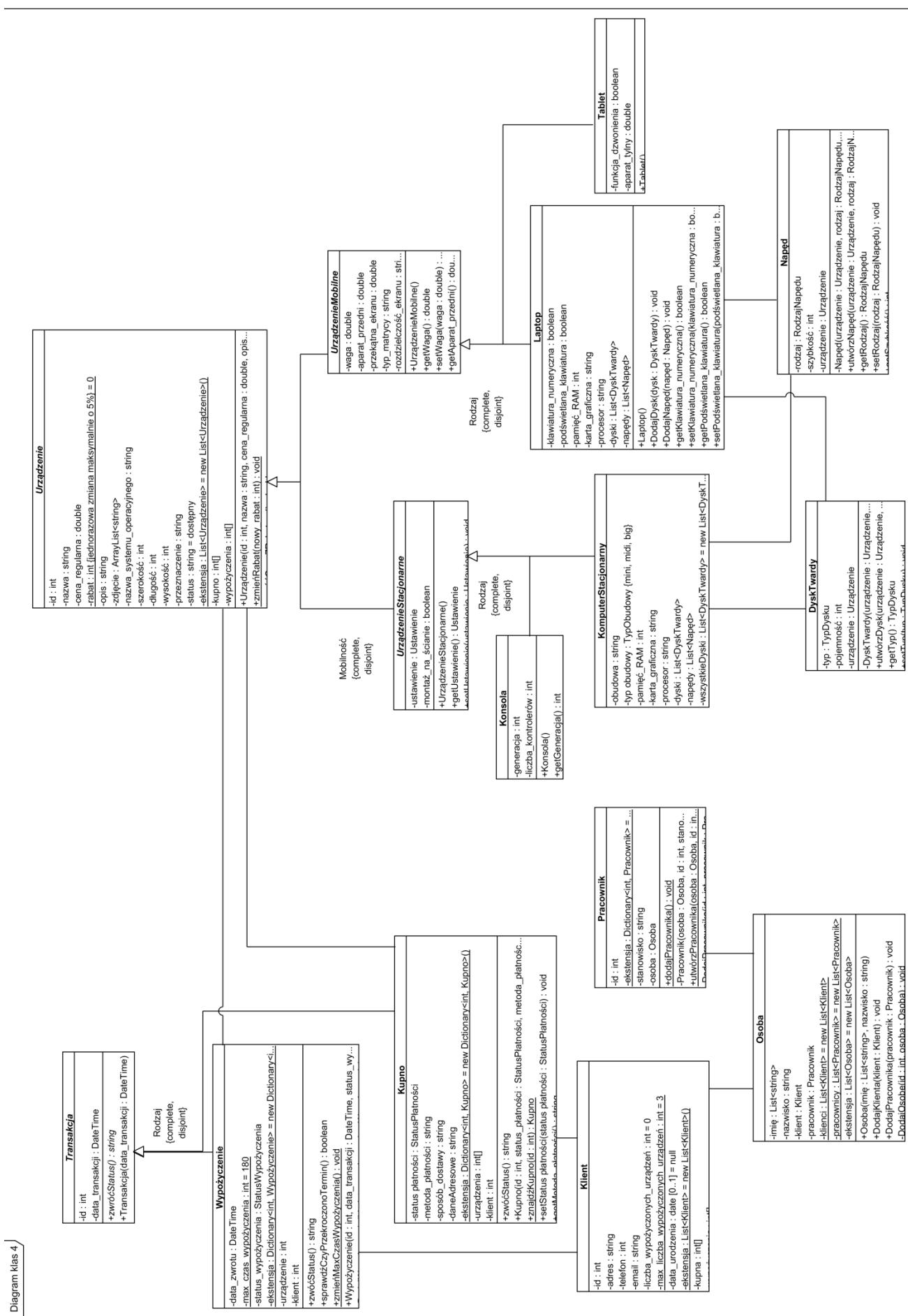
```

```
        if (KomputerStacjonarny.WszystkieNapędy.Contains(napęd))
            throw new Exception("Ten napęd jest już powiązany z innym
                                  urządzeniem");
        napędy.Add(napęd);
        KomputerStacjonarny.WszystkieNapędy.Add(napęd);
    }
}
```

- **metody zmieniające lub ustalające wartości atrybutów** zostaną zamienione na właściwości ustawiające *set*
- **w celu dostępu do niektórych prywatnych atrybutów** zostaną dodane właściwości *get*, które będą je zwracać
- **metody mające na celu tworzenie obiektów** zostaną usunięte, gdyż tę rolę będzie pełnił konstruktor danej klasy

16. Schemat logiczny po uwzględnieniu decyzji projektowych:

Widok ogólny:



Szczegółowe klasy:

Transakcja
-id : int
-data_transakcji : DateTime
+zwróćStatus() : string
+Transakcja(data_transakcji : DateTime)

Wypożyczenie
<p>-data_zwrotu : DateTime <u>-max_czas_wypożyczenia : int = 180</u> <u>-status_wypożyczenia : StatusWypożyczenia</u> <u>-ekstensja : Dictionary<int, Wypożyczenie> = new Dictionary<int, Wypożyczenie>()</u> <u>-urządzenie : int</u> <u>-klient : int</u></p> <p>+zwóćStatus() : string +sprawdźCzyPrzekroczenoTermin() : boolean +zmieńMaxCzasWypożyczenia() : void +Wypożyczenie(id : int, data_transakcji : DateTime, status_wypożyczenia : StatusWypożyczenia, urządzeniId : int, klientId : int) -DodaWypożyczenie(id : int, wypożyczenie : Wypożyczenie) : void -USuńWypożyczenie(id : int) : void +ZnajdźWypożyczenie(id : int) : Wypożyczenie +getData_zwrotu() : DateTime +setData_zwrotu(data_zwrotu : DateTime) : void +setStatus_wypożyczenia(status_wypożyczenia : StatusWypożyczenia) : void +getMax_czas_wypożyczenia() : int</p>

Kupno
<p>-status_płatności : StatusPłatności -metoda_płatności : string -sposób_dostawy : string -daneAdresowe : string -ekstensja : Dictionary<int, Kupno> = new Dictionary<int, Kupno>() -urządzenia : int[] -klient : int</p> <p>+zwóćStatus() : string +Kupno(id : int, status_płatności : StatusPłatności, metoda_płatności : string, sposób_dostawy : string, daneAdresowe : string, urządzenia : int[], klient : int) +znajdźKupno(id : int) : Kupno +setStatus_płatności(status_płatności : StatusPłatności) : void +getMetoda_płatności() : string +setMetoda_płatności(metoda_płatności : string) : void +getSposób_dostawy() : string +setSposób_dostawy(sposób_dostawy : string) : void +getDaneAdresowe() : string +setDaneAdresowe(daneAdresowe : string) : void -DodaKupno(id : int, kupno : Kupno) : void -USuńKupno(id : int) : void +ZnajdźKupno(id : int) : Kupno</p>

Osoba
<p>-imię : List<string> -nazwisko : string -klient : Klient -pracownik : Pracownik -kienci : List<Klient> = new List<Klient> -pracownicy : List<Pracownik> = new List<Pracownik> -ekstensja : List<Osoba> = new List<Osoba></p> <p>+Osoba(imię : List<string>, nazwisko : string) +DodaKlienta(klient : Klient) : void +DodajPracownika(pracownik : Pracownik) : void -DodaOsobę(id : int, osoba : Osoba) : void -USuńOsobę(id : int) : void +ZnajdźOsobę(id : int) : Osoba +getImię() : List<string> +setImię(imię : List<string>) : void +getNazwisko() : string +setNazwisko(nazwisko : string) : void +getKlient() : Klient +getPracownik() : Pracownik</p>

Klient
<p>-id : int -adres : string -telefon : int -email : string -liczba_wypożyczonych_urządzeń : int = 0 <u>-max_liczba_wypożyczonych_urządzeń : int = 3</u> <u>-data_urodzenia : date [0..1] = null</u> <u>-ekstensja : List<Klient> = new List<Klient>()</u> <u>-kupna : int[]</u> <u>-wypożyczenia : int[]</u> <u>-osoba : Osoba</u></p> <p><u>+zmieńMaxLiczbaWypUrządzeń() : void</u> <u>+zwróćWiek() : int</u> <u>-Klient(osoba : Osoba, id : int, adres : string, telefon : int, email : string, kupnalds : int[], wypożyczenials : int[])</u> <u>+utwórzKlienta(osoba : Osoba, id : int, adres : string, telefon : int, email : string, kupnalds : int[], wypożyczenials : int[]) : Klient</u> <u>-DodajKlienta(id : int, klient : Klient) : void</u> <u>-UsuńKlienta(id : int) : void</u> <u>+znajdźKlienta(id : int) : Klient</u> <u>+getAdres() : string</u> <u>+setAdres(adres : string) : void</u> <u>+getTelefon() : int</u> <u>+setTelefon(telefon : int) : void</u> <u>+getEmail() : string</u> <u>+setEmail(email : string) : void</u> <u>+getKupna() : int[]</u> <u>+setKupna(kupna : int[]) : void</u> <u>+getWypożyczenia() : int[]</u> <u>+setWypożyczenia(wypożyczenia : int[]) : void</u> <u>+getData_urodzenia() : date</u> <u>+setData_urodzenia(data_urodzenia : date) : void</u> <u>+getMax_liczba_wypożyczonych_urządzeń() : int</u></p>

Pracownik
<p>-id : int <u>-ekstensja : Dictionary<int, Pracownik> = new Dictionary<int, Pracownik>()</u> -stanowisko : string -osoba : Osoba</p> <p><u>+dodajPracownika() : void</u> <u>-Pracownik(osoba : Osoba, id : int, stanowisko : string)</u> <u>+utwórzPracownika(osoba : Osoba, id : int, stanowisko : string) : Pracownik</u> <u>-DodajPracownika(id : int, pracownik : Pracownik) : void</u> <u>-UsuńPracownika(id : int) : void</u> <u>+znajdźPracownika(id : int) : Pracownik</u></p>

<i>Urządzenie</i>
<pre> -id : int -nazwa : string -cena_regularna : double -cetak : int fiedorazowa zmiana maksymalnie o 5% = 0 -opis : string -zdjęcie : Array<string> -nazwa_systemu_operacyjnego : string -szerokość : int -długość : int -wysokość : int -przeznaczenie : string -status : string = dostępny -ekstensja : List<Urządzenie> = new List<Urządzenie>() -kupno : int[] +wypożyczania : int[] +Urządzenie(id : int, nazwa : string, cena : double, opis : string, zdjęcie : string, przeznaczenie : string, szerokość : int, długość : int, wysokość : int, kupnialds : int, wypożyczenialds : int[]) +zmieńCenęNowy_rabat : int : void +zwrocCenęZRabatem() : double +dodajDostępnych() : void +usunZłyDostępnych() : void +przejdźdajUrządzeniami() : List<Urządzenie> +przejdźdajUrządzeniaprzekształcenie() : List<Urządzenie> +DodajUrządzenie(id : int, urządzenie : Urządzenie) : void +UsuńUrządzenie(id : int) : void +ZnajdźUrządzenie(id : int) : Urządzenie +getNazwa() : string +setNazwa(nazwa : string) : void +getCena_regularna() : double +regularna(cena_regularna : double) : void +getOpis() : string +setOpis(opis : string) : void +getZdjęcie() : Array<string> +setZdjęcie(zdjęcie : Array<string>) : void +getNazwa_systemu_operacyjnego() : string +setNazwa_systemu_operacyjnego(nazwa_systemu_operacyjnego : string) : void +getSzerokość() : int +getDługość() : int +setDługość(długość : int) : void +getWysokość() : int +setWysokość(wysokość : int) : void +getPrzeznaczenie() : string +setPrzeznaczenie(przeznaczenie : string) : void +getStatus() : string +setStatus(status : string) : void +getKupno() : int[] +setKupno(kupno : int[]) : void +getWypożyczenia() : int[] +setWypożyczenia(wypożyczenia : int[]) : void </pre>

UrządzenieStacjonarne
-ustawienie : Ustawienie
-montaż_na_ścianie : boolean
+UrządzenieStacjonarne()
+getUstawienie() : Ustawienie
+setUstawienie(ustawienie : Ustawienie) : void
+getMontaż_na_ścianie() : boolean
+setMontaż_na_ścianie(montaż_na_ścianie : boolean) : void

UrządzenieMobilne
-waga : double
-aparat_przedni : double
-przekątna_ekranu : double
-typ_matrycy : string
-rozdzielncość_ekranu : string
+UrządzenieMobilne()
+getWaga() : double
+setWaga(waga : double) : void
+getAparat_przedni() : double
+setAparat_przedni(aparat_przedni : double) : void
+getPrzekątna_ekranu(przekątna_ekranu : double) : void
+getTyp_matrycy() : string
+setTyp_matrycy(typ_matrycy : string) : void
+getRozdzielncość_ekranu() : string
+setRozdzielncość_ekranu(rozdzielncość_ekranu : string) : void

Konsola
-generacja : int
-liczba_kontrolerów : int
+Konsola()
+getGeneracja() : int
+setGeneracja(generacja : int) : void
+getLiczba_kontrolerów() : int
+setLiczba_kontrolerów(liczba_kontrolerów : int) : void

Tablet
-funkcja_dzwoniienia : boolean
-aparat_tylny : double
+Tablet()
+getFunkcja_dzwoniienia() : boolean
+setFunkcja_dzwonienia(funkcja_dzwoniienia : boolean) : void
+getAparat_tylny() : double
+setAparat_tylny(aparat_tylny : double) : void

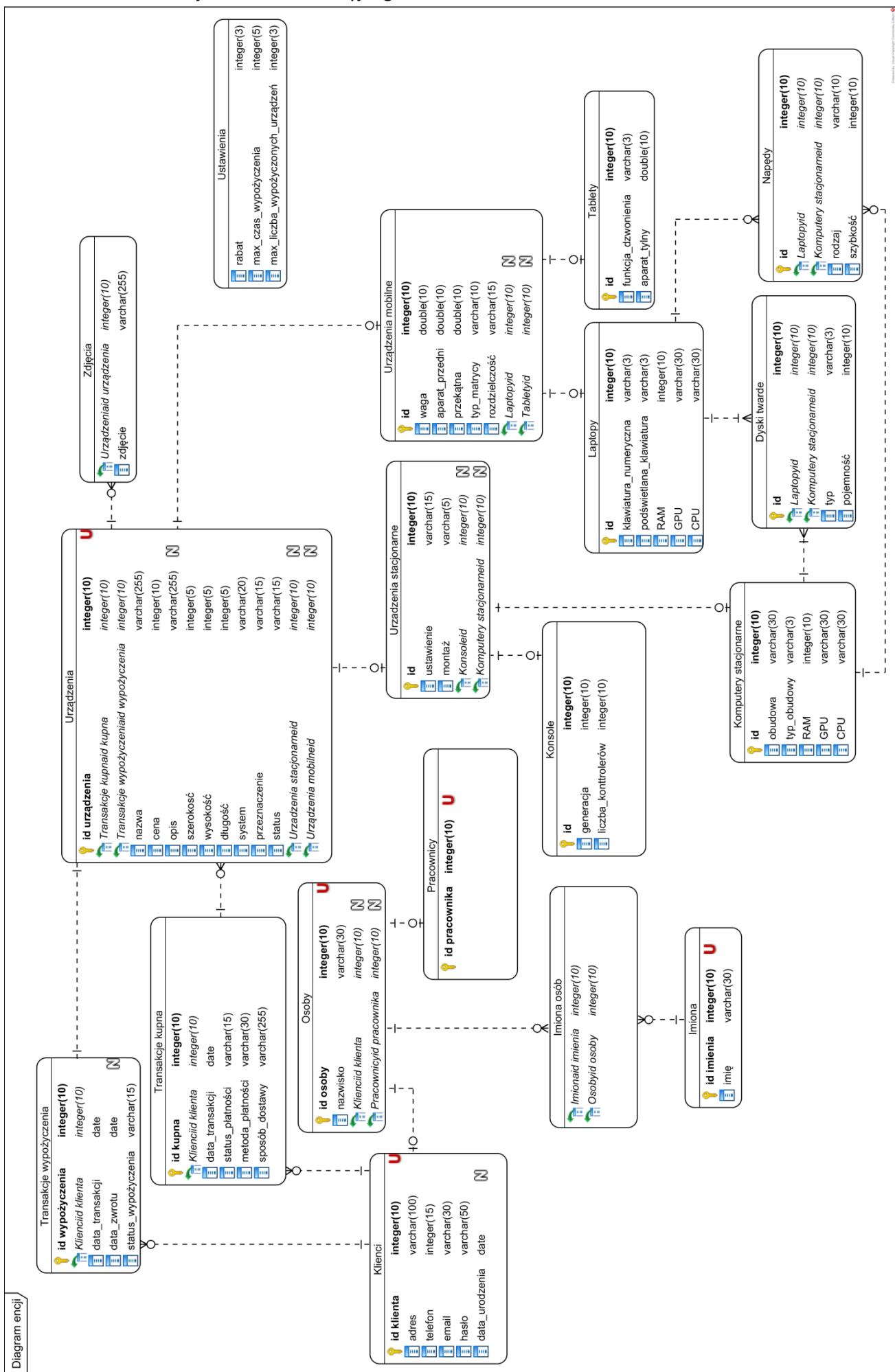
KomputerStacjonarny
-obudowa : string
-typ obudowy : TypObudowy {mini, midi, big}
-pamięć_RAM : int
-karta_graficzna : string
-procesor : string
-dyski : List<DyskTwardy>
-napędy : List<Napęd>
-wszystkieDyski : List<DyskTwardy> = new List<DyskTwardy>
-wszystkieNapędy : List<Napęd> = new List<Napęd>
+KomputerStacjonarny()
+DodajDysk(dysk : DyskTwardy) : void
+DodajNapęd(napęd : Napęd) : void
+getObudowa() : string
+setObudowa(obudowa : string) : void
+getTyp_obudowy() : TypObudowy
+setTyp_obudowy(typ_obudowy : TypObudowy) : void
+getPamięć_RAM() : int
+setPamięć_RAM(pamięć_RAM : int) : void
+getKarta_graficzna() : string
+setKarta_graficzna(karta_graficzna : string) : void
+getProcesor() : string
+setProcesor(procesor : string) : void
+getWszystkieDyski() : List<DyskTwardy>
+setWszystkieDyski(wszystkieDyski : List<DyskTwardy>) : void
+getWszystkieNapędy() : List<Napęd>
+setWszystkieNapędy(wszystkieNapędy : List<Napęd>) : void

Laptop
-klawiatura_numeryczna : boolean
-podświetlana_klawiatura : boolean
-pamięć_RAM : int
-karta_graficzna : string
-procesor : string
-dyski : List<DyskTwardy>
-napędy : List<Napęd>
+Laptop()
+DodajDysk(dysk : DyskTwardy) : void
+DodajNapęd(napęd : Napęd) : void
+getKlawiatura_numeryczna() : boolean
+setKlawiatura_numeryczna(klawiatura_numeryczna : boolean) : void
+getPodświetlana_klawiatura() : boolean
+setPodświetlana_klawiatura(podświetlana_klawiatura : boolean) : void
+getPamięć_RAM() : int
+setPamięć_RAM(pamięć_RAM : int) : void
+getKarta_graficzna() : string
+setKarta_graficzna(karta_graficzna : string) : void
+getProcesor() : string
+setProcesor(procesor : string) : void

DyskTwardy
-typ : TypDysku
-pojemność : int
-urządzenie : Urządzenie
-DyskTwardy(urządzenie : Urządzenie, typ : TypDysku, pojemność : int)
+utwórzDysk(urządzenie : Urządzenie, rodzaj : TypDysku, szybkość : int) : DyskTwardy
+getTyp() : TypDysku
+setTyp(typ : TypDysku) : void
+getPojemność() : int
+setPojemność(pojemność : int) : void
+getUrządzenie() : Urządzenie
+setUrządzenie(urządzenie : Urządzenie) : void

Napęd
-rodzaj : RodzajNapędu
-szybkość : int
-urządzenie : Urządzenie
-Napęd(urządzenie : Urządzenie, rodzaj : RodzajNapędu, szybkość : int)
+utwórzNapęd(urządzenie : Urządzenie, rodzaj : RodzajNapędu, szybkość : int) : Napęd
+getRodzaj() : RodzajNapędu
+setRodzaj(rodzaj : RodzajNapędu) : void
+getSzybkość() : int
+setSzybkość(szybkość : int) : void
+getUrządzenie() : Urządzenie
+setUrządzenie(urządzenie : Urządzenie) : void

17. Transformacja do modelu relacyjnego



Kod:

```
// Transakcja
using System;
namespace Package8 {
    public abstract class Transakcja {
        private int id;
        private DateTime data_transakcji;

        public abstract string ZwróćStatus(); // metoda abstrakcyjna ZwróćStatus() z klasy Transakcja
        public Transakcja(int id, DateTime data_transakcji) {
            this.id = id;
            this.data_transakcji = data_transakcji;
        }
    }
}

// Wypożyczenie
using System;
using System.Collections.Generic;
namespace Package8 {
    // typ enumeryczny wykorzystywany w klasie Wypożyczenie
    public enum StatusWypożyczenia { wypożyczony, zwrócony };

    public class Wypożyczenie : Transakcja {
        private DateTime data_zwrotu;
        private static int max_czas_wypożyczenia = 180; // atrybut max czas wypożyczenia z
        klasy Wypożyczenie; zgodnie z wymaganiami jego wartość początkowa została ustalona na 180
        private StatusWypożyczenia status_wypożyczenia; // atrybut status_wypożyczenia z klasy
        Wypożyczenie o typie StatusWypożyczenia

        private static Dictionary<int, Wypożyczenie> ekstensja = new Dictionary<int,
        Wypożyczenie>(); // ekstensja klasy Wypożyczenie
        private int urządzenie;
        private int klient;

        public static void ZmieńMaxCzasWypożyczenia() {
            throw new System.Exception("Not implemented");
        }

        public Wypożyczenie(int id, DateTime data_transakcji, StatusWypożyczenia
        status_wypożyczenia, int urządzenieId, int klientId) : base(id, data_transakcji) {
            this.status_wypożyczenia = status_wypożyczenia;
            if (Urządzenie.ZnajdźUrządzenie(urządzenieId).Przeznaczenie !=
            Przeznaczenie.Wypożyczenie)
                throw new Exception("Urządzenie nie jest przeznaczone do wypożyczenia");
            this.urządzenie = urządzenieId;
            this.klient = klientId;
            DodajWypożyczenie(id, this); // dodanie do ekstensji
        }

        private static void DodajWypożyczenie(int id, Wypożyczenie wypożyczenie)
        {
```

```

        ekstensja.Add(id, wypożyczenie);
    }
    private static void UsuńWypożyczenie(int id)
    {
        ekstensja.Remove(id);
    }
    public static Wypożyczenie ZnajdźWypożyczenie(int id)
    {
        return ekstensja[id];
    }

    // przesłonięta metoda ZwróćStatus() w klasie Wypożyczenie
    public override string ZwróćStatus()
    {
        return status_wypożyczenia.ToString();
    }

    public DateTime Data_zwrotu
    {
        get
        {
            return data_zwrotu;
        }

        set
        {
            data_zwrotu = value;
        }
    }

    public StatusWypożyczenia Status_wypożyczenia
    {
        set
        {
            status_wypożyczenia = value;
        }
    }

    public static int Max_czas_wypożyczenia
    {
        get
        {
            return max_czas_wypożyczenia;
        }
    }
}

}

// Kupno
using System;
using System.Collections.Generic;
namespace Package8 {
    //typ enumeryczny wykorzystywany w klasie Kupno
    public enum StatusPłatności { nieopłacone, opłacone };

    public class Kupno : Transakcja {
        private StatusPłatności status_płatności; // atrybut status_płatności z klasy Kupno o
        typie StatusPłatności
        private string metoda_płatności;
        private string sposób_dostawy;
    }
}

```

```
private string daneAdresowe;

private static Dictionary<int, Kupno> ekstensja = new Dictionary<int, Kupno>(); //  
ekstensja klasy Kupno  
private int[] urządzenia;  
private int klient;

public StatusPłatności Status_płatności  
{  
    set  
    {  
        status_płatności = value;  
    }  
}

public string Metoda_płatności  
{  
    get  
    {  
        return metoda_płatności;  
    }  
  
    set  
    {  
        metoda_płatności = value;  
    }  
}

public string Sposób_dostawy  
{  
    get  
    {  
        return sposób_dostawy;  
    }  
  
    set  
    {  
        sposób_dostawy = value;  
    }  
}

public string DaneAdresowe  
{  
    get  
    {  
        return daneAdresowe;  
    }  
  
    set  
    {  
        daneAdresowe = value;  
    }  
}

public Kupno(int id, DateTime data_transakcji, StatusPłatności status_płatności, string  
metoda_płatności,  
            string sposób_dostawy, string daneAdresowe, int[] urządzeniaIds, int klientId) :  
base(id, data_transakcji) {  
    this.status_płatności = status_płatności;
```

```

        this.metoda_płatności = metoda_płatności;
        this.sposób_dostawy = sposób_dostawy;
        this.daneAdresowe = daneAdresowe;
        foreach (int item in urządzenia)
            if (Urządzenie.ZnajdźUrządzenie(urządzenieId).Przeznaczenie != Przeznaczenie.Kupno)
                throw new Exception("Niektóre z urządzeń nie są przeznaczone do kupna");
        this.urządzenia = urządzeniaIds;
        this.klient = klientId;
        DodajKupno(id, this); // dodanie do ekstensji
    }
    private static void DodajKupno(int id, Kupno kupno)
    {
        ekstensja.Add(id, kupno);
    }
    private static void UsuńKupno(int id)
    {
        ekstensja.Remove(id);
    }
    public static Kupno ZnajdźKupno(int id)
    {
        return ekstensja[id];
    }

    public override string ZwróćStatus()
    {
        return status_płatności.ToString();
    }
}

// Osoba
using System;
using System.Collections.Generic;
namespace Package8 {
    public class Osoba {
        private List<string> imię; // atrybut imię z klasy Osoba
        private string nazwisko;

        private static List<Osoba> ekstensja = new List<Osoba>();

        private Klient klient;
        private Pracownik pracownik;

        private static List<Klient> wszyscyKlienci = new List<Klient>();
        private static List<Pracownik> wszyscyPracownicy = new List<Pracownik>();

        public Osoba(List<string> imię, string nazwisko)
        {
            this.imię = imię;
            this.nazwisko = nazwisko;

            DodajOsobę(this);
        }

        public Klient Klient
        {
            get

```

```
        {
            return klient;
        }
    }

    public Pracownik Pracownik
    {
        get
        {
            return pracownik;
        }
    }

    public List<string> Imię
    {
        get
        {
            return imię;
        }

        set
        {
            imię = value;
        }
    }

    public string Nazwisko
    {
        get
        {
            return nazwisko;
        }

        set
        {
            nazwisko = value;
        }
    }

    private static void DodajOsobę(Osoba osoba)
    {
        ekstensja.Add(osoba);
    }

    private static void UsuńOsobę(Osoba osoba)
    {
        ekstensja.Remove(osoba);
    }

    public void DodajKlienta(Klient klient) {
        if(wszyscyKlienci.Contains(klient))
            throw new Exception("Klient należy już do innej osoby");
        this.klient = klient;
        wszyscyKlienci.Add(klient);
    }

    public void DodajPracownika(Pracownik pracownik) {
        if(wszyscyPracownicy.Contains(pracownik))
            throw new Exception("Pracownik należy już do innej osoby");
        this.pracownik = pracownik;
        wszyscyPracownicy.Add(pracownik);
    }
}
```

```
    }

}

// Klient
using System;
using System.Collections.Generic;
namespace Package8 {
    public class Klient : Osoba {
        private int id;
        private string adres;
        private int telefon;
        private string email;
        private int liczba_wypożyczonych_urządzeń = 0;
        private static int max_liczba_wypożyczonych_urządzeń = 3; // atrybut max liczba
wypożyczonych urządzeń z klasy Klient; zgodnie z wymaganiami wartość początkowa atrybutu została
ustalona na 3
        private DateTime data_urodzenia = null; // atrybut data_urodzenia z klasy Klient

        private static Dictionary<int, Klient> ekstensja = new Dictionary<int, Klient>();
        private int[] kupna;
        private int[] wypożyczenia;

        private Osoba osoba;

        public string Adres
        {
            get
            {
                return adres;
            }

            set
            {
                adres = value;
            }
        }

        public int Telefon
        {
            get
            {
                return telefon;
            }

            set
            {
                telefon = value;
            }
        }

        public string Email
        {
            get
            {
                return email;
            }
        }
    }
}
```

```

        set
    {
        email = value;
    }
}

public DateTime Data_urodzenia
{
    get
    {
        return data_urodzenia;
    }

    set
    {
        data_urodzenia = value;
    }
}

public static void ZmieńMaxLiczbaWypUrządzeń() {
    throw new System.Exception("Not implemented");
}
public int ZwróćWiek() {
    if (data_urodzenia == null)
        return -1;
    return DateTime.Now.Year - data_urodzenia.Year;
} // metoda ZwróćWiek() z klasy Klient

private Klient(Osoba osoba, int id, string adres, int telefon, string email, int[]
kupnaIds, int[] wypożyczeniaIds) {
    this.osoba = osoba;
    this.id = id;
    this.adres = adres;
    this.telefon = telefon;
    this.email = email;
    this.kupna = kupnaIds;
    this.wypożyczenia = wypożyczeniaIds;
    DodajKlienta(id, this);
}
public static Klient UtwórzKlienta(Osoba osoba, int id, string adres, int telefon,
string email, DateTime data_urodzenia, int[] kupnaIds, int[] wypożyczeniaIds) {
    if(osoba == null)
        throw new Exception("Osoba nie istnieje");
    Klient k = new Klient(osoba, id, adres, telefon, email, data_urodzenia,
kupnaIds, wypożyczeniaIds);
    osoba.DodajKlienta(k);
    return k;
}

private static void DodajKlienta(int id, Klient klient)
{
    ekstensja.Add(id, klient);
}
private static void UsuńKlienta(int id)
{
    ekstensja.Remove(id);
}

```

```
        }
    public static Klient ZnajdzKlienta(int id) {
        return ekstensja[id];
    }
}

// Pracownik
using System;
using System.Collections.Generic;
namespace Package8 {
    public class Pracownik {
        private int id;
        private string stanowisko;

        private static Dictionary<int, Pracownik> ekstensja = new Dictionary<int, Pracownik>();

        private Osoba osoba;

        public string Stanowisko
        {
            get
            {
                return stanowisko;
            }

            set
            {
                stanowisko = value;
            }
        }

        private Pracownik(Osoba osoba, int id, string stanowisko)
        {
            this.osoba = osoba;
            this.id = id;
            this.stanowisko = stanowisko;
            DodajPracownika(id, this);
        }

        public static Pracownik UtworzPracownika(Osoba osoba, int id, string stanowisko) {
            if(osoba == null)
                throw new Exception("Osoba nie istnieje");
        }
    }
}
```

```

        Pracownik p = new Pracownik(osoba, id, stanowisko);
        osoba.DodajPracownika(p);
        return p;
    }

private static void DodajPracownika(int id, Pracownik pracownik)
{
    ekstensja.Add(id, pracownik);
}

private static void UsuńPracownika(int id)
{
    ekstensja.Remove(id);
}

public static Pracownik ZnajdźPracownika(int id)
{
    return ekstensja[id];
}

}

}

// Urządzenie
using System;
using System.Collections.Generic;
namespace Package8 {
    // typy enumeryczne wykorzystywane w klasie Urządzenie
    public enum Przeznaczenie { kupno, wypożyczenie };
    public enum StatusUrządzenia { dostępny, kupiony, wypożyczony };

    public abstract class Urządzenie {
        private int id;
        private string nazwa;
        private double cena_regularna;
        private static int rabat = 0; // atrybut rabat z klasy Urządzenie; jego wartość
        początkowa wynosi 0
        private string opis = null; // atrybut opis z klasy Urządzenie
        private List<string> zdjęcie; // atrybut zdjęcie z klasy Urządzenie
        private string nazwa_systemu_operacyjnego;
        private int szerokość;
        private int długość;
        private int wysokość;
        private Przeznaczenie przeznaczenie; // atrybut przeznaczenie z klasy Urządzenie o
        typie Przeznaczenie
        private StatusUrządzenia status = StatusUrządzenia.dostępny; // atrybut status z klasy
        Urządzenie o typie StatusUrządzenia

        // implementacja ekstensji klasy Urządzenie
        private static Dictionary<int, Urządzenie> ekstensja = new Dictionary<int,
        Urządzenie>();
    }

    private int[] kupna;
}

```

```
private int[] wypożyczenia;

public Urządzenie(int id, string nazwa, double cena_regulararna, string opis, List<string>
zdjęcie,
                     string nazwa_systemu_operacyjnego, int szerokość, int długość, int wysokość,
Przeznaczenie przeznaczenie, int[] kupnaIds, int[] wypożyczeniaIds)
{
    this.id = id;
    this.nazwa = nazwa;
    this.cena_regulararna = cena_regulararna;
    this.opis = opis;
    this.zdjęcie = zdjęcie;
    this.nazwa_systemu_operacyjnego = nazwa_systemu_operacyjnego;
    this.szerokość = szerokość;
    this.długość = długość;
    this.wysokość = wysokość;
    this.przeznaczenie = przeznaczenie;
    this.kupna = kupnaIds;
    this.wypożyczenia = wypożyczeniaIds;
    DodajUrządzenie(this);
}

public string Nazwa
{
    get
    {
        return nazwa;
    }

    set
    {
        nazwa = value;
    }
}

public double Cena_regulararna
{
    get
    {
        return cena_regulararna;
    }

    set
    {
        cena_regulararna = value;
    }
}

public string Opis
{
    get
    {
        return opis;
    }

    set
    {
        opis = value;
    }
}
```

```
public string Nazwa_systemu_operacyjnego
{
    get
    {
        return nazwa_systemu_operacyjnego;
    }

    set
    {
        nazwa_systemu_operacyjnego = value;
    }
}

public int Szerokość
{
    get
    {
        return szerokość;
    }

    set
    {
        szerokość = value;
    }
}

public int Długość
{
    get
    {
        return długość;
    }

    set
    {
        długość = value;
    }
}

public int Wysokość
{
    get
    {
        return wysokość;
    }

    set
    {
        wysokość = value;
    }
}

public Przeznaczenie Przeznaczenie
{
    get
    {
        return przeznaczenie;
    }
}
```

```
        set
    {
        przeznaczenie = value;
    }
}

public StatusUrządzenia Status
{
    get
    {
        return status;
    }

    set
    {
        status = value;
    }
}

public int[] Kupna
{
    get
    {
        return kupna;
    }

    set
    {
        kupna = value;
    }
}

public int[] Wypożyczenia
{
    get
    {
        return wypożyczenia;
    }

    set
    {
        wypożyczenia = value;
    }
}

public static List<Urządzenie> PrzeglądajUrządzenia()
{
    List<Urządzenie> l = new List<Urządzenie>();
    foreach (var item in ekstensja)
        l.Add(item.Value);
    return l;
}

public static List<Urządzenie> PrzeglądajUrządzenia(Przeznaczenie przeznaczenie)
{
    List<Urządzenie> l = new List<Urządzenie>();
    foreach (var item in ekstensja)
    {
        if(item.Value.Przeznaczenie == przeznaczenie)
            l.Add(item.Value);
    }
}
```

```

        return 1;
    }

    // metoda ZmieńRabat ustawiająca wartość atrybutu klasowego Rabat z klasy Urządzenie;
metoda sprawdza, czy nowa wartość rabatu nie różni się więcej niż o 5% aktualnej wartości rabatu
    public static void ZmieńRabat(int nowy_rabat) {
        if(nowy_rabat > this.rabat * 1.05 || nowy_rabat < this.rabat * 0.95)
            throw new Exception("Zmiana rabatu większa niż 5%");
        rabat = nowy_rabat;
    }
    public double ZwróćCenęZRabatem() {
        return cena_regularna - rabat * cena_regularna;
    } // metoda ZwróćCenęZRabatem() z klasy Urządzenie
public void DodajDoListyDostępnych() {
    throw new System.Exception("Not implemented");
}
public void UsuńZListyDostępnych() {
    throw new System.Exception("Not implemented");
}

private static void DodajUrządzenie(int id, Urządzenie urządzenie) {
    ekstensja.Add(id, urządzenie);
}
private static void UsuńUrządzenie(int id) {
    ekstensja.Remove(id);
}
public static Urządzenie ZnajdźUrządzenie(int id)
{
    return ekstensja[id];
}

}

}

// UrządzenieStacjonarne
using System;
namespace Package8 {
    // typ enumeryczny wykorzystywany w klasie Urządzenie_stacjonarne
    public enum Ustawienie { pionowo, poziomo };

    public abstract class UrządzenieStacjonarne : Urządzenie {
        private Ustawienie ustawienie; // atrybut ustawienie z klasy UrządzenieStacjonarne o
typie Ustawienie
        private bool montaż_na_ścianie; // atrybut montaż_na_ścianie z klasy
UrządzenieStacjonarne o typie bool

        public UrządzenieStacjonarne(int id, string nazwa, double cena_regularna, string opis,
List<string> zdjęcie, string nazwa_systemu_operacyjnego, int szerokość, int długość, int
wysokość, string przeznaczenie, int[] kupnaIds, int[] wypożyczeniaIds,
        Ustawienie ustawienie, bool montaż_na_ścianie) : base(id, nazwa, cena_regularna,
opis, zdjęcie, nazwa_systemu_operacyjnego, szerokość, długość, wysokość, przeznaczenie,
kupnaIds, wypożyczeniaIds)
    {

```

```

        this.ustawienie = ustawienie;
        this.montaż_na_ścianie = montaż_na_ścianie;
    }

    public Ustawienie Ustawienie
    {
        get
        {
            return ustawienie;
        }

        set
        {
            ustawienie = value;
        }
    }

    public bool Montaż_na_ścianie
    {
        get
        {
            return montaż_na_ścianie;
        }

        set
        {
            montaż_na_ścianie = value;
        }
    }
}

}

// UrządzenieMobilne
using System;
namespace Package8 {
    public abstract class UrządzenieMobilne : Urządzenie {
        private double waga;
        private double aparat_przedni;
        private double przekątna_ekranu;
        private string typ_matrycy;
        private string rozdzielcość_ekranu;

        public UrządzenieMobilne(int id, string nazwa, double cena_regulararna, string opis,
List<string> zdjęcie, string nazwa_systemu_operacyjnego, int szerokość, int długość, int
wysokość, string przeznaczenie, int[] kupnaIds, int[] wypożyczeniaIds, double waga, double
aparat_przedni,
        double przekątna_ekranu, string typ_matrycy, string rozdzielcość_ekranu) : base(id,
nazwa, cena_regulararna, opis, zdjęcie, nazwa_systemu_operacyjnego, szerokość, długość, wysokość,
przeznaczenie, kupnaIds, wypożyczeniaIds)
        {
            this.waga = waga;
            this.aparat_przedni = aparat_przedni;
            this.przekątna_ekranu = przekątna_ekranu;
            this.typ_matrycy = typ_matrycy;
            this.rozdzielcość_ekranu = rozdzielcość_ekranu;
        }

        public double Waga
        {

```

```
        get
    {
        return waga;
    }

        set
    {
        waga = value;
    }
}

public double Aparat_przedni
{
    get
    {
        return aparat_przedni;
    }

    set
    {
        aparat_przedni = value;
    }
}

public double Przekątna_ekranu
{
    get
    {
        return przekątna_ekranu;
    }

    set
    {
        przekątna_ekranu = value;
    }
}

public string Typ_matrycy
{
    get
    {
        return typ_matrycy;
    }

    set
    {
        typ_matrycy = value;
    }
}

public string Rozdzielcość_ekranu
{
    get
    {
        return rozdzielcość_ekranu;
    }

    set
    {
        rozdzielcość_ekranu = value;
    }
}
```

```
        }
    }

}

// Konsola
using System;
namespace Package8 {
    public class Konsola : Urządzenie_stacjonarne {
        private int generacja;
        private int liczba_kontrolerów;

        public Konsola(int id, string nazwa, double cena_regulararna, string opis, List<string> zdjęcie, string nazwa_systemu_operacyjnego, int szerokość, int długość, int wysokość, string przeznaczenie, int[] kupnaIds, int[] wypożyczeniaIds,
                      Ustawienie ustawienie, bool montaż_na_ścianie, int generacja, int liczba_kontrolerów) : base(id, nazwa, cena_regulararna, opis, zdjęcie, nazwa_systemu_operacyjnego, szerokość, długość, wysokość, przeznaczenie, kupnaIds, wypożyczeniaIds, ustawienie,
                      montaż_na_ścianie) {
            this.generacja = generacja;
            this.liczba_kontrolerów = liczba_kontrolerów;
        }

        public int Generacja
        {
            get
            {
                return generacja;
            }

            set
            {
                generacja = value;
            }
        }

        public int Liczba_kontrolerów
        {
            get
            {
                return liczba_kontrolerów;
            }

            set
            {
                liczba_kontrolerów = value;
            }
        }
    }

}

// KomputerStacjonarny
using System;
using System.Collections.Generic;
namespace Package8 {
```

```

// typ enumeryczny wykorzystywany w klasie KomputerStacjonarny
public enum TypObudowy { mini, midi, big };

public class KomputerStacjonarny : UrządzenieStacjonarne {
    private string obudowa;
    private TypObudowy typ_obudowy; // atrybut typ_obudowy z klasy KomputerStacjonarny o
typie TypObudowy
    private int pamięć_RAM;
    private string karta_graficzna;
    private string procesor;
    private List<DyskTwardy> dyski = new List<DyskTwardy>();
    private List<Napęd> napędy = new List<Napęd>();

    private static List<DyskTwardy> wszystkieDyski = new List<DyskTwardy>();
    private static List<Napęd> wszystkieNapędy = new List<Napęd>();

    public KomputerStacjonarny(int id, string nazwa, double cena_regulararna, string opis,
List<string> zdjęcie, string nazwa_systemu_operacyjnego, int szerokość, int długość, int
wysokość, string przeznaczenie, int[] kupnaIds, int[] wypożyczeniaIds, Ustawienie ustawienie,
bool montaż_na_ścianie,
        string obudowa, TypObudowy typ_obudowy, int pamięć_RAM, string karta_graficzna,
string procesor) : base(id, nazwa, cena_regulararna, opis, zdjęcie, nazwa_systemu_operacyjnego,
szerokość, długość, wysokość, przeznaczenie, kupnaIds, wypożyczeniaIds, ustawienie,
montaż_na_ścianie)
    {
        this.obudowa = obudowa;
        this.typ_obudowy = typ_obudowy;
        this.pamięć_RAM = pamięć_RAM;
        this.karta_graficzna = karta_graficzna;
        this.procesor = procesor;
    }
}

```

```

public string Obudowa
{
    get
    {
        return obudowa;
    }

    set
    {
        obudowa = value;
    }
}

public TypObudowy Typ_obudowy
{
    get
    {
        return typ_obudowy;
    }

    set
    {
        typ_obudowy = value;
    }
}

```

```
}

public int Pamięć_RAM
{
    get
    {
        return pamięć_RAM;
    }

    set
    {
        pamięć_RAM = value;
    }
}

public string Karta_graficzna
{
    get
    {
        return karta_graficzna;
    }

    set
    {
        karta_graficzna = value;
    }
}

public string Procesor
{
    get
    {
        return procesor;
    }

    set
    {
        procesor = value;
    }
}

public static List<DyskTwardy> WszystkieDyski
{
    get
    {
        return wszystkieDyski;
    }
}

public static List<Napęd> WszystkieNapędy
{
    get
    {
        return wszystkieNapędy;
    }
}

public void DodajDysk(DyskTwardy dysk) {
    if (!dyski.Contains(dysk))
    {
```

```

        if (wszystkieDyski.Contains(dysk))
            throw new Exception("Ten dysk jest już powiązany z innym urządzeniem");
        dyski.Add(dysk);
        wszystkieDyski.Add(dysk);
    }
}

public void DodajNapęd(Napęd napęd) {
    if (!napędy.Contains(napęd))
    {
        if (wszystkieNapędy.Contains(napęd))
            throw new Exception("Ten napęd jest już powiązany z innym urządzeniem");
        napędy.Add(napęd);
        wszystkieNapędy.Add(napęd);
    }
}

}

// Laptop
using System;
using System.Collections.Generic;
namespace Package8 {
    public class Laptop : Urządzenie_mobilne {
        private bool klawiatura_numeryczna; // atrybuty klawiatura_numeryczna oraz
podświetlana_klawiatura z klasy Laptop o typie bool
        private bool podświetlana_klawiatura;
        private int pamięć_RAM;
        private string karta_graficzna;
        private string procesor;
        private List<DyskTwardy> dyski = new List<DyskTwardy>();
        private List<Napęd> napędy = new List<Napęd>();

        public Laptop(int id, string nazwa, double cena_regulararna, string opis, List<string>
zdjęcie, string nazwa_systemu_operacyjnego, int szerokość, int długość, int wysokość, string
przeznaczenie, int[] kupnaIds, int[] wypożyczeniaIds, double waga,
            double aparat_przedni, double przekątna_ekranu, string typ_matrycy, string
rozdzielcość_ekranu, bool klawiatura_numeryczna, bool podświetlana_klawiatura, int pamięć_RAM,
            string karta_graficzna, string procesor) : base(id, nazwa, cena_regulararna, opis, zdjęcie,
nazwa_systemu_operacyjnego, szerokość, długość, wysokość, przeznaczenie, kupnaIds,
wypożyczeniaIds, waga, aparat_przedni, przekątna_ekranu, typ_matrycy, rozdzielcość_ekranu)
        {
            this.klawiatura_numeryczna = klawiatura_numeryczna;
            this.podświetlana_klawiatura = podświetlana_klawiatura;
            this.pamięć_RAM = pamięć_RAM;
            this.karta_graficzna = karta_graficzna;
            this.procesor = procesor;
        }

        public bool Klawiatura_numeryczna
        {
            get
            {
                return klawiatura_numeryczna;
            }
            set
            {
                klawiatura_numeryczna = value;
            }
        }
    }
}

```

```
        }

    }

public bool Podświetlana_klawiatura
{
    get
    {
        return podświetlana_klawiatura;
    }

    set
    {
        podświetlana_klawiatura = value;
    }
}

public int Pamięć_RAM
{
    get
    {
        return pamięć_RAM;
    }

    set
    {
        pamięć_RAM = value;
    }
}

public string Karta_graficzna
{
    get
    {
        return karta_graficzna;
    }

    set
    {
        karta_graficzna = value;
    }
}

public string Procesor
{
    get
    {
        return procesor;
    }

    set
    {
        procesor = value;
    }
}

public void DodajDysk(DyskTwardy dysk)
{
    if (!dyski.Contains(dysk))
    {
        if (KomputerStacjonarny.WszystkieDyski.Contains(dysk))
```

```

        throw new Exception("Ten dysk jest już powiązany z innym urządzeniem");
        dyski.Add(dysk);
        KomputerStacjonarny.WszystkieDyski.Add(dysk);
    }
}
public void DodajNapęd(Napęd napęd)
{
    if (!napędy.Contains(napęd))
    {
        if (KomputerStacjonarny.WszystkieNapędy.Contains(napęd))
            throw new Exception("Ten napęd jest już powiązany z innym urządzeniem");
        napędy.Add(napęd);
        KomputerStacjonarny.WszystkieNapędy.Add(napęd);
    }
}

}

// Tablet
using System;
using System.Collections.Generic;
namespace Package8 {
    public class Tablet : Urządzenie_mobilne {
        private bool funkcja_dzwonienia; // atrybut funkcja_dzwonienia z klasy Tablet o typie
bool
        private double aparat_tylny;

        public Tablet(int id, string nazwa, double cena_regularna, string opis, List<string>
zdjęcie, string nazwa_systemu_operacyjnego, int szerokość, int długość, int wysokość, string
przeznaczenie, int[] kupnaIds, int[] wypożyczeniaIds, double waga,
double aparat_przedni, double przekątna_ekranu, string typ_matrycy, string
rozdzielcość_ekranu, bool funkcja_dzwonienia, double aparat_tylny) : base(id, nazwa,
cena_regularna, opis, zdjęcie, nazwa_systemu_operacyjnego, szerokość, długość, wysokość,
przeznaczenie, kupnaIds, wypożyczeniaIds, waga, aparat_przedni, przekątna_ekranu, typ_matrycy,
rozdzielcość_ekranu)
{
    this.funkcja_dzwonienia = funkcja_dzwonienia;
    this.aparat_tylny = aparat_tylny;
}

public bool Funkcja_dzwonienia
{
    get
    {
        return funkcja_dzwonienia;
    }

    set
    {
        funkcja_dzwonienia = value;
    }
}

public double Aparat_tylny
{
    get
    {
        return aparat_tylny;
    }
}
}
```

```

        }

        set
        {
            aparat_tylny = value;
        }
    }

}

// DyskTwardy
using System;
namespace Package8 {
    // typ enumeryczny wykorzystywany w klasie DyskTwardy
    public enum TypDysku { HDD, SSD };

    public class DyskTwardy {
        private TypDysku typ; // atrybut typ z klasy DyskTwardy o typie TypDysku
        private int pojemność;
        private Urządzenie urządzenie;

        private DyskTwardy(Urządzenie urządzenie, TypDysku typ, int pojemność) {
            this.urządzenie = urządzenie;
            this.typ = typ;
            this.pojemność = pojemność;
        }
        public DyskTwardy UtwórzDysk(Urządzenie urządzenie, TypDysku rodzaj, int szybkość) {
            if(urządzenie == null)
                throw new Exception("Urządzenie nie istnieje");
            DyskTwardy d = new Dysk_twardy(urządzenie, rodzaj, pojemność);
            urządzenie.DodajDysk(d);
            return d;
        }
    }

}

// Napęd
using System;
namespace Package8 {
    // typ enumeryczny wykorzystywany w klasie Napęd
    public enum RodzajNapędu { DVD, bluray };

    public class Napęd {
        private RodzajNapędu rodzaj; // atrybut rodzaj z klasy Napęd o typie RodzajNapędu
        private int szybkość;
        private Urządzenie urządzenie;

        private Napęd(Urządzenie urządzenie, RodzajNapędu rodzaj, int szybkość) {
            this.urządzenie = urządzenie;
            this.rodzaj = rodzaj;
            this.szybkość = szybkość;
        }
        public Napęd UtwórzNapęd(Urządzenie urządzenie, RodzajNapędu rodzaj, int szybkość) {
            if(urządzenie == null)

```

```
        throw new Exception("Urządzenie nie istnieje");
        Napęd n = new Napęd(urządzenie, rodzaj, szybkość);
        urządzenie.DodajNapęd(n);
        return n;
    }

}
```