a

1CD

Projekt „Inteligentny wózek widłowy”

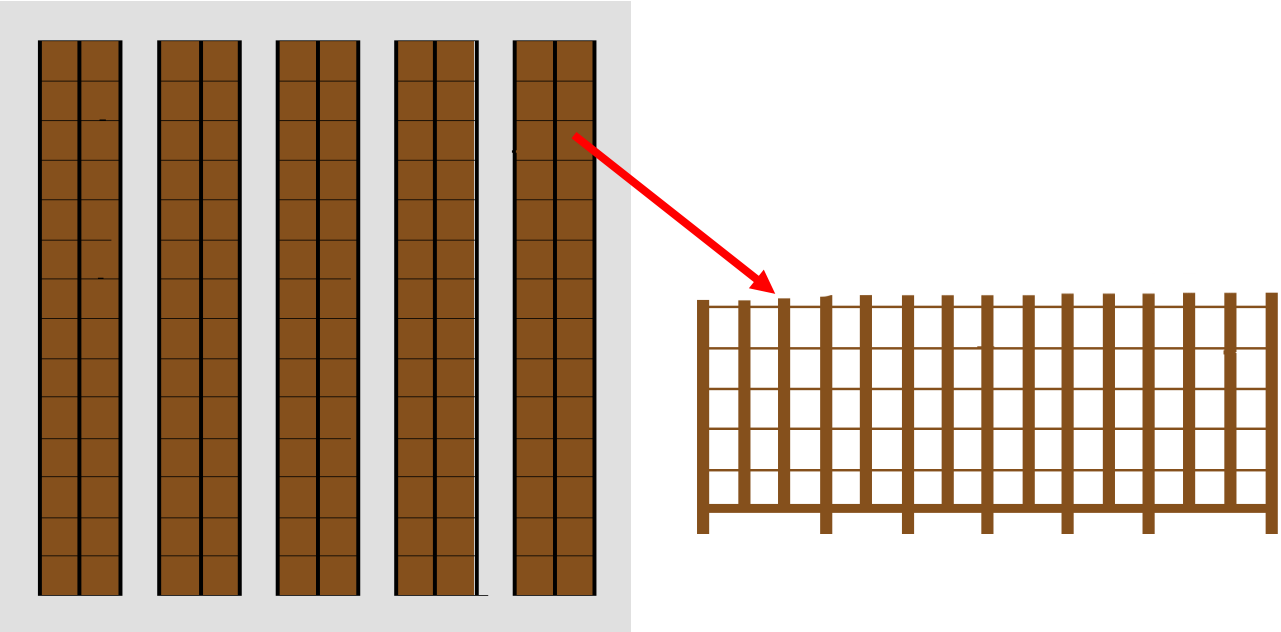
Damian Czapiewski, Mikołaj Mański, Zbigniew Pilak, Szymon Rzepa

Spis treści

Dokumentacja użytkownika

Opis projektu:

Inteligentny wózek widłowy jest to program napisany w języku JAVA. Wózek widłowy będzie poruszał się po magazynie (tablicy trójwymiarowej - tablica dwuwymiarowa ze ścieżką dla wózka widłowego i dodatkowy wymiar z półkami) pełnym półek i przedmiotów. Jego ścieżką będzie podłoga (szare pola) a przedmioty będą umieszczone na półkach. Wózek nie będzie mógł poruszać się po półkach.



Zadaniem inteligentnego wózka widłowego jest przenoszenie przedmiotów w odpowiednie miejsce magazynu na podstawie poleceń wprowadzanych z klawiatury przez użytkownika. Ponadto użytkownik będzie mógł tworzyć nowe przedmioty oraz je usuwać z magazynu co nastąpi po przez przeniesienie przedmiotu w odpowiednie miejsce.

Użytkownik będzie wpisywał polecenia w języku naturalnym do odpowiedniego pola tekstowego. Wózek będzie mógł na jego podstawie przenieść oraz znaleźć odpowiedni przedmiot. Jeżeli polecenie będzie niekonkretne to wózek widłowy poprosi użytkownika o doprecyzowanie polecenia. Polecenia użytkownika powinny umożliwiać zidentyfikowanie przedmiotu na podstawie jego cech (wielkość, kolor, kształt), będą interpretowane na podstawie słowo-klucz. Po każdym poleceniu wózek widłowy będzie generował krótką odpowiedz jakie zmiany. Program będzie posiadał interfejs graficzny. Z lewej strony znajduje się lista dostępnych przedmiotów które posiadają unikalne id. Na środku znajduje się graficzna wersja magazynu. Na dolę jest wiersz poleceń przez które użytkownik komunikuje się z wózkiem widłowym.

Sposób działania programu:

Użytkownik podaje przez wiersz poleceń, rozkazy co ma wykonywać wózek widłowy.

*//**
Tutaj będą sposoby interpretacji języka naturalnego na polecenia wózka
**//*

Metoda, która interpretuje wprowadzony tekst to NaturalLanguage.Interpret(). Metoda przyjmuje wprowadzony tekst i zwraca odpowiedź bota.

Na początku, jeżeli metoda Interpret() jest wywoływana pierwszy raz, to ładowane są dwie rzeczy:

a) słowa kluczowe przechowywane w pliku tekstowym o specjalnej strukturze. W tym pliku są zapisane słowa kluczowe, ich synonimy oraz każde słowo kluczowe posiada jeszcze typ słowa kluczowego (instrukcja, parametr, kolor...);

b) ewentualnie jeszcze jedna rzecz, która na razie nie jest zbyt istotna.

Słowa kluczowe w programie są reprezentowane poprzez klasę Keyword. Synonimy danego słowa kluczowego są reprezentowane w tablicy będącej składową klasy Keyword. Klasa NaturalLanguage zawiera z kolei tablicę, która zawiera wszystkie słowa kluczowe.

Poza słowami kluczowymi, które są ładowane z pliku, są jeszcze inne słowa kluczowe:
a) kiedy użytkownik tworzy nowy obiekt, to nazwa danego obiektu staje się słowem kluczowym (np. jeżeli zostaje dodany obiekt o nazwie "skrzynia", to powstaje nowe słowo kluczowe "skrzynia") (tak na marginesie: jeżeli obiekt zostanie dodany poprzez przycisk w programie ("Dodaj obiekt"), to Wasza funkcja dodająca obiekt będzie musiała wywołać jakąś moją metodę, która będzie odpowiedzialna za dodanie nowego keywordu będącego nazwą dodanego obiektu, żeby słowa kluczowe zostały zaktualizowane w momencie dodania obiektu);
b) istnieje również jedno specjalne słowo kluczowe, które reprezentuje każdą liczbę (dzięki temu jeżeli zostanie podana waga jakiegoś obiektu, która jest liczbą, to ta liczba zostanie wyłapana przez analizator).

Klasa Keyword posiada składową "type", która reprezentuje typ słowa kluczowego. Te typy to:
a) INSTRUCTION - instrukcja taka jak "znajdź", "przełóż" itd.;
b) PARAMETER - słowa w rodzaju "kolor", "waga", "nazwa" itd.;
c) COLOR - słowa w rodzaju "czerowny", "niebieski" itd.;
d) NAME - słowo będące nazwą jakiegoś obiektu, te słowa kluczowe nie są ładowane

z pliku tylko są tworzone "dynamicznie" przy każdym dodaniu nowego obiektu;
e) INTEGER - tylko jedno słowo kluczowe będzie posiadało ten typ - słowo specjalne reprezentujące liczbę;
f) ... i chyba będą jeszcze jakieś typy...

Keyword posiada metodę check(String), która sprawdza, czy String podany jako argument pasuje do danego słowa kluczowego. "Pasuje do danego słowa kluczowego" oznacza, że string jest podobny do słowa kluczowego albo do któregoś z synonimów. To, czy ciągi znaków są do siebie podobne jest mierzone algorytmem Levenshteina (http://pl.wikipedia.org/wiki/Odleg%C5%82o%C5%9B%C4%87_Levenshteina), który dla dwóch ciągów zwraca liczbę, która w skrócie określa jak bardzo dane ciągi się ze sobą różnią (czym większa liczba tym bardziej się różnią). Poprzez algorytm Levenshteina zostają rozwiązane dwa problemy:

a) słowa w języku polskim się odmieniają - można by stworzyć bardzo dużo synonimów, ale dla nazw obiektów (które również są słowami kluczowymi - tak jak napisałem wyżej) nie da się stworzyć synonimów; algorytm Levenshteina rozwiązuje problem;
b) użytkownik robi literówki - algorytm Levenshteina mierzy podobieństwo ciągów, a nie czy ciągi są dokładnie sobie równe; dzięki temu jeżeli użytkownik wpisze np. "czrwonny" zamiast "czerwony", to to przejdzie.

Teraz wracając do metody Interpret(), to pierwszą rzeczą jaka jest robiona po załadowaniu słów kluczowych (i ewentualnie innych rzeczy) jest oczywiście podzielenie wprowadzonego tekstu na słowa. Następnie metoda Interpret() (lub któraś z metod wywoływanych przez Interpret()) sprawdza, jakie słowa kluczowe występują we wprowadzonym tekście i zapisuje je do tablicy found_keywords. Więc teraz mamy już informację, jakie słowa kluczowe znajdują się w tekście i w jakiej kolejności (w takiej, w jakiej są w tablicy).

Teraz szukamy pierwszego słowa kluczowego typu INSTRUCTION ("znajdź", "przenieś") w tablicy found_keywords (w przyszłości bot będzie reagował na sytuację, w której są dwa słowa typu INSTRUCTION w found_keywords, na razie zakładamy że jest jedno, a jak będzie więcej to algorytm na razie zakłada że to pierwsze jest prawidłowe).

Dalsze postępowanie algorytmu jest zależne od tego, jakie jest dane słowo kluczowe typu INSTRUCTION, ale w przypadku niektórych instrukcji działanie prawdopodobnie będzie dosyć podobne.

Na przykład dla słowa "znajdź" będzie następujące:

Załóżmy, że wprowadzony tekst to będzie: "Znajdź obiekt o nazwie Skrzynia".

Wtedy zawartość tablicy found_keywords będzie następująca: [znajdź, nazwa, skrzynia]. Tablica found_keywords zawiera obiekty typu Keyword.

Teraz jeżeli instrukcja brzmi "znajdź", to oczywiście będziemy wywoływać funkcję find() (czy tam search()), która już nie należy do analizatora. Ta funkcja przyjmuje dwa argumenty: po czym szukamy i czego szukamy. Więc musimy znaleźć te informacje.

Aby znaleźć pierwszy argument do funkcji find() (po czym szukamy), algorytm znajduje pierwsze słowo kluczowe typu PARAMETER (słowa typu: "nazwa", "kolor", "waga"). W tym przypadku będzie to słowo "nazwa". Na podstawie tego, co tutaj znaleźliśmy już wiemy, co przekazać do funkcji find() (czy tam search()...) jako pierwszy argument, który określa po czym szukamy danego obiektu.

Jeżeli będzie więcej słów kluczowych tego typu, to na razie liczy się pierwsze słowo (później być może znajdziemy lepsze rozwiązanie).

Teraz aby znaleźć drugi argument do funkcji find() (czego szukamy), algorytm znajduje pierwsze słowo kluczowe typu innego niż INSTRUCTION i PARAMETER (wszystkie pozostałe słowa reprezentują jakąś "wartość"). W tym przypadku będzie to "skrzynia".

Zatem zostanie wywołana funkcja find() w następujący sposób (oczywiście mniej więcej):
find("nazwa", "skrzynia").

Inny przykład:

Wprowadzony tekst = "Znajdź obiekt, którego waga wynosi 5 kg".
found_keywords = [znajdź, waga, liczba]
pierwsze słowo kluczowe typu PARAMETER = waga
pierwsze słowo kluczowe typu innego niż INSTRUCTION i PARAMETER = liczba

Teraz jeżeli tutaj pierwszym słowem kluczowym innym niż INSTRUCTION i PARAMETER jest liczba, to robimy mały wyjątek i przed wywołaniem funkcji find() pobieramy informację, jaka to jest liczba i dopiero to przekazujemy do funkcji find(). (na razie nie będę wchodził w szczegóły, jak to będzie pobierane)

Zatem wywołanie będzie następujące:

find(waga, 5)

Teraz może pojawić się następujące pytanie:

Co dla takiego wprowadzonego tekstu: "Znajdź skrzynię"?

found_keywords w tym przypadku to będzie [znajdź, skrzynia].

Nie ma informacji na temat pierwszego argumentu, czyli tego czego szukamy. Co zrobić? Oczywiście algorytm może się domyśleć na podstawie typu słowa kluczowego reprezentującego drugi argument. To znaczy: drugi argument = skrzynia, a skrzynia to słowo kluczowe typu NAME, zatem pierwszy argument dla funkcji find() to będzie "nazwa".

Tak samo, jeżeli na przykład wprowadzony tekst = "Znajdź czerwony obiekt". Tablica found_keywords = [znajdź, czerwony]. Słowo kluczowe "czerwony" jest typu COLOR zatem algorytm się domyśla, że pierwszy argument dla funkcji find() to "color".

Co się stanie jeżeli nie będziemy posiadali informacji również na temat drugiego argumentu funkcji find()? Czyli wprowadzony tekst to będzie na przykład po prostu "Znajdź". Co wtedy? Wtedy bot zwróci informację, że nie została podana wystarczająca liczba informacji aby poszukać jakiegoś obiektu.

Jeżeli zostanie podana wystarczająca liczba informacji, to bot zwróci w odpowiedzi użytkownikowi informacje na temat obiektu, które zostaną zwrócone przez funkcję find().

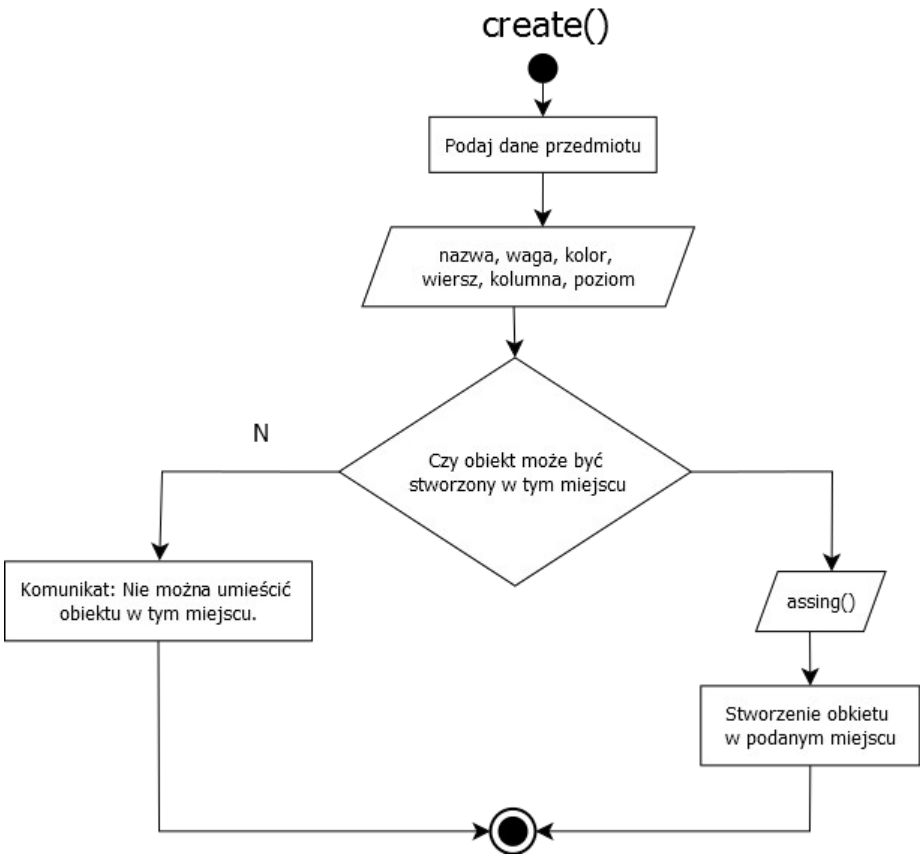
Pozostałe instrukcje będą działać podobnie. Instrukcja "znajdź" jest chyba najbardziej skomplikowana.

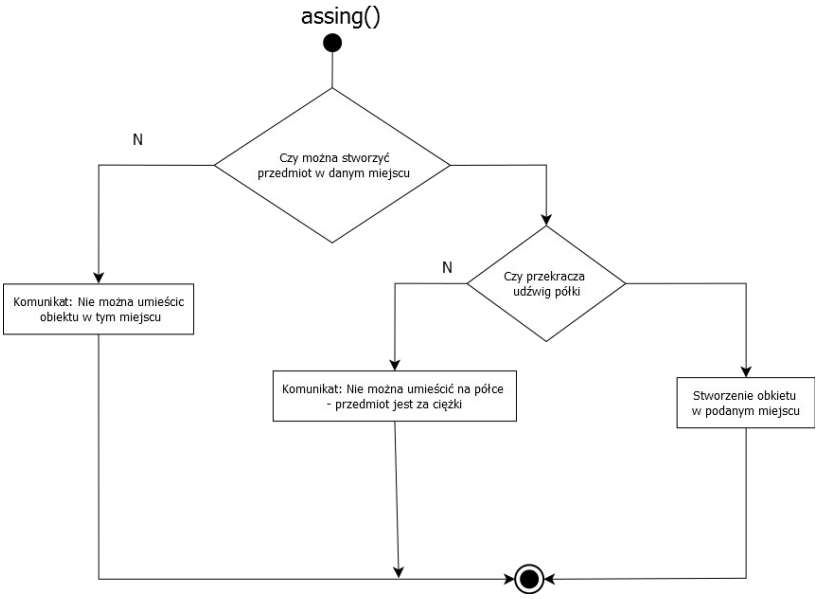
Opis funkcji programu:

Użytkownik programu będzie mógł wybrać jedną z kilku dostępnych opcji:

Tworzenie przedmiotu (create())

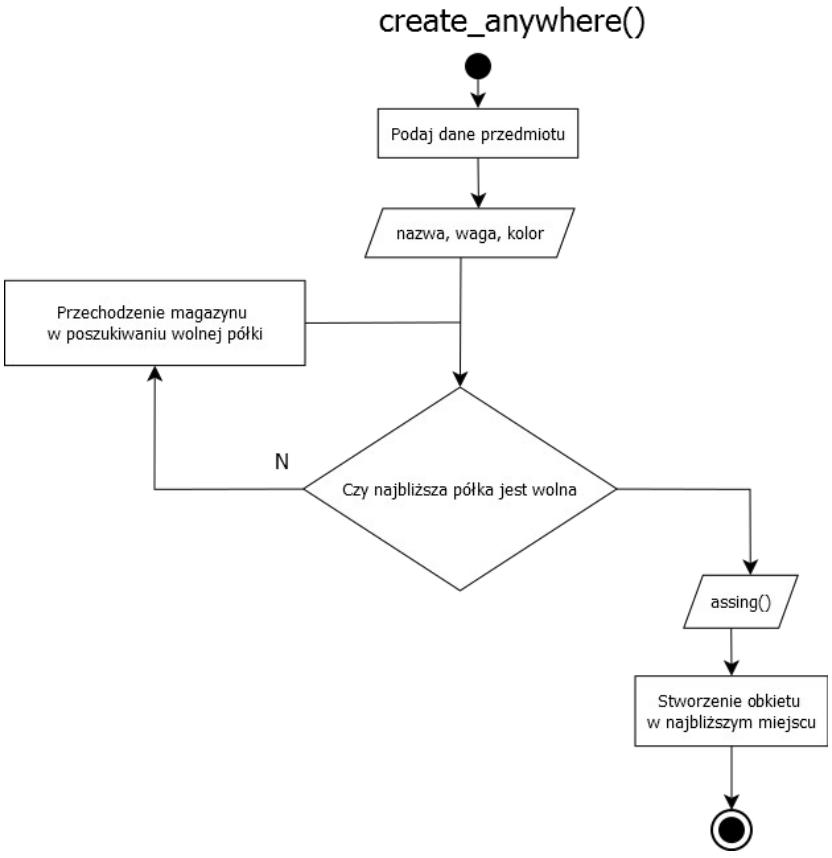
Po wybraniu tej opcji użytkownik wprowadza parametry przedmiotu takie jak nazwa, waga, kolor oraz podaje współrzędne miejsca i numer półki na której ma być umieszczony przedmiot. Program sprawdza czy w danym miejscu można go umieścić – jeśli nie (w tym miejscu nie ma półek lub półka jest zajęta) to na ekranie zostanie wypisany stosowny komunikat, a jeśli tak to obiekt zostanie odłożony przez wózek widłowy w podane miejsce.





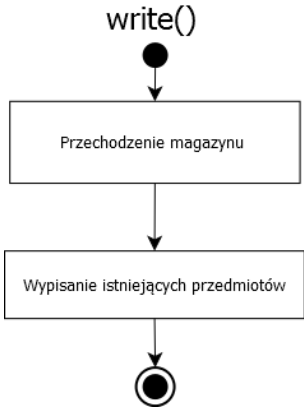
Tworzenie przedmiotu w dowolnym miejscu (create_anywhere())

Metoda ta działa w podobny sposób jak tworzenie przedmiotu jednak użytkownik nie musi określać miejsca w którym ma być stworzony przedmiot – jest on odkładany na pierwszej wolnej półce.



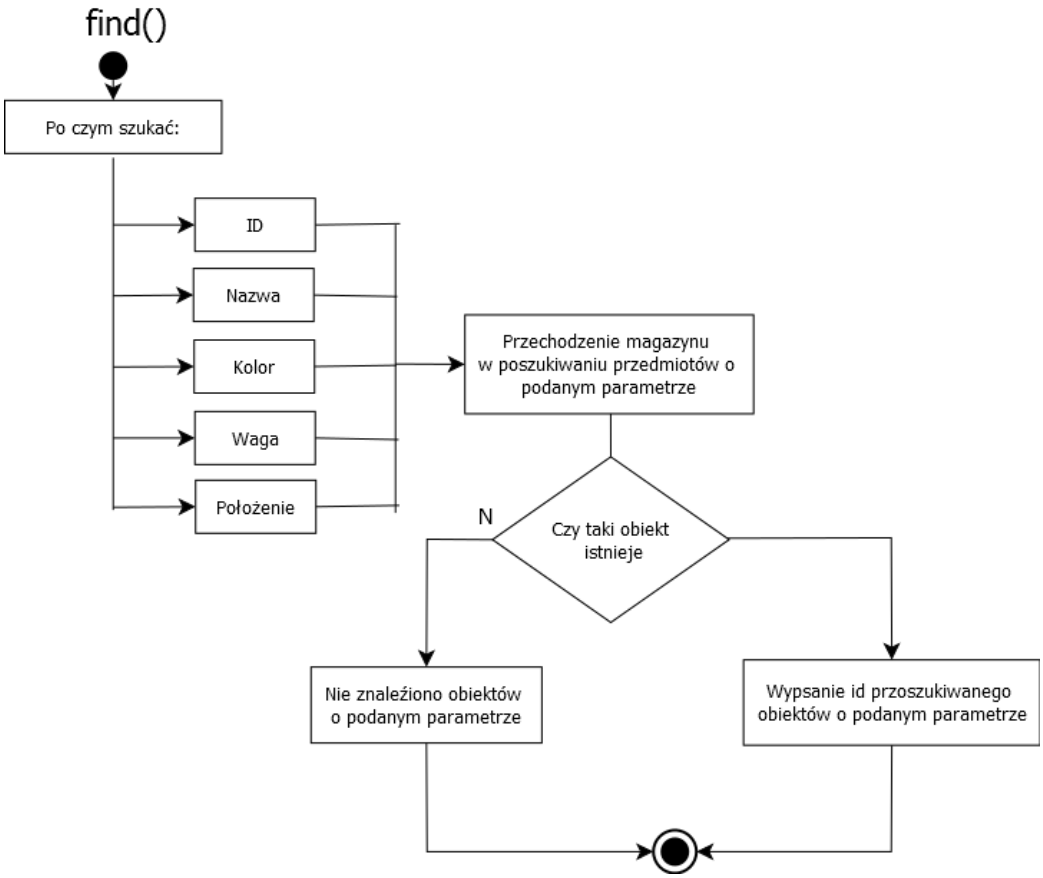
Wypisywanie przedmiotów (write())

Na ekranie zostaje wypisana lista i informacje o przedmiotach znajdujących się w magazynie.



Znajdowanie przedmiotów (find())

Funkcja ta wykorzystywana przez inne metody (przenoszenie, usuwanie, zamianę). Użytkownik podaje cechę produktu i program przeszukuje magazyn w poszukiwaniu obiektów pasujących do wzorca.

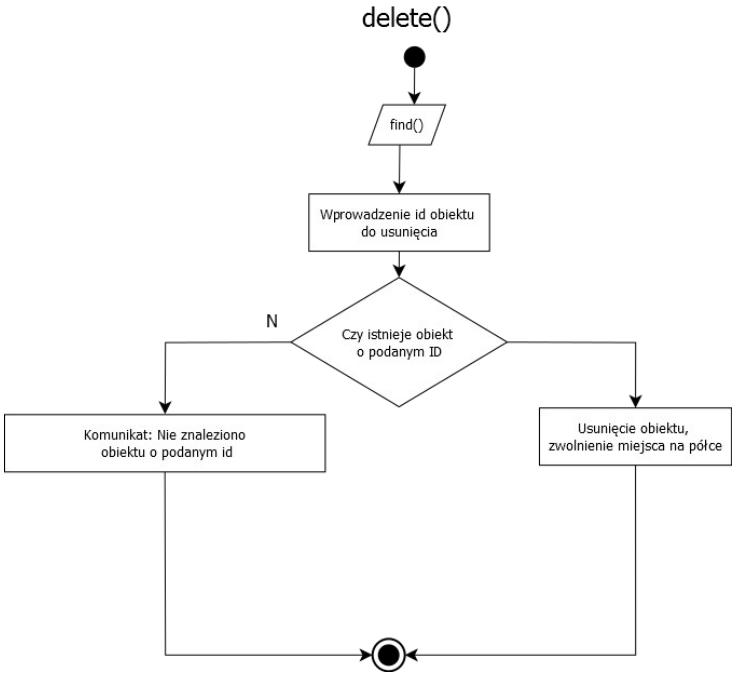


Przenoszenie przedmiotu (move())

Najpierw użytkownik podaje jedną z cech przedmiotu. Program wyszukuje przedmioty o wybranej cesze korzystając z funkcji find() i jeśli istnieją to wypisuje je na ekranie. Następnie użytkownik podaje unikalne id obiektu i miejsce w które dany obiekt ma zostać przeniesiony. Jeśli to miejsce jest zajęte to na ekranie zostanie wypisany stosowny komunikat, a jeśli jest wolne to wózek widłowy przeniesie przedmiot.

Usuwanie przedmiotu (delete())

Najpierw użytkownik podaje jedną z cech przedmiotu. Program wyszukuje przedmioty o wybranej cesze korzystając z funkcji find() i jeśli istnieją to wypisuje je na ekranie. Następnie użytkownik podaje unikalne id obiektu, który ma zostać usunięty a miejsce, w którym się znajdował zostaje zwolnione.



Zamiana przedmiotów (replace())

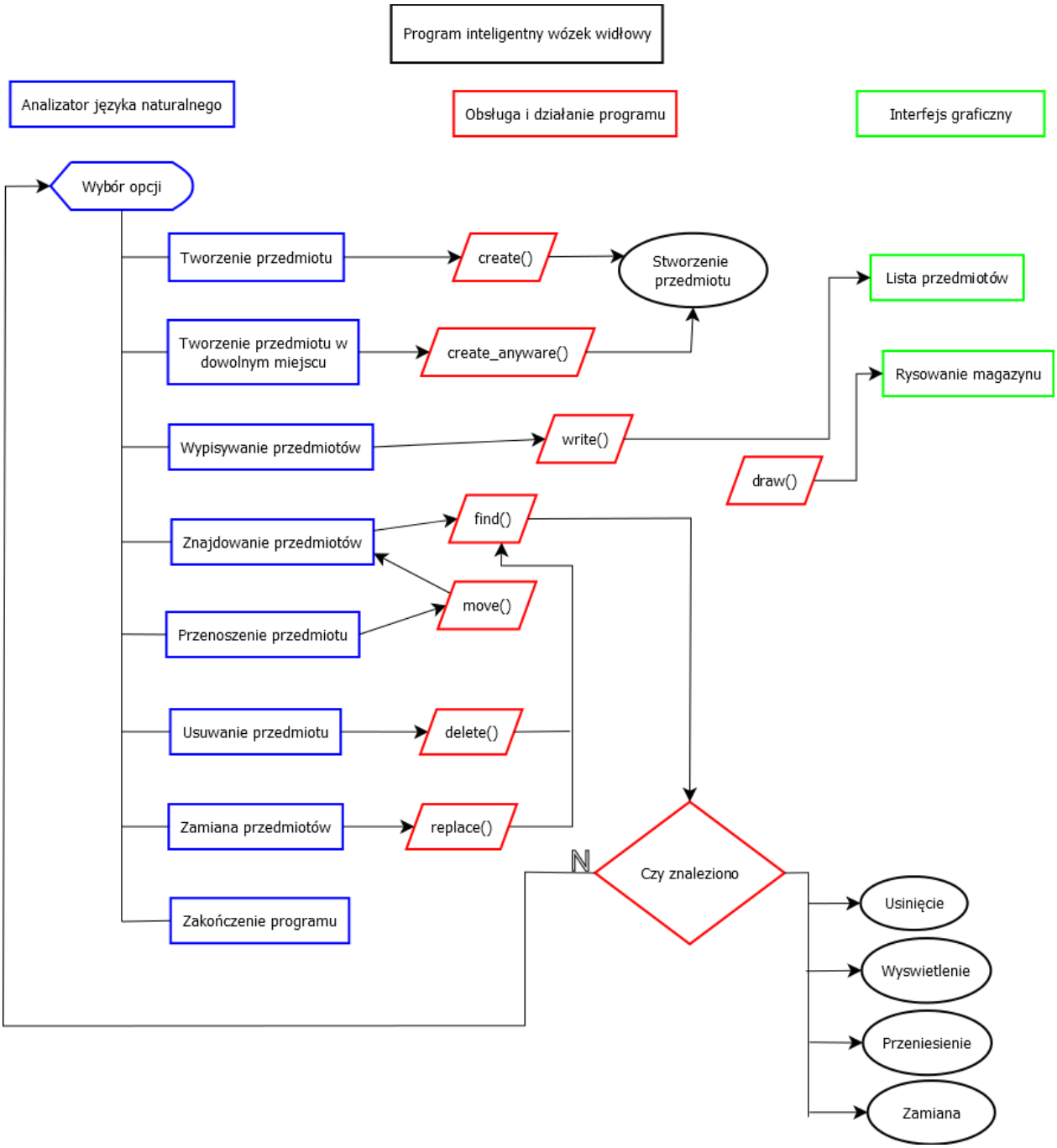
Najpierw użytkownik podaje cechy przedmiotów, które mają być zamienione. Program wyszukuje przedmioty o wybranych cechach korzystając z funkcji find() i jeśli istnieją to wypisuje je na ekranie. Następnie użytkownik podaje unikalne id obiektów i wózek widłowy zamienia miejscami przedmioty wybrane przez użytkownika.

Zakończenie programu

Powoduje zamknięcie aplikacji

Diagram programu:

Diagram programu przedstawiający sposób działania:



Testy działania programu:

Po uruchomieniu programu na ekranie wyświetla się interfejs graficzny na którym jest narysowany magazyn oraz menu:

1) Stwórz obiekt i umieść go w magazynie

2) Stwórz obiekt i umieść go w dowolnym miejscu

3) Usuń obiekt

4) Znajdź obiekt

5) Przenieś obiekt

6) Zamień miejscami obiekty

7) Wypisz obiekty w magazynie

8) Rozrysuj magazyn

9) Zakończ

1) Użytkownik wprowadza dane przedmiotu:

Podaj dane przedmiotu nr: 1
Nazwa:
Waga:
Kolor:
Gdzie umieścić obiekt?
wiersz:
kolumna:
poziom:

Po wprowadzaniu poprawnych danych wyświetlone zostają informacje o nowopowstałym przedmiocie, np:

Dane obiektu:
Nr obiektu: 1
Nazwa: opona
Waga: 3.0
Kolor: czarny
Położenie: pole [1,1] na półce nr 1

Po stworzeniu przedmiotu o wadze większej niż 70 kg:

komunikat: „Nie można umieścić na półce - przedmiot jest za ciężki” , powrót do menu

Po stworzeniu przedmiotu w niedozwolonym miejscu:

komunikat: „Nie można umieścić obiektu w tym miejscu” , powrót do menu

2) Użytkownik wprowadza dane przedmiotu:

Podaj dane przedmiotu nr: 1
Nazwa:
Waga:
Kolor:

Po wprowadzaniu poprawnych danych wyświetlone zostają informacje o nowopowstałym przedmiocie (jw.):

Po stworzeniu przedmiotu o wadze większej niż 70 kg:

komunikat: „Nie można umieścić na półce - przedmiot jest za ciężki” , powrót do menu

3) Wykorzystanie wyszukiwania z pkt. 4). Następnie po wypisaniu informacji o wyszukanych obiektach użytkownik jest proszony o podanie id obiektu do usunięcia:

komunikat: „wprowadź id obiektu, który chcesz usunąć”

Po wprowadzeniu id nieistniejącego obiektu:

komunikat: „Nie znaleziono obiektu o podanym id” , powrót do menu

Po wprowadzeniu id istniejącego obiektu, obiekt zostaje usunięty.

4) Zapytanie o parametr przedmiotu i wybór opcji wyszukiwania:

Po czym szukać?:

1) id
2) nazwa
3) waga
4) kolor
5) położenie

Przy wybraniu innej wartości:

komunikat: „Błędne dane”, powrót do menu

Użytkownik wprowadza parametr np. nazwę:

Podaj nazwę przedmiotu:

Jeśli obiekty o podanym parametrze nie istnieją:

komunikat: „Nie znaleziono obiektu o podanym id/nazwie/wadze/kolorze/w podanym miejscu” , powrót do menu

Jeśli takie obiekty istnieją informacje o nich zostają wypisane na ekranie.

5) Wykorzystanie wyszukiwania z pkt. 4). Następnie po wypisaniu informacji o wyszukanych obiektach użytkownik jest proszony o podanie id obiektu do przeniesienia:

komunikat: „wprowadź id obiektu, który chcesz przenieść”

Po wprowadzeniu id nieistniejącego obiektu:

komunikat: „Nie znaleziono obiektu o podanym id” , powrót do menu

Po wprowadzeniu id istniejącego obiektu:

Wprowadź współrzędne pola gdzie chcesz przenieść obiekt

wiersz:

kolumna:

nr półki:

Jeżeli użytkownik poda miejsce w którym nie ma półek lub pólka jest zajęta:

komunikat: „Nie można przenieść obiektu w podane miejsce” , powrót do menu

Jeśli wprowadzone dane są poprawne to przedmiot zostaje przeniesione w miejsce wybrane przez użytkownika.

6) Wykorzystanie wyszukiwania z pkt. 4) dla pierwszego obiektu. Następnie po wypisaniu informacji o wyszukanych obiektach użytkownik jest proszony o podanie id obiektu, który chce zamienić:

pierwszy obiekt:

wprowadź id obiektu, który chcesz zamienić

Po wprowadzeniu id nieistniejącego obiektu:

komunikat: „Nie znaleziono obiektu o podanym id” , powrót do menu

Po wprowadzeniu id istniejącego obiektu w ten sam sposób następuje wybór drugiego obiektu. Jeśli wprowadzone dane są poprawne to przedmioty zostają zamienione miejscami.

7) Na ekranie zostają wypisane dane obiektów znajdujących się w magazynie, np.:

Dane obiektu:
Nr obiektu: 1
Nazwa: deska
Waga: 5.0
Kolor: brazowy
Położenie: pole [1,1] na półce nr 3

8) Na ekranie zostanie rozrysowany magazyn z uwzględnionymi półkami i drogą przeznaczoną dla wózka widłowego.

9) Powoduje zakończenie aplikacji.

Przy wybraniu innej wartości:

komunikat: „Błędne dane” , powrót do menu

Dokumentacja techniczna

Indeks hierarchiczny

Hierarchia klas

1. wozekwidlowy.Cart
2. wozekwidlowy.Object
3. wozekwidlowy.Storehouse
    a. wozekwidlowy.Shelf

4. Canvas
    a. wozekwidlowy.Forklift

5. JTextArea
    a. wozekwidlowy.InformationPanel
    b. wozekwidlowy.CommandlinePanel

Indeks klas

Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

1. **wozekwidlowy.Cart**
2. **wozekwidlowy.CommandlinePanel**
3. **wozekwidlowy.Forklift**
4. **wozekwidlowy.InformationPanel**
5. **wozekwidlowy.Object**
6. **wozekwidlowy.Shelf**
7. **wozekwidlowy.Storehouse**

Dokumentacja klas

Dokumentacja klasy wozekwidlowy.Cart

Metody publiczne

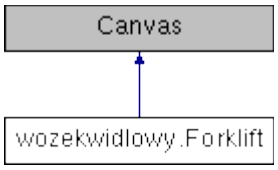
- void **set_coordinates** (int x, int y)
- int **get_wsp_x** ()
- int **get_wsp_y** ()

Dokumentacja klasy wozekwidlowy.CommandlinePanel

Diagram dziedziczenia dla wozekwidlowy.CommandlinePanel

Dokumentacja klasy wozekwidlowy.Forklift

Diagram dziedziczenia dla wozekwidlowy.Forklift



Statyczne metody publiczne

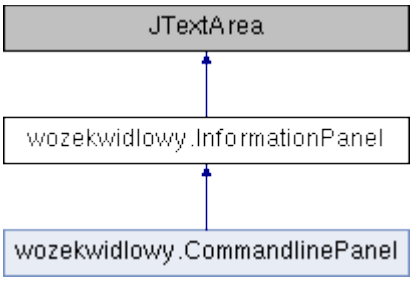
- static void **main** (String[] args) throws IOException

Dokumentacja funkcji składowych

static void wozekwidlowy.Forklift.main (String[] args) throws IOException `[static]`

Dokumentacja klasy wozekwidlowy.InformationPanel

Diagram dziedziczenia dla wozekwidlowy.InformationPanel



Metody publiczne

- short **getActualInfoCount** ()
- void **addInfo** (String info)
- String **getInfo** (int infoNumber)
- void **printInfo** ()
- void **removeInfo** ()
- void **removeInfo** (int infoNumber)

Atrybuty chronione

- Dimension **panelSize** = Toolkit.getDefaultToolkit().getScreenSize()

Dokumentacja funkcji składowych

void wozekwidlowy.InformationPanel.addInfo (String info)

short wozekwidlowy.InformationPanel.getActualInfoCount ()

String wozekwidlowy.InformationPanel.getInfo (int infoNumber)

void wozekwidlowy.InformationPanel.removeInfo ()

void wozekwidlowy.InformationPanel.removeInfo (int infoNumber)

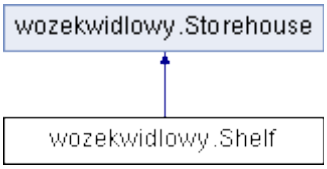
Dokumentacja klasy wozekwidlowy.Object

Metody publiczne

- int **get_id** ()
- String **get_name** ()
- double **get_weight** ()
- String **get_color** ()
- int **get_x** ()
- int **get_y** ()
- int **get_numberOfShelf** ()
- void **set_nazwa** (String name)
- void **set_weight** (double weight)
- void **set_color** (String color)
- void **set_x** (int x)
- void **set_y** (int y)
- void **set_numberOfShelf** (int numberOfShelf)
- void **set_coordinates** (int x, int y, int numberOfShelf)

Dokumentacja klasy wozekwidlowy.Shelf

Diagram dziedziczenia dla wozekwidlowy.Shelf

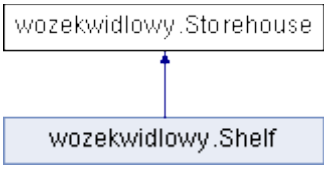


Metody publiczne

- double **get_lifting** ()
- **Object get_Object** ()
- void **writeInfo** ()
- void **assign** (**Object** object, **Storehouse** field[][][], int x, int y, int level)

Dokumentacja klasy wozekwidlowy.Storehouse

Diagram dziedziczenia dla wozekwidlowy.Storehouse



Metody publiczne

- void **set_isThereShelf** (boolean isThereShelf)+
- void **set_isThereObject** (boolean isThereObject)
- boolean **get_isThereShelf** ()
- boolean **get_isThereObject** ()
- abstract **Object get_Object** ()
- void **create** (**Storehouse** field[][][]) throws IOException
- void **create_anywhere** (**Storehouse** field[][][]) throws IOException
- void **delete** (**Storehouse** field[][][], **Storehouse** storehouse) throws IOException
- void **move** (**Storehouse** field[][][], **Storehouse** storehouse) throws IOException
- void **replace** (**Storehouse** field[][][], **Storehouse** storehouse) throws IOException
- void **write** (**Storehouse** field[][][])
- void **draw** (**Storehouse** field[][][])
- void **case_id** (**Storehouse** field[][][], String return_) throws IOException
- void **case_name** (**Storehouse** field[][][]) throws IOException
- void **case_weight** (**Storehouse** field[][][]) throws IOException
- void **case_color** (**Storehouse** field[][][]) throws IOException
- void **case_coordinates** (**Storehouse** field[][][]) throws IOException
- void **find** (**Storehouse** field[][][]) throws IOException

Atrybuty chronione

- boolean **m_isThereObject** = false