

# Symulator Parku Wodnego

## Dokumentacja

### SPIS TREŚCI

|   |          |
|---|----------|
| <b>1. Temat projektu</b>                        | <b>2</b> |
| <b>2. Założenia projektu</b>                    | <b>2</b> |
| 2.1 Pracownicy                                  | 2        |
| 2.2 Atrakcje                                    | 2        |
| 2.3 Klienci                                     | 3        |
| 2.4 Jednostka czasu                             | 3        |
| <b>3. Architektura</b>                          | <b>3</b> |
| 3.1 Opis ogólny                                 | 3        |
| 3.2 Klasa Client                                | 4        |
| 3.3 Schemat dziedziczenia dla klasy pracownika  | 4        |
| 3.4 Schemat dziedziczenia dla klasy atrakcji    | 5        |
| 3.5 Diagram przepływu informacji między klasami | 6        |
| <b>4. Opis działania symulacji</b>              | <b>6</b> |
| 4.1 Przebieg symulacji                          | 6        |
| 4.2 Instrukcja obsługi                          | 7        |
| 4.3 Opis danych wejściowych                     | 7        |
| 4.4 Opis danych wyjściowych                     | 7        |
| <b>5. Aspekty techniczne</b>                    | <b>8</b> |
| 5.1 Technologie                                 | 8        |
| 5.2 Biblioteka STL                              | 8        |
| <b>6. Wyjątki</b>                               | <b>8</b> |
| <b>7. Sposób testowania</b>                     | <b>8</b> |
| <b>8. Podział ról</b>                           | <b>9</b> |
| <b>9. Podsumowanie</b>                          | <b>9</b> |
| <b>10. Autorzy</b>                              | <b>9</b> |

# 1. Temat projektu

Projekt symuluje działanie parku wodnego. Zawiera różnorodne atrakcje dostępne dla klientów. Pracownicy parku wodnego odpowiadają za jego prawidłowe funkcjonowanie. Symulacja przez określoną ilość czasu przedstawia działania klientów parku wodnego, umożliwiając im przyjsie do parku, skorzystania z atrakcji i wyjścia po określonym czasie. Za skorzystanie z usług klient musi wnieść odpowiednią opłatę.

## 2. Założenia projektu

### 2.1 Pracownicy

- Pracownicy parku wodnego to: kasjerzy, instruktorzy, ratownicy
- Każdy pracownik ma swoje unikalne ID
- Każdy kasjer jest przypisany do kasy o numerze takim samym jak jego ID
- Instruktor ma określony poziom kwalifikacji:
  - 1 - podstawowy
  - 2 - średni
  - 3 - zaawansowany
- Instruktor jest dowolnie wybierany przez klienta, w tym samym czasie instruktor może trenować dowolną liczbę klientów
- Kwalifikacje ratownika można stwierdzić po ilości przepracowanych lat

### 2.2 Atrakcje

- W parku wodnym mogą znajdować się atrakcje 10 rodzajów:
  1. *Aqua Bar*
  2. *Bouncy Castle*
  3. *Jacuzzi*
  4. *Paddling Pool*
  5. *Pontoon Ride*
  6. *Sauna*
  7. *Slide*
  8. *Beginner Swimlane*
  9. *Standard Swimlane*
  10. *Pro Swimlane*
- Atrakcje podzielone są na przeznaczone dla dzieci (klienci poniżej 18 roku życia) oraz dorosłych
- Każda atrakcja posiada unikalną nazwę
- Każdy rodzaj atrakcji posiada określoną cenę biletu za wejście
- Atrakcje mają sprecyzowaną ilość jednostek czasu, jakie klienci mogą maksymalnie na nich spędzić wedle zakupionego biletu
- Tory pływackie (... *Swimlane*) posiadają 3 rodzaje głębokości, odpowiednie do umiejętności pływackich odwiedzających je klientów

## 2.3 Klienci

- Klient ma unikalną nazwę, zawierającą jego numer przyścia do parku, przykładowo „Klient nr 1”
- Klienci w parku mogą mieć od 5 do 100 lat
- Klienci mogą wejść na atrakcje tylko przystosowane do swojego wieku:
  - mniej niż 18 lat - atrakcje dla dzieci
  - co najmniej 18 lat - atrakcje dla dorosłych
- Każdy klient ma określony poziom umiejętności pływania:
  - 1 - początkujący
  - 2 - średniozaawansowany
  - 3 - zaawansowany
- W zależności od poziomu umiejętności pływania klient może wybrać przystosowany dla siebie tor pływacki
- W momencie wejścia do parku wodnego klient kupuje bilet wejściowy na określoną liczbę jednostek czasu

## 2.4 Jednostka czasu

- Za jednostkę czasu przyjmujemy moment, w którym każdy klient może wykonać jakąś czynność
- W każdej jednostce czasu nowy klient przychodzi do parku
- Jeśli klient obecnie nie jest na żadnej atrakcji, wybiera na którą chciałby pójść
- Jeśli klient do tej pory nie wykupił instruktora, podejmuje decyzję czy chce go teraz zakupić, jeśli tak, może dowolnie wybrać którego
- Jeśli klient spędził w parku wodnym już wszystkie wykupione jednostki czasu, wychodzi z obiektu
- Symulacja trwa podaną na wejściu ilość jednostek czasu, nie musi się zakończyć w momencie wyjścia wszystkich klientów

# 3. Architektura

## 3.1 Opis ogólny

Główną klasą projektu odpowiadającą za przeprowadzenie symulacji jest klasa *Simulation*. Klasa logger służy wypisywaniu komunikatów opisujących przebieg symulacji, a także przekierowywaniu ich do plików.

Bazy danych są przechowywane w klasach:

- *EmployeesList* - baza danych pracowników
- *AttractionsDB* - baza danych atrakcji
- *ClientDB* - baza danych klientów

Każda z nich zawiera wektor wskaźników inteligentnych do obiektów, a także metody umożliwiające między innymi dodanie lub usunięcie danego obiektu z bazy danych.

Klasa *Client* przechowuje informację o kliencie parku wodnego. Klasa *Receipt* zawiera listę atrakcji, z których skorzystał klient, informację o wybranym instruktorze oraz łączny koszt pobytu klienta w parku wodnym.

Klasa *Client* oraz podział klas pracowników i atrakcji został przedstawiony w następnych punktach.

## 3.2 Klasa Client

Klasa *Client* przechowuje podstawowe dane klienta parku wodnego:

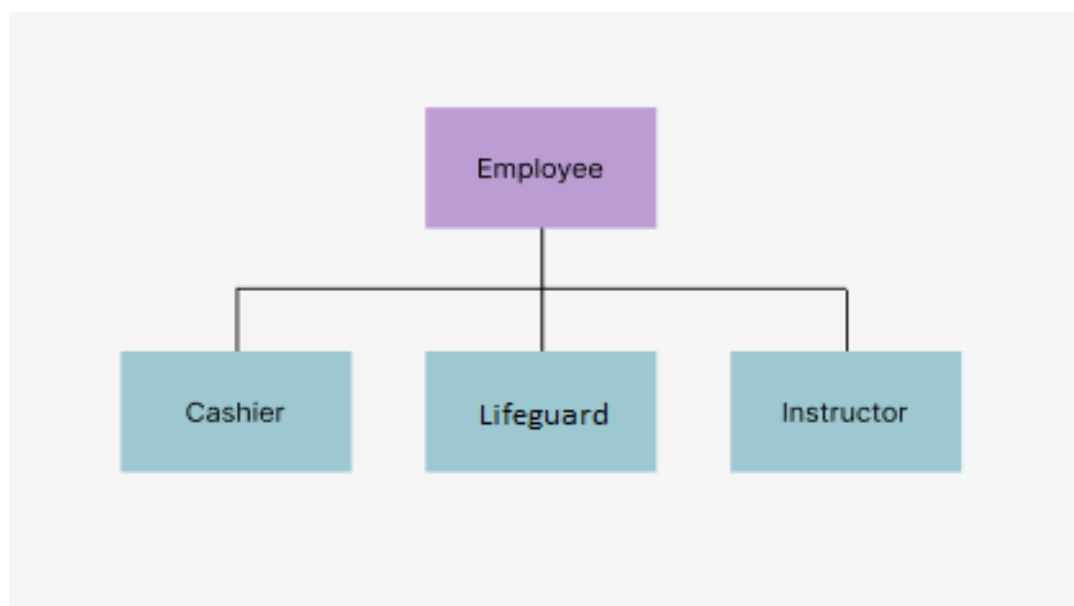
- Nazwa wraz z ID - przykładowo: Klient nr 1
- Umiejętności pływackie
- Wiek
- Liczba jednostek czasu, jakie klient spędzi w parku
- Liczba jednostek czasu, które spędzi na danej atrakcji
- Paragon

Dodatkowo klasa *Client* posiada metody:

- Gettery
- Settery
- Losowy wybór atrakcji
- Losowy wybór instruktora - decyzja czy wybrać, jeśli tak to decyzja którego

## 3.3 Schemat dziedziczenia dla klasy pracownika

Na górze klasa bazowa. Fioletowym kolorem zaznaczone zostały klasy abstrakcyjne.

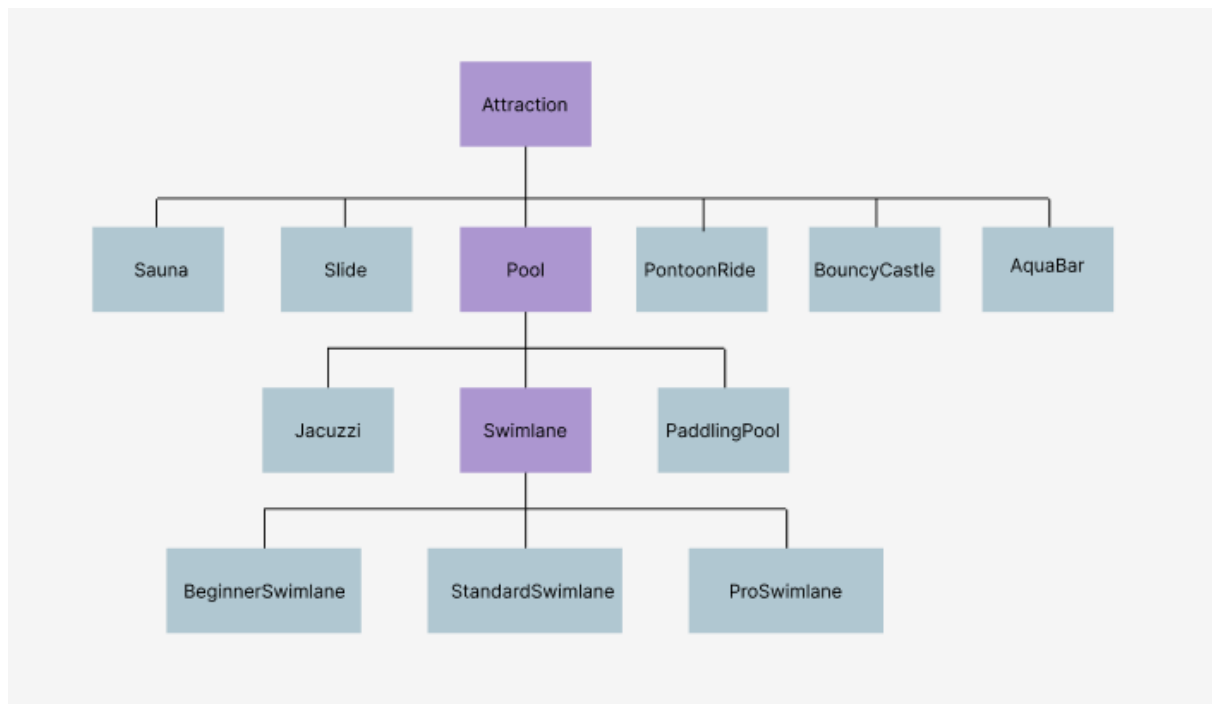


By zapewnić prawidłowe funkcjonowanie pracowników w Parku Wodnym, utworzono klasę *Employee*, po której dziedziczą klasy: *Cashier*, *Lifeguard*, *Instructor*.

Wymienione klasy zawierają między innymi:

- Klasa *Employee* - zawiera nazwę pracownika, unikalne id, a także gettery i settery
- Klasa *Cashier* - dodatkowo zawiera numer kasy przypisanej do kasjera
- Klasa *Instructor* - dodatkowo zawiera poziom umiejętności instruktora
- Klasa *Lifeguard* - dodatkowo zawiera ilość przepracowanych lat przez pracownika

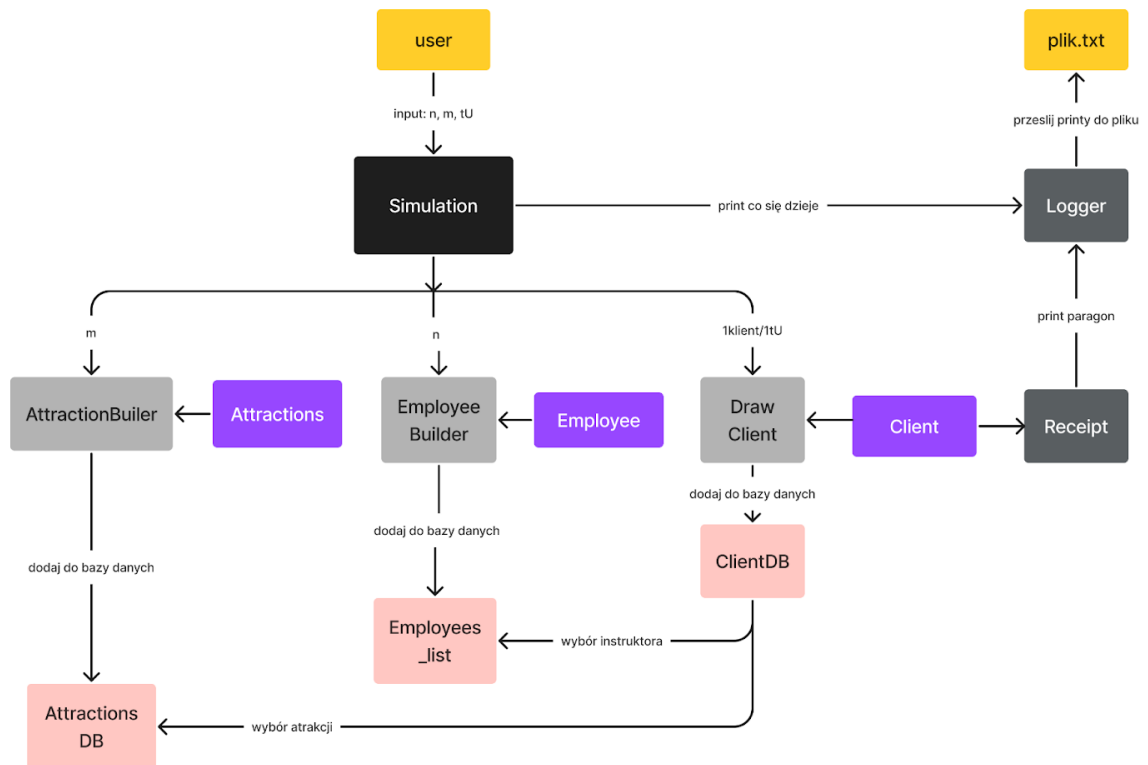
### 3.4 Schemat dziedziczenia dla klasy atrakcji



Klasa *Attraction* została utworzona w celu reprezentowania atrakcji w Parku Wodnym. Klasy po niej dziedziczące są już konkretnymi rodzajami atrakcji.

- Klasa *Attraction* zawiera:
  - Unikalną nazwę stanowiącą jej id,
  - Cenę biletu wstępu,
  - Ilość jednostek czasu ile można na niej spędzić według biletu,
  - Flagę czy jest ona dla dzieci,
  - Gettery i settery.
- Klasy dziedziczące dodatkowo po klasie *Swimlane* - dodano właściwość: głębokość toru pływackiego.

### 3.5 Diagram przepływu informacji między klasami



## 4. Opis działania symulacji

### 4.1 Przebieg symulacji

W celu symulowania działania parku wodnego, z poziomu pliku main.cpp tworzymy obiekt klasy *Simulation* i uruchamiamy symulację. Dzięki klasom *AttractionDB*, *EmployeesList*, *ClientDB* generowane są obiekty potrzebne do przeprowadzenia symulacji.

Symulacja opiera się na działaniu pętli for. Każda jej iteracja to imitacja wirtualnie mijającego czasu. W każdym kroku przychodzi nowy klient do parku wodnego, a także każdy klient może wykonać:

- Wybór nowej atrakcji, jeśli aktualnie nie jest na żadnej
- Wybór instruktora, jeśli do tej pory go nie wybrał
- Wyjście z parku wodnego, jeśli upłynęły już jego jednostki czasu

## 4.2 Instrukcja obsługi

W celu rozpoczęcia symulacji należy skompilować program, a następnie uruchomić go z poziomu konsoli programu, wraz z argumentami wejścia.

## 4.3 Opis danych wejściowych

Argumenty wejścia:

- $n$  - ilość pracowników w parku wodnym
- $m$  - ilość atrakcji w parku wodnym
- $time\_units$  - ilość jednostek czasu, przez które ma trwać symulacja

Plik tekstowy:

Służy do budowy bazy danych atrakcji. Każda linijka pliku jest w formacie <typ atrakcji>,<nazwa>, np:

*Slide,Zjeżdżalnia Twister*

*Slide,Zjeżdżalnia Boa Topiciel*

*Sauna,Sauna Duszek*

*Sauna,Sauna Goraczka Sobotniej Nocy*

## 4.4 Opis danych wyjściowych

Przebieg symulacji jest wyświetlany na konsolę w czasie rzeczywistym. Dodatkowo cała symulacja jest zapisywana do pliku tekstowego: *simulation\_log.txt*

Przykładowe logi::

```
##### SYMULACJA PARK WODNY #####

----- TWORZENIE ATRAKCJI -----
1. Standardowy tor plywacki 2 dla dzieci?: TAK
2. Poczatkujacy tor plywacki 1 dla dzieci?: TAK
3. Standardowy tor plywacki 1 dla dzieci?: TAK
4. Bar Pod Palemka dla dzieci?: NIE
5. Standardowy tor plywacki 4 dla dzieci?: TAK
6. Jacuzzi Kociol Czarownicy dla dzieci?: NIE
7. Sauna Duszek dla dzieci?: NIE
8. Brodzik Mala Syrenka dla dzieci?: TAK
9. Sauna Finska dla dzieci?: NIE
10. Brodzik Ratunku jestem Rybka dla dzieci?: TAK
-----

----- TWORZENIE PRACOWNIKOW -----
1. Instructor 0
2. Cashier 1
3. Lifeguard 2
4. Instructor 3
5. Cashier 4
-----
```

```
_____ Symulacja 23 jednostka czasu _____

Przychodzi klient (Client nr 23)
Na 16 jednostki czasu
Jego poziom umiejetnosci plywackich: 1
Wiek: 69

Klient (Client nr 7)
Wybral atrakcje: Dmuchaniec Wielki Banan

Wychodzi klient (Client nr 11)
----- Paragon -----
Standardowy tor plywacki 2 --- cena: 14
Instructor 3 --- cena: 30
| | | | | | | Cena laczna: 44
-----

Klient (Client nr 23)
Wybral atrakcje: Jacuzzi Czara Ognia
```

## 5. Aspekty techniczne

### 5.1 Technologie

Projekt został stworzony w języku programowania C++, korzystając ze środowiska Visual Studio Code.

Korzystano również z systemu kontroli wersji git. Cały projekt jest dostępny na GitLabie.

### 5.2 Biblioteka STL

W projekcie wykorzystane zostały elementy biblioteki standardowej takie jak:

- `<vector>` - do przechowywania np. wskaźników w bazach danych
- `<algorithm>` - w celu użycia metody *find*
- `<string>` - do pracy z łańcuchami znaków

## 6. Wyjątki

Sytuacje wyjątkowe zostają rozpoznane i rzucone jako jeden z własnych napisanych wyjątków, a następnie odpowiednio obsługiwane. O wystąpieniu niektórych sytuacji niekrytycznych, informuje nas komunikat wyświetlony na konsoli.

Przykładowe sytuacje wyjątkowe i rzucane od nich wyjątki:

- *EmptyNameException* - rzucony przy próbie ustawienia imienia na pusty ciąg znaków
- *NegativeArgumentException* - rzucony przy próbie ustawienia parametru na liczbę ujemną
- *CannotFindEmployeeException* - rzucony przy próbie modyfikacji lub usunięcia nieistniejącego pracownika
- *InvalidAttractionsListException* - rzucony gdy klient nie mógł dołączyć na żadną atrakcję, na przykład gdy klient był dorosły a wszystkie atrakcje były dla dzieci
- *CannotOpenFileException* - rzucony przy błędzie otwarcia pliku

## 7. Sposób testowania































Testowanie klas głównych zostało szczegółowo przeprowadzone za pomocą różnorodnych testów jednostkowych z wykorzystaniem biblioteki *Catch*.

Działanie elementów bardziej złożonych, jak na przykład symulacja, na bieżąco, tj. równoległe z implementacją, było sprawdzane za pomocą funkcji pomocniczych w main.



## 8. Podział ról

Organizacja pracy i rozdzielanie zadań była wykonywana z pomocą serwera GitLab. Poniższe przykładowe Issues, które były podzielone równomiernie.

|  |   |
|--|---|
| Zwolnić pamięć przed zakończeniem symulacji<br>#14 · created 6 days ago by dferfeck          | CLOSED   0<br>updated 21 hours ago  |
| Stworzyć klasę losującą atrybuty klienta<br>#12 · created 1 week ago by dferfeck             | CLOSED   1  0<br>updated 1 week ago        |
| Stworzyć klasę dodającą do bazy danych n pracowników<br>#11 · created 1 week ago by dferfeck | CLOSED   0<br>updated 1 week ago  |
| Stworzyć klasę dodającą do bazy danych m atrakcji<br>#10 · created 1 week ago by dferfeck    | CLOSED   0<br>updated 1 week ago  |
| Wypisywanie logów<br>#9 · created 1 month ago by dferfeck                                    | CLOSED   0<br>updated 2 days ago  |
| Stworzyć klasę Symulacja<br>#8 · created 1 month ago by dferfeck                             | CLOSED   1  0<br>updated 1 week ago        |
| Odczytywanie argumentów wywołania programu<br>#7 · created 1 month ago by dferfeck           | CLOSED  1  0<br>updated 1 week ago  |
| Stworzyć klasę paragon<br>#6 · created 1 month ago by dferfeck                               | CLOSED   1  0<br>updated 3 weeks ago       |
| Stworzenie bazy danych klientów<br>#4 · created 1 month ago by dferfeck                      | CLOSED   0<br>updated 1 week ago  |
| Stworzenie klasy Klient<br>#3 · created 1 month ago by dferfeck                              | CLOSED   1  0<br>updated 3 weeks ago       |
| Dodanie klas: atrakcje<br>#2 · created 1 month ago by mczech1                                | CLOSED   1  0<br>updated 1 week ago        |
| Dodanie klas pracowników<br>#1 · created 1 month ago by dferfeck                             | CLOSED   1  0<br>updated 4 weeks ago |

## 9. Podsumowanie

Projekt udało się skończyć realizując wszystkie postawione wymagania. Park Wodny zawiera pracowników, atrakcje, klientów, a przebieg symulacji jest przedstawiany na konsoli i w pliku.

Tworząc nasz projekt spędziłyśmy wiele czasu by jak najlepiej dopracować nasz kod. Musiałyśmy podjąć decyzję jakie stworzymy klasę, uzgodnić hierarchię, jakich technologii będziemy używać. Biorąc pod uwagę, że była to praca w zespole dwuosobowym, zwracaliśmy uwagę aby wspólnie ustalić główny koncept i równomiernie rozłożyć zadania.

Tworzenie projektu rozwinęło nasze umiejętności z programowania obiektowego, a efekt końcowy jest zadowalający.

## 10. Autorzy

Projekt wykonany przez:

- Dominika Ferfecka
- Magdalena Czech