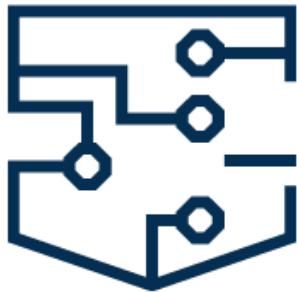


Programowanie Obiektowe



Laboratorium 4 – modyfikacja przykładowej aplikacji

Tomasz Bieliński
2024-03-13



Przygotowanie do zajęć w domu

Celem tego laboratorium będzie rozwinięcie prostej aplikacji napisanej w języku Java. Na początku zostaną omówione niektóre elementy języka Java.

Zakresy widoczności

W języku Java są 4 zakresy widoczności:

- Publiczny oznaczany słowem kluczowym **public** – pole, metoda lub klasa oznaczone tym modyfikatorem jest dostępne **dla każdej innej klasy z każdego pakietu**,
- Pakietu oznaczany przez brak modyfikatora zakresu widoczności – pole, metoda lub klasa tego typu jest widoczne **dla wszystkich klas z tego samego pakietu**,
- Chroniony oznaczany słowem kluczowym **protected** – pole, metoda lub klasa oznaczone tym modyfikatorem jest dostępne **tylko dla klasy w, której jest ono/ona zdefiniowane lub dla klas pochodnych**,
- Prywatny oznaczany słowem kluczowym **private** – pole, metoda lub klasa oznaczone tym modyfikatorem jest dostępne **tylko dla klasy w której jest ono/ona zdefiniowane**.

Typ wyliczeniowy

W języku Java również jest dostępny typ wyliczeniowy, jednak różni się on od analogicznego typu w języku C++. W języku C++ typ wyliczeniowy jest traktowany jak typ prosty, natomiast w języku Java jest to **typ referencyjny**. Przykład przedstawia Listing 1.

Listing 1 Przykład typu wyliczeniowego

```
package pl.edu.pg.eti.ksg.po.lab2.biegpolesie;

/**
 * Rodzaje terenu na jakie mogą natknąć się uczestnicy biegu po lesie
 * @author TB
 */
public enum RodzajTerenu {
    /**
     * Teren po którym łatwo jest się porsuwać
     */
    DROGA,
    /**
     * Nieco mnie przebieżna niż {@link
pl.edu.pg.eti.ksg.po.lab2.biegpolesie.RodzajTerenu#DROGA}
     */
    SCIEZKA,
    /**
     * Teren w którym trzeba się ostrożnie pruszać
     */
    WYSOKI_LAS,
    /**
     *
    
```



```

        * Teren trudny
        */
NISKI_LAS,

< /**
     * Teren niebezpieczny, prawie niemożliwy do przebycia
     */
BAGNO;
}

```

Kolekcje

W bibliotece standardowej języka Java znajduje się dużo gotowych klas implementujących kolekcję. Kolekcje implementują całą hierarchię interfejsów, które standaryzują operacje na kolekcjach niezależnie od wybranej implementacji. Listing 2 przedstawia zastosowanie klasy `java.util.HashSet` poprzez interfejs `java.util.Set`.

Listing 2 Zastosowanie klasy `java.util.HashSet` rzutowanej na interfejs `java.util.Set`

```

package pl.edu.pg.eti.ksg.po.lab2;

< /**
     * ...
     */
public class JavaLab2 {
    public static void main(String[] args) {

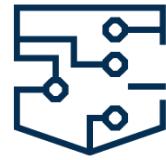
        Set<Uczestnik> uczestnicy = new HashSet<Uczestnik>();
        uczestnicy.add(new Student("Alojzy", "Mechanik",
Czlowiek.Plec.MEZCZYZNA));
        uczestnicy.add(new Student("Ada", "Lovelace",
Czlowiek.Plec.KOBIETA));
        uczestnicy.add(new Student("Jan", "Elektronik",
Czlowiek.Plec.MEZCZYZNA));
        uczestnicy.add(new StudentKSG("Piotr", "Teledetekcyjny",
Czlowiek.Plec.MEZCZYZNA));

        /*
         * ...
         */

        for(Uczestnik u : grupa.getUczestnicy()) {
            System.out.println(u);
        }
    }
}

```

Warto zwrócić uwagę na linię



```
Set<Uczestnik> uczestnicy = new HashSet<Uczestnik>();
```

Interfejs `java.util.Set` oraz klasa `java.util.HashSet` są typami generycznymi. Oznacza to, że jako parametr przyjmują one typ. Dzięki temu można powiedzieć, że obiekt na który wskazuje referencja `uczestnicy` zawiera elementy implementujące interfejs `Uczestnik`. Bez funkcjonalności typów generycznych konieczne byłoby rzutowanie obiektów na klasę `java.lang.Object`, a podczas odczytywania obiektów z kolekcji trzeba byłoby rzutować je z powrotem właściwą klasę lub interfejs (np. `Uczestnik`).

W Java 7 wprowadzono tzw. „Diamond operator”, który pozwala na skracanie zapisu tworzenia niektórych obiektów generycznych.

```
Set<Uczestnik> uczestnicy = new HashSet<>();
```

W pakiecie `java.util` istnieje cała hierarchia interfejsów dotyczących kolekcji¹. Dobrym przykładem elementu tej hierarchii jest interfejs `java.lang.Iterable`, który pozwala iterować po kolekcjach w następujący sposób:

```
for (Uczestnik u : grupa.getUczestnicy()) {
    System.out.println(u);
}
```

Klasy wewnętrzne, anonimowe, lambda wyrażenia, interfejsy funkcjonalne

Założmy istnienie następującego interfejsu (Listing 3).

Listing 3 Interfejs SluchaczZdarzen

```
package pl.edu.pg.eti.ksg.po.lab2.sup;
/*
 * Adnotacja @FunctionalInterface została wprowadzona w Javie 8.
 * Służy ona do oznaczania interfejsów z jedną metodą. Na takie
 * interfejsy mogą zostać przekonwertowane lambda wyrażenia oraz metody
 * mające taką samą listę argumentów oraz typ zwracany.
 */
@FunctionalInterface
public interface SluchaczZdarzen {
    /*
     * Metoda interfejsu funkcjonalnego może mieć dowolną listę
     * parametrów oraz typ zwracany.
     */
    public void wystapiloZdarzenie(int priorytet, String nazwa);
}
```

Interfejs ten zastosowano aby zaprezentować w jaki sposób zwykła klasa może subskrybować zdarzenia (Listing 4 oraz Listing 5).

Listing 4 ZwyklaKlasa która obsługuje zdarzenia poprzez interfejs SluchaczZdarzen na 6 różnych sposobów

```
package pl.edu.pg.eti.ksg.po.lab2.sup;
```

¹ <https://docs.oracle.com/javase/8/docs/api/java/util/package-tree.html>, sekcja „Interface Hierarchy”



```

public class ZwyklaKlasa {

    private static int liczbaInstancji = 0;

    private int numerInstancji;

    /**
     * Klasa wewnętrzna ma dostęp do wszystkich pól i metod obiektu oraz
     * klasy. Nie musi implementować interfejsu. Może mieć dowolny zasięg
     * widoczności.
     */
    private class KlasaWewnętrzna implements SluchaczZdarzen
    {
        private int poleKlasyWewnentrznej;

        public KlasaWewnętrzna(int poleKlasyWewnentrznej) {
            this.poleKlasyWewnentrznej = poleKlasyWewnentrznej;
        }

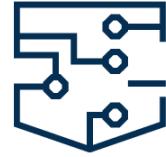
        @Override
        public void wystapiloZdarzenie(int priorytet, String nazwa) {
            System.out.println("KlasaWewnętrzna obsługuje zdarzenie "
                    +nazwa+ " o priorytecie "+priorytet);
            System.out.println("Wartosci pól:");
            System.out.println("liczbaInstancji: "+ liczbaInstancji);
            System.out.println("numerInstancji: "+ numerInstancji);
            System.out.println("poleKlasyWewnentrznej: "+
                    poleKlasyWewnentrznej);
        }
    }

    /**
     * Statyczna klasa wewnętrzna ma tylko dostęp
     * do wszystkich statycznych pól i metod klasy.
     * Nie musi implementować interfejsu.
     * Może mieć dowolny zasięg widoczności.
     */
    private static class StatycznaKlasaWewnętrzna
            implements SluchaczZdarzen
    {
        private int poleStatycznejKlasyWewnentrznej;

        public StatycznaKlasaWewnętrzna(int p) {
            this.poleStatycznejKlasyWewnentrznej = p;
        }

        @Override
        public void wystapiloZdarzenie(int priorytet, String nazwa) {
            System.out.println("StatycznaKlasaWewnętrzna"+
                    " obsługuje zdarzenie "+nazwa+
                    " o priorytecie "+priorytet);
            System.out.println("Wartosci pól:");
            System.out.println("liczbaInstancji: "+ liczbaInstancji);
        }
    }
}

```



```

        //System.out.println("numerInstancji: "+ numerInstancji);
        System.out.println("poleStatycznejKlasyWewnentrznej: "+
                           poleStatycznejKlasyWewnentrznej);
    }

}

public ZwyklaKlasa() {
    liczbaInstancji++;
    numerInstancji = liczbaInstancji;
}

public SluchaczZdarzen klasaWewnetrzna()
{
    return new KlasaWewnetrzna(2*numerInstancji);
}

public SluchaczZdarzen statycznaKlasaWewnetrzna()
{
    return new StatycznaKlasaWewnetrzna(3*liczbaInstancji);
}

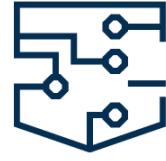
public SluchaczZdarzen klasaAnonimowa()
{
    /**
     * Obiekt klasy anonimowej tworzony w metodzie obiektu
     * nadziennego ma dostęp do wszystkich pól i metod analogcznie
     * do klasy wewnętrznej. Klasy anonimowe mogą być utworzone na
     * bazie każdej klasy po której można dziedziczyć, bądź
     * interfejsu. Najczęściej jednak klasy te są
     * tworzone na bazie klas abstrakcyjnych i interfejsów.
     */
    return new SluchaczZdarzen();

    int poleKlasyAnonimowej = 5*numerInstancji;

    @Override
    public void wystapiloZdarzenie(int priorytet, String nazwa)
    {
        System.out.println("KlasaAnonimowa obsługuje zdarzenie " +
                           +nazwa+ " o priorytecie "+priorytet);
        System.out.println("Wartosci pól:");
        System.out.println("liczbaInstancji: "+ liczbaInstancji);
        System.out.println("numerInstancji: "+ numerInstancji);
        System.out.println("poleKlasyAnonimowej: "+
                           poleKlasyAnonimowej);
    }
}
};

public void zwyklaMetoda(int priorytet, String nazwa)
{
    System.out.println("zwyklaMetoda obsługuje zdarzenie "+nazwa +
                       +" o priorytecie "+priorytet);
    System.out.println("Wartosci pól:");
    System.out.println("liczbaInstancji: "+ liczbaInstancji);
    System.out.println("numerInstancji: "+ numerInstancji);
}

```



```

public static void zwyklaStatycznaMetoda(int priorytet,
                                         String nazwa)
{
    System.out.println("zwyklaStatycznaMetoda obsługuje zdarzenie "
        +"nazwa" o priorytecie "+priorytet);
    System.out.println("Wartosci pól:");
    System.out.println("liczbaInstancji: "+ liczbaInstancji);
}
}

```

Listing 5 Przykład subskrypcji zdarzenia oraz jego wywołania

```

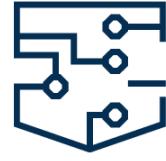
package pl.edu.pg.eti.ksg.po.lab2.sup;
import java.util.LinkedList;
import java.util.List;
/**
 * @author TB
 */
public class Javalab2_sup {
    public static void main(String[] args) {
        int a = 5;
        double b = 3.14;
        ZwyklaKlasa instancjaA = new ZwyklaKlasa();
        ZwyklaKlasa instancjaB = new ZwyklaKlasa();

        List<SluchaczZdarzen> listaSluchaczy = new LinkedList<>();
        listaSluchaczy.add(instancjaA.klasaWewnętrzna());
        listaSluchaczy.add(instancjaA.statycznaKlasaWewnętrzna());
        listaSluchaczy.add(instancjaA.klasaAnonimowa());
        /**
         * Wyrażenia lambda wprowadzono w Javie 8. Można przyjąć,
         * że jest to skrócona wersja klasy anonimowej
         * dla interfejsu oznaczonego anotacją
         * @FunctionalInterface
         */
        listaSluchaczy.add((int p, String n) -> {
            System.out
                .println("Wyrażenie Lambda obsługuje zdarzenie "+n
                    +" o priorytecie "+p);
            System.out.println("Wartosci pól:");
            System.out.println("wartość a z funkcji main: "+ a);
            System.out.println("wartość b z funkcji main: "+ b);
        });

        /**
         * Wprowadzone w Javie 8. Pozwala przekonwertować metodę
         * obiektu na klasę implementującą interfejs funkcyjonalny.
         */
        listaSluchaczy.add(instancjaB::zwyklaMetoda);

        /**
         * Wprowadzone w Javie 8. Pozwala przekonwertować metodę
         * statyczną na klasę implementującą interfejs funkcyjonalny
         */
        listaSluchaczy.add(ZwyklaKlasa::zwyklaStatycznaMetoda);
    }
}

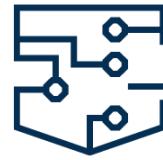
```



```
String nazwa = "Test Interfejsów";
int priorytet = 8;
for(SluchaczZdarzen sl : listaSluchaczy)
{
    sl.wystapiloZdarzenie(priorytet, nazwa);
}

}
```

Zdarzenia są jedną z zaawansowanych technik programowania obiektowego. Przykład rejestracji zdarzeń, który uwidacznia Listing 5, jest nieco uproszczony. Zazwyczaj klasa która generuje zdarzenia posiada prywatną kolekcję obiektów obsługujących zdarzenie, zaś dostęp do niej jest poprzez metodę publiczną. W języku Java zdarzenia nie są wspierane specjalną konstrukcją językową – trzeba do tego celu wykorzystać interfejsy. W języku C# do obsługi służy specjalne pole klasy oznaczone słowem kluczowym **event**, które ukrywa całą implementację obsługi zdarzeń.



Zadania na laboratorium

Zadania laboratoryjne będą polegać na modyfikacji istniejącej aplikacji. Aplikacją jest prosty symulator wycieczek górskich (i nie tylko). Aplikacja w uproszczony sposób symuluje zachowanie grupy w różnych rodzajach terenu oraz przy spotkaniu z różnymi atrakcjami turystycznymi. Grupa składa się z przewodnika oraz uczestników. Ma ona do pokonania pewną trasę zamkniętą w obiekcie klasy Wycieczka. Trasa składa się z kilku lub kilkunastu etapów. Każdy z nich może być wędrówką lub atrakcją turystyczną. Mają one następujące właściwości:

- Wędrówka ma określoną długość w GOTach² (pole klasy Wedrowka) oraz jak wpływa na prędkość grupy (metoda klasy Wedrowka). Wędrówka powoduje zwiększenie zmęczenia uczestników.
- Atrakcja turystyczna ma określony czas zwiedzania. Ponieważ grupa pozostaje w miejscu powoduje ona regenerację zmęczenia uczestników.

Każdy uczestnik ma określony poziom zmęczenia w skali od 0 do 1. Bazowa prędkość poruszania się uczestnika jest obliczana w następujący sposób:

$$(1 - \text{poziomZmeczenia}) * \text{maksymalnaPredkosc}$$

Gdzie maksymalna prędkość jest zależna od klasy uczestnika. Zmęczenie domyślnie jest modyfikowane w następujący sposób:

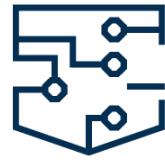
- Dla czasu t spędzonego na wędrówce jest ono zwiększane o $t/18h$,
- Dla czasu t spędzonego na/w atrakcji turystycznej jest ono zmniejszane o $t/18h$

Prędkością grupy jest prędkość poruszania się najwolniejszego uczestnika. Każda wędrówka ma określoną trudność nawigacyjną w skali 1-5. Przewodnik ma określone umiejętności nawigacyjny w skali od 1-5. Jeżeli umiejętności przewodnika są niższe niż wymaga tego teren grupa otrzymuje karę do prędkości.

Prędkość grupy jest modyfikowana w następujący sposób:

1. Przez metodę **modyfikujPredkoscGrupy** klasy **Wedrowka**,
2. Za każdy punkt różnicy między umiejętnościami nawigacyjnymi przewodnika a trudnością terenu (na niekorzyść przewodnika) prędkość grupy jest zmniejszana o 20%.

² Punkty GOT, to jednostka pierwotnie przewidziana do rozliczania osiągnięć turystycznych na poszczególne Górskie Odznaki Turystyczne PTTK (https://pl.wikipedia.org/wiki/G%C3%B3rska_Odznaka_Turystyczna_PTTK). 1 GOT jest zdefiniowany jako 1 km trasy po płaskim terenie (lub będąc precyzyjnym jeden kilometr w rzucie na płaszczyznę) lub 100 m podejścia pod górę. Przykładowo: trasa licząca 1 km, która zawiera 100 m podejścia pod górę jest warta 2 GOTy. Jednostka ta jest przydatna do określania w przybliżeniu wysiłku na trasach pieszych w górach i znajduje zastosowanie poza Odznaką Turystyczną. **W skrócie: 1 GOT można traktować jako równoważnik 1 km w górach.**



Symulator oblicza ile czasu zajęło pokonanie danego odcinka. Czas ten wpływa na zmęczenie uczestników grupy. Jeżeli, któryś uczestnik jest na tyle zmęczony, że nie może iść dalej, to grupa zostaje na noc w miejscu w którym się znajduje (można to traktować jako niepowodzenie 😞).

Symulator bierze pod uwagę także atrakcje turystyczne, które wpływają na regenerację zmęczenia.

Większość powyższych mechanizmów jest już zaimplementowana. Część z nich można zmodyfikować poprzez dziedziczenie i nadpisywanie metod. Brakujące mechanizmy będą do zaimplementowania w poniższych zadaniach.

Uwaga! Zadania przedstawione poniżej mogą być modyfikowane przez prowadzącego laboratorium.

Zadanie 1 (2 punkty)

Dodaj klasę abstrakcyjną **Atrakcja**, która będzie opisywała atrakcje turystyczne. Klasa **Atrakcja** powinna implementować interfejs **ElementWycieczki**. Klasa **Atrakcja** powinna zwierać pole opisujące jak długo (w godzinach) uczestnicy będą ją zwiedzać.

Ponadto:

- W interfejsie **Uczestnik** dodaj metodę **reagujNaAtrakcje**, która będzie analogiczna do metody **reagujNaWedrowke**. Zaimplementuj te metodę we wszystkich klasach nie abstrakcyjnych implementujących interfejs **Uczestnik**. Pamiętaj o uwzględnieniu regeneracji zmęczenia.
- W metodzie symuluj klasy **SymulatorWycieczki** dodaj obsługę **Atrakcji** analogicznie do obsługi **Wedrowki**. Obsługa obiektów klasy **Atrakcja** powinna być jednak krótka i prosta – należy zalogować na wyjściu wystąpienie atrakcji turystycznej, a następnie wywołanie metody **reagujNaAtrakcje** dla każdego uczestnika,
- Dodaj klasę **DrewniaCerkiew** do pakietu **pl.edu.pg.eti.ksg.po.lab2.symulatorwycieczki.gory.beskidy**, ustaw jej czas zwiedzania na 0,5 h. Od komentuj jej zastosowanie w domyśle symulowanej wycieczce.

Przetestuj powyższe zmiany i zweryfikuj, czy uczestnicy regenerują siły zwiedzając cerkiew.

Zadanie 2 (1 punkt)

Zaimplementuj własną wycieczkę turystyczną/górską (np. swoją ulubioną), która będzie miała przynajmniej 8 elementów, z czego:

- Będą to przynajmniej trzy różne podklasy klasy **Wedrowka**,
- Oraz przynajmniej dwie różne podklasy klasy **Atrakcja** (wliczając w to ewentualnie **DrewnianaCerkiew**).

W razie braku pomysłów posłuż się informacjami w dodatku A na str. 12. Ułóż dowolną trasę o długości od 15 do 30 GOTów.



Zadanie 3 (1 punkt)

Zaimplementuj przynajmniej 4 klasy (poza istniejącymi) implementujące interfejs **Uczestnik** (moga także dziedziczyć po klasie **Człowiek**) na podstawie opisów w dodatku B na str. 17. Można zastosować własne pomysły. Przy implementacji szczegółów reakcji na atrakcje lub wędrówki pamiętaj umieszczeniu stosowanego komentarza uczestnika. Nie zapomnij o uwzględnieniu zmęczenia/regeneracji. O ile to możliwe nie kopuj kodu z klasy bazowej, tylko w razie potrzeby odnoś się do jej implementacji za pomocą słowa kluczowego **super**.

Podpowiedzi:

- Aby zmienić ogólnie tempo regeneracji lub przyrostu zmęczenia oraz maksymalnej prędkości poruszania się wywołaj konstruktor klasy bazowej z odpowiednimi parametrami.
- Jeżeli uczestnik w danym terenie lub w danej atrakcji ma inny przyrost zmęczenia lub inne tempo regeneracji, to dla danego przypadku pomnóż czas przez odpowiednio dobraną stałą i wywołaj z tym parametrem odpowiednio metody aktualizujZmeczenie lub regeneruj.

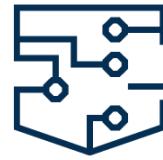
Zadanie 4 (1 punkt)

Dodaj do kodu interfejs funkcyjonalny **SluchaczPostepow**. W klasie **SymulatorWycieczki** dodaj pole zawierające zbiór (Set) obiektów implementujących klasę **SluchaczPostepow**. Dodaj metodę pozwalającą na dodanie słuchacza z zewnątrz. Na końcu iteracji głównej pętli metody **symuluj** dodaj wywołanie metod **aktualizujPostep** we wszystkich elementach dodanego wcześniej zbioru.

Zaprezentuj w metodzie **main** subskrypcję tego zdarzenia. Metoda subskrybująca może wyświetlać np. pasek postępu. Wykonaj to dowolną metodą przedstawioną w sekcji *Klasy wewnętrzne, anonimowe, lambda wyrażenia, interfejsy funkcyjonalne*.

Listing 6 Interfejs funkcyjonalny SluchaczPostepow

```
@FunctionalInterface
public interface SluchaczPostepow {
    void aktualizujPostep(ElementWycieczki elementWycieczki, int lp, int
liczbaElementow);
}
```



Dodatek A. Przykładowe typy wędrówki i atrakcji turystycznych

Uwaga: Pola oznaczone „????” są pozostawione inwencji twórczej czytelnika :).

Góry

Typy wędrówki i atrakcji turystycznych wspólne dla wszystkich gór.

Tabela 1 Przykładowe typy wędrówki dla gór

Proponowana Nazwa Klasa	Opis	Modyfikacja prędkości grupy	Trudność nawigacyjna
Droga	Zwykła droga gruntowa w lesie lub poza lasem	Prędkość bez zmian	1
Las	Przejście na przełaj przez łatwy do przejścia las	Prędkość jest mnożona razy 0,5	3
GestyLas	Przejście na przełaj przez gęsty las w którym nawigacja jest trudniejsza niż w zwykłym lesie.	???	5
PrzeprawaPrzezRzeke	Przejście przez większą rzekę w poprzek lub wędrówka z biegiem rzeki.	???	???

Tabela 2 Przykładowe typy atrakcji dla gór

Proponowana Nazwa Klasa	Opis	Czas spędzony na atrakcji	Dodatkowe informacje
Panorama	Panorama górska, którą można podziwiać przy dobrej widoczności. Niektórzy przewodnicy opowiadają grupie jakie szczyty są widoczne.	10 min (1/6 h)	brak
Schronisko	Miejsce odpoczynku, dla uproszczenia traktowane tak samo jak atrakcja turystyczna.	???	brak



Beskidy

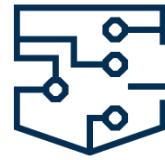
Góry lesiste rozciągające się od Śląska po Podkarpacie.

Tabela 3 Przykładowe typy wędrówki dla Beskidów

Proponowana Nazwa Klasy	Opis	Modyfikacja prędkości grupy	Trudność nawigacyjna
Blotostrada	Błotostrada – leśna droga zryta przez ciężki sprzęt do zrywki drewna. Błotostrada wskazuje jedynie kierunek marszu. Ze względu na duże ilości błota trzeba iść skrajem drogi.	Prędkość jest mnożona razy 0,75	2
PoleBarszczuSosnowskiego	Pole na którym rośnie dziko w dużych ilościach niebezpieczna roślina – Barszcz Sosnowskiego. Grupa musi kluczyć pomiędzy roślinami tak aby uniknąć poparzeń.	Prędkość jest mnożona razy 0,75	3
LakaPelneMuch	Łąka pełna gryzących much (Jusznic). Grupa porusza się szybko aby uniknąć pogryzień.	???	???
Tory	Tory kolejki wąskotorowej. Są mniej wygodne niż droga, ale pozwalają na łatwą nawigację.	???	???

Tabela 4 Przykładowe typy atrakcji dla Beskidów

Proponowana Nazwa Klasy	Opis	Czas spędzony na atrakcji	Dodatkowe informacje
DrewnianaCerkiew	Mała drewniana świątynia prawosławna lub grekokatolicka, wzniesiona przez lokalną ludność	0,5 h	Jako dodatkowe pole klasy warto zatrzymać nazwę miejscowości w której znajduje się cerkiew.



	Łemków lub Bojków. Architektoniczna atrakcja turystyczna.		
Cmentarz ZI Wojny	Cmentarz z I Wojny Światowej. Miejsce pamięci oraz atrakcja architektoniczna.	20 min (1/3 h)	Jako dodatkowe pole klasy warto zatrzymać nazwę miejscowości w której znajduje się cmentarz.
Chatka Studencka	Miejsce noclegu i odpoczynku o obniżonym standardzie ale o podwyższonym klimacie ☺.	???	Jako dodatkowe pole klasy warto umieścić nazwę klubu lub koła studenckiego, który/które opiekuje się chatką.
Elektrownia Wodna	Elektrownia szczytowo-pompowa lub elektrownia na zaporze. Atrakcja technologiczna.	???	Jako dodatkowe pole klasy warto umieścić nazwę rzeki na której elektrownia się znajduje.

Sudety

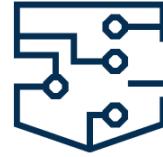
Góry w południowo zachodniej Polsce.

Tabela 5 Przykładowe typy wędrówki dla Sudetów

Proponowana Nazwa Klasa	Opis	Modyfikacja prędkości grupy	Trudność nawigacyjna
WietrznaGran	Szlak prowadzący grzbietem górkim. Wiejący wiatr przyspiesza lub spowalnia wędrówkę.	Prędkość zwiększana lub zmniejsza o 1/3	1
BagnoGorskie	Teren podmokły znajdujący w piętrze kosodrzewiny.	???	???

Tabela 6 Przykładowe typy atrakcji dla Sudetów

Proponowana Nazwa Klasa	Opis	Czas spędzony na atrakcji	Dodatkowe informacje
Przepasc	Atrakcja analogiczna do panoramy górskiej,	10 min (1/6 h)	brak



	ale nie należy wychodzić poza barierki.		
WychodnieSkalne	Wychodnie skalne – formacje skalne, interesujący obiekt geologiczny.	???	brak
ObserwatoriumMeteorlogiczne	Rozwinęta wersja schroniska turystycznego z ładnym widokiem.	???	brak

Biebrza

Teren wybitnie nie górska. Są to mokradła i rozlewiska wokół rzeki Biebrzy na Podlasiu. Dla wędrówek po zalanych terenach wokół Biebrzy jako dodatkowy parametr można dodać poziom wody (od 0 do 75 cm) – poziom wody może wpływać na prędkość grupy. Poniżej jednak założono stały wpływ typu wędrówki na prędkość grupy.

Tabela 7 Przykładowe typy wędrówki dla Biebrzy

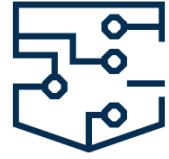
Proponowana Nazwa Klasy	Opis	Modyfikacja prędkości grupy	Trudność nawigacyjna
BlotnistadDroga	Zwykła droga, lekko podmokła,	Prędkość mnożona razy 0,9.	1
ZalanaLakaTrawiasta	Zwykła łąka trawiasta zalana wodą.	Prędkość mnożona razy 1/3	2
ZalanaLakaTurzyc	Łąka pełna turzyc. Poruszanie się jest utrudnione, gdyż trzeba uważać aby nie potknąć się o wystające kępy traw.	???	???
ZalaneTrzciny	Podobnie jak zalana trawiasta łąka, ale ze względu na wysokie trzciny trudniej jest się nawigować.	???	???
ZalanyLas	Zalany las w którym ciężko jest się poruszać i nawigować.	???	4

Tabela 8 Przykładowe typy atrakcji dla Biebrzy

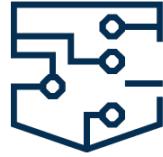
Proponowana Nazwa Klasy	Opis	Czas spędzony na atrakcji	Dodatkowe informacje



GDAŃSK UNIVERSITY
OF TECHNOLOGY



Wieża Widokowa	Wieża widokowa pozwalająca obserwować gniazdujące na bagnach ptaki.	20 min (1/3 h)	brak
----------------	---	----------------	------



Dodatek B. Opisy klas implementujących interfejs Uczestnik

Uwaga: Pola oznaczone „???” są pozostawione inwencji twórczej czytelnika :).

Tabela 9 Tabela opisująca różne klasy uczestników. Regeneracja (Reg.) opisuje jak regeneruje się uczestnik po zadanym czasie t, Przyrost Zmęczenia (Przyr. Zm.) opisuje jak zwiększa się zmęczenie po czasie t, ponadto są jeszcze kolumny Maksymalna Prędkość (Maks. Pr.) oraz Umiejętności Nawigacyjne (Um .Naw.) oraz Specjalne właściwości.

Nazwa Klasy	Opis	Reg.	Przyr. Zm.	Maks. Pr.	Um. Naw.	Specjalne właściwości
Czlowiek	Klasa abstrakcyjna, baza dla innych uczestników.	t/18 h	t/18 h	N/D	N/D	brak
Student	Dziedziczy po klasie Czlowiek.	???	???	4 GOT/h	3	brak
StudentKSG	Dziedziczy po klasie Student. Student lepiej zaznajomiony z terenem. Przystosowany do badań hydrologicznych	???	???	4 GOT/h	???	Podczas pobytu na terenach podmokłych i rzecznych zbiera dane na temat przewodności wody (EC) i jej pH. Potrzebne mu są do pracy Magisterskiej.
PrzewodnikStudencki	Dziedziczy po klasie Student. Skończył kurs na przewodnika beskidzkiego. Umie lepiej nawigować, szybciej się porusza	???	???	5 GOT/h	4	Opowiada o cerkwiach, cmentarzach i panoramach.
PrzewodnikBeskidzki	Dziedziczy po klasie PrzewodnikStudencki. Zdał egzaminy państwowego na przewodnika beskidzkiego. Ma większe doświadczenie.	t/16 h	t/18 h	5 GOT/h	???	Tak jak PrzewodnikStudencki
LesnyBiegacz	Szybko się porusza i dobrze się nawiguje. Interesuje się cmentarzami z I Wojny Światowej	???	???	6 GOT/h	???	Ze względu na swoje zainteresowania podczas pobytu na atrakcji turystycznej „Cmentarz z I wojny Światowej” regeneruje



						się dwa razy szybciej niż zwykłe.
BagiennyBiegacz	Szybko się porusza i dobrze się nawiguje. Ma duże doświadczenie w poruszaniu się po bagnach i w terenie podmokłym.	???	???	???	???	Na terenach podmokłych mączy się dwa razy wolniej.
BeskidzkiPiechur	Bardzo powoli się mączy, ale też bardzo powoli się regeneruje. Interesuje się cerkwiami.	t/20 h	t/20 h	???	???	W drewnianych cerkwiach regeneruje się dwa razy szybciej.
MistrzPanoram	Zna wszystkie panoramy górskie.	???	???	???	???	Na panoramach regeneruje się dwa razy szybciej.
CzlowiekZKontuzja	Wolniej poruszający się uczestnik wycieczki.	???	???	???	???	brak
Hydrolog	Wykonuje badania terenów podmokłych. Ma zezwolenie na poruszanie się poza szlakiem w Biebrzańskim Parku Narodowym	???	???	???	???	Podczas pobytu na terenach podmokłych i rzecznych zbiera dane na temat przewodności wody (EC) i jej pH do celów naukowych.
Informatyk	Pomaga Hydrologowi. Ma zezwolenie na poruszanie się poza szlakiem w Biebrzańskim Parku Narodowym	???	???	???	???	Zapisuje na kartce pomiary od Hydrologa w celu późniejszego wprowadzenia ich do systemu GIS.