Fakulta informatiky a informačných technológií STU, Bratislava Ilkovičova 2, 842 16 Bratislava 4

Objektovo orientované programovanie

Správa o realizácii projektu

Golf Tour Planner

Frederik Duvač
2. semester
13.5.2023

1. Zámer projektu

Program bude slúžiť ako plánovač ciest pre golfistov. Cieľom bude umožniť používateľom naplánovať najlepší spôsob prepravy medzi hráčom určenými miestami a vygenerovať najefektívnejšiu trasu na základe jeho alebo tímového rozpočtu.

Hráči si budú môcť vytvoriť cestovný plán pre rôzne miesta, ktoré chcú navštíviť a program im odporučí najlepšiu trasu, aby mohli cestovať čo najefektívnejšie. Okrem najkratšej trasy by program mohol odporučiť aj najrýchlejšiu trasu s pomocou dát o cestách.

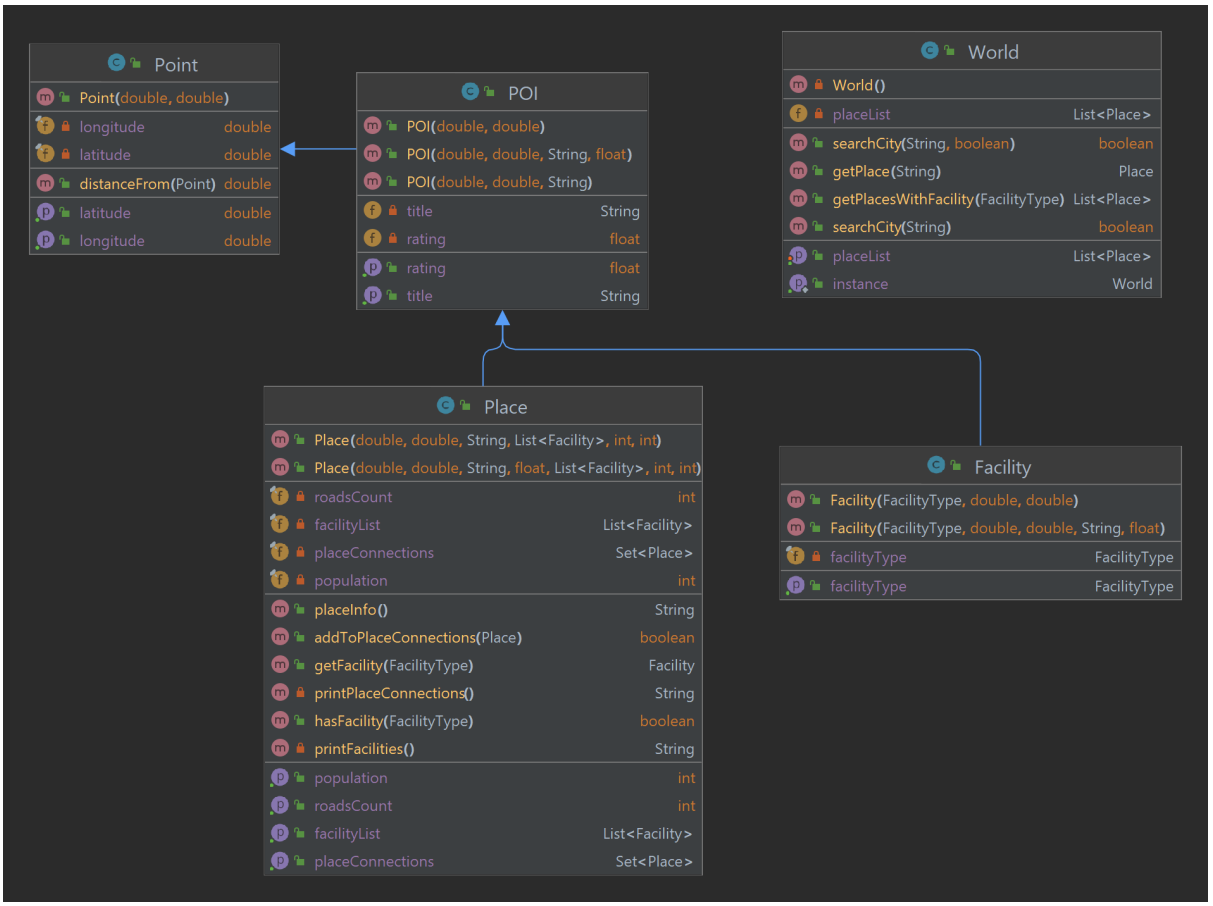
Na základe rozpočtu program rozhodne napríklad o tom akým dopravným prostriedkom sa bude hráč presúvať medzi jednotlivými miestami po svete. Ďalej program naplánuje na základe miesta a času konania turnaja či je potrebné zabezpečiť ubytovanie v hoteli niekde v okolí alebo môže odporučiť rôzne reštaurácie, atrakcie alebo body záujmu v blízkosti trasy na základe preferencií používateľa.

Navrhovaný program pre naplánovanie vlastnej golfovej túry by mohol byť veľmi užitočný pre golfistov alebo aj iných cestovateľov a mohol by poskytnúť veľa zaujímavých funkcionalít pre optimalizáciu ich cestovných plánov a zároveň ponúka mnoho ďalších možností ako rozšíriť tento projekt, medzi ktorými môžeme spomenúť napríklad integrácia reálnych online máp, počasie, atď.

2. UML

2.1. World

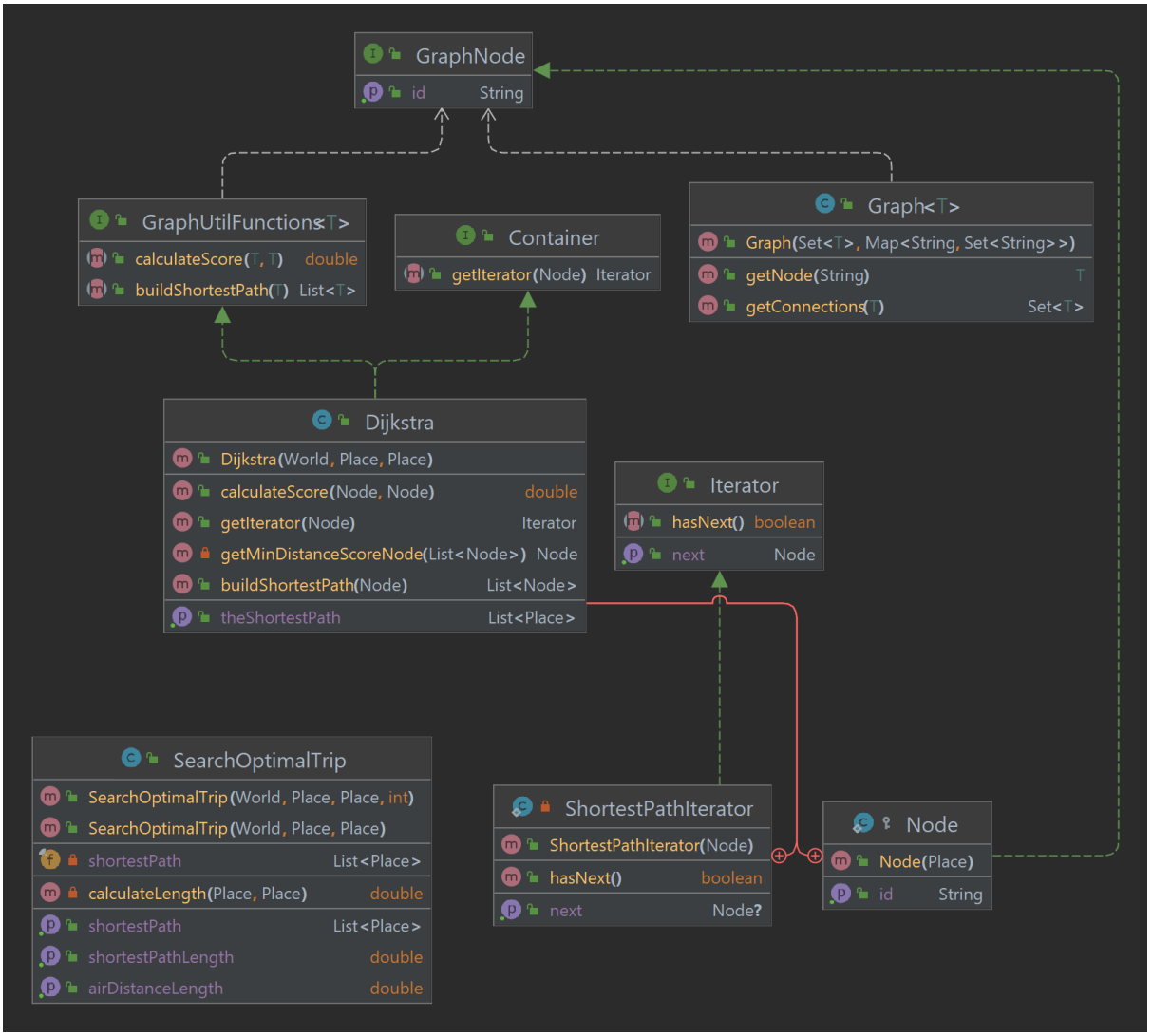
Trieda `World` reprezentuje svet alebo mapu miest. Základný stavebný prvok je trieda `Point`, ktorá obsahuje základné informácie ako sú latitude a longitude. Od tejto triedy dedí trieda `POI`, ktorá obsahuje najviac informáciu o názve a popularite daného miesta záujmu. Triedy `Place` a `Facility` sú detmi triedy `POI` z dôvodu, že samotné miesto a alebo zariadenie, ktoré môžu mestá obsahovať sú samé o sebe bodmi záujmu.



2.2. Pathfinding

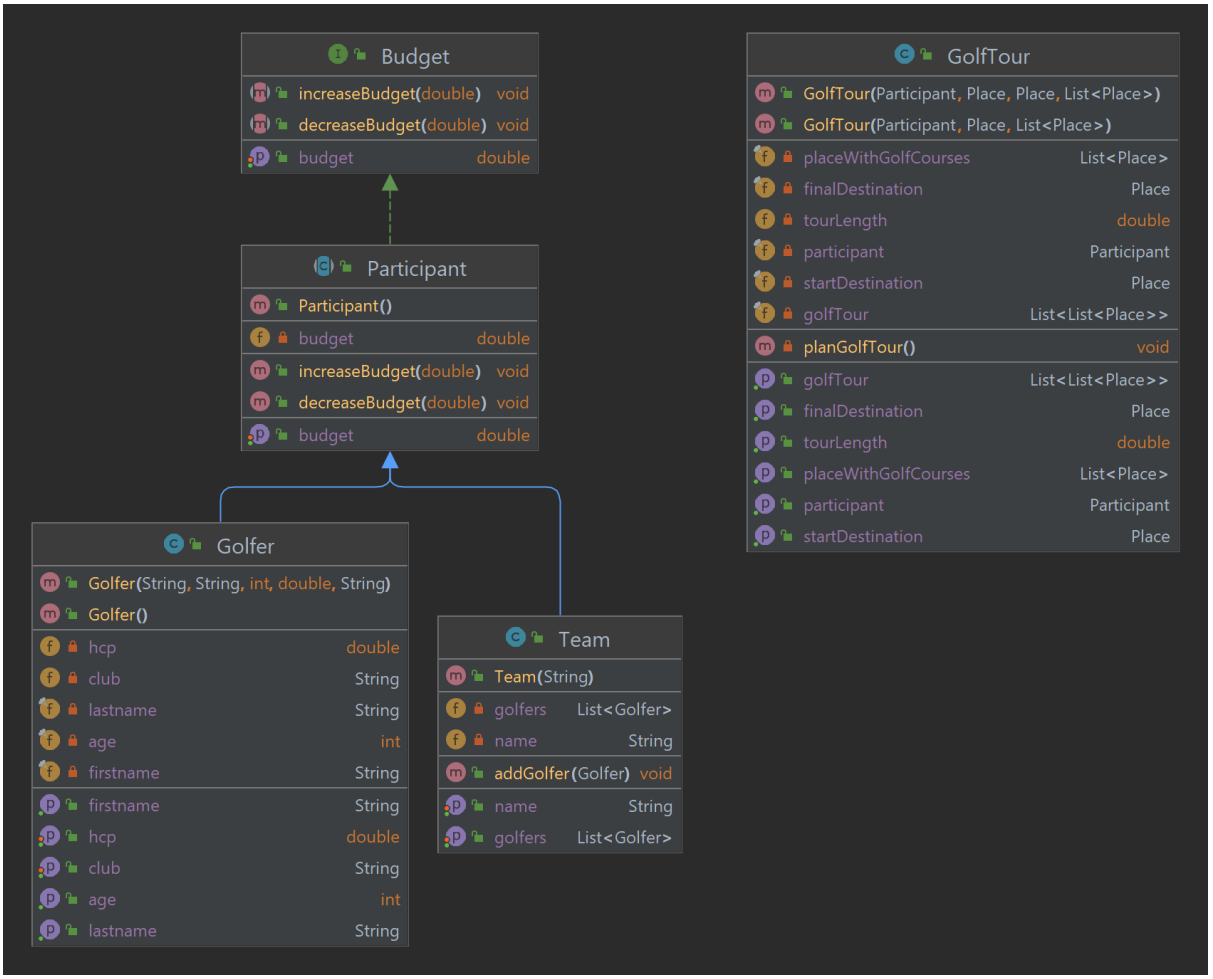
Hlavná trieda `SearchOptimalTrip` je zodpovedná za nastavenie Dijkstrovho algoritmu a spracováva a poskytuje informácie poskytnuté triedou. Základná stavebná jednotka na reprezentáciu miest na mape je generická trieda `Graph`, ktorá vie o všetkých uzloch grafu, reprezentovaných vnorenou triedou `Node` v triede `Dijkstra` a prepojeniach o každom z nich.

Trieda `Dijkstra` implementuje funkcie potrebné pre algoritmus a rozhranie `Container` potrebné pre iterator pattern. Iterator pattern je použitý na zostavenie najkratšej trasy zistenej Dijkstrovým algoritmom. Iterator pattern reprezentuje vnorená trieda `ShortesthPathIterator` implementujúca základné funkcie z rozhrania `Iterator`.



2.3. Golf Tour

Trieda `GolfTour` vytvorí golfovú túru pre účastníkov, ktorými je buď solo hráč `Golfer` alebo tím reprezentovaný triedou `Team`. Obe triedy dedia od rodiča `Participant`, ktorý implementuje rozhranie `Budget`.



3. Splnené kritéria

3.1. Hlavné kritéria

Zo zámeru projektu program dokáže vygenerovať náhodné miesta na mape a vytvoriť najkratšie trasy medzi mestami za pomoci Dijkstrovho algoritmu. Hráč si môže vytvoriť vlastnú golfovú túru. Zo zámeru nebolo obsiahnute plánovanie najefektívnejsiej trasy, kedze dáta o cestach neboli implementované, ani dopravné prostriedky. Taktiež neboli pre nedostatok času implementované mechanizmy na zabavenie účastníka popri cestovaní. Každopádne program je v stave kedy by sa dali podobne veci doprogramovať relatívne jednoducho bez výrazných zmien jadra programu.

V programe sú vhodne uplatnené objektovo-orientované mechanizmy ako sú dedenie, polymorfizmus, zapuzdrenie a agregácia.

Organizácia kódu a kvalita java dokumentácie je na vysokej úrovni.

3.2. Ďalšie kritéria

A. Použitie návrhových vzorov okrem návrhového vzoru Singleton – každý implementovaný návrhový vzor sa počíta ako splnenie jedného ďalšieho kritériá, ale implementácia všetkých návrhových vzorov sa posudzuje maximálne na úrovni splnenia troch ďalších kritérií

Implementovaný bol Iterator pattern.

```
public interface Iterator {

   boolean hasNext();

   Dijkstra.Node getNext();

}

public interface Container {

   Iterator getIterator(Dijkstra.Node node);

}

private static class ShortestPathIterator implements Iterator {

   Node actualNode;

   public ShortestPathIterator(Node actualNode) {

       this.actualNode = actualNode;

   }

   @Override

   public boolean hasNext() {

       return actualNode.previousNode != null;

   }

   @Override

   public Node getNext() {

       if (this.hasNext()) {

           actualNode = actualNode.previousNode;

           return actualNode;

       }
```

```
        return null;
    }
}
public List<Node> buildShortestPath(Node finalNode) {
    List<Node> shortestPath = new ArrayList<>();
    shortestPath.add(finalNode);
    Iterator shortestPathIterator = getIterator(finalNode);
    while (shortestPathIterator.hasNext()) {
        shortestPath.add(shortestPathIterator.getNext());
    }
    Collections.reverse(shortestPath);
    return shortestPath;
}
@Override
public Iterator getIterator(Node object) {
    return new ShortestPathIterator(object);
}
```

B. Ošetrenie mimoriadnych stavov prostredníctvom vlastných výnimiek – stačí jedna vlastná výnimka, ale musí byť skutočne vyhadzovaná a ošetrovaná

Implementovaných bolo viacero vlastných výnimiek. Nachádzajú sa v package exceptions. Použitie je väčšinou v `MainApp` a `WorldFXMLController` triedach.

C. Poskytnutie grafického používateľského rozhrania oddelene od aplikačnej logiky a s aspoň časťou spracovateľov udalostí (handlers) vytvorenou manuálne – počíta sa ako splnenie dvoch ďalších kritérií

Implementované, vlastné udalosti sa nachádzajú napr. v `MainApp` triede.

D. Explicitné použitie viacniťovosti (multithreading) – spustenie vlastnej nite priamo alebo prostredníctvom API vyššej úrovne (trieda **Task** a pod.)

Implementované na oddelenie grafických častí programu od aplikačných na separátnych vláknach.

E. Použitie generickosti vo vlastných triedach – implementácia a použitie vlastnej generickej triedy (ako v príklade spájaného zoznamu poskytnutého k prednáške 5)

Implementovaná generická trieda `Graph` reprezentujúca mestá a cesty na mape.

```
public class Graph<T extends GraphNode> {

   private final Set<T> nodes;

   private final Map<String, Set<String>> connections;



   public Graph(Set<T> nodes, Map<String, Set<String>>
connections) {

       this.nodes = nodes;

       this.connections = connections;

   }



   public T getNode(String id) {

       return nodes.stream()

               .filter(node -> node.getId().equals(id))

               .findFirst()

               .orElseThrow(() -> new IllegalArgumentException("No
node found with ID: " + id));

   }
```

```
    public Set<T> getConnections(T node) {

        return connections.get(node.getId()).stream()

                .map(this::getNode)

                .collect(Collectors.toSet());

    }

}
```

F. Explicitné použitie RTTI – napr. na zistenie typu objektu alebo vytvorenie objektu príslušného typu (ako v hre s obrami a rytiermi pri zisťovaní počtu bytostí)

Použité v prípade účastníka golfovej túry (`Golfer` alebo `Team` rodič `Participant`) v triede `GolfTour`. alebo v `MainApp` pri vytváraní účastníka golfovej túry.

```
if (participant instanceof Team) {

    participant.decreaseBudget(fee * ((Team)
participant).getGolfers().size());

} else {

    participant.decreaseBudget(fee);

}
```

G. Použitie vhniezdených tried a rozhraní – počíta sa iba použitie v aplikačnej logike, nie v GUI, pričom rozhrania musia byť vlastné (jedna možnosť je v príklade vnútorných tried k prednáške 4)

Vhniezdené triedy `Node` a `ShortestPathIterator` sú použité v triede Dijkstra. Trieda `GolfCourseListViewItem` je použitá v `MainApp` triede.

H. Použitie lambda výrazov alebo referencií na metódy (method references) – počíta sa iba použitie v aplikačnej logike, nie v GUI (jedna možnosť je v príklade referencií na metódy a lambda výrazov k prednáške 4)

Použitie napr. v tirede `Graph` a `Dijkstra`. Inak vo veľkom používané v GUI :).

I. Použitie implicitnej implementácie metód v rozhraniach (default method implementation)

   Nepoužité

J. Použitie aspektovo-orientovaného programovania (AspectJ)

   Nepoužité

K. Použitie serializácie

   Nepoužité

4. Zoznam verzií programu

- dijkstra - predbežná verzia projektu
- javafx setup - pridanie knižníc javafx, prvotné vytvorenie GUI pre program
- data generator improvement - GUI zobrazenie miest na mape a v liste
- exceptions - GUI vyhľadanie najkratšej trasy a zobrazenie na mape
- search places - možnosť vyhľadať mestá
- bug in generator caused different paths fixed - reprezentácia miest na mape generickou triedou Graph, zobrazenie vyhľadávaného miesta na mape, správne poradia miest v najkratšej trase, fixnutý generator
- iterator pattern for build the shortest path - implementacia iterator patternu do Dijkstra triedy na zostavenie najkratšej trasy
- multithreading for generating places and navigation - implementácia app logiky na vlastné vlákna a GUI na fx vlákno
- java doc - napísanie java dokumentácie pre doteraz naprogramované objekty a metódy
- golf tour logic and ui, bug fix generate only place with unique title, sort list views - implementácia vytvorenia golfovej túry a rozdelenie typu na solo a team, GUI pre vygenerovanie golfovej túry a zobrazenie na mape
- age exception, console print removed - GUI pre zobrazenie informácii o golfovej túre
- project report and java doc - pridanie správy projektu a java dokumentácie