Ministry of Science and Education of Russian Federation

Peter the Great St. Petersburg Polytechnic University

—

Institute of Applied Mathematics and Mechanics

**Department «Information security of computer systems»**

**Laboratory work №1**

**CREATING A UML DIAGRAM**

course «OOP»

Student Sinyapkin B.G.

Gr. 33656/5

<*signature*>

Instructor Chernov A.U.

<*signature*>

Saint-Petersburg

2018

1. **Purpose of work**

Study of types of relationships between objects and classes, introduction to basic elements of the definition, presentation, design and modeling software systems using the UML language, gaining the skill development of UML diagrams for application applications.

1. **Formulation of the problem**

According to the variant of the task, select entities related to the specified subject area. Construct a UML diagrams. Implement a demo interactive program for described visual representation.

1. **Results**

**3.1 Attribute and operations tables**

|  |  |
| --- | --- |
| **Entity name** | Route |
| **Entity Type** | Class |
| **Comment** | Represents a route, on which transport goes. |
| **Attributes** | -t\_amount : *unsigned*  -route\_tt : std::*map<const char\*,const char\*>*  +route\_name : const char\* |
| **Operations** | +get\_amount(void) : unsigned  -time\_to\_i(t: char\*) : int  +operator++(void) : void  +operator—(void) : void  +get\_arrived\_time(stop: const char\*, cur: const char\*) : void |

|  |  |
| --- | --- |
| **Entity name** | Transport |
| **Entity Type** | Class |
| **Comment** | Represents a transport, which attached to the park and goes along the route. |
| **Attributes** | -appended\_park : park\*  -cur\_route : route\*  -number : const char\* |
| **Operations** | +get\_number() : const char\*  +operator==(b : transport&) : bool |

|  |  |
| --- | --- |
| **Entity name** | Park |
| **Entity Type** | Class |
| **Comment** | Represents a transport park, which has one or many transports and traffic controller. |
| **Attributes** | -transport\_map : std::map<const char\*, transport>  +park\_name : const char\*  +tc : traffic\_controller |
| **Operations** | +add\_vehicle(vehicle : transport) : void  +is\_belongs\_vehicle(vehicle : transport) : bool  +park\_size(void) : size\_t |

|  |  |
| --- | --- |
| **Entity name** | Traffic controller |
| **Entity Type** | Class |
| **Comment** | Represents a traffic controller, which raise alerts, when transport is crash or delay on their route. |
| **Attributes** | -alerts : std::map<alert\*> |
| **Operations** | +raise\_alert(t\_num : const char\*)  +show\_alerts(void) : void |

|  |  |
| --- | --- |
| **Entity name** | Alert |
| **Entity Type** | Interface |
| **Comment** | Represents an accident that happened with the transport along the route. |
| **Attributes** | #t\_num : const char\* |
| **Operations** | +show\_alert(void) : virtual void |

|  |  |
| --- | --- |
| **Entity name** | Crash |
| **Entity Type** | Class |
| **Comment** | Represents a crash, which occurred on the route. |
| **Attributes** |  |
| **Operations** | +show\_alert(void) : void |

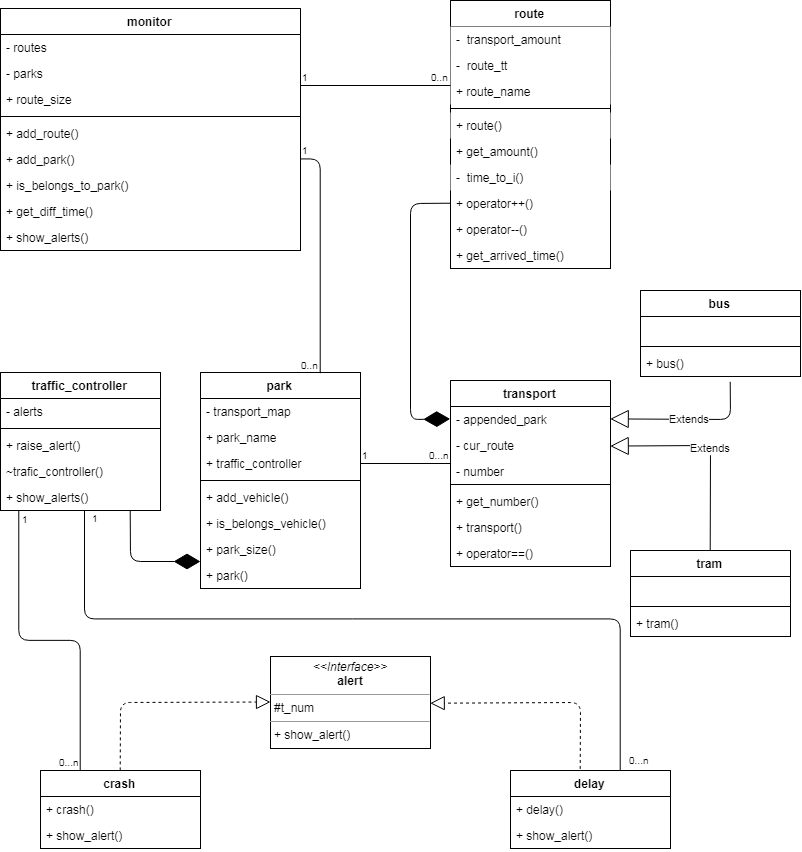
|  |  |
| --- | --- |
| **Entity name** | Delay |
| **Entity Type** | Class |
| **Comment** | Represents a transport delay on the route. |
| **Attributes** |  |
| **Operations** | +show\_alert(void) : void |

|  |  |
| --- | --- |
| **Entity name** | Bus |
| **Entity Type** | Class |
| **Comment** | This is a transport that is a bus. |
| **Attributes** |  |
| **Operations** |  |

|  |  |
| --- | --- |
| **Entity name** | Tram |
| **Entity Type** | Class |
| **Comment** | This is a transport that is a tram. |
| **Attributes** |  |
| **Operations** |  |

|  |  |
| --- | --- |
| **Entity name** | Monitor |
| **Entity Type** | Class |
| **Comment** | Represents a monitoring system for public transport. |
| **Attributes** | -routes : std::map<const char\*, route\*>  -parks : std::map<const char\*, park\*>  +route\_size : |
| **Operations** | +route\_size(route\_name : const char \*) : void  +add\_route(new\_route : route \*) : void  +add\_park(new\_park : park\*) : void  +is\_belongs\_to\_park(t\_num : const char\*, p\_num : const char\*)  +get\_diff\_time(r\_name : const char\*, s\_name : const char\*, cur\_time : const char\*) : void  +show\_alerts(park\_name : const char\*) : void |

* 1. **UML diagram**

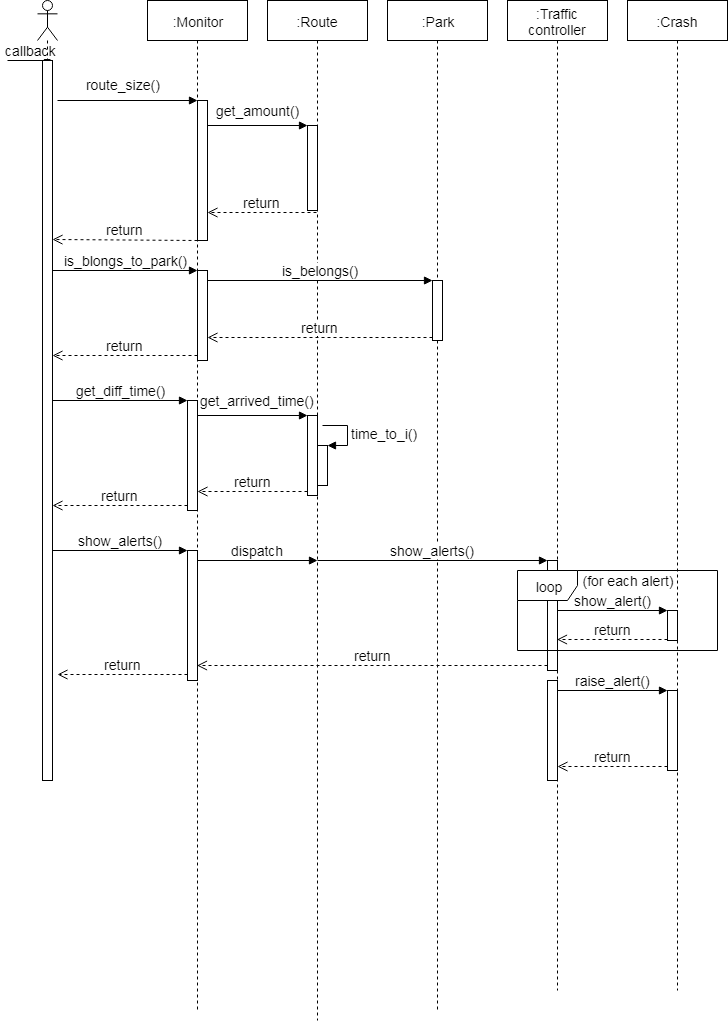
****

*Picture 1.*

* 1. **Table for UML interaction diagram**

|  |  |  |  |
| --- | --- | --- | --- |
| **Message number** | **Object – sender of the message** | **Object – receiver of the message** | **Name** |
| 1 | Monitor | Route | get\_amount  (*Information about the number of vehicles on the route*) |
| 2 | Monitor | Park | is\_belongs  (*Information about the attachment of this transport to this park*) |
| 3 | Monitor | Route | get\_diff\_time  (*Getting the waiting time of the vehicle stop in the route*) |
| 4 | Park | Traffic controller | show\_alerts  (*Getting the information aboute the crashes and delay on the route*) |
| 5 | Park | Transport | get\_number  (*Getting the transport registration mark*) |
| 6 | Traffic controller | Crash, Delay | raise\_alert  (*Recording an emergency*) |
| 7 | Traffic controller | Crash, Delay | show\_alert  (*Getting information about the alert*) |

* 1. **UML interaction diagram**

****

*Picture 2.*

**3.5 Source code**

Source code of this visual representation locate in section “Attachments”.

1. **Conclusion**

During the execution of this work, skills of construction UML diagrams and relationships between objects and classes have been mastered.

1. **Attachment**

typedef std::map<const char\*, const char\*> timetable;

class route

{

public:

route(const char \*init\_route, timetable init\_tt ) :

route\_name(init\_route),

transport\_amount(0),

route\_tt(init\_tt) {}

const char \*route\_name;

unsigned get\_amount() { return transport\_amount; }

void operator++() { transport\_amount++; }

void operator--() { transport\_amount--; }

void get\_arrived\_time(const char \*stop\_name, const char \*cur\_time)

{

try

{

if (route\_tt.find(stop\_name) == route\_tt.end())

throw(std::out\_of\_range(""));

int diff = time\_to\_i(route\_tt[stop\_name]) - time\_to\_i(cur\_time);

if (diff < 0)

throw(std::exception("Transport passed this stop.\n"));

else

{

cout << "Transport will arrived in " << diff / 60

<< " hour(s) "

<< diff % 60

<< " minutes." << endl;

}

}

catch (const std::out\_of\_range&)

{

cout << "This stop is not in the route." << endl;

}

catch (const std::exception& e)

{

cout << e.what() << endl;

}

}

private:

unsigned transport\_amount;

timetable route\_tt;

//HH:MM

int time\_to\_i(const char \*time\_str)

{

string temp\_str(time\_str);

string hour = temp\_str.substr(0, temp\_str.find(":"));

string minute = temp\_str.substr(temp\_str.find(":") + 1, temp\_str.length());

return std::atoi(hour.c\_str()) \* 60 + std::atoi(minute.c\_str());

}

};

class park;

class transport

{

public:

transport(const char \*numb, park \*ap) :

cur\_route(0), number(numb), appended\_park(ap) {}

transport(const char \*numb, route \*t\_route, park\* ap) :

cur\_route(t\_route), number(numb), appended\_park(ap)

{

cur\_route->operator++();

}

~transport() {}

const char\* get\_number() { return number; }

bool operator==(transport& b)

{

return (\*this).get\_number() == b.get\_number() ? true : false;

}

private:

transport() {}

park \*appended\_park;

route \*cur\_route;

const char \*number;

};

class bus : public transport

{

public:

bus(const char \*numb, park \*ap) :

transport(numb, ap) {}

bus(const char \*numb, route \*t\_route, park\* ap) :

transport(numb, t\_route, ap) {}

};

class tram : public transport

{

public:

tram(const char \*numb, park \*ap) :

transport(numb, ap) {}

tram(const char \*numb, route \*t\_route, park\* ap) :

transport(numb, t\_route, ap) {}

};

/\*number, class trnsport\*/

typedef std::pair<const char\*, transport> t\_id;

class alert

{

public:

alert(const char \*at\_num) : t\_num(at\_num) {}

~alert() {}

virtual void show\_alert() = 0;

protected:

const char \*t\_num;

};

struct crash : public alert

{

crash(const char \*at\_num) : alert(at\_num) {}

void show\_alert()

{

cout << "type: " << typeid(\*this).name()

<< ", registration mark: "

<< t\_num << endl;

}

};

struct delay : public alert

{

delay(const char \*at\_num) : alert(at\_num) {}

void show\_alert()

{

cout << "type: " << typeid(\*this).name()

<< ", registration mark: "

<< t\_num << endl;

}

};

class traffic\_controller

{

public:

template<typename alert\_type>

void raise\_alert(const char \*t\_num)

{

alert\_type \*new\_alert = new alert\_type(t\_num);

alerts.push\_back(new\_alert);

}

void show\_alerts()

{

if (alerts.empty()) cout << "Alerts empty." << endl;

else

{

for (auto i = alerts.begin(); i != alerts.end(); i++)

{

(\*i)->show\_alert();

}

}

}

~traffic\_controller()

{

for (auto i = alerts.begin(); i != alerts.end(); i++) delete (\*i);

}

private:

std::vector<alert\*> alerts;

};

class park

{

public:

traffic\_controller tc;

park(const char \*reg\_mark) : park\_name(reg\_mark) {}

~park() {}

const char \*park\_name;

void add\_vehicle(transport vehicle)

{

t\_map.insert(t\_id(vehicle.get\_number(), vehicle));

}

bool is\_belongs(transport vehicle)

{

if (t\_map.find(vehicle.get\_number()) == t\_map.end()) return false;

else return true;

}

bool is\_belongs(const char \*transport\_number)

{

if (t\_map.find(transport\_number) == t\_map.end()) return false;

else return true;

}

size\_t park\_size() { t\_map.size(); }

private:

/\*number, class transport\*/

std::map<const char\*, transport> t\_map;

};

class monitor

{

public:

void add\_route(route \*new\_route) { routes[new\_route->route\_name] = new\_route; }

void add\_park(park \*new\_park) { parks[new\_park->park\_name] = new\_park; }

void route\_size(const char \*route\_name)

{

try

{

if (routes.find(route\_name) == routes.end())

throw(std::out\_of\_range(""));

cout << "Route \"" << route\_name << "\" size: "

<< routes[route\_name]->get\_amount();

}

catch (const std::exception&)

{

cout << "Route \"" << route\_name << "\" does not exist." << endl;

}

}

void is\_belongs\_to\_park(const char \*transport\_number, const char \*park\_name)

{

try

{

if (parks.find(park\_name) == parks.end())

throw(std::out\_of\_range(""));

cout << "Belongs \"" << transport\_number

<< "\" to \"" << park\_name << "\" ? : "

<< parks[park\_name]->is\_belongs(transport\_number);

}

catch (const std::out\_of\_range&)

{

cout << "Park \"" << park\_name << "\" does not exist." << endl;

}

}

void get\_diff\_time(const char \*route\_name,

const char \*stop\_name,

const char \*cur\_time)

{

try

{

if (routes.find(route\_name) == routes.end())

throw(std::out\_of\_range(""));

routes[route\_name]->get\_arrived\_time(stop\_name, cur\_time);

}

catch (const std::exception&)

{

cout << "Route \"" << route\_name << "\" does not exist." << endl;

}

}

void show\_alerts(const char \*park\_name)

{

try

{

if (parks.find(park\_name) == parks.end())

throw(std::out\_of\_range(""));

parks[park\_name]->tc.show\_alerts();

}

catch (const std::exception&)

{

cout << "Park \"" << park\_name << "\" does not exist." << endl;

}

}

private:

/\*park name\*/

std::map<const char\*, route\*> routes;

/\*route name\*/

std::map<const char\*, park\*> parks;

};