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 Gr. 33656/5	*<signature>*	Sinyapkin B.G.
Instructor	*<signature>*	Chernov A.U.

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.

2. **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.

3. **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	

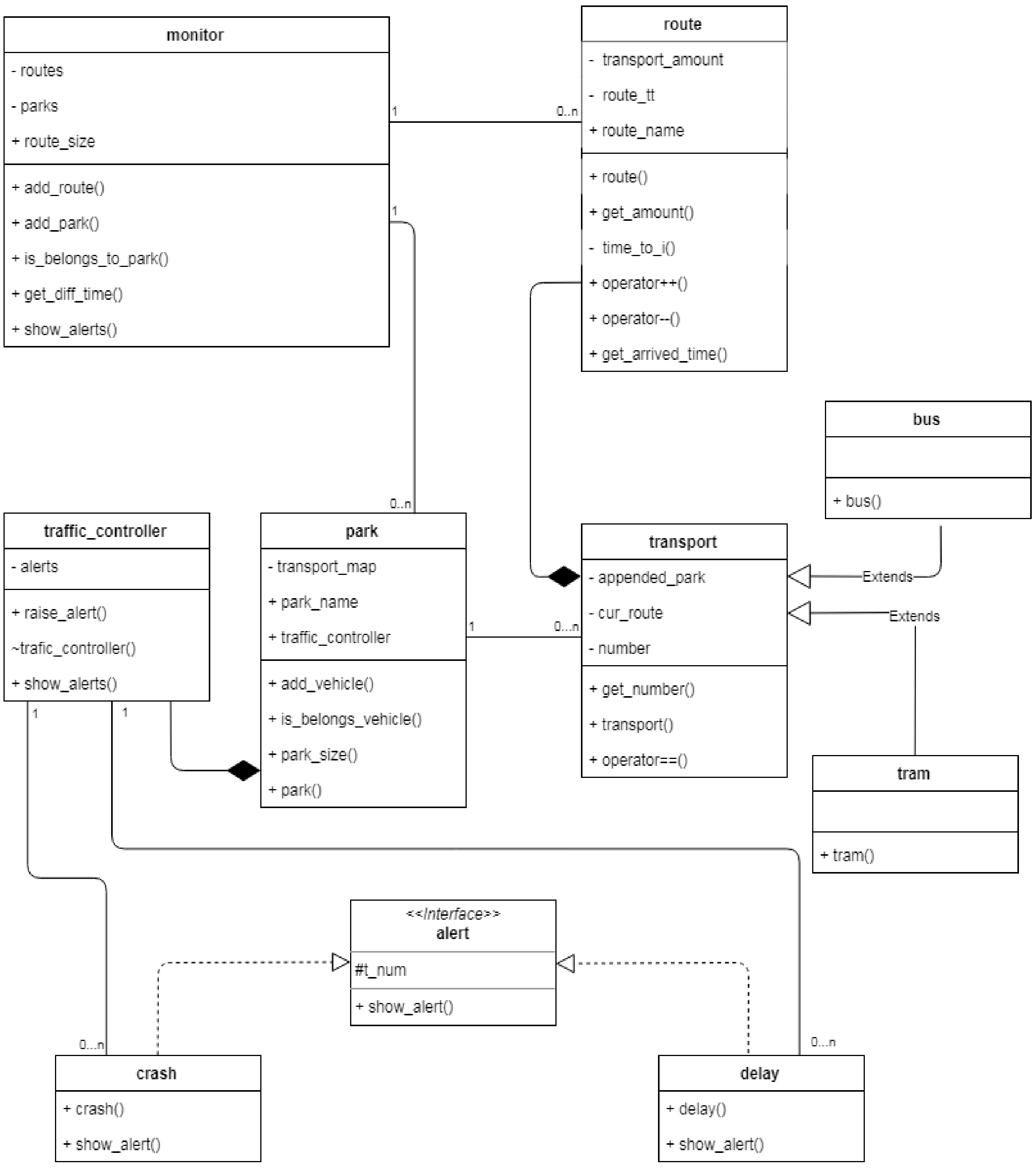
Entity name	Tram
Entity Type	Class
Comment	This is a transport that is a tram.
Attributes	
Operations	

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`

3.2 UML diagram

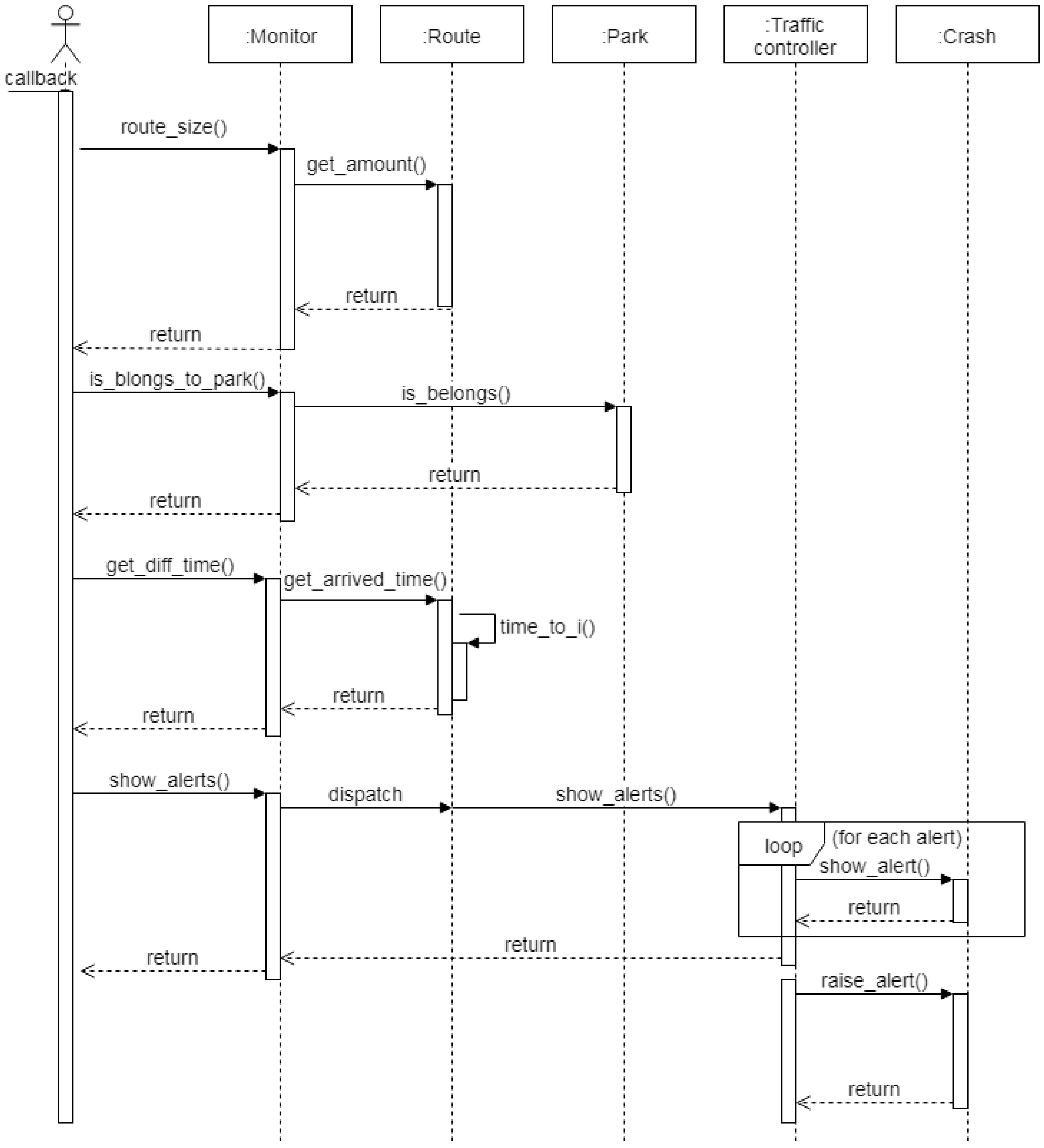


Picture 1.

3.3 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*)

3.4 UML interaction diagram



Picture 2.

3.5 Source code

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

4. Conclusion

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

5. 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;
        };
```