**Федеральное агентство связи**

Ордена Трудового Красного Знамени

федеральное государственное бюджетное образовательное учреждение высшего образования   
«Московский технический университет связи и информатики»

Кафедра «Информатика»

Лабораторная работа №3: Алгоритм A\* («A star»)

по дисциплине «Кроссплатформенные технологии программирования»

Если вы когда-нибудь играли в какую-либо игру на компьютере на основе карт, то вы, вероятно, сталкивались с органами компьютерного управления, которые умеют самостоятельно рассчитывать путь из пункта А в пункт Б. На самом деле это обычная распространенная проблема как в играх, так и в других видах программного обеспечения - поиск пути от начального местоположения до пункта назначения с успешным преодолением препятствий.

Выполнил: студент

группы БСТ1803

Шевцов Никита

Вариант №24

Москва 2020

**Содержание**

[1 Задание на разработку программы 3](#_Toc53411121)

[2 Разработанные функции и классы 6](#_Toc53411122)

[2.1 Класс Location 6](#_Toc53411123)

[2.1.1 Метод int hashCode 6](#_Toc53411124)

[2.1.2 Метод boolean equals 6](#_Toc53411125)

[2.2. Класс AStarState 6](#_Toc53411126)

[2.2.1 Метод int numOpenWaypoints 6](#_Toc53411127)

[2.2.2 Метод Waypoint getMinOpenWaypoint 6](#_Toc53411128)

[2.2.3 Метод boolean addOpenWaypoint 7](#_Toc53411129)

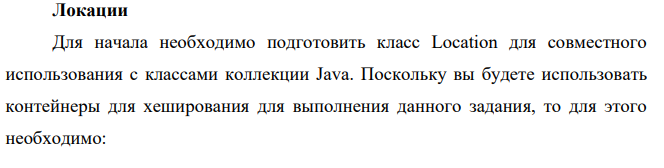
[2.2.4 Метод boolean isLocationClosed 7](#_Toc53411130)

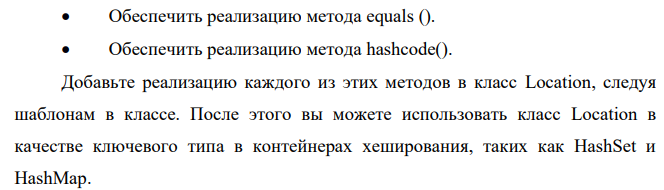
[2.2.5 Метод void closeWaypoint 7](#_Toc53411131)

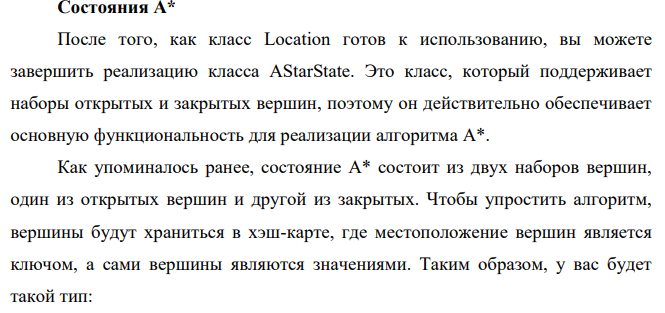
[3 Код программы 7](#_Toc53411132)

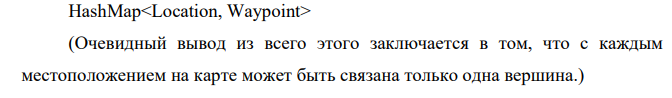
[4 Результат работы программы 9](#_Toc53411133)

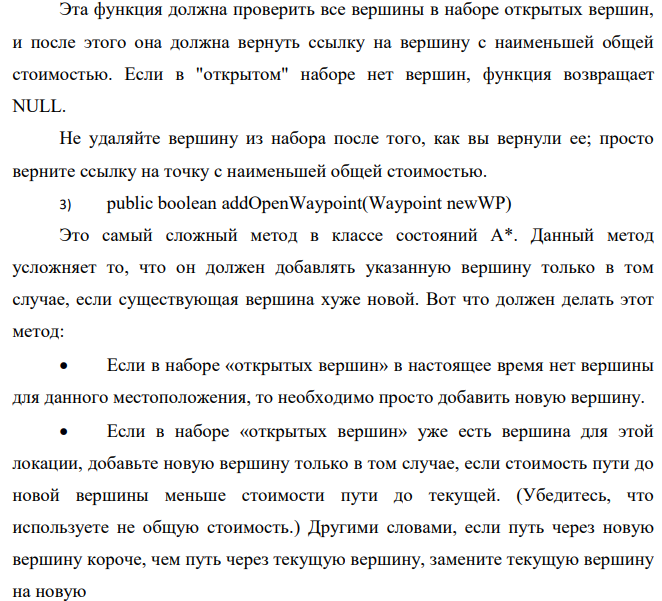
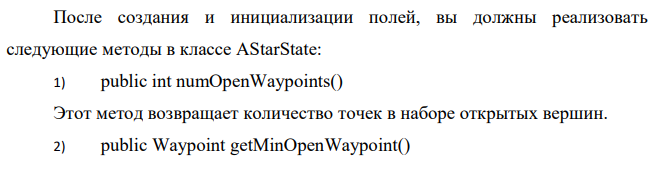
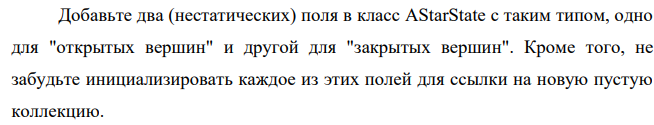
# Задание на разработку программы

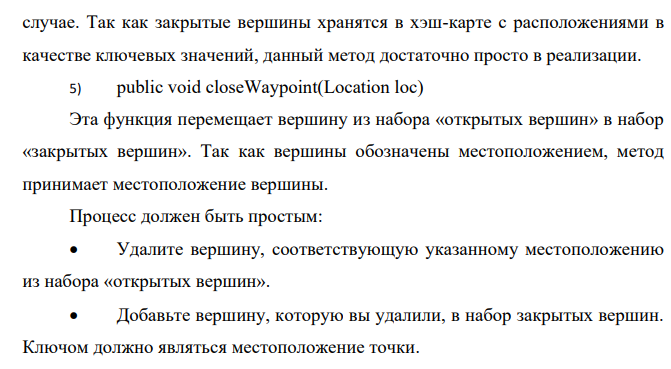
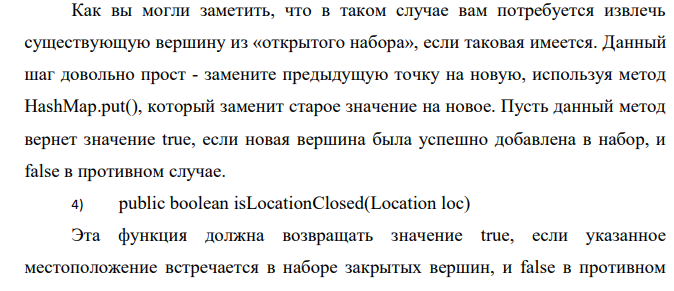


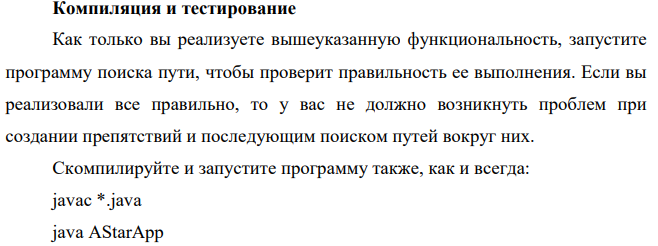












# Разработанные функции и классы

## Класс Location

Класс описывающий координатную точку на 2д карте.

* + 1. Метод int hashCode

Возвращает хэш данной точки

* + 1. Метод boolean equals

Сравнивает классы Location, возвращает **true** если точки равны, иначе **false**

## 2.2. Класс AStarState

* + 1. Метод int numOpenWaypoints

Этот метод возвращает количество точек в наборе открытых вершин

* + 1. Метод Waypoint getMinOpenWaypoint

Возвращает ссылку на вершину с наименьшей общей стоимостью, из набора “открытых” вершин. Если набор вершин пуст, возвращает **NULL**

* + 1. Метод boolean addOpenWaypoint

Метод добавляющий вершины. Возвращает **true,** если вершина успешно добавлена, иначе **false**

* + 1. Метод boolean isLocationClosed

Метод возвращает **true**, если местоположение встречается в наборе закрытых вершин, иначе **false**

* + 1. Метод void closeWaypoint

Перемещает вершину из набора “открытых” в набор “закрытых”

# Код программы

**public** **class** Location

{

/\*\* Сравнение локации с другим объектом

\* **@param** obj - объект для сравнения

\* **@return** <code>true</code>, если сравнение происходит с собой или координаты локаций совпадают,

\* <code>false</code> в противном случае \*/

@Override

**public** **boolean** equals(Object obj) {

**boolean** result = **false**;

**if** (**this** == obj) result = **true**;

**if** (getClass() == obj.getClass()) {

Location difloc = (Location) obj;

**if** (**this**.xCoord == difloc.xCoord & **this**.yCoord == difloc.yCoord) result = **true**;

}

**return** result;

}

/\*\* Получить хеш локации

\* **@return** Хеш-код локации \*/

@Override

**public** **int** hashCode() {

**int** result = 17, prime = 31;

result = result \* prime + **this**.xCoord;

result = result \* prime + **this**.yCoord;

**return** result;

}

}

**public** **class** AStarState

{

/\*\*

\* This method scans through all open waypoints, and returns the waypoint

\* with the minimum total cost. If there are no open waypoints, this method

\* returns <code>null</code>.

\*\*/

**public** Waypoint getMinOpenWaypoint()

{

Waypoint minCostWp = **null**;

**for** (Waypoint elem : openWp.values()) {

**if** (minCostWp == **null** || minCostWp.getTotalCost() > elem.getTotalCost()) minCostWp = elem;

}

**return** minCostWp;

}

/\*\*

\* This method adds a waypoint to (or potentially updates a waypoint already

\* in) the "open waypoints" collection. If there is not already an open

\* waypoint at the new waypoint's location then the new waypoint is simply

\* added to the collection. However, if there is already a waypoint at the

\* new waypoint's location, the new waypoint replaces the old one <em>only

\* if</em> the new waypoint's "previous cost" value is less than the current

\* waypoint's "previous cost" value.

\*\*/

**public** **boolean** addOpenWaypoint(Waypoint newWP)

{

Waypoint target = openWp.get(newWP.loc);

**if** (target == **null**) {

openWp.put(newWP.loc, newWP);

**return** **true**;

}

**if** (target.getPreviousCost() > newWP.getPreviousCost()) {

openWp.replace(newWP.loc, newWP);

**return** **true**;

}

**return** **false**;

}

/\*\* Returns the current number of open waypoints. \*\*/

**public** **int** numOpenWaypoints()

{

**return** openWp.size();

}

/\*\*

\* This method moves the waypoint at the specified location from the

\* open list to the closed list.

\*\*/

**public** **void** closeWaypoint(Location loc)

{

closedWp.put(loc, openWp.remove(loc));

}

/\*\*

\* Returns true if the collection of closed waypoints contains a waypoint

\* for the specified location.

\*\*/

**public** **boolean** isLocationClosed(Location loc)

{

**return** closedWp.containsKey(loc);

}

}

# Результат работы программы

