Development Documentation

路径结构

```
|_ pj2
    |_ readme
        |_ Development Documentation.pdf
    |_material
        |_ stations.txt(储存站点信息)
    |_source code
        |_ Core
            |_ Main.java
            |_ Solve.java
        |_ Data
            |_ Metro.java
            |_ Station.java
```

具体类及作用

Main.java

```java
public class Main {
    public static void main(String[] args) {
        Solve solve = new Solve();
        Scanner scanner = new Scanner(System.in);
        System.out.println("起点-->终点，请输入a\n起点-->中转站-->终点，请输入b\n退出，请输入任意内容");
        String sc = scanner.nextLine();
        switch (sc){
            case "a":
                solve.direct();break;
            case "b":
                solve.transfer();break;
            case "x":
                System.exit( status: 0);
        }
    }
}
```

方法：

public static void main(String[] args)；作为程序执行的入口

Solve.java

```
class Solve {
    private Station departure, transfer, terminal;
    private Metro metro;

    Solve() { metro = new Metro(); }

    void direct(){...}

    void transfer(){...}

    private void enterStations(){...}

    //dijkstra算法
    private void dijkstra(Station s){...}

    private void printPath(Station terminal){...}
    }
```

数据域：

用于储存起点站、中转站、终点站的 Station **departure, transfer, terminal**

用于储存整张地铁网络的 Metro **metro**

方法：

Solve()；用于构造 **Solve** 对象，并初始化 **metro**

void direct()；用于规划从 **departure** 到 **terminal** 的路线

void transfer()；用于规划从 **departure** 到 **transfer** 再到 **terminal** 的路线

private void enterStations()；用于从控制台获取用用户输入的起点站，中转站，终点站，并从 **metro** 中取出对应的 station 作为 **departure**、**transfer**、**terminal**

private void dijkstra(Station s)；通过递归找到单源到所有点的最短路径，并对所有非 **departure** 的点的 **path** 和 **dist** 进行更改

private void printPath(Station terminal)；通过递归打印出路线经过的每一站站名及所乘坐线路

Metro.java

```java
public class Metro{
    private Station[] stations = new Station[324];

    public Metro() { init(); }

    public void refresh(){...}

    //从txt文件读取数据，并生成相应站点
    private void init(){...}

    public Station getStation(String name){...}

    //直接获取当前station的最小unknown邻站点，无则返回null
    public Station minimum(){...}
}
```

数据域:

用于储存所有站点的 Station 数组 **stations**

方法:

public Metro()；用于构造 **Metro** 对象,

public void refresh()；通过 foreach 调用 **stations** 数组中的每个 **Station** 的 **refresh()**方法来将每个 **station** 进行重置

private void init();通过读取 *stations.txt* 获取所有站点信息并生成 **station** 对象，并对 **stations** 数组进行初始化

public Station getStation(String name)；通过传入的站点名称来从 **stations** 数组中返回相应的 **Station**

public Station minimum()；返回 **stations** 中当前与起点站最近的 unknown 的站点

Station.java

```
public class Station {
    private int dist;
    private Map<String,Integer> adjacent;
    private Map<String,String> line;
    private boolean known;
    private String name;
    private Station path;

    Station(String str){...}

    void refresh(){...}

    private void init(String str){...}

    public String getName() { return name; }

    public boolean isKnown() { return !known; }

    public void setKnown(boolean known) { this.known = known; }

    public Map<String, Integer> getAdjacent() { return adjacent; }

    public Map<String, String> getLine() { return line; }

    public int getDist() { return dist; }

    public void setDist(int dist) { this.dist = dist; }

    public Station getPath() { return path; }

    public void setPath(Station path) { this.path = path; }
```

Station 作为构成图与储存信息的最小节点

数据域:

用于储存站点到起点的距离的 int **dist**

用于储存相邻站点名称与到该相邻站点所需时间的 Map<String, Integer> **adjacent**

用于储存相邻站点名称与到相邻站点所走路线的 Map<String, String> **line**

用于储存被访问状态的 boolean **known**

用于储存站点名称的 String **name**

用于储存前驱站点对象的 Station **Path**

方法:

getName(); isKnown(); setKnown(); getAdjacent(); getLine(); getDist(); setDist(); getPath(); setPath();都是对数据进行封装

private void init(String str); 通过读入 Str 的内容初始化相关数据

void refresh(); 将节点中因寻找最短第一条路径改变数据全部重置