```java
package ir.shayandaneshvar;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.layout.AnchorPane;
import javafx.stage.Stage;

public class Main extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) throws Exception {
        AnchorPane root = FXMLLoader
                .load(getClass().getResource( name: "/main.fxml"));
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.setTitle("Graph Coloring Problem - Tehran Map");
        stage.getIcons().add(new Image(getClass().getResource( name: "/images/map" +
                ".jpg").toURI().toURL().toString()));
        stage.show();
    }
}
```

کلاس Main :این کلاس مسئول شروع برنامه و لود view است که به فرم fxml است.

فایل fxml که به کمک Scene Builder ساخته شده در پوشه resources به نام main.fxml ذخیره شده است.

```java
    private Map<Integer, Circle> circleMap;
    private Integer[][] matrixGraph;
    private final Color[] colors = new Color[4];
    private Color[] verticesColor;

    @FXML
    void handleColoring() {...}
    private void updateMap() {...}
    private boolean startColoring(int i, int numberOfColors) {...}
    private boolean promising(int i, Color color) {...}

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {...}
}
```

این کلاس مسئول کنترل فلوی برنامه است و الگوریتم اصلی در این کلاس نوشته شده است.

circleMap عدد هر منطقه را به دایره متناظر به آن که روی نقشه برای نمایش رنگ منطقه گذاشته شده استفاده شده است، این نگاشت شدن در متد initialize انجام شده است.

matrixGraph ماتریس مجاورت گراف نقشه است که در متد initialize به صورت دستی وارد شده است.

colors رنگ های انتخابی کاربر که 4 تا می باشد (حداکثر تعداد رنگ برای رنگ کردن هر نوع گراف) که در هر بار زدن روی دکمه color! رنگ های استخراج شده از color picker ها درون این آریه ریخته میشود.

verticesColors نیز برای رنگ کردن گراف به کار می رود، و رنگ هر راس گراف درون آن قرار میگیرد.

متد handleColoring() : متصل است به دکمه color! و الگوریتم را شروع می کند و از 2 رنگ شروع کرده و تا 4 متد startColoring را فراخوانی می کند و هر جا تعداد رنگ ها برای رنگ کردن نقشه کافی بود متوقف شده و متدupdateMap را فراخوانی میکند تا رنگ دایره های مشخص شده در نقشه آپدیت شود.

متد updateMap : این متد پس از اجرای کامل الگوریتم، فراخوانی می شود و رنگ هرکدام از دایره های مشخص شده را آپدیت میکند.

متد startColoring : الگوریتم اصلی در این متد قرار دارد که مشابه شبه کد بررسی شده در کلاس، در هر نوبت به راس i ام رنگی می دهد که رئوس مجاور با آن چنین رنگی ندارند، اگر این کار ممکن باشد نوبت راس i + 1 می شود و به صورت بازگشتی دوباره صدا زده می شود، در صورتی که ممکن نباشد رنگ داده شده بازپس گرفته می شود و مقدار false بازگردانده می شود تا درخت حل الگوریتم عقبگرد کند و راه های دیگر را امتحان کند، در صورتی که کل درخت حل الگوریتم پیمایش شود و در نهایت false بازگردانده شود به این معنی است که تعداد رنگ ها برای رنگ کردن گراف نقشه کافی نبوده است، بحث همرنگ بودن رئوس مجاور در متد promising بررسی میشودو

متدpromising : همه رئوس را چک میکند و در صورتی که راسی همرنگ و مجاور با راس داده شده باشد مقدار false برمیگرداند، در غیر این صورت true برمیگرداند.

متد initialize : برای مقدار دهی اولیه به ماتریس مجاورت گراف نقشه و نیز نگاشت کردن عدد هر منطقه به دایره آن در ابتدای برنامه استفاده شده است.

نمونه اجرا: