

实验4 抽象与接口

1 实验目的

- (1) 熟悉抽象函数与接口的声明和使用方法；
- (2) 理解面向抽象（接口）编程的思维方式。

2 实验环境

开发环境：JDK 8.0（或更高版本）+JavaFX

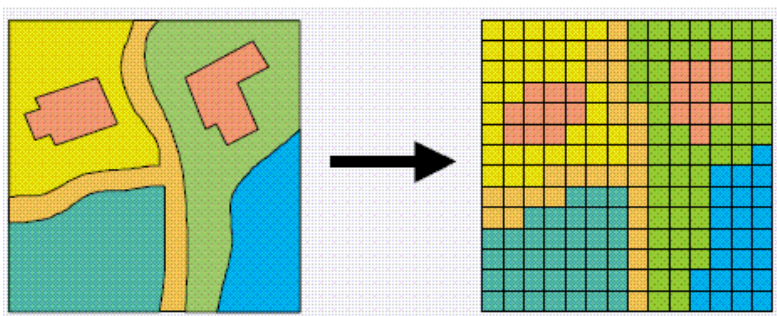
开发工具：Eclipse

设计工具：StarUML（或PlantUML等其他工具）

3 实验内容

3.1 栅格数据操作

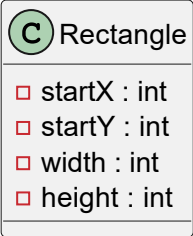
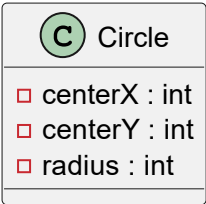
背景知识：地理信息系统中主要存在两种数据形式：矢量和栅格。矢量数据是在直角坐标中，用 (x,y) 坐标表示地图图形或地理实体的位置和形状的数据。栅格数据通过将世界分割成在格网上布局的离散方形或矩形像元来表示地理要素。每个像元都具有一个值，用于表示该位置的某个特征，例如温度、高程或光谱值。在一些应用场景下需要将矢量数据转换为栅格数据，并对栅格进行一些操作，例如在地图编辑和管理阶段采用矢量数据，在发布的地图服务中将矢量地图转换为不同比例的栅格图片。



问题描述：（1）设计一些支持 栅格化 的矢量形状对象，例如 圆形、矩形、正方形、三角形、多边形 等。（2）转换得到的 栅格 对象支持合并（merge）、相交（intersect）和裁剪（clip）等操作。（3）通过图形界面或控制台进行效果展示。

1 识别对象

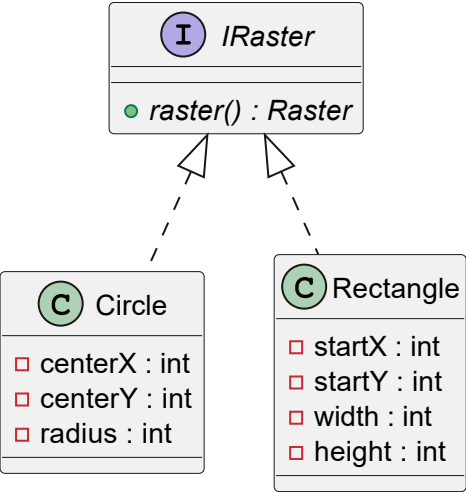
矢量形状对象以 圆形 和 矩形 为例，圆形的主要状态包括中心坐标和半径，矩形的状态包括左上角坐标、长度和宽度。栅格对象包含位置（起点坐标）信息、长度、宽度，另外还需要存储每个像元的值信息，例如1表示有，0表示无。



2 栅格化行为

每个形状的栅格化操作是有差异的，这里设计一个接口 IRaster 表示栅格化：

```
public interface IRaster {  
    Raster raster(); // 转换成栅格  
}
```



3 栅格操作声明

这里主要关注栅格数据的裁剪、相交和合并三种操作：

C Raster
<div><div>□ startX : int</div><div>□ startY : int</div><div>□ width : int</div><div>□ height : int</div><div>□ values : int[][]</div></div>
<div><div>● intersect(other : Raster) : Raster</div><div>● merge(other : Raster) : Raster</div><div>● clip(clipped : Raster) : Raster</div></div>

4 栅格化实现

以圆形为例，首先基于圆的外切正方形创建一个空的栅格对象，再基于点与圆的关系对栅格中的像元赋值。

```

public class Circle extends Shape{
    public int centerX,centerY, radius;

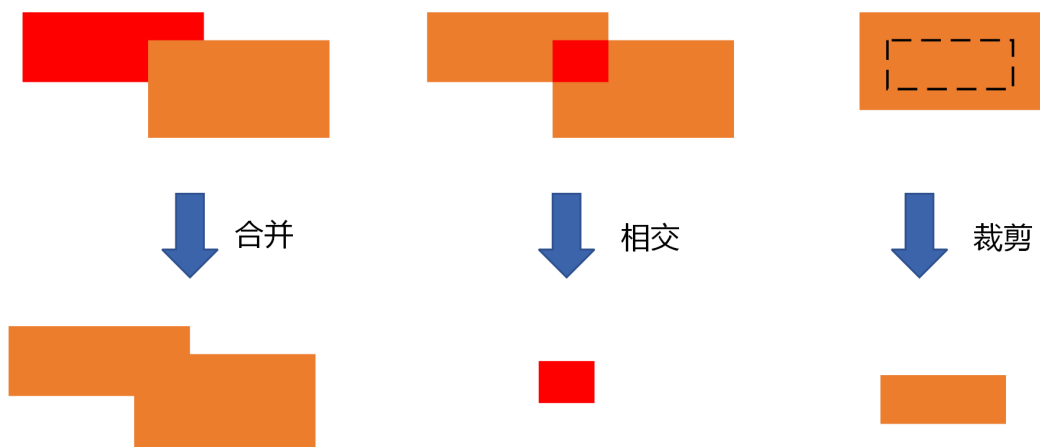
    public Circle(int x, int y, int r) {
        centerX = x;
        centerY = y;
        radius = r;
    }

    @Override
    public Raster raster() {
        int minx = centerX - radius;
        int miny = centerY - radius;
        int maxx = centerX + radius;
        int maxy = centerY + radius;
        int rasterW = maxx - minx + 1;
        int rasterH = maxy - miny + 1;
        // 基于外切正方形创建白底栅格对象
        Raster r = new Raster(minx, miny, rasterW, rasterH);
        // 遍历像元并赋值
        for(int x = minx; x < minx + rasterW; x++) {
            for(int y = miny; y < miny + rasterH; y++) {
                double dis = Math.sqrt(Math.pow(x - centerX, 2) + Math.pow(y - centerY, 2));
                if(dis <= radius) {
                    r.setValue(x, y, 1);
                }
            }
        }
        return r;
    }
}

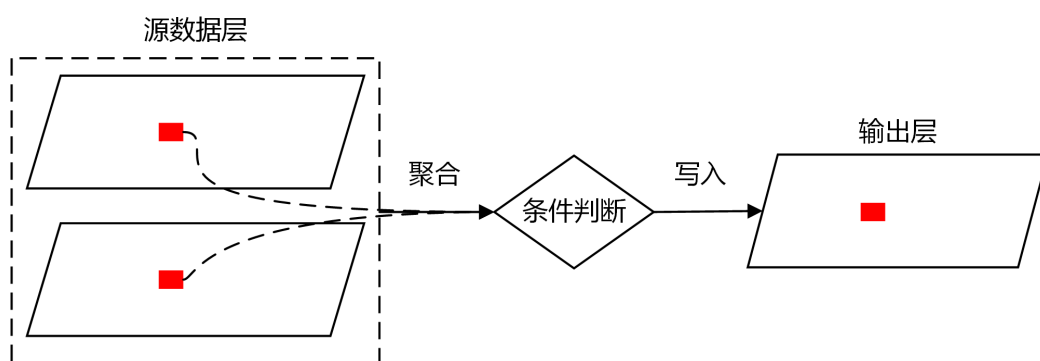
```

5 栅格操作实现

三种栅格操作如下图所示：



三种操作的共性流程如下图所示，源数据表示待操作的数据，例如两个待合并的栅格；来自不同源数据的像元值聚合到一个判断器中，根据特定规则判断是否要将值写入新的栅格数据中。



共性操作 `project` 实现如下，表示将源数据集 `ins` 根据判断器 `pre` 的要求将像元值投影到新数据 `out` 中。判断器设计为一个函数式接口，测试函数 `test` 表示将 (x,y) 位置上进行多个栅格数据像元值的聚合判断。

```

public class Raster {
    // 带判断条件的值投影
    private Raster project(Raster out, Raster[] ins, Predicate pre) {
        for(int y = out.getStartY(); y < out.getEndY(); y++) {
            for(int x = out.getStartX(); x < out.getEndX(); x++) {
                if(pre.test(ins, x, y)) {
                    out.setValue(x, y, 1);
                }
            }
        }
        return out;
    }
    // 判断接口
    @FunctionalInterface
    interface Predicate {
        boolean test(Raster[] ins, int x, int y);
    }
}

```

栅格合并操作，先创建一个合并后的白底栅格数据，再将两个栅格数据的像元值聚合输出到白底栅格中，判断逻辑为并集。

```

public Raster merge(Raster other) {
    int startX = Math.min(this.getStartX(), other.getStartX());
    int startY = Math.min(this.getStartY(), other.getStartY());
    int endX = Math.max(this.getEndX(), other.getEndX());
    int endY = Math.max(this.getEndY(), other.getEndY());
    Raster merged = new Raster(startX, startY, endX - startX, endY - startY);
    return project(merged, new Raster[] {this, other}, (ins, x, y) ->{
        return ins[0].getValue(x, y) == 1 || ins[1].getValue(x, y) == 1;
    });
}

```

栅格交集操作与合并操作类似，判断逻辑为交集。（遗留问题：修剪栅格数据中的无值边框）

```

public Raster intersect(Raster other) {
    int startX = Math.min(this.getStartX(), other.getStartX());
    int startY = Math.min(this.getStartY(), other.getStartY());
    int endX = Math.max(this.getEndX(), other.getEndX());
    int endY = Math.max(this.getEndY(), other.getEndY());
    Raster intersected = new Raster(startX, startY, endX - startX, endY - startY);
    return project(intersected, new Raster[] {this, other}, (ins, x, y) ->{
        return ins[0].getValue(x, y) == 1 && ins[1].getValue(x, y) == 1;
    });
}

```

栅格裁剪操作尝试自己完成：

```

// 基于矩形区域裁剪栅格数据
public Raster clip(Raster clipped) {
}

```

6 图形界面及运行

可以采用图形界面或控制台界面来验证功能的实现效果。

初始化矢量形状对象，并转换为栅格对象。

```

Circle c = new Circle(100, 100, 50);
Rectangle r = new Rectangle(120, 120, 80, 40);

```

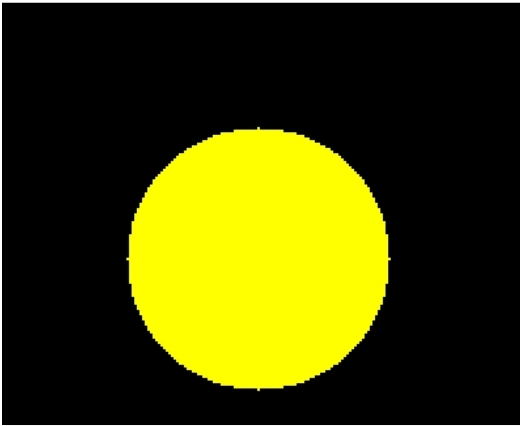
验证栅格对象的转换功能：

```

Raster r1 = c.raster();
view.render(r1);

```

 栅格化




```
Raster r2 = r.raster();  
view.render(r2);
```

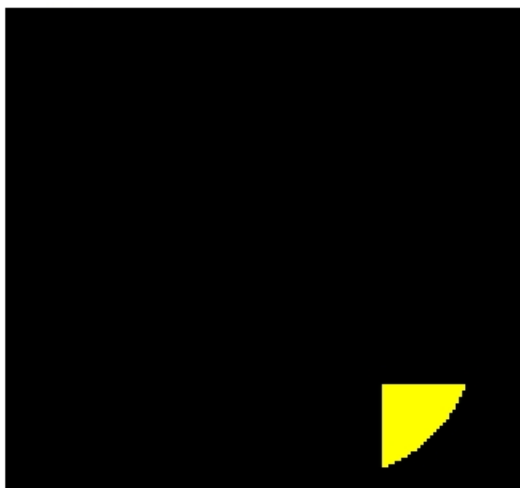
 栅格化



验证栅格相交操作：

```
view.render(r1.intersect(r2));
```

 栅格化



验证栅格合并操作：


```
view.render(r1.merge(r2));
```

 栅格化



7 完善实验并整理报告

- 1、基于UML绘制程序类图；
- 2、完善矩形类，包括栅格化功能；
- 3、增加一个几何形状类，例如椭圆、三角形等；
- 4、完善栅格裁剪操作；
- 5、思考并阐述程序中接口的用法和作用；
- 6、其他思考。

4 实验要求

4.1 实验评价

- 1、必须完成实验3.1的所有步骤，实验中的程序能正常运行。
- 2、提交实验报告，作为评价的主要依据。
- 2、遇到问题时能及时与指导老师沟通并解决问题。

4.2 实验报告

本次实验需要提交实验报告至FTP服务器对应文件夹。一律采用给定的word模板编写，报告名称规范：学号+姓名，例如2023001小明。

5 实验教学录屏

暂无