Lesson 8: Abstract Classes, Interfaces, and File I/O

Marcin Kurzyna

Lecture Goals

By the end of this lesson, students should be able to:

- Understand what an **abstract class** is and why it is useful.
- Declare and use **interfaces**.
- Build a class hierarchy for geometric shapes.
- Generate a simple SVG file using Java.
- Read from and write to files.

1 Abstract Classes

An **abstract class** is a class that:

- cannot be instantiated,
- may contain abstract methods (methods without a body),
- may contain normal methods.

Abstract classes are useful when you want to:

- define a common base for several subclasses,
- force subclasses to implement certain methods.

Example: Abstract Shape

```
abstract class Shape {
    protected String color;

    Shape(String color) {
        this.color = color;
    }

    public abstract double area();
    public abstract String toSvg();
}
```

2 Interfaces

An **interface** defines a set of methods that classes must implement.

Example: Drawable interface

```
interface Drawable {
    String toSvg();
}
```

A class can implement multiple interfaces.

3 Shape Hierarchy for SVG

We now build a simple hierarchy:

- `Shape` (abstract)
- `Circle`
- `Rectangle`
- `Polygon`

Each class implements `toSvg()` to generate SVG markup.

3.1 Circle

```
class Circle extends Shape implements Drawable {
    private double cx, cy, r;

    Circle(String color, double cx, double cy, double r) {
        super(color);
        this.cx = cx;
        this.cy = cy;
        this.r = r;
    }

    @Override
    public double area() {
        return Math.PI * r * r;
    }

    @Override
    public String toSvg() {
        return "<circle cx=\"" + cx + "\" cy=\"" + cy +
               "\" r=\"" + r + "\" fill=\"" + color + "\" />";
    }
}
```

3.2 Rectangle

```
class Rectangle extends Shape implements Drawable {
    private double x, y, width, height;

    Rectangle(String color, double x, double y, double width,
       double height) {
        super(color);
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }

    @Override
    public double area() {
        return width * height;
    }

    @Override
    public String toSvg() {
        return "<rect x=\"" + x + "\" y=\"" + y +
               "\" width=\"" + width + "\" height=\"" + height +
               "\" fill=\"" + color + "\" />";
    }
}
```

4 Generating an SVG Document

A simple SVG file looks like this:

```
<svg width="500" height="500"
     xmlns="http://www.w3.org/2000/svg">
    ... shapes go here ...
</svg>
```

Example of creating an SVG file in Java:

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.List;

class SvgWriter {

    public static void save(String filename, List<Drawable>
       shapes)
            throws IOException {
        FileWriter writer = new FileWriter(filename);

        writer.write("<svg width=\"500\" height=\"500\" "
                     + "xmlns=\"http://www.w3.org/2000/svg\">\n")
                        ;
```

```
        for (Drawable d : shapes) {
            writer.write("  " + d.toSvg() + "\n");
        }

        writer.write("</svg>");
        writer.close();
    }
}
```

5 File Reading

Reading a file line by line:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

class FileReaderExample {
    public static void readFile(String filename) {
        try (BufferedReader br = new BufferedReader(new
           FileReader(filename))) {
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Summary

In this lesson you learned:

- Abstract classes define a common foundation for subclasses.
- Interfaces specify required behaviors for classes.
- A shape hierarchy can use both abstract classes and interfaces.
- SVG files can be generated directly from Java methods.
- Java supports file reading and writing using classes like `FileWriter`, `FileReader`, and `BufferedReader`.

6 Exercises

1. Create your own `Triangle` class extending `Shape` and implementing `toSvg()`.
2. Add stroke color and stroke width to `Shape` and update all subclasses.
3. Write a program that reads SVG element data from a text file and constructs the corresponding objects.
4. Modify the `SvgWriter` so it automatically sets SVG width and height based on the shapes inside.
5. (Challenge) Create an `Animation` interface and let shapes optionally implement an animation method that outputs SVG animation tags.