Conceptual Modeling and Programming in GIScience

Lecture 8: GUI, File Readers, and Model-View-Controller

Yingjing Huang

yingjing.huang@univie.ac.at

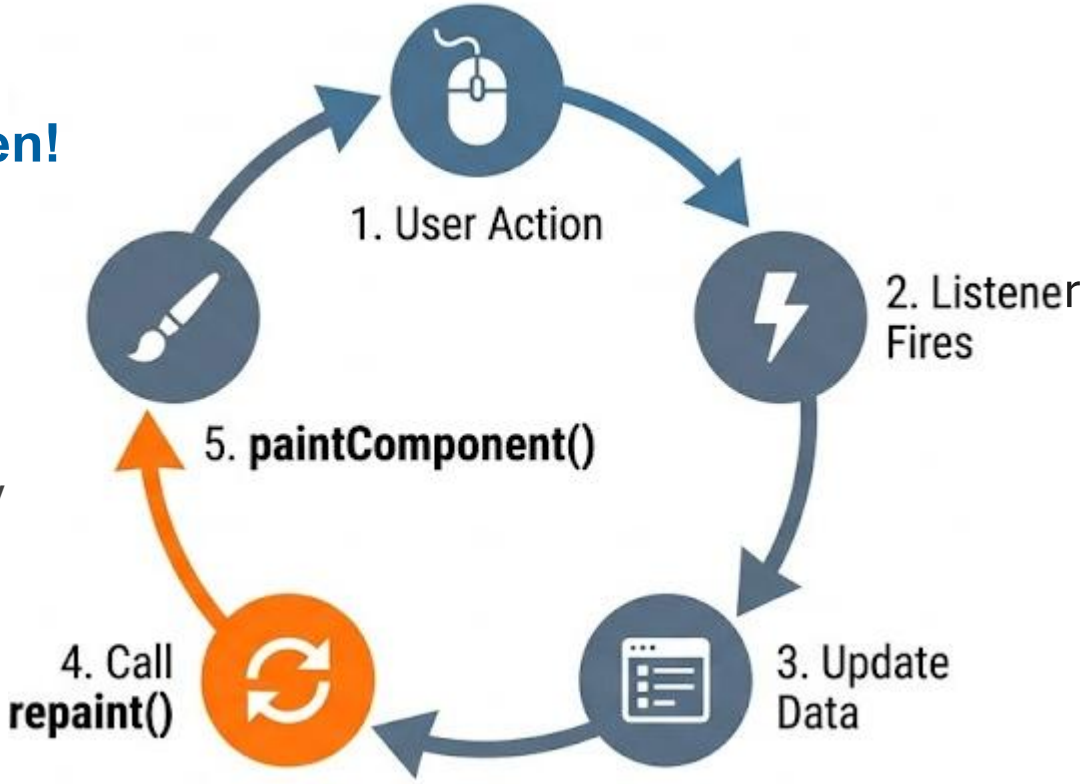
2026S – June 8, 2026

What You Will Learn in This Lecture

- Learn about your first (and most important) design pattern: **Model-View-Controller**
- Understand **Readers (and Writers)** as frameworks to give access to streams of data
- Learn how to interpret **images as grids of pixel data** in Java
- Understand how to use **Exceptions** to handle unpredictable real-world errors
- Learn how to translate the **SIR epidemic model** into a cellular automaton simulation

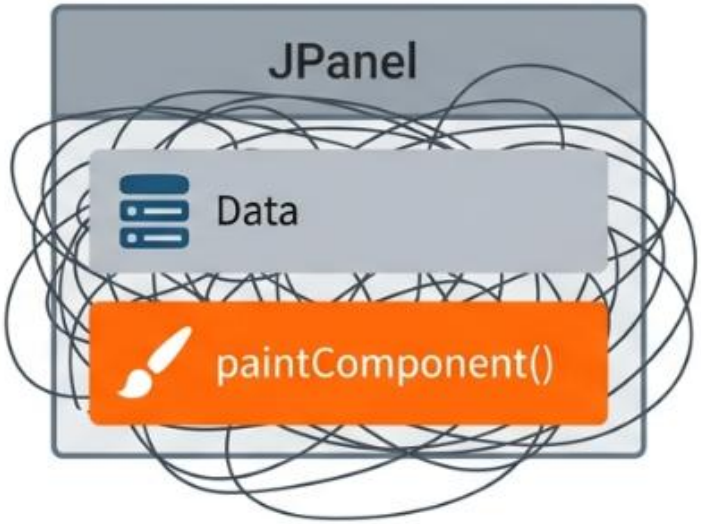
Data Changed? Tell the Screen!

- Screen does NOT update automatically
- You must call **repaint()** explicitly
- Java will then trigger ***paintComponent()***

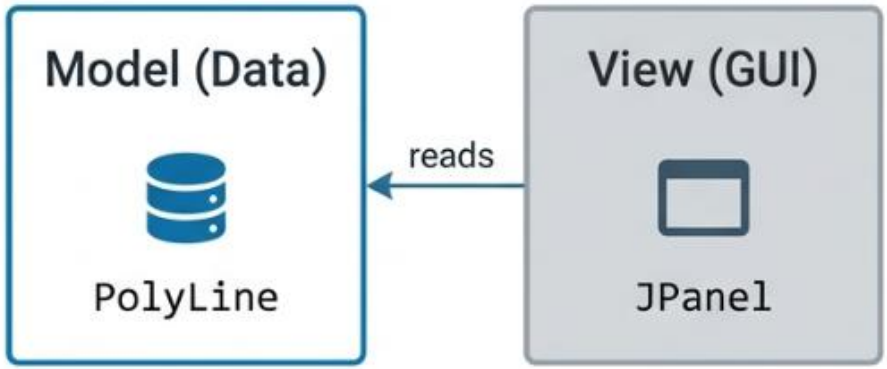


Recap: Keep Model and View Separate

Tangled



Separated

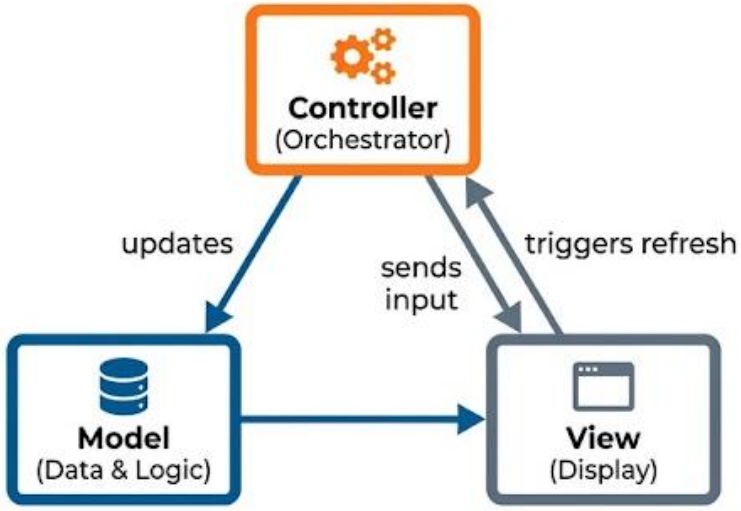


Note:
No import javax. swing.* in your core domain classes!

Model-View-Controller (MVC)

A design pattern that separates application logic into three interconnected layers

- **Model:** Holds Data & Business Logic
- **View:** Handles Display & Visuals
- **Controller:** Manages User Input & Execution Flow



The Model

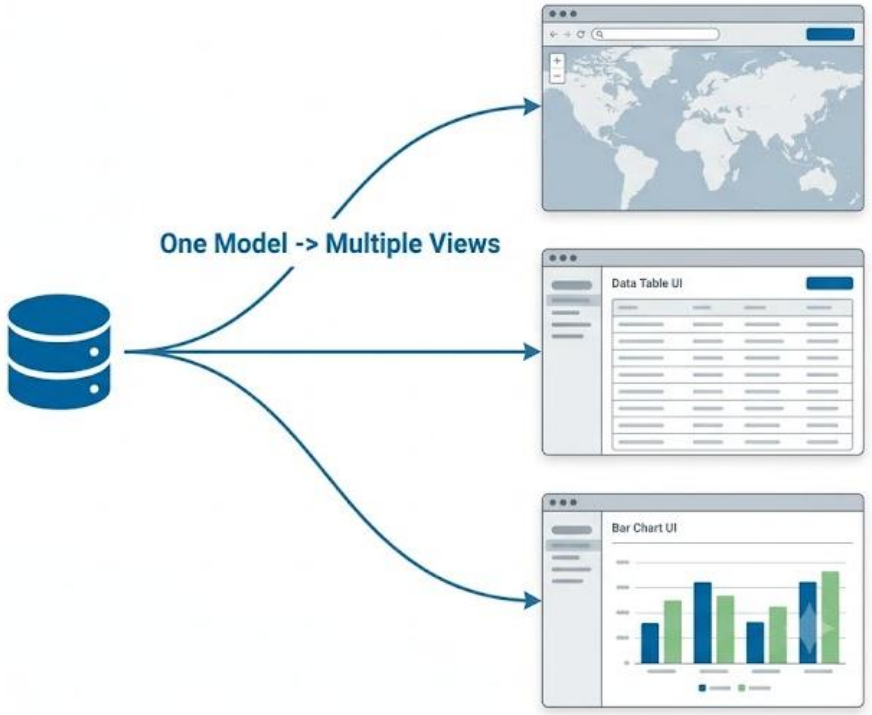
- Application-driven abstraction of the real world
- **Defines objects**, their states, and behaviors
- Completely **independent** of the User Interface
- Can it **run as a command-line tool** without changing code?

No GUI code in the Model

- **NO** `import javax.swing.*` inside the Model.

The View

- Renders the Model to the screen
- **Captures** raw user input (e.g., clicks, text input)
- Contains **NO** business logic
- **Simply reports events** (“A button was clicked”) to the **Controller**



One Model can have multiple Views (Map, Table, Chart) simultaneously

The Controller

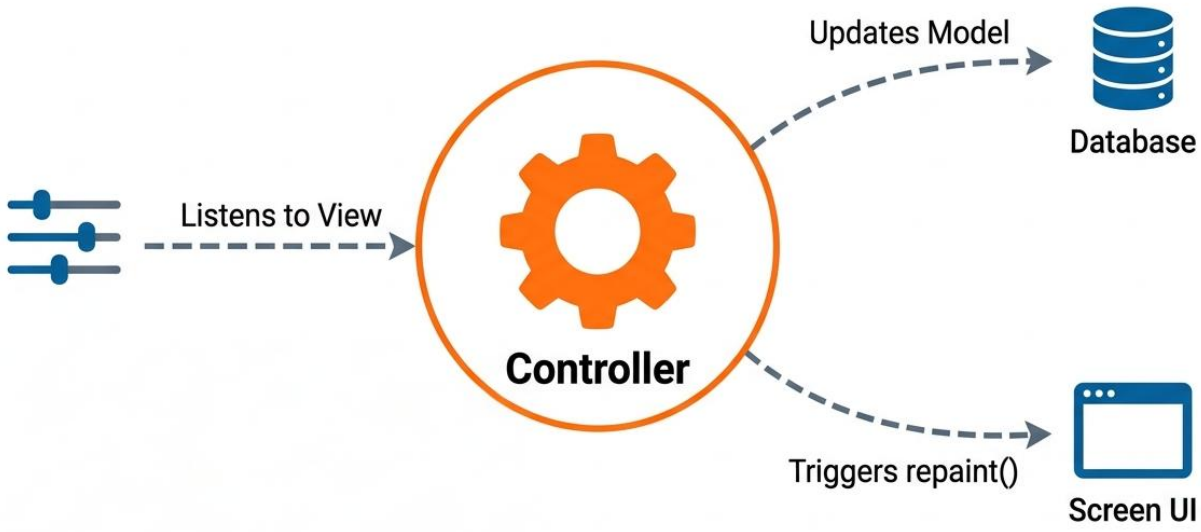
Acts as the glue between Model and View

Workflow:

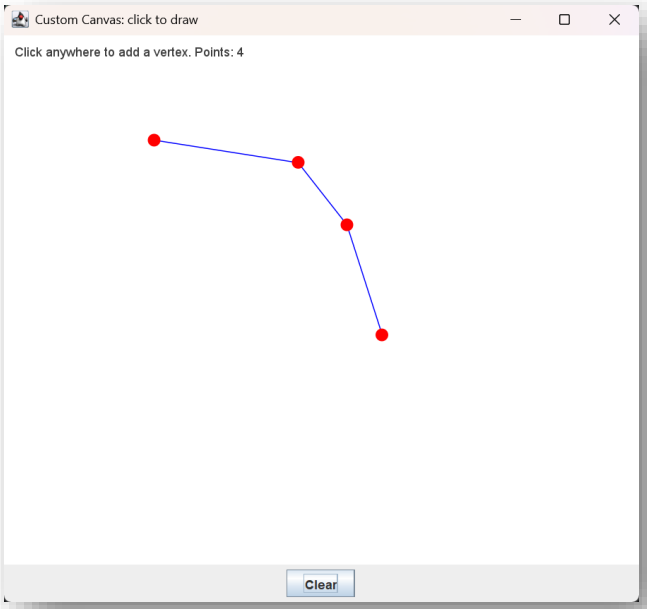
- Listens to the View for user inputs
- Interprets inputs and triggers operations
- Updates the Model accordingly
- Tells the View to refresh (*repaint()*)

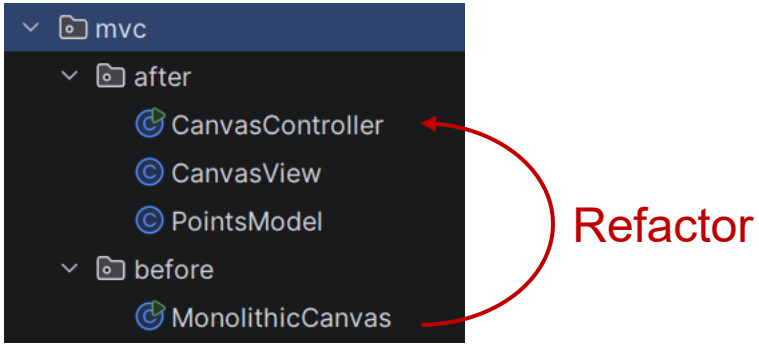
Manges execution flow, timers,

and program shutdown



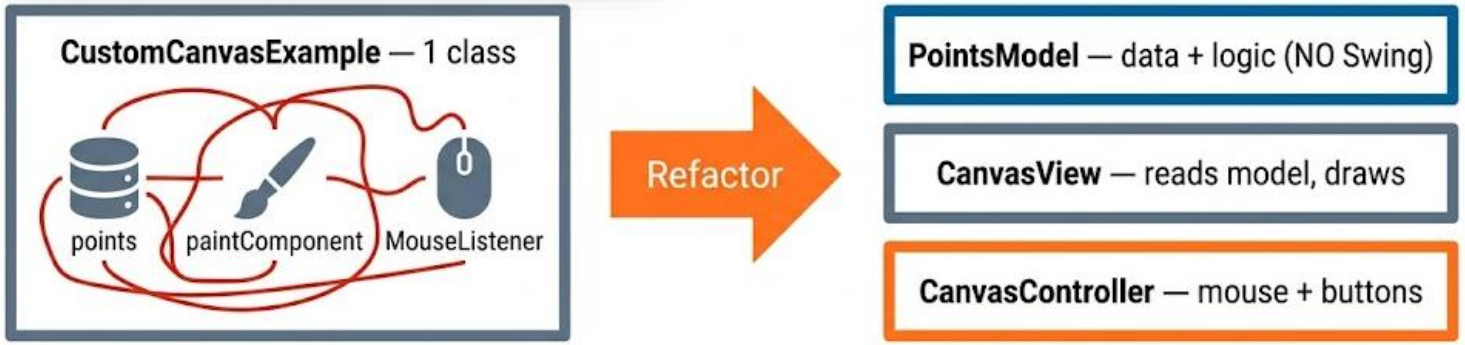
Example: CustomCanvasExample from Week 7





They look the same in the GUI

But MVC has clear structure



Check the details in Week 8's Code Examples

The Payoff of MVC

Reusability

Swap the GUI without touching core data.

Testability

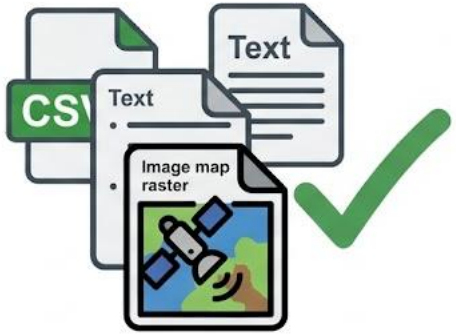
Test business logic without opening a window.

Clarity

Every file has exactly one reason to change.

Transition: Data Comes from Outside

Hardcoded Arrays



External Files (java.io)

From typing data by yourself to read data from external files

BufferedReader

Wraps the FileReader to add efficiency

FileReader

Opens the file and provides basic character access

```
// Step 1: Open the file using a wrapper chain
BufferedReader reader = new BufferedReader(new FileReader("data.csv"));

// Variable to temporarily store the content of each line
String line;

// Step 2: Loop through the file line by line
while ((line = reader.readLine()) != null) {
    // Process or display the current line
    System.out.println("Processing: " + line);
}

// Step 3: CRITICAL! Release system resources
reader.close()
```

readLine()

- Returns one full line of text at a time
- Typically used inside a loop until it returns null (End of File)

Always close() the reader when finished to prevent memory leaks

BufferedWriter

Wraps the FileWriter to add efficiency

FileWriter

Opens (or creates) the file and accepts characters to write

```
String wktRoute = "LINESTRING(116.38 39.90, 116.40 39.92, 116.43 39.95)";
String outputFile = "output_route.txt";

try (BufferedWriter writer = new BufferedWriter(new FileWriter(outputFile))) {

    // Write the header or summary info
    writer.write("--- Digitized Route Summary ---");
    writer.newLine(); // Inserts a platform-dependent line separator

    // Write the actual geospatial data (WKT)
    writer.write(wktRoute);
    writer.newLine();

    System.out.println("Data successfully written to buffer.");

} catch (IOException e) {
    System.err.println("Failed to write file: " + e.getMessage());
}
```

write(...) / newLine()

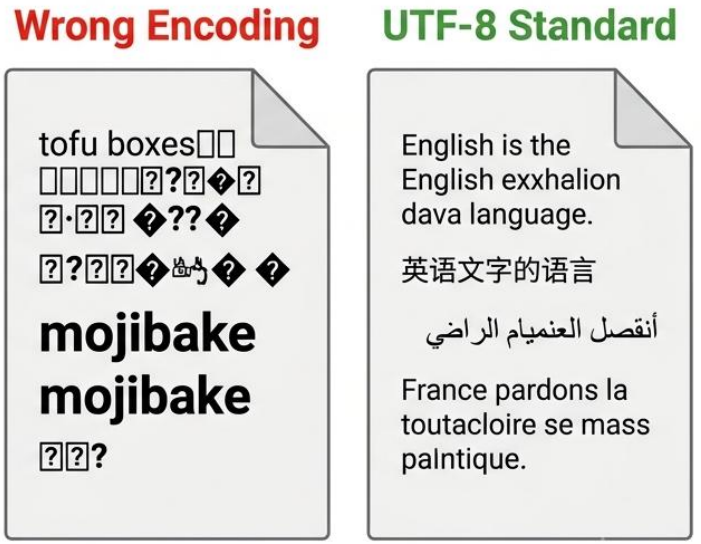
Output text one piece or one line at a time

try-with-resources: try (BufferedWriter w = ...) { ... }

auto-closes the file even if an error occurs

Programs read raw bytes and interpret them via **encoding standards**

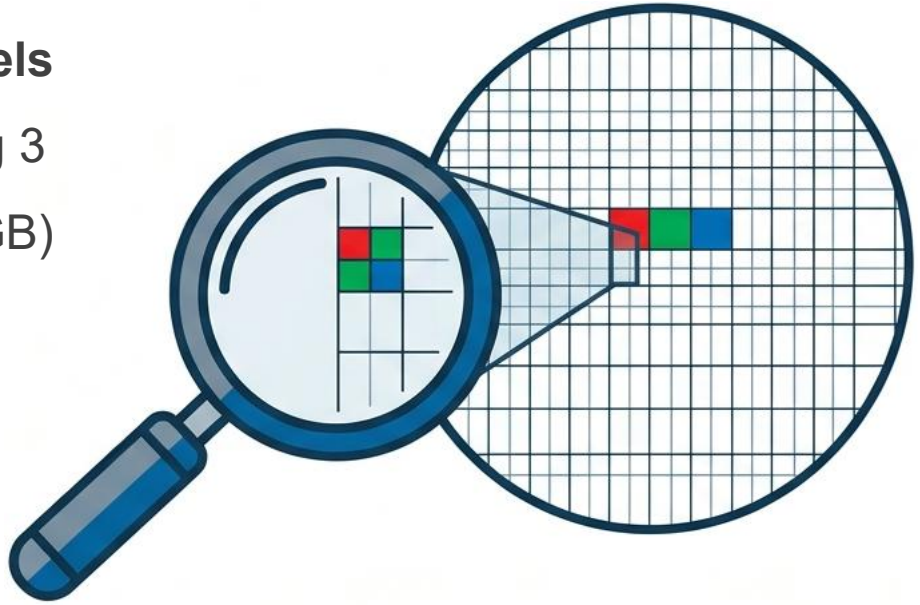
- **ASCII:** Older standard for basic English
- **UTF-8:** Modern standard, encodes **characters** in every language



Always use UTF-8 as the safe default

Binary Data: Images

- An image is simply a **2D grid of pixels**
- Each pixel is typically encoded using 3 numbers: Red, Green, and Blue (RGB)
- Values range from 0 to 255
- Memory cost:
  - **1 Pixel = 3 Bytes**
  - A 1000 x 1000 pixel image requires 3,000,000 bytes (3MB) in memory uncompressed



ImageIO Class

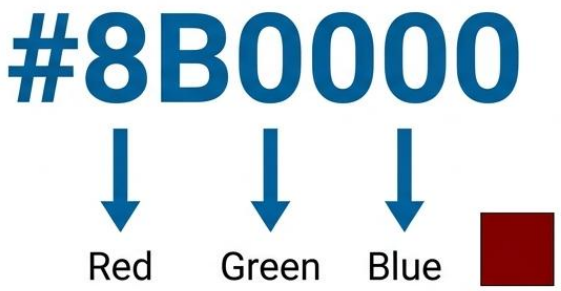
ImageIO.read(): Loads files (JPEG, PNG) into a *BufferedImage* object

```
// 1. Load the file into memory
BufferedImage img = ImageIO.read(new File("map.png"));

// 2. Extract the color of a specific pixel
int pixelColor = img.getRGB(x, y);
```

Hex Colors

- Commonly used encoding format: e.g., #8B0000
- Uses 16 digits (0-9, A-F)



Radiometric Resolution in Remote Sensing

Determines how many distinct intensity values a sensor records per channel

- **Standard Images (8-bit):** 256 levels per channel
- **GIS Satellites (12/16-bit):** 4,096 to 65,536 levels (e.g., Landsat, Sentinel)

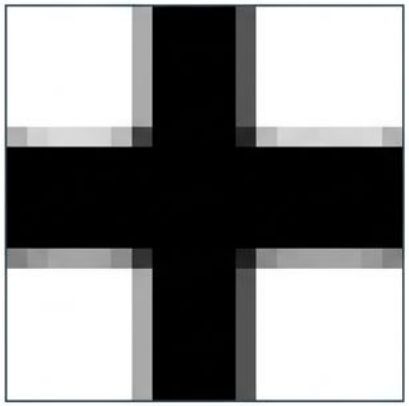
Why it matters: Higher resolution detects subtle ecological/geological differences that disappear when compressed to 8 bits

Portable Gray Map (PGM)

- Stores pixel values as plain text numbers in rows and columns
- You can read the image data in a basic text editor

An "Image" and "Raw Data" are the same thing, just interpreted differently

Binary Image (PGM View)



Images are just 2D data arrays

ASCII Text File View

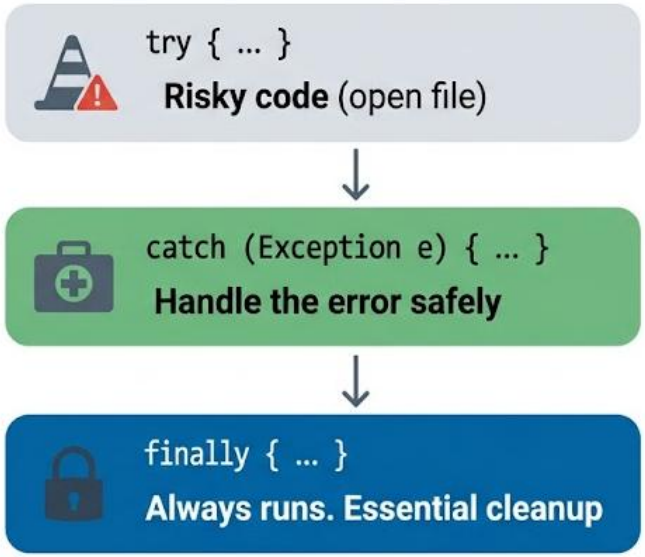
```
 0   0  90 190  98  80
 0  88  95 193  91  80
80  94  95 192  94  92
80  92  95 199  92  94
90 102 100 199 102 103
90 104 104 199 104 104
90 102 104 199 104 104
90  64  93 103  96  66
60  64  93 182  96  66
60  66  93 182  96  66
90  98  93 100  98  98
```

File operations introduce external factors outside your control:

These are **NOT** logic bugs. You cannot prevent them; you must **REACT** to them

Java Exceptions: Objects thrown by Java when a failure occurs, halting normal execution

The try-catch-finally Block




```
try {
    // The Danger Zone: Risky operations
    BufferedReader reader = new BufferedReader(new
FileReader("map.txt"));
    // ...
} catch (IOException e) {
    // The Safety Net: Handle the error without crashing
    System.out.println("File not found! Using default grid.");
} finally {
    // The Cleanup: Always runs, success or failure
    System.out.println("Execution finished.");
}
```

Checked Exceptions

External factors beyond control. The compiler **FORCES you to handle them.**

- External factors beyond your control
- The Java compiler **FORCES** you to handle them
- Use *throws* in the method signature to pass the responsibility up to the caller

Unchecked Exceptions

Logic errors in your code. The compiler expects you to **FIX** the bug.

- Programming/logic errors in your code
- The compiler expects you to **FIX** the bug, not just catch it

1. Fallback

Use safe default values.

2. Retry

Wait briefly and try again.

3. Skip

Log error and continue batch.

```
int port;

try {
    // Attempt to read custom configuration
    port = Integer.parseInt(readConfigFromFile("config.txt"));
} catch (IOException | NumberFormatException e) {
    // FALLBACK: If file is missing or corrupted, use a sensible default
    System.err.println("Warning: Config missing/invalid. Falling back to default port 8080.");
    port = 8080;
}
```

NEVER leave a catch block empty!

1. Fallback

Use safe default values.

2. Retry

Wait briefly and try again.

3. Skip

Log error and continue batch.

```java
int maxRetries = 3;
int attempt = 0;
boolean success = false;

while (!success) {
    try {
        attempt++;
        connectToNetwork(); // Might throw NetworkException
        success = true;     // If we reach here, connection succeeded
    } catch (NetworkException e) {
        if (attempt >= maxRetries) {
            System.err.println("Error: Network failed after " + maxRetries + " attempts.");
            throw e; // Give up and rethrow the exception
        }
        System.out.println("Network dropped. Retrying in 1 second... (Attempt " + attempt + ")");
        Thread.sleep(1000); // Wait 1 second before retrying
    }
}
```

NEVER leave a catch block empty!

1. Fallback

Use safe default values.

2. Retry

Wait briefly and try again.

3. Skip

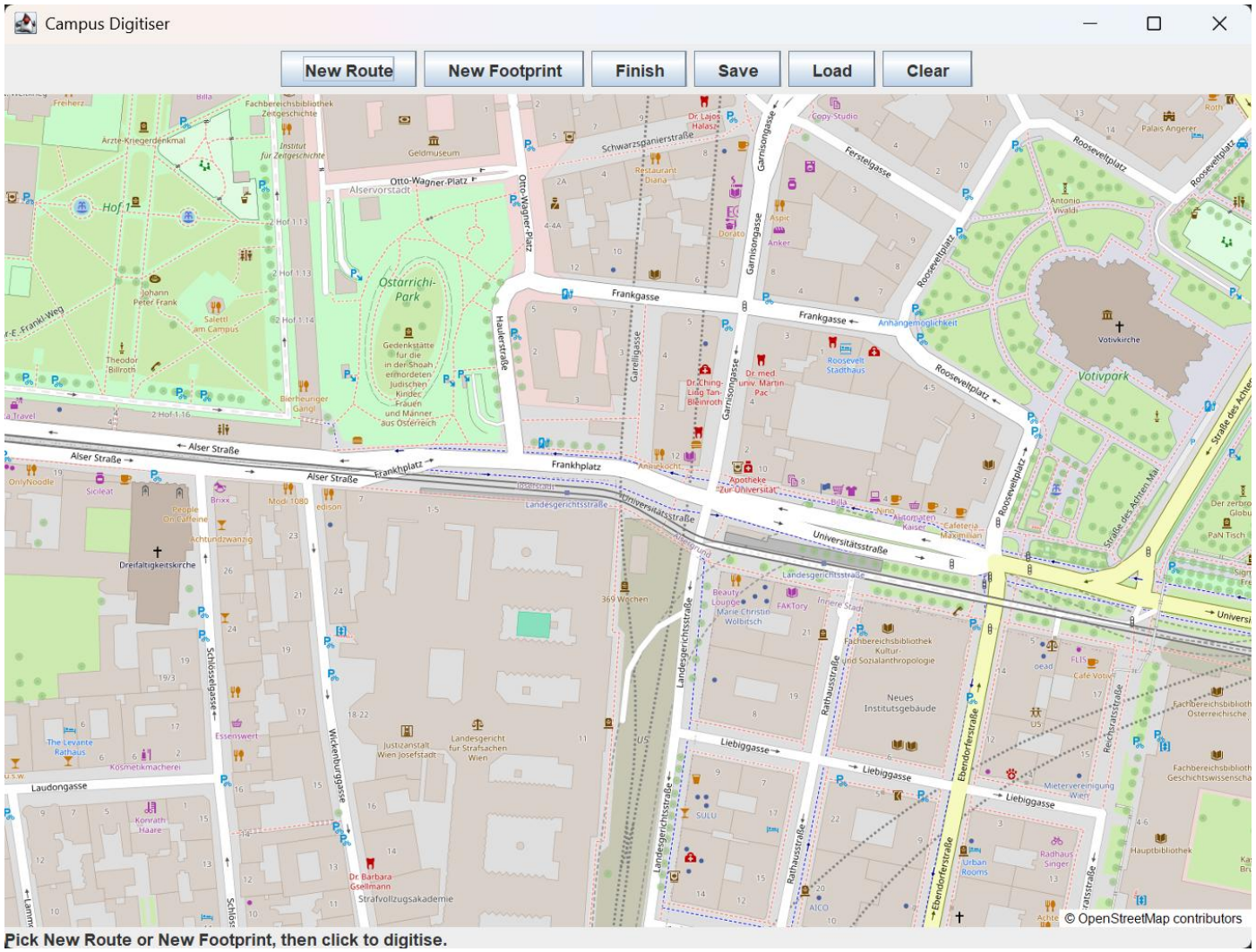
Log error and continue batch.

```java
List<String> csvRows = loadCsvData();

for (int i = 0; i < csvRows.size(); i++) {
    try {
        // This might throw DataFormatException if a row is corrupted
        processRow(csvRows.get(i));
    } catch (DataFormatException e) {
        // SKIP: Log the specific error and row number, then 'continue' the loop
        System.err.printf("Row %d is corrupted (Skipped). Error: %s%n", i + 1, e.getMessage());
        // The loop naturally moves to the next iteration (i++)
    }
}
```

NEVER leave a catch block empty!

Putting It Together: The Campus Digitiser



Check the details in Week 8's Code Examples

Build an **SIR epidemic simulation**

which is a simplified version of agent-based epidemic models

(Huang et al., Simulating SARS, 2004) https://jasss.soc.surrey.ac.uk/7/4/2.html

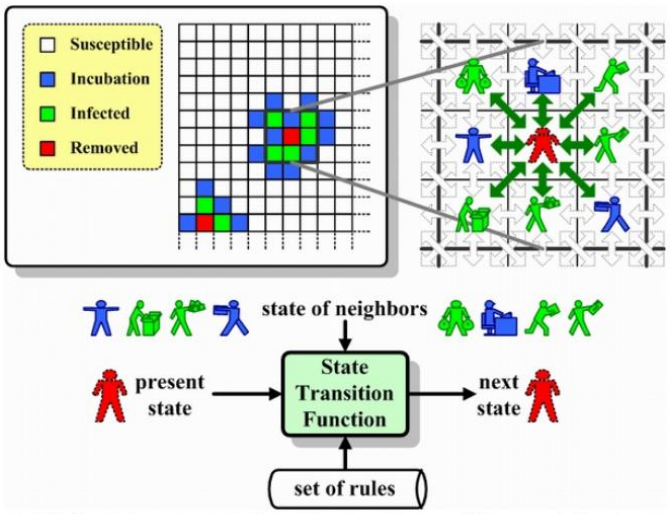


Figure 2. Cellular Automata, state transition function, and Moore neighborhood concept

- Every cell is empty, or a person who is Susceptible/ Infectious / Recovered
- **Each Day**: people near the sick may be infected, the sick recover, and everyone moves to nearby empty cells
- A **stay-at-home order** keeps a fraction of people from moving
- **QUESTION**: Can strict stay-at-home orders effectively contain the epidemic?

The Assignment: Your Tasks

PROVIDED

- ✓ MVC skeleton
- ✓ GUI controls + wiring
- ✓ Timer
- ✓ Random seeding
- ✓ Movement logic

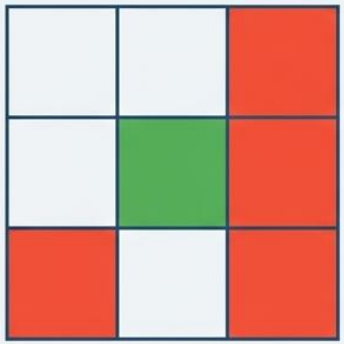
Your Tasks

- ☐ countInfectiousNeighbours()
- ☐ nextState()
- ☐ step() – copy and swap
- ☐ GUI layout

Task 1

countInfectiousNeighbours(row, col)

- Look at the 8 neighbours; skip the cell itself
- Skip neighbours that fall off the grid edge (row/col < 0 or >= size)
- Count how many are Infectious



skip the centre;
skip off-grid cells

count the red neighbours
→ k = 3

Task 2

nextState(current, infectiousNeighbours)

— the SIR rule:

- S → I with probability $1 - (1 - \beta)^k$
  k = infectious neighbours
  β = infectionRate
- I → R with probability γ (recoveryRate)
- R stays R

Roll the dice with

random.nextDouble() < probability

Recap: The Copy-and-Swap Pattern

```
public void tick() {
    boolean[][] next = new boolean[ROWS][COLS];    // ① fresh

    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            next[i][j] = nextStateOf(i, j);        // ② read grid, write next
        }
    }

    grid = next;                                   // ③ swap — single assignment
}
```

- grid is **untouched** during the whole computation
- next is filled in based on grid's current state — consistent, by construction
- One assignment at the end commits the new generation

Other names for the same idea:

double-buffering (graphics), immutable update (functional programming).

step()

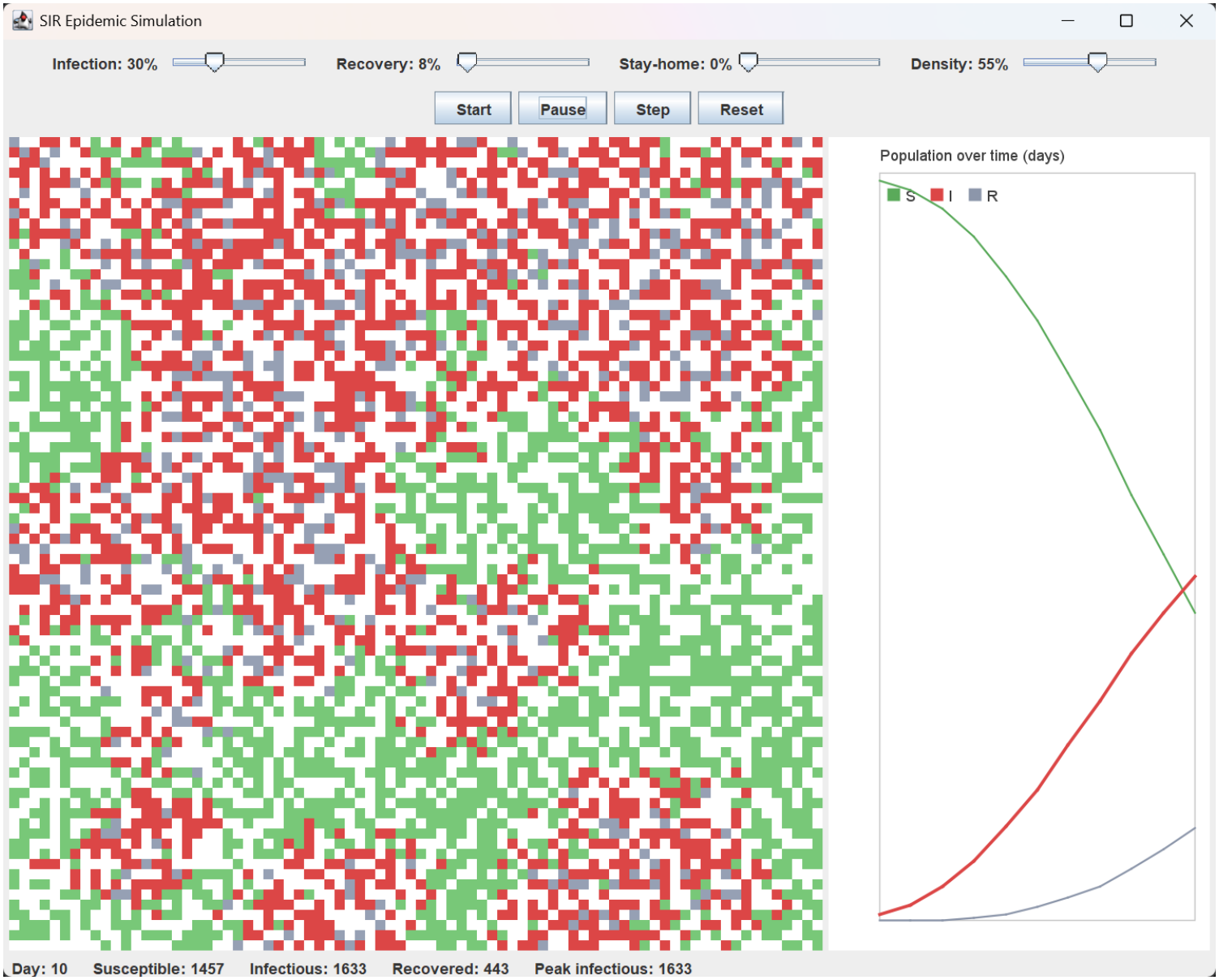
advance one day with the copy-and-swap pattern

1. Make a fresh grid: *int[][] next = new int[size][size];*
2. For every cell of the current grid:
   - if it is EMPTY
     → leave *next[r][c]* empty;
   - otherwise
     → *next[r][c] = nextState(current, countInfectiousNeighbours(r, c));*
3. Swap it in: *grid = next;*
4. Keep the provided *move()* call at the end of *step()*

Task 4: Design Your Own GUI

- The pieces are already created and wired for you: four **sliders** (infection, recovery, stay-home, density), four **buttons** (Start / Pause / Step / Reset), the **map** (view), the **live chart** (chart), and the **status line**
- A **bare-bones starter layout** (buttons + grid + status) is already there so the app runs and you can test your model
  — extend it: add the sliders and the chart, and arrange it your way
- Put the sliders/buttons in a JPanel, put the map + chart side by side, then give the frame a **BorderLayout**
- Because the wiring is done, your buttons work no matter how you arrange them
  — **so make it look the way you want!**
- *Optional*: restyle the cell colours in *SirView.colorFor()*

Example layout



Submit & Grading

- Submit **FirstNameLastNameSIR.zip** (your *.java files)
- Individual work
- **Deadline**: Jun 29th 23:00 PM (Vienna Time)

- **Graded on**:
  - correct SIR rules
  - copy-and-swap **(most weight)**
  - a working, usable GUI
  - readable, commented code