**a) Comparison of Constructors with Java's Garbage Collection vs C++ Destructors**

| **Feature** | **Constructors** | **Java Garbage Collection** | **C++ Destructors** |
| --- | --- | --- | --- |
| **Purpose** | Initialize object state during creation | Automatically reclaim unused memory | Explicitly release resources (memory, files) |
| **Invocation** | Called explicitly via new keyword | Runs automatically in JVM background thread | Called explicitly via delete keyword |
| **Control** | Developer-defined logic | JVM-controlled scheduling (non-deterministic) | Developer-controlled deterministic cleanup |
| **Resource Handling** | Allocates resources | Only manages *memory* (ignores non |  |

**Java GC Weakness**:  
Java GC cannot handle non-memory resources (e.g., open files). Explicit cleanup is required using try-finally/try-with-resources, unlike C++ destructors which guarantee cleanup.

### b) St. Mary’s Hospital EMR System Implementation

#### i) this Keyword for Naming Conflicts

public class Patient {

private String id;

private String allergyNotes;

public Patient(String id, String allergyNotes) {

// 'this' resolves conflict between parameter and instance variable

this.id = id;

this.allergyNotes = allergyNotes;

}

}

**Explanation**:  
this.id refers to the instance variable, while id is the constructor parameter. This avoids ambiguity when names are identical.

#### **ii) Static Methods & Method Overloading**

1. **Static Method**  
   **Definition**: A method bound to the class (not instances), called via ClassName.method().  
   **Hospital Use**: Shared utility for BMI calculation.

public class MedicalUtils {

// Static method: No patient instance needed

public static double calculateBMI(double weightKg, double heightM) {

return weightKg / (heightM \* heightM);

}

}

// Usage: MedicalUtils.calculateBMI(70, 1.75);

1. **Method Overloading**  
   **Definition**: Multiple methods with the same name but different parameters.  
   **Hospital Use**: Generate patient reports in different formats.

public class ReportGenerator {

// Text report

void generateReport(String patientId) {

System.out.println("Text Report for: " + patientId);

}

// HTML report (overloaded)

void generateReport(String patientId, boolean asHtml) {

if(asHtml)

System.out.println("<html>Report for " + patientId + "</html>");

}

// PDF report (overloaded)

void generateReport(String patientId, String format) {

System.out.println(format + " Report for " + patientId);

}

}

### c) final vs finally

| **Keyword** | **Purpose** | **Example** |
| --- | --- | --- |
| final | Restricts modification (variables, methods, classes) | java<br>public final class MedicalConstants {<br> public static final double MAX\_DOSE = 500.0; // Unmodifiable constant<br>}<br> |
| finally | Ensures code executes after try-catch, regardless of exceptions | java<br>try {<br> generateReport(); // May throw IOException<br>} finally {<br> file.close(); // Always executed<br>}<br> |

**Medical Use Cases**:

* final: Prevent alteration of critical constants (e.g., MAX\_DOSE).
* finally: Guarantee file closure after report generation to prevent resource leaks.

### d) Medical Log File Reader Program

import java.io.\*;

public class MedicalLogReader {

public static void main(String[] args) {

File medicalLogFile = new File("medical\_logs.txt");

readAndDisplayFile(medicalLogFile);

}

public static void readAndDisplayFile(File file) {

try (BufferedReader reader = new BufferedReader(new FileReader(file))) {

String line;

System.out.println("--- MEDICAL LOG START ---");

while ((line = reader.readLine()) != null) {

System.out.println(line);

}

System.out.println("--- MEDICAL LOG END ---");

}

catch (FileNotFoundException e) {

System.err.println("Error: File not found. " + e.getMessage());

}

catch (IOException e) {

System.err.println("Error reading file: " + e.getMessage());

}

}

}