OOP Final Assignment

# Simulating Memory Leak in Java

To demonstrate the effect of different heap sizes on memory usage, we can use the following code:

public void simulateMemoryLeak() {

List<Student> students = new ArrayList<>();

int index = 1;

Runtime rt = Runtime.getRuntime();

try {

while (true) {

Student s = new Student("Student" + index, String.valueOf(index), new DatePicker(), "Semester" + index);

students.add(s);

System.out.printf("[%d] free memory: %d%n", index++, rt.freeMemory());

}

} catch (OutOfMemoryError e) {

System.out.println("Out of memory at index " + (index - 1));

System.out.printf("Total memory used: %d bytes%n", rt.totalMemory() - rt.freeMemory());

}

}

This code simulates a memory leak by creating new Student objects in an infinite loop until the application runs out of memory. The amount of memory required to create each object is not known in advance and may vary depending on the system.

When running this code with a heap size of **-Xmx750m**, we get the following output:

Text

Description automatically generated

This indicates that the application ran out of memory after creating 500077 Student objects and that the total amount of memory used was approximately 750 MB.

When running the same code with a heap size of **-Xmx375m**, we get the following output:

Text

Description automatically generated

This indicates that the application ran out of memory after creating 250048 Student objects and that the total amount of memory used was approximately 350 MB.

As you can see from these results, reducing the heap size can have a significant impact on how much memory an application can use before running out of memory.