Design Description - Team 15

Jaime Martinez (jmartinez66)

Jane McDowell (jmcdowell7)

Katie Addison (kpeatman3)

Kevin Jones (kjones314)

Overview

The program Gallhp.Demo is a cross platform java application that simulates the diffusion of heat through a uniform two-dimensional metal plate by approximating Fick's Law and providing a graphical representation to the user. The program is accessible to the user via a graphical interface, which allows the user the ability to customize the execution by selecting one of four separate programs. The programs differ in data type utilization and methodology, each of which has an impact on performance and precision. The user further customizes the outcome of the program by providing specific values to be used in the computation of plate temperatures. The user must supply the following values: an integer value representing the dimension of the grid (number of columns and rows), and values for each of the initial edge temperatures of the plate (top, bottom, left and right). The results of the computation are displayed to the user in a window. An intriguing aspect in the program design is how do we accomplish optimum results with respect to performance and precision? Possible approaches to achieving optimum results include explicitly setting the variables which control the number of iterations for simulating diffusion and the minimum amount of relative change for the computation to continue as well as limiting the user in the grid dimension input variable. The team chose to confine the maximum number of iterations to 100 and confine the relative change stopping criteria to 1.0. Additionally, the team chose to limit the user grid size input to 25.

Program Characteristics

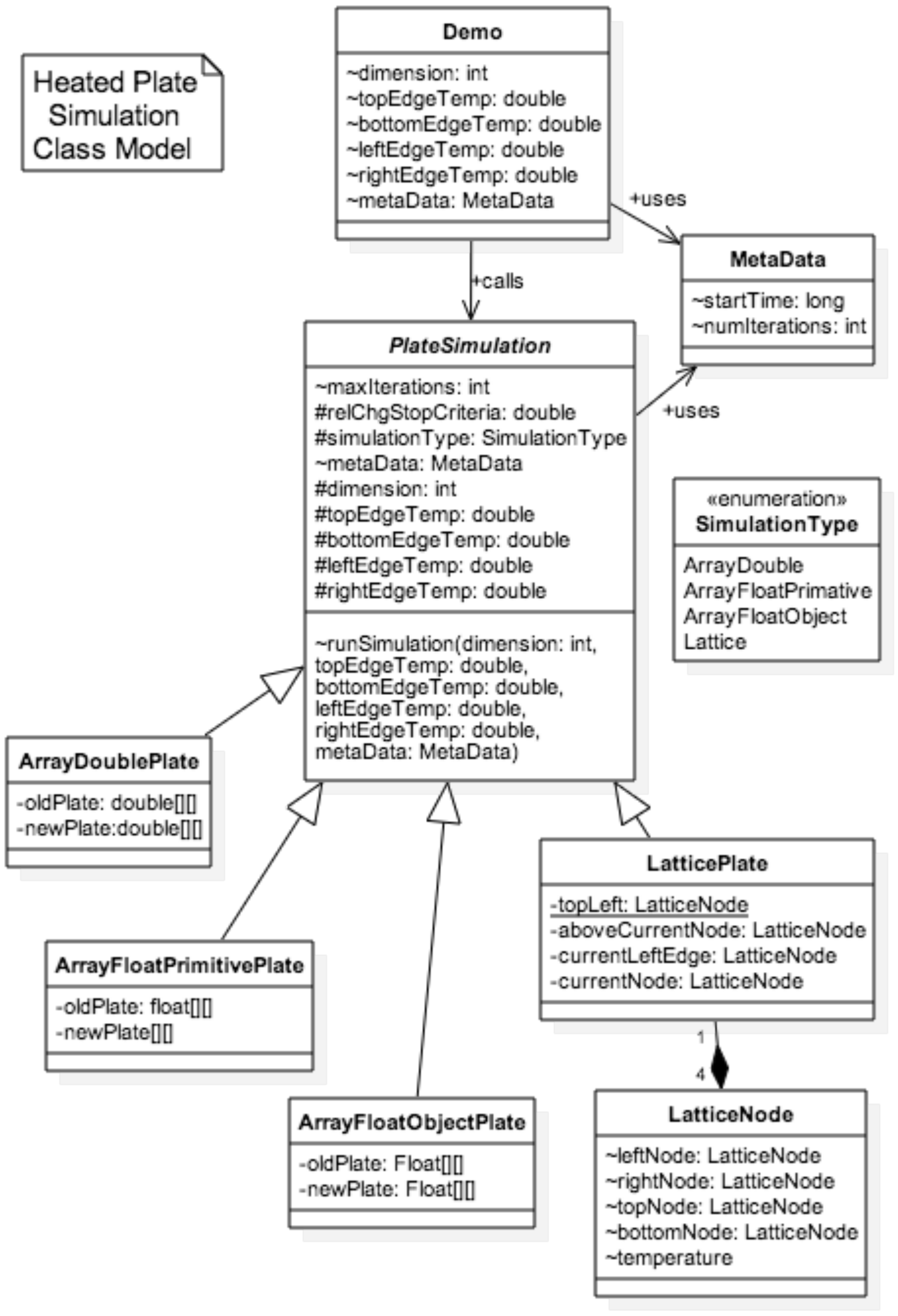
Characteristics / Programs	**Tpdahp**	**Tpfahp**	**Twfahp**	**Tpdohp**	**Gallhp**
Lines of Code	260	265	268	344	889
File Size (bytes)	11,639	9,011	9,382	15,430	137,517
# of Classes	1	1	1	3	8
Average # of methods (per class)	7	7	7	2.33	3.75
Average # of attributes (per class)	8	8	8	6	7.13
# of inter-class dependencies	0	0	0	2	5

Static Description (dependency table)

Gallhp.Demo has eight main classes. The main Demo class creates and manages the GUI which collects all the data from the user and which calls the classes that actually perform the simulations. PlateSimulation is a conceptual class that is a generalization for all of the simulation subclasses and provides the common interface. ArrayDoublePlate, ArrayFloatPrimitivePlate, ArrayFloatObjectPlate, and LatticePlate all perform the simulation described by their titles and print the results. LatticePlate uses LatticeNode to keep track of each node in the lattice. MetaData is used to hold information to facilitate gathering runtime and memory consumption data from program execution.

Source Class	**Dependency Type**	**Target Class**
Demo	instantiates	MetaData
Demo	calls	ArrayDoublePlate
Demo	calls	ArrayFloatPrimitivePlate
Demo	calls	ArrayFloatObjectPlate
Demo	calls	LatticePlate
ArrayDoublePlate	uses as a parameter	MetaData
ArrayFloatPrimitivePlate	uses as a parameter	MetaData
ArrayFloatObjectPlate	uses as a parameter	MetaData
LatticePlate	uses as a parameter	MetaData
LatticePlate	creates	LatticeNode

UML Class Model design diagram



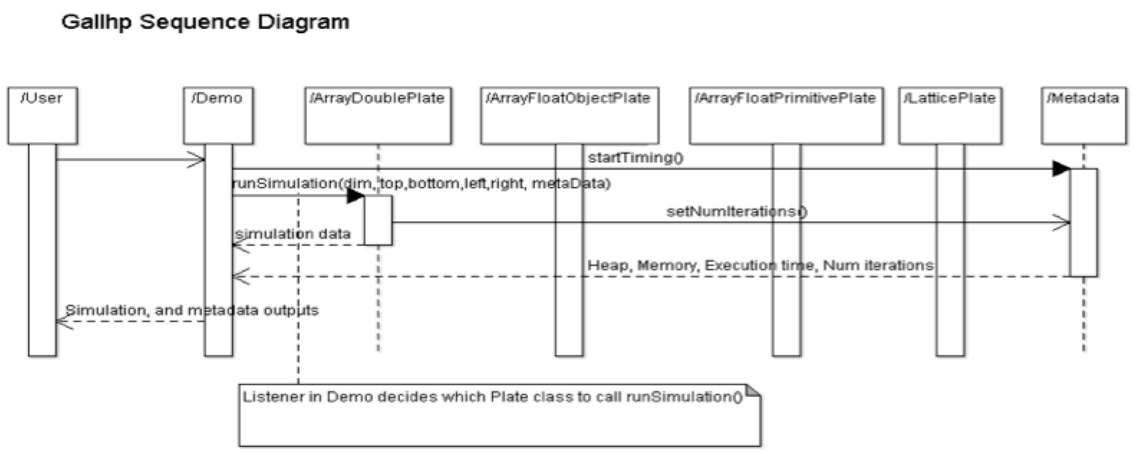
Dynamic Description

When the user runs the Demo GUI, selects the desired variant, enters the grid and edge values, and clicks the run button, Demo class first instantiates the Metadata class and stores the current time. Demo then calls the runSimulation method of the selected variant passing in the variables the user entered in the text fields along with the Metadata instance. Each variant class: ArrayFloatObjectPlate, ArrayDoublePlate, ArrayFloatPrimitivePlate, and LatticePlate have runSimulation, diffuse, swap, and print methods. All variants except Lattice plate also contain an initialize method. The Lattice constructor does the initialization for the LatticePlate class.

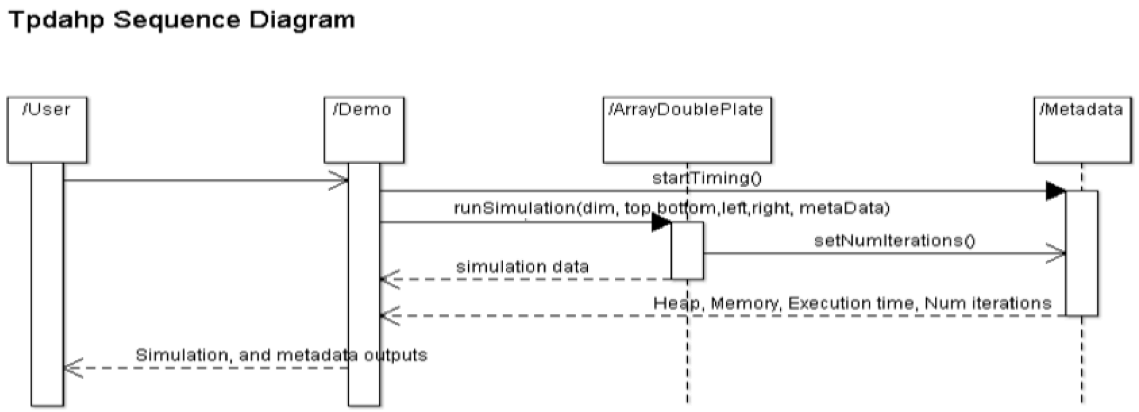
Inside of the runSimulation method, the plates are initialized per user input and diffused until either 100 iterations have occurred or the relative change between iterations is less than or equal to 1.0. The number of iterations are captured and massed back to the Metadata instance and the table printout of temperature is returned to the Demo class in the form of a String.

The Demo class then calls the print method within the Metadata instance which returns: amount of free and used memory in bytes and megabytes, total available memory in bytes and megabytes, total execution time, and number of iterations performed. The returns from runSimulation and print methods are displayed onto a textarea component.

Sequence Diagram



Gallhp Sequence Diagram
/User
/Demo
/ArrayDoublePlate
/ArrayFloatObjectPlate
/ArrayFloatPrimitivePlate
/LatticePlate
/Metadata
startTiming()
runSimulation(dim, top,bottom,left,right, metaData)
setNumIterations()
simulation data
Heap, Memory, Execution time, Num iterations
Simulation, and metadata outputs
Listener in Demo decides which Plate class to call runSimulation()



Tpdahp Sequence Diagram
/User
/Demo
/ArrayDoublePlate
/Metadata
startTiming()
runSimulation(dim, top,bottom,left,right, metaData)
setNumIterations()
simulation data
Heap, Memory, Execution time, Num iterations
Simulation, and metadata outputs

Low Level Design Considerations

The programs used to simulate heat diffusion for the most part utilized similar algorithms and data structures. In terms of data structures, a two dimensional array was utilized to play the part of a metal plate cut up into equal sized cells in three of the programs. For the fourth program, complex objects were utilized to represent a lattice of nodes, each interlocked with its direct four neighbors. Both approaches have their merits and detriments. The array approach focuses on simplicity, with no direct links from one node to another and easy look up via the arrays indexing system. The lattice of complex objects approach requires a more intricate look up system, in a sense similar to a linked list, or a quadruply linked list, that can be referenced via a root node; each node knows about its direct neighbors, and the tails of the lattice are simply the edges of the plate.

In terms of algorithms, this can be analyzed on a per function basis. Summarily, two plates are represented via either 2d arrays, or lattices of node objects. One would serve as the "old" or original plate, and the other as the "new" or most up to date plate. With these two containers the diffusion logic can be applied by setting updated values in the "new" array based on its original value in the "old" with its neighbors. This is done via the diffuse method:

- **Diffuse:** Iterates over all the points in the "old" container, and updates each corresponding cell of the "new" container by calculating the average of its neighboring cells; if a cell is on the edge of the plate, then it uses the static temperature of that edge to perform the average.

In order to be able to run the diffuse method over and over again, a swap, or essentially copy of the "new" container would have to be done over to the "old". For this very purpose the swapping method is used:

- **Swap/Copy:** Iterates over all cells in both the old and new plate, and then updates the “old” plate with the corresponding values of “new”.

Eventually the heat diffusion will stabilize and the temperature of each cell will be the same; an ending condition is reached when calling diffuse has no change, insignificant change occurred, or the safety cap of 100 iterations has been reached.

Non-functional Requirements

Requirement	Description	Decision
Performance	Program execution shall be optimized to produce the results without noticeable delay to the user.	Limiting the grid size to 25 and maxIterations in the diffusion simulation to 100 produces optimal runtimes without sacrificing precision or causing memory drain
Precision	Computational precision shall be optimized to provide accurate results.	Limiting the relChgStopCriteria variable to 1.0 provides a high level of accuracy without causing memory drain or significantly impacting runtimes
Evolvability	The program shall allow for future enhancements.	The design of Gallhp.Demo provides simple and clear organization of the different classes lending to future enhancements in which different implementation solutions can be achieved. The public class MetaData provides the runtime and heap utilization statistics for all of the underlying classes.
Reliability	Errors shall be handled efficiently and the program shall perform properly and without failure.	To ensure reliability and correctness, the use of non-standard Java libraries shall be excluded. In addition, setting maxIterations and limiting the user input on grid size prevents the program from functioning improperly and/or potentially crashing.
Usability	The graphical interface shall be intuitive and responsive; the resultant display following program execution shall be a clear representation of the simulation of diffusion.	The resultant display of the simulation of diffusion produces the temperature values in the requested grid size with both horizontal and vertical scroll bar features for viewing ease.
Environmental	The program must be able to compile and run on Java 1.6 and on Ubuntu 3.11.0-17 64-bit architecture.	In addition to satisfying these exact requirements, the program has been tested and successfully compiles and runs on other platforms including mac OS X 64-bit and Windows 7 64-bit.

GUI Design

The GUI is fairly simple and straightforward. A drop down list in the top left corner allows the user to select which variant they would like to run. The dropdown list was chosen over other options such as: checkboxes, radio buttons, and written text to simply validation and provide users a defined list of options. Additionally, there are five text fields that accept user input for dimension and edge values. The text fields are validated and prompt the user of invalid entries. The GUI was implemented full screen, with the text area (area showing the results of the simulation) taking up approximately 85% of the screen, in order to provide the user ample space to view larger grid dimensions. It is fairly common to see buttons at the bottom of applications such as: Next, Continue, Start, etc. Thus, a "Run" button was added at the bottom. The GUI interacts directly with four classes. The listener attached to the drop down list determines the selected variant and stores the selection. When the run button is pressed, its listener uses the stored variant selection to call one of the four corresponding classes' run simulation method passing in the variable the user entered in the text fields.

Reflection

The resulting program produced by Team 15 fully satisfies the criteria detailed in the assignment description with respect to implementation as well as conforms to proper Java coding conventions. The decision to limit the user input with respect to grid dimension size as well as explicitly setting the values for the maximum number of iterations in the diffusion simulation and the relative change criteria produced the overall desired results without sacrificing precision or performance. Not limiting the grid dimension value nor setting the maximum number of iterations produced significantly longer runtimes, especially in the Lattice implementation; whereas, not setting the relative change criteria consumed more memory and eventually caused program crashes due to OutOfMemory errors. This occurred in conjunction with a significantly large grid dimension. The memory consumption of the program could be altered by varying the heap size during program execution, however it was decided to use the default values provided by the Virtual Machine and the other OS platforms the program was executed on. The use of the MetaData class to calculate the runtime and memory statistics further streamlined the program and prevented code duplication. The use of the garbage collector in the MetaData class was removed as it produced inconsistent results. An opportunity for improvement existed with respect to testing and bug handling, as testing was performed manually without the use of tools and a defect log was not opened. However, GitHub was used as a software repository and change was properly documented. Despite the time constraints and working in different time zones, the team worked very well together, had a high productivity level and delivered a quality design and program.