Assignment 1: Complexity and Sorting

Description

In this assignment, you'll work in groups to create a sorting algorithm for geometric shapes. You'll also implement and perform experimental analysis (benchmarking) on six sorting algorithms.

Warning: This project is significantly larger and more complex than the assignments in your previous semesters, do NOT wait until the last week of the deadline to start!

Equipment and Materials

For this assignment, you will need:

- Eclipse and Java 8

Instructions

This assignment consists of three parts, to be completed **outside** of class time. See the course outline and Brightspace for due dates. Complete this assignment with your assigned group.

Part A: Create a Sorting Application (90 marks)

Using the specifications below, create an application to sort any object according to a specific property. You'll then implement and perform experimental analysis (benchmarking) on six sorting algorithms:

- Bubble
- Insertion
- Selection
- Merge
- Quick
- A sorting algorithm of your choice

You must research the sorting algorithm of your choice and ensure that it's **significantly different** from the other five. Include a detailed description of this sort's algorithm and a complexity analysis outlined in the *Submission Deliverables* section of this document.

Part B: Complete an Application Evaluation as a Group (5 marks)

After completing your sorting application, check your work against the provided marking criteria. Your instructor will refer to your group's self-evaluation when grading the assignment and will provide further feedback and grade adjustments as needed. Your instructor is responsible for awarding the group's final grade.

1. Open the Marking Criteria document (MarkingCriteria_Assignment1.docx) and save a copy with your group's name.
2. As a group, discuss how well you met each criterion and assign yourselves a mark for each row in the table. You may include a short, point form, explanation for your mark in the Notes column.

   Note: This is an opportunity for you to identify any missing pieces according to the marking criteria, make sure you do this step diligently to avoid losing any marks that is clearly stated in this document.
3. Save this file for submission to Brightspace along with your completed code.

Part C: Complete a Peer Assessment (5 marks)

Each student must also complete a peer assessment of their group members. Your instructor will provide further submission details.

Assignment Specifications

Important: Read the specifications very carefully. If you are uncertain about any of the requirements, discuss it with your instructor.

1. Import the assignment1StartingCode project into Eclipse.
2. Create an abstract class in Java that represents a three-dimensional geometric shape. Using the Strategy pattern discussed in class:
   - Implement the **compareTo()** method of the Comparable interface to compare two shapes by their height, and the **compare()** method of the Comparator interface to compare two shapes by their base area and volume.
   - The compare type will be provided as input from the command line to your program: **h** for height, **v** for volume, and **a** for base area via the -t option.
3. Write a testing program that will read a text file (entered at the command line via the -f option) of random shapes that adds them to an array (not ArrayList).

   Notes:
   - All shapes must be manipulated as elements of the corresponding collection.
   - The first value in the first line of the data file contains the number of shapes in that file.
   - A shape in the file is represented as follows on each line (the values are separated by spaces):
     - One of: Cylinder, Cone, Prism or Pyramid

  - Cylinders and Cones are followed by a double value representing the height and another double value representing radius.

    e.g., Cylinder 9431.453 4450.123 or Cone 674.2435 652.1534

  - Pyramids are followed by a double value representing the height and another double value representing edge length.

    e.g., Pyramid 6247.53 2923.456

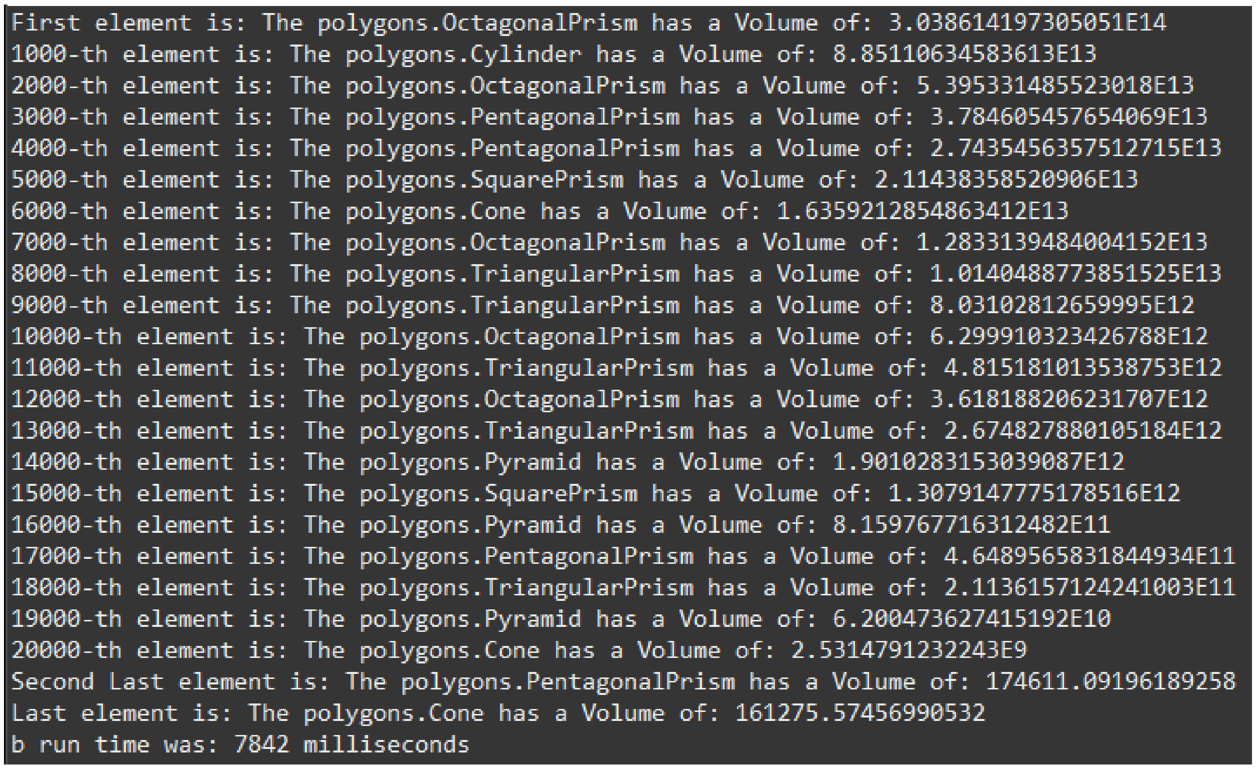
  - Prisms are specified by the type of base polygon (SquarePrism, TrianglarPrism, PentagonalPrism, OctagonalPrism), a double value representing the height and another double value representing edge length.

    e.g., SquarePrism 8945.234 3745.334

- See the formulas at the end of this document for information on how to calculate the base area and volume for each type of shape.

4. Your testing application should then invoke the utility methods to re-arrange the shape objects according to the compare type from the **largest to smallest** (descending order).
5. Your testing program should print the time that it took to sort the collection of objects for each of the six sorting algorithms, including a measurement unit (e.g., milliseconds).

   Note: Have this benchmarking done outside of the sorting algorithms such that the sorting code can be executed without this explicit output.
6. The program should also display to the console – the first sorted value and last sorted value, and every thousandth value in between, for example:



```
First element is: The polygons.OctagonalPrism has a Volume of: 3.038614197305051E14
1000-th element is: The polygons.Cylinder has a Volume of: 8.85110634583613E13
2000-th element is: The polygons.OctagonalPrism has a Volume of: 5.395331485523018E13
3000-th element is: The polygons.PentagonalPrism has a Volume of: 3.784605457654069E13
4000-th element is: The polygons.PentagonalPrism has a Volume of: 2.7435456357512715E13
5000-th element is: The polygons.SquarePrism has a Volume of: 2.11438358520906E13
6000-th element is: The polygons.Cone has a Volume of: 1.6359212854863412E13
7000-th element is: The polygons.OctagonalPrism has a Volume of: 1.2833139484004152E13
8000-th element is: The polygons.TriangularPrism has a Volume of: 1.0140488773851525E13
9000-th element is: The polygons.TriangularPrism has a Volume of: 8.03102812659995E12
10000-th element is: The polygons.OctagonalPrism has a Volume of: 6.299910323426788E12
11000-th element is: The polygons.TriangularPrism has a Volume of: 4.815181013538753E12
12000-th element is: The polygons.OctagonalPrism has a Volume of: 3.618188206231707E12
13000-th element is: The polygons.TriangularPrism has a Volume of: 2.674827880105184E12
14000-th element is: The polygons.Pyramid has a Volume of: 1.9010283153039087E12
15000-th element is: The polygons.SquarePrism has a Volume of: 1.3079147775178516E12
16000-th element is: The polygons.Pyramid has a Volume of: 8.159767716312482E11
17000-th element is: The polygons.PentagonalPrism has a Volume of: 4.6489565831844934E11
18000-th element is: The polygons.TriangularPrism has a Volume of: 2.1136157124241003E11
19000-th element is: The polygons.Pyramid has a Volume of: 6.200473627415192E10
20000-th element is: The polygons.Cone has a Volume of: 2.5314791232243E9
Second Last element is: The polygons.PentagonalPrism has a Volume of: 174611.09196189258
Last element is: The polygons.Cone has a Volume of: 161275.57456990532
b run time was: 7842 milliseconds
```

7. Implement the sorting algorithms as part of a utility package as one class or one class per algorithm. The sorts must be able to accept an array of Comparables as its argument, not the base Shape class or just any generic type T.

   Note: Ensure the sorts isn't dependent on the testing application and that it can be re-used in the future, i.e., your sorting methods can be invoked similar to the Arrays.sort() method for any objects that implements the comparable interface.

8. To run the program, the user should be able enter a command via command line in the following case insensitive format:

   ```
   java -jar Sort.jar -ffile_name -tv -sb
   ```

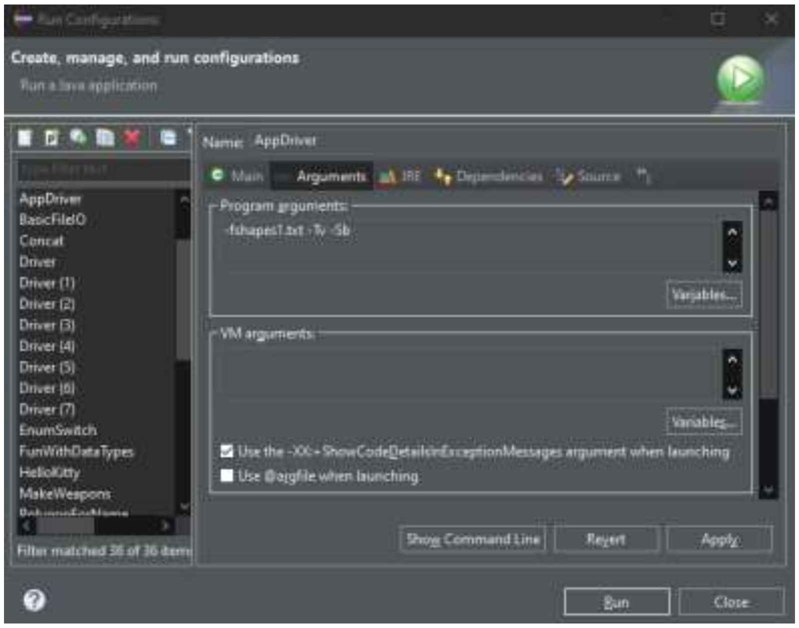
   - -f or -F followed by file_name (the file name and path) with no spaces
   - -t or -T followed by v (volume), h (height) or a (base area) with no spaces
   - -s or -S followed by b (bubble), s (selection), i (insertion), m (merge), q (quick) or z (your choice of sorting algorithm) with no spaces

   The program must be able to handle these command line arguments no matter the order (order insensitive). The following are examples of valid inputs:

   - `java -jar Sort.jar -fshapes1.txt -Tv -Sb`
   - `java -jar Sort.jar -ta -sQ -f"res\shapes1".txt`
   - `java -jar Sort.jar -tH -F"C:\temp\shapes1.txt" –sB`

9. If the user enters an incorrect command line argument according to the rules above, the program should display a meaningful message to help the user in correcting the error.

   You can test this command line in Eclipse using the "Run Configurations" tool under the "Arguments" tab:



Name: AppDriver

AppDriver
BasicFileIO
Concat
Driver
Driver (1)
Driver (2)
Driver (3)
Driver (4)
Driver (5)
Driver (6)
Driver (7)
EnumSwitch
FunWithDataTypes
HelloKitty
MakeWeapons

Filter matched 36 of 36 items

Program arguments:
-fshapes1.txt -Tv -Sb

VM arguments:

☑ Use the -XX:+ShowCodeDetailsInExceptionMessages argument when launching
☐ Use @argfile when launching

10. You will want to test your sorting algorithms starting with the smallest data set (shapes1.txt) to check the results.
11. Make sure your code works with the other test data! (i.e., shapes2.txt and shapes3.txt)

Note: Your instructor will use a different test data file to evaluate your application.

12. Challenge! Try your code out with the BIG data set (shapesBig.txt)! Does it still run correctly? :D

Submission Deliverables

Note: This is a group assessment, so only one submission is required per group. If there are multiple submissions, your instructor will only evaluate the last submission. See Brightspace for the exact due date and time.

Your group's submission should include a zipped folder named as the assignment number and your group number (e.g., A1Group3.zip). The zipped folder should contain the following items:

1) An executable Java Archive file (.jar) for your sorting application called **Sort.jar**.
2) A **readMe.txt** file with instructions on how to install and use your sorting program.
3) The project should have completed javadoc using the "-private" option when generated. Place the output in the doc directory of the project.
4) A folder containing the complete export Eclipse project directory. Refer to the exact instructions of how to export your project from Lab 0.

   Note: Do NOT include any of the test data files in your upload (e.g., shapesBig.txt).
5) A **mySort.txt** file (or any document format, e.g., word, pdf etc.) that provides a detailed description of the sorting algorithm of your choice along with its complexity analysis.
   - Your analysis should include the steps of the algorithm in **pseudocode**, with a **counting analysis** (the number of operations performed in each step).
   - **Note:** Make sure to write this in your own words and NOT copy or rephrase from another source.
6) The completed Marking Criteria document containing your group's evaluation of your application. All group members should be present when completing this document and it acts as your "sign-off" (approval) of the assignment submission.

No late assignments will be accepted.

Formulas

Cylinder

- $base\ area\ = \pi r^2$
- $volume = \pi r^2 h$

Cone

- $base\ area\ = \pi r^2$
- $volume = \frac{1}{3}\pi r^2 h$

Pyramid

- $base\ area\ = s^2$
- $volume = \frac{1}{3}s^2 h$

Prisms

Square Base

- $base\ area\ = s^2$
- $volume = s^2 h$

Equilateral Triangle Base

- $base\ area\ A = \frac{s^2\sqrt{3}}{4}$
- $volume = \mathrm{A}h$

Pentagon Base

- $base\ area\ A = \frac{5s^2 \tan{(54°)}}{4}$
- $volume = \mathrm{A}h$

Octagon Base

- $base\ area\ A = 2\left(1+\sqrt{2}\right)\mathrm{s}^2$
- $volume = \mathrm{A}h$