Task

Fill a collection with several regular shapes (circle, regular triangle, square, regular hexagon).
Determine the smallest bounding box, which contains all the shapes, and its sides parallel with an x or y axis.

Each shape can be represented by its center and side length (or radius), if we assume that one side of the polygons are parallel with x axis, and its nodes lies on or above this side.
Load and create the shapes from a text file. The first line of the file contains the number of the shapes, and each following line contain a shape. The first character will identify the type of the shape, which is followed by the center coordinate and the side length or radius.
Manage the shapes uniformly, so derive them from the same super class.

Description of the task

We can divide the task into 3 stages:
1. Loading shapes from a text file (in my case it is data.txt): First line specifies how many shapes are in collection. Then, each line starts with a character that clarifies the type of shape (‘C’ for circle, ‘T’ for triangle, ‘S’ for square, ‘H’ for hexagon). This will be followed by the x and y coordinates of the shape’s center and either the side length or the radius (for circles).
2. An abstract shape class: Every shape should be derived from the same superclass in order to handle the forms consistently. This will enable the user to handle them consistently, irrespective of their particular nature.
3. Bounding box: In each shape we create method to calculate the minimum and maximum coordinates. To make things simple, we assume that every polygon has a side that is always parallel to the x-axis, and that the vertices of all of them are located on or above this side.

Description of the methods

read(String fileName):
Shape data can be read using this method from a given file. It starts by reading the total number of shapes, and then iteratively reads the size (either radius or side length) and coordinates of each shape (Circle, Square, Triangle, Hexagon). It generates a matching Shape object and adds it to the shapes list based on the type of shape.

report():
A list of every shape kept in the shapes ArrayList is printed by this procedure. It obtains a string representation for each Shape object in the list by iterating through them and calling their toString() method.

getMinMaxBBCoordinates():
The coordinates of the smallest bounding box that can contain every form kept in the shapes ArrayList are determined using this function. It sets the maximum possible double value for the minimum and 0 for maximum x and y values respectiveley. Next, iterating through each shape, it modifies the minimum and maximum values based on the bounding box coordinates that are obtained using the getBBCoordinates() method. Lastly, it writes to the console the coordinates of the smallest bounding box. This helps to comprehend the general spatial scope of the database's shapes.

getBBCoordinates():
Returns Coordinate object which includes minimum and maximum coordinates of the shapes.
Circle:
Min X: This is the furthest left point of the circle, which is calculated by subtracting the radius from the central x-coordinate: x - radius.
Min Y: This is the lowest point of the circle, calculated by subtracting the radius from the central y-coordinate: y - radius.
Max X: This is the furthest right point of the circle, calculated by adding the radius to the central x-coordinate: x + radius.

Max Y: This is the highest point of the circle, calculated by adding the radius to the central y-coordinate: y + radius.

Triangle:

Height: The height of an equilateral triangle is given by the formula: $\frac{\sqrt{3}}{2} \times \text{side}$.

Min X: The triangle's base extends equally on both sides from the center, so the minimum x is $x - \frac{side}{2}$.

Max X: The maximum x is $x + \frac{\text{side}}{2}$.

Min Y: The minimum y (bottom of the triangle) is found by subtracting one-third of the height from the center $y - \frac{\text{height}}{3}$.

Max Y: The maximum y (top of the triangle) is two-thirds of the height above the center $y + \frac{2 \times \text{height}}{3}$.

Hexagon:

Height: The height of a regular hexagon is also $\frac{\sqrt{3}}{2} \times \text{side}$, similar to the triangle.

Min X: The leftmost point is x - height, since the horizontal distance between two opposite vertices is twice the height.

Max X: The rightmost point is x + height.

Min Y: The bottommost point is y - side, as the distance from the center to the flat edge of the hexagon is the length of the side.

Max Y: The topmost point is y + side.

Square:

Min X: Since the square's side is symmetric around the center, the minimum x is $x - \frac{\text{side}}{2}$.

Max X: The maximum x is $x + \frac{\text{side}}{2}$.

Min Y: The minimum y is $y - \frac{\text{side}}{2}$.

Max Y: The maximum y is $y + \frac{\text{side}}{2}$.

<u>toString():</u>

Returns a string that includes the x and y values, as well as the bounding box coordinates (minX, minY, maxX, maxY) obtained from the getBBCoordinates() abstract method. Used for debugging or logging the details of Shape objects.

Class diagram

Testing

Test #1:
AS A user, I WANT TO calculate the bounding box for multiple shapes, GIVEN a collection, WHEN shape data is provided, THEN output correct bounding box dimensions.

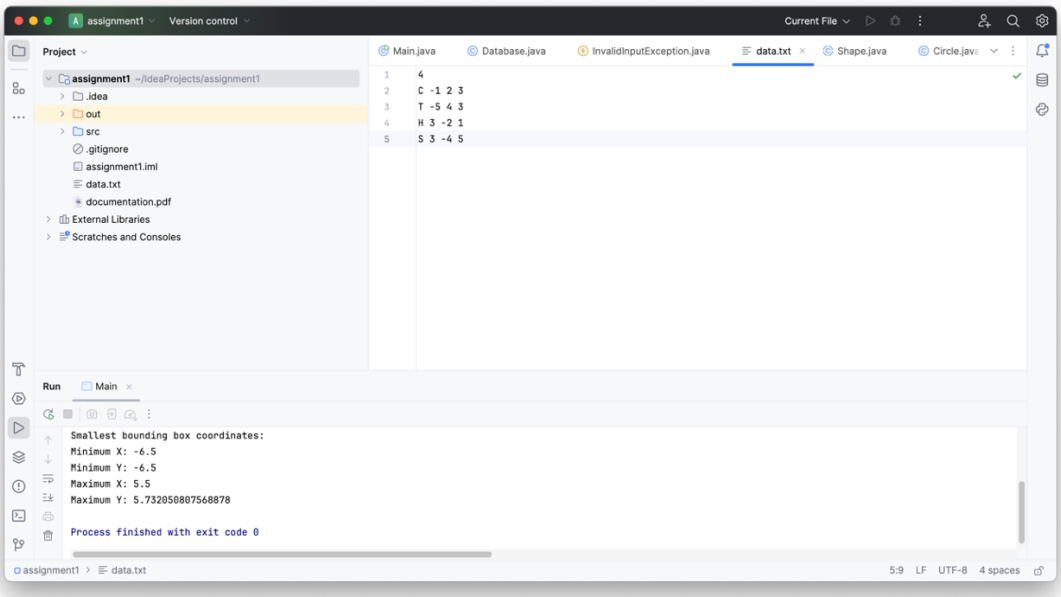
Input:

Shape	x	y	side length (radius)
C	-1	2	3
T	-5	4	3
H	3	-2	1
S	3	-4	5

Output:

minX	minY	maxX	maxY
-6.5	-6.5	5.5	5.732050807568878

Screenshot:



Test #2:
AS A user, I WANT TO calculate the bounding box for a circle, GIVEN center coordinates and radius, WHEN correct input is provided, THEN output bounding box dimensions.

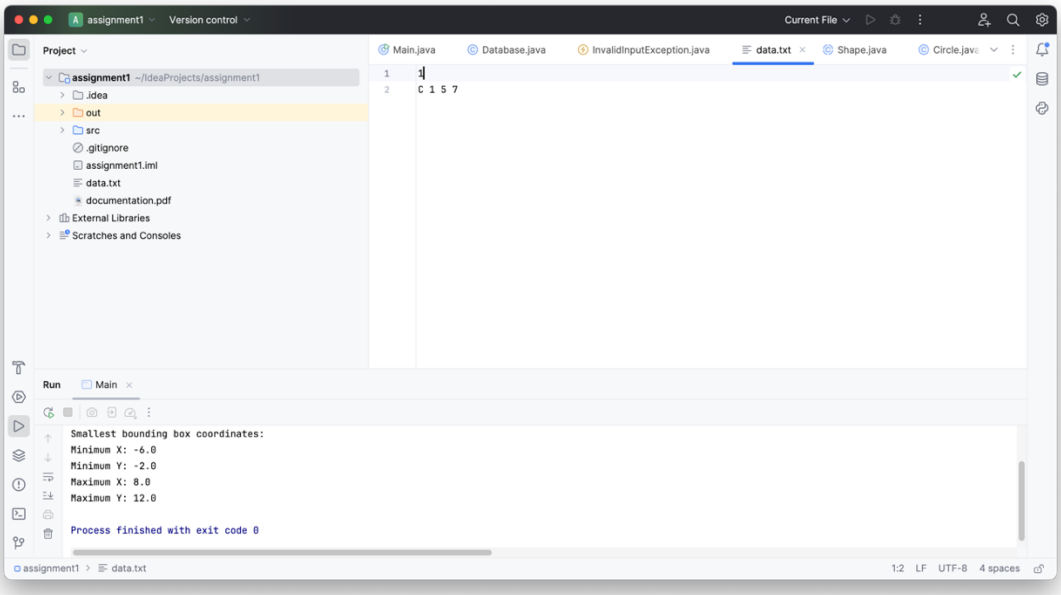
Input:

Shape	x	y	side length (radius)
C	1	5	7

Output:

minX	minY	maxX	maxY
-6.0	-2.0	8.0	12.0

Screenshot:



Test #3:
AS A user, I WANT TO calculate the bounding box for multiple shapes, GIVEN different shapes, WHEN center coordinates and sizes are provided, THEN output smallest bounding box.

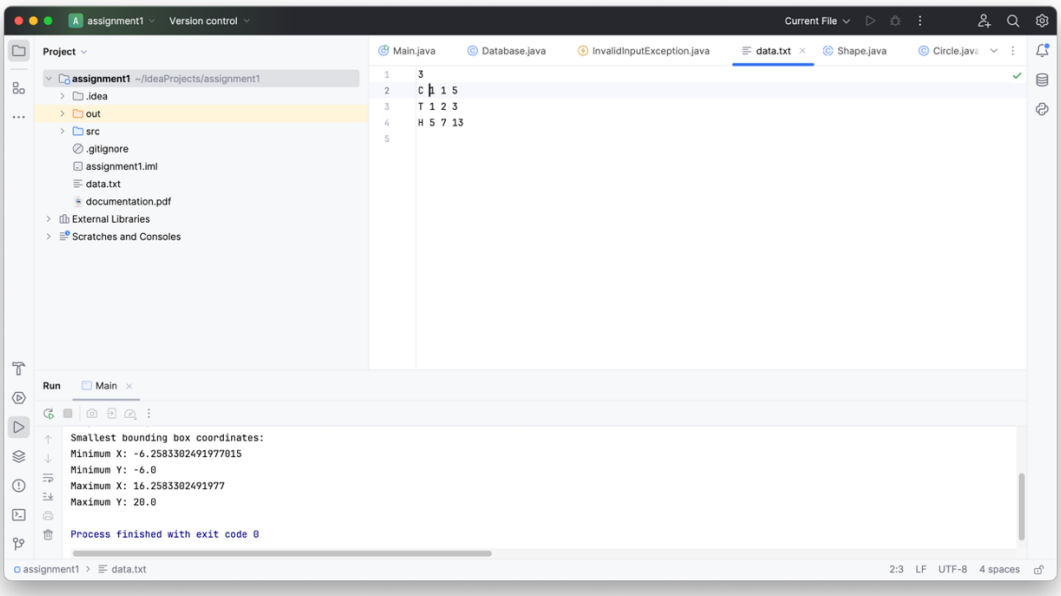
Input:

Shape	**x**	**y**	**side length (radius)**
C	1	1	5
T	1	2	3
H	5	7	13

Output:

minX	**minY**	**maxX**	**maxY**
-6.2583302491977015	-6.0	16.2583302491977	20.0

Screenshot:



Test #4:

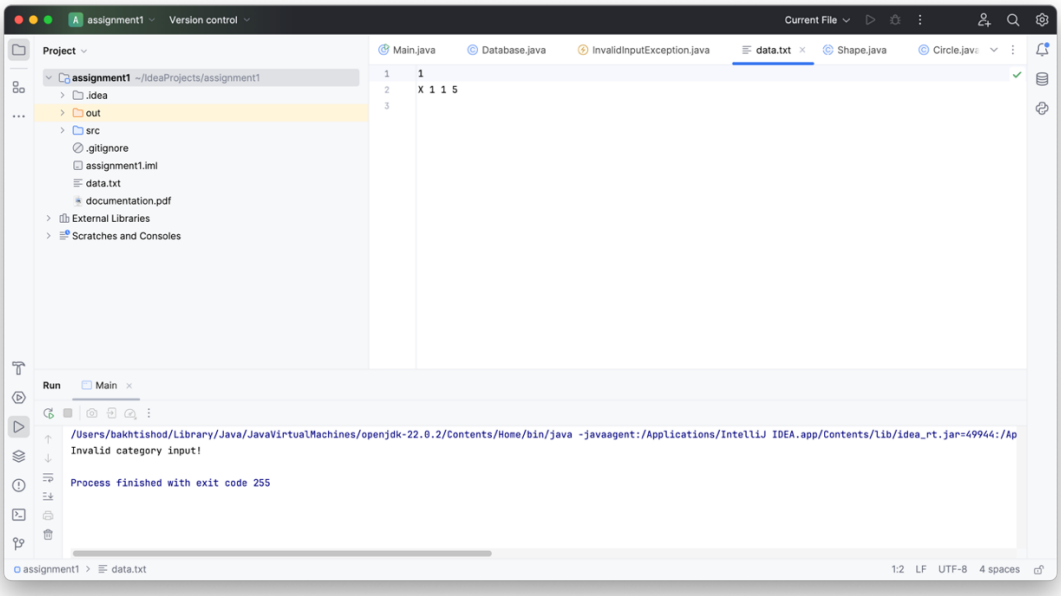
AS A user, I WANT TO receive an error for invalid shape type, GIVEN an unsupported shape, WHEN incorrect type is input, THEN return “Invalid category input!”

Input:

Shape	x	y	side length (radius)
X	1	1	5

Output:
Invalid category input!

Screenshot:



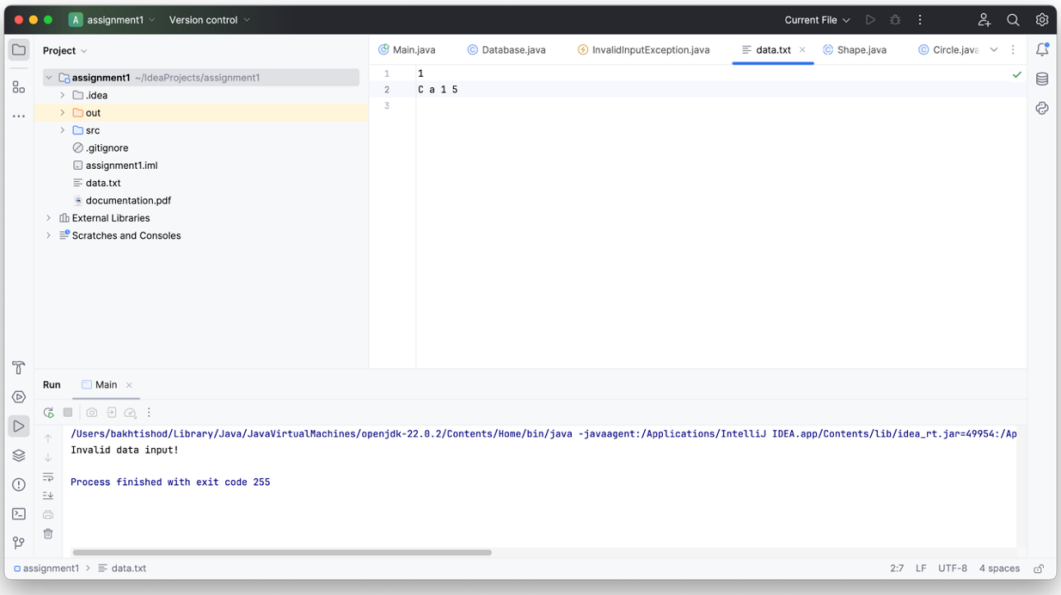
Test #5:
AS A user, I WANT TO receive an error for invalid data input, GIVEN non-numeric values, WHEN invalid characters are input, THEN return “Invalid data input!”

Input:

Shape	x	y	side length (radius)
C	a	1	5

Output:
Invalid data input!

Screenshot:



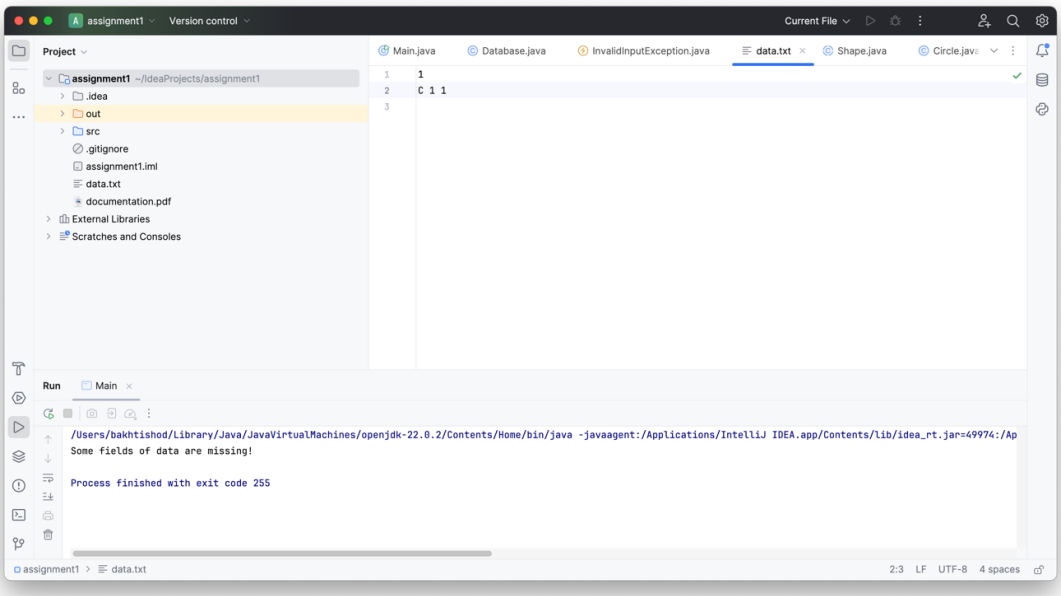
Test #6:
AS A user, I WANT TO receive an error for incomplete data, GIVEN missing fields, WHEN shape details are incomplete, THEN return “Some fields of data are missing!”

Input:

Shape	**x**	**y**	**side length (radius)**
C	1	1	

Output:
Some fields of data are missing!

Screenshot:



Test #7:
AS A user, I WANT TO calculate the bounding box for different shapes, GIVEN correct input, WHEN shapes are provided, THEN output correct bounding box.

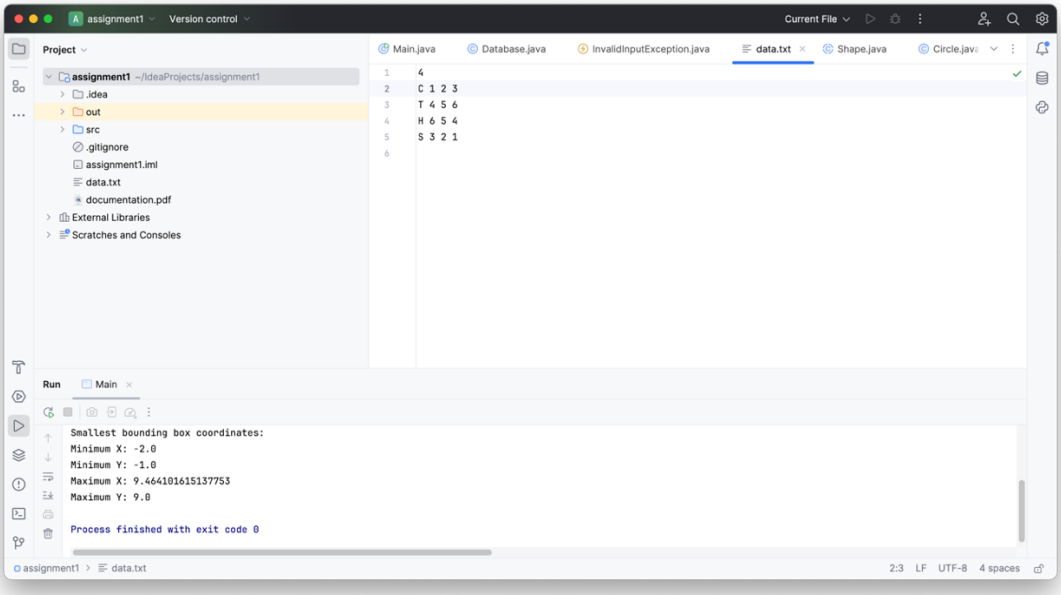
Input:

Shape	x	y	side length (radius)
C	1	2	3
T	4	5	6
H	6	5	4
S	3	2	1

Output:

minX	minY	maxX	maxY
-2.0	-1.0	9.464101615137753	9.0

Screenshot:



Test #8:
AS A user, I WANT TO calculate the bounding box for shapes centered at the same point, GIVEN the same center, WHEN sizes are provided, THEN output bounding box.

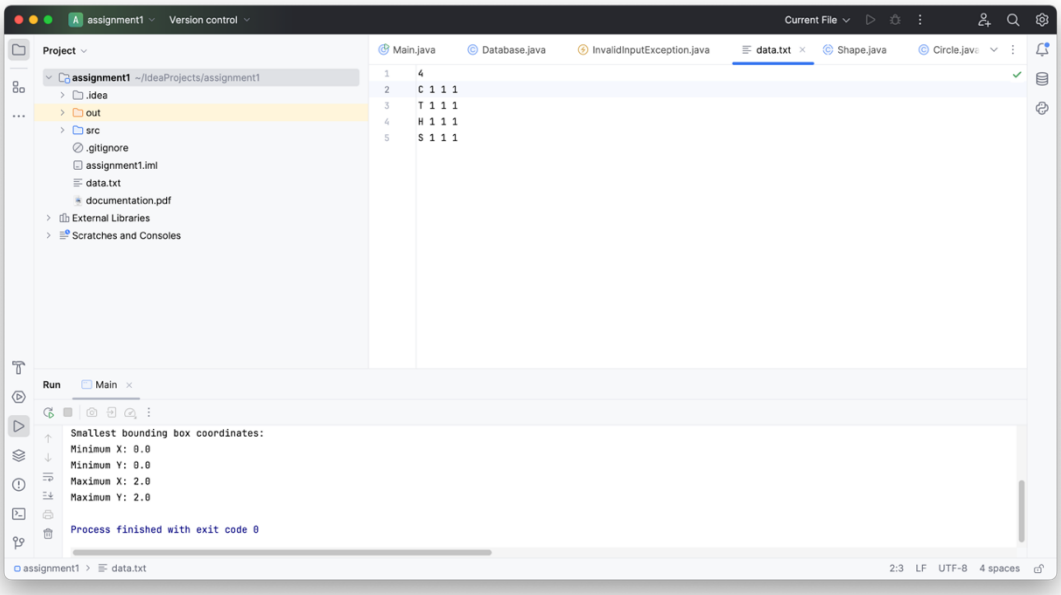
Input:

Shape	x	y	side length (radius)
C	1	1	1
T	1	1	1
H	1	1	1
S	1	1	1

Output:

minX	minY	maxX	maxY
0.0	0.0	2.0	2.0

Screenshot:



Test #9:
AS A user, I WANT TO calculate the bounding box for zero-sized shapes, GIVEN zero size, WHEN input is zero, THEN output all coordinates as zero.

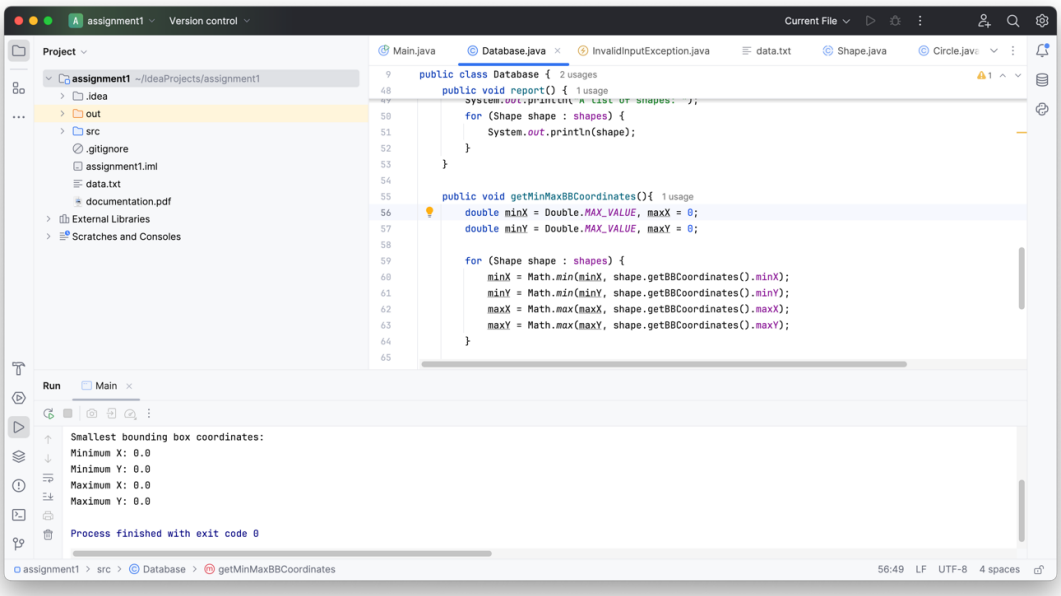
Input:

Shape	x	y	side length (radius)
C	0	0	0
T	0	0	0
H	0	0	0
S	0	0	0

Output:

minX	minY	maxX	maxY
0.0	0.0	0.0	0.0

Screenshot:



Test #10:

AS A user, I WANT TO calculate the bounding box for large shapes, GIVEN large shapes, WHEN large coordinates and sizes are input, THEN output correct bounding box.

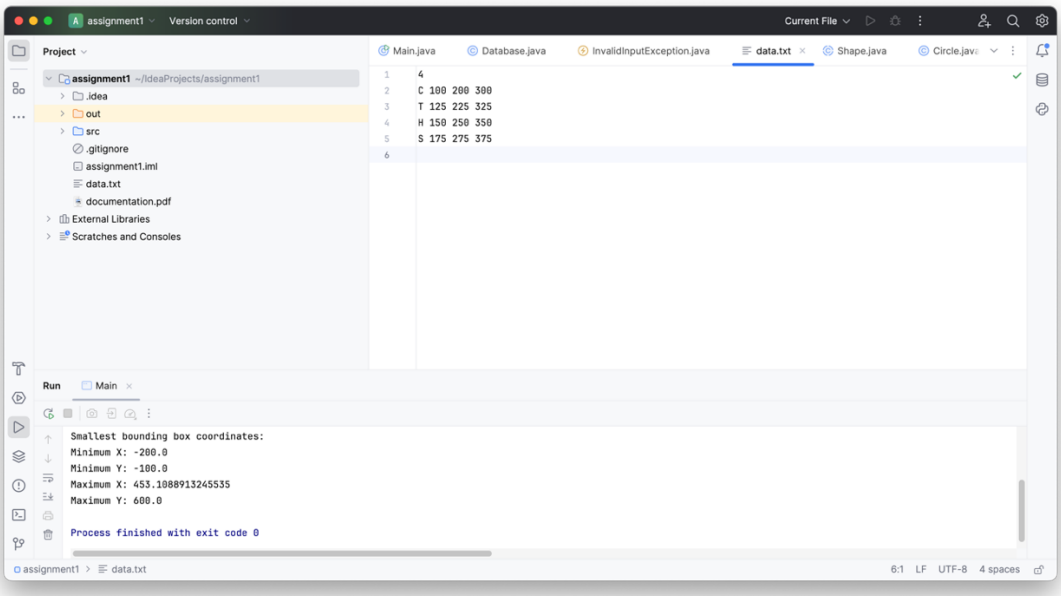
Input:

Shape	x	y	side length (radius)
C	100	200	300
T	125	225	325
H	150	250	350
S	175	275	375

Output:

minX	minY	maxX	maxY
-200.0	-100.0	453.1088913245535	600.0

Screenshot:



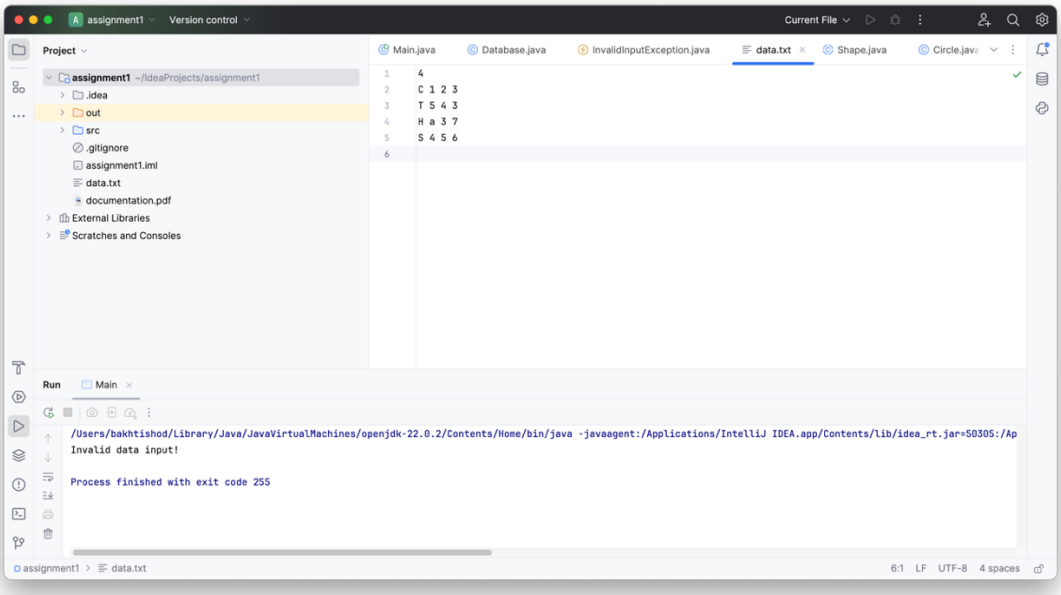
Test #11:
AS A user, I WANT TO receive an error for invalid data, GIVEN letters in the coordinates, WHEN non-numeric data is input, THEN return “Invalid data input!”

Input:

Shape	x	y	side length (radius)
C	1	2	3
T	5	4	3
H	a	3	7
S	4	5	6

Output:
Invalid data input!

Screenshot:



Test #12:
AS A user, I WANT TO calculate the bounding box for shapes with large coordinates, GIVEN large center and side values, WHEN large data is input, THEN output bounding box.

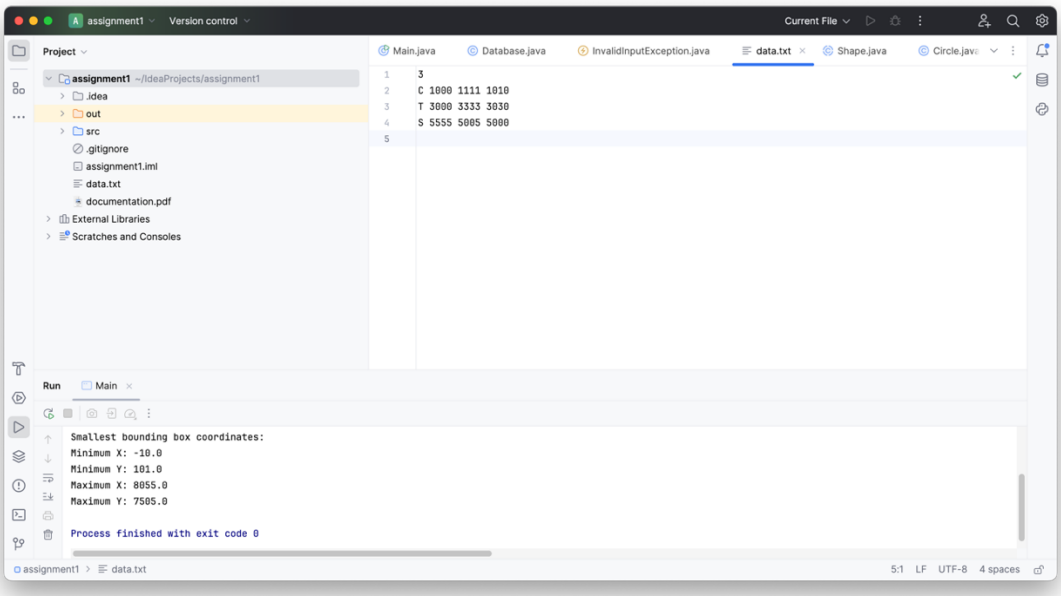
Input:

Shape	**x**	**y**	**side length (radius)**
C	1000	1111	1010
T	3000	3333	3030
S	5555	5005	5000

Output:

minX	**minY**	**maxX**	**maxY**
-10.0	101.0	8055.0	7505.0

Screenshot:



Test #13:

AS A user, I WANT TO calculate the bounding box for a mix of shapes, GIVEN shapes of varying sizes, WHEN data is provided, THEN output the smallest bounding box.

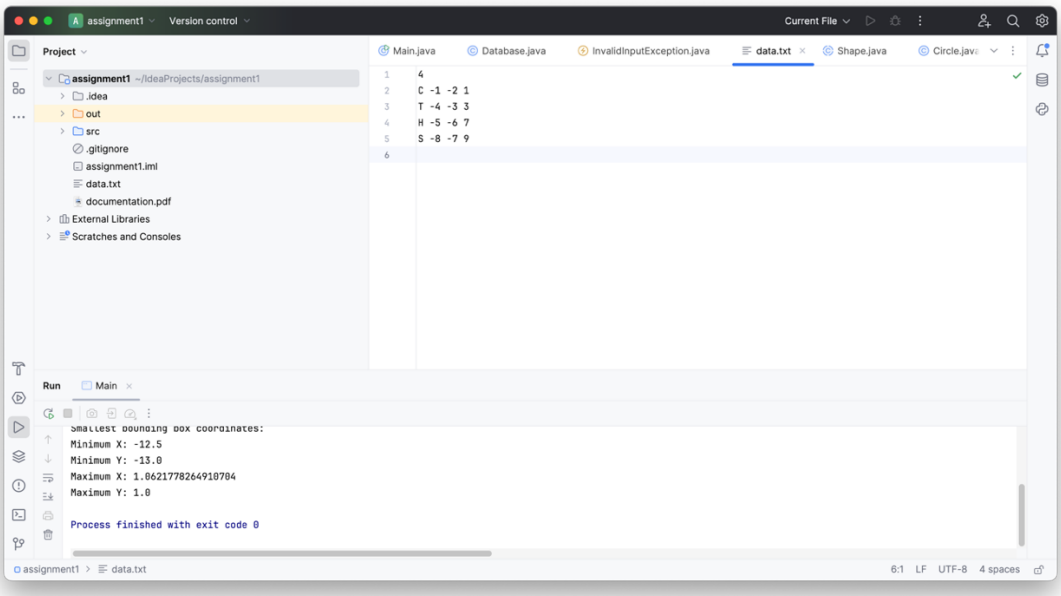
Input:

Shape	x	y	side length (radius)
C	-1	2	3
T	-5	4	3
H	3	-2	1
S	3	-4	5

Output:

minX	minY	maxX	maxY
-12.5	-13.0	1.0621778264910704	1.0

Screenshot:



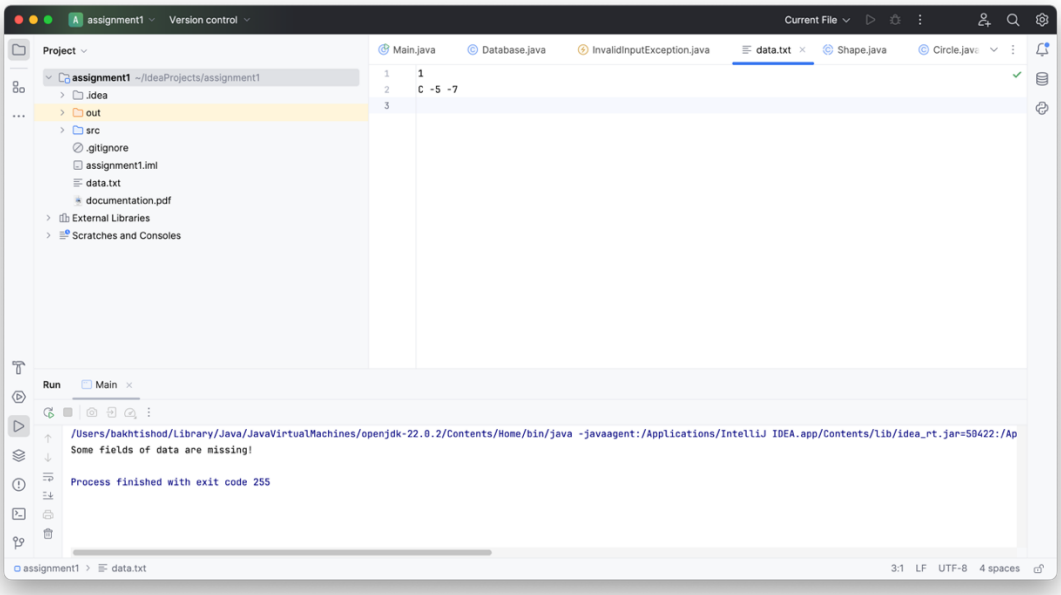
Test #14:
AS A user, I WANT TO receive an error for missing shape data, GIVEN a circle with missing size, WHEN incomplete data is input, THEN return “Some fields of data are missing!”

Input:

Shape	x	y	side length (radius)
C	-5	-7	

Output:
Some fields of data are missing!

Screenshot:



Test #15:
AS A user, I WANT TO receive an error for negative size, GIVEN a shape with negative values, WHEN negative size is input, THEN return “Side (or radius) cannot be negative!”

Input:

Shape	**x**	**y**	**side length (radius)**
C	-5	-7	-9

Output:
Side (or radius) cannot be negative!

Screenshot: