**Class Point2D:**

1. **Member Variables:**

**a. \_x:** A private double variable that represents the x-coordinate of the 2D point.

**b. \_y:** A private double variable that represents the y-coordinate of the 2D point.

1. **Constructors:**

**a. Point2D():** This is a default constructor with no parameters. It initializes the **\_x** and **\_y** variables with default double values (0.0). This constructor is not commonly used, as it creates a point at the origin (0,0).

**b. Point2D(double x, double y):** This is a parameterized constructor that takes two double values **x** and **y** as input and initializes the **\_x** and **\_y** variables with these input values. This constructor is commonly used to create a **Point2D** object with specific x and y coordinates.

1. **Properties:**

**a. X:** This is a public property that allows getting and setting the **\_x** value. It provides access to the x-coordinate of the 2D point. When you read the property (**get**), it returns the current value of **\_x**, and when you set the property (**set**), it updates the value of **\_x** with the provided input value.

**b. Y:** This is a public property that allows getting and setting the **\_y** value. It provides access to the y-coordinate of the 2D point. When you read the property (**get**), it returns the current value of **\_y**, and when you set the property (**set**), it updates the value of **\_y** with the provided input value.

**Class Point:**

1. **Member Variables:**

**a. \_x:** A private double variable that represents the x-coordinate of the 3D point.

**b. \_y:** A private double variable that represents the y-coordinate of the 3D point.

**c. \_z:** A private double variable that represents the z-coordinate of the 3D point.

1. **Constructors:**

**a. Point():** This is a default constructor with no parameters. It initializes the **\_x**, **\_y**, and **\_z** variables with default double values (0.0). This constructor is not commonly used, as it creates a point at the origin (0,0,0).

**b. Point(double x, double y, double z):** This is a parameterized constructor that takes three double values **x**, **y**, and **z** as input and initializes the **\_x**, **\_y**, and **\_z** variables with these input values. This constructor is commonly used to create a **Point** object with specific x, y, and z coordinates.

1. **Properties:**

**a. X:** This is a public property that allows getting and setting the **\_x** value. It provides access to the x-coordinate of the 3D point. When you read the property (**get**), it returns the current value of **\_x**, and when you set the property (**set**), it updates the value of **\_x** with the provided input value.

**b. Y:** This is a public property that allows getting and setting the **\_y** value. It provides access to the y-coordinate of the 3D point. When you read the property (**get**), it returns the current value of **\_y**, and when you set the property (**set**), it updates the value of **\_y** with the provided input value.

**c. Z:** This is a public property that allows getting and setting the **\_z** value. It provides access to the z-coordinate of the 3D point. When you read the property (**get**), it returns the current value of **\_z**, and when you set the property (**set**), it updates the value of **\_z** with the provided input value.

**Class Curve3D:**

1. **Member Variables:**

a. **\_curvePt**: A private List of **Point** objects. This list stores the points that define the 3D curve.

1. **Constructor:**

a. **Curve3D()**: This is the default constructor with no parameters. It initializes the **\_curvePt** list as an empty list, ready to store **Point** objects.

1. **Properties:**

a. **SectionPts**: This is a public property that allows getting the list of points that make up the 3D curve. When you read the property (**get**), it returns the reference to the **\_curvePt** list, allowing you to access the points that define the curve.

1. **Public Methods:**

a. **Addpoints(Point pt2D)**: This method allows adding a new **Point** object to the **\_curvePt** list, effectively extending the 3D curve. It takes a **Point** object (**pt2D**) as input and appends it to the list.

**Class Mapping:**

1. **Member Variables:**

a. **\_indexvalue**: A private integer variable used to store an index value or some identifier.

b. **\_dvalue**: A private double variable used to store a numerical value associated with the mapping.

c. **\_tankname**: A private string variable used to store the name or identifier of a tank related to the mapping.

1. **Constructors:**

a. **Mapping()**: This is the default constructor with no parameters. It initializes **\_indexvalue** and **\_dvalue** with their default values (0 for integers, 0.0 for doubles) and leaves **\_tankname** as null (since strings default to null).

b. **Mapping(int inDexValue, double values)**: This is a parameterized constructor that takes an integer **inDexValue** and a double **values** as input. It initializes **\_indexvalue** with the provided **inDexValue** and **\_dvalue** with the provided **values**. The **\_tankname** remains uninitialized and will be set separately using the property.

1. **Properties:**

a. **IndexValue**: This is a public property that allows getting and setting the **\_indexvalue**. It provides access to the index value or identifier stored in the **Mapping** object. When you read the property (**get**), it returns the current value of **\_indexvalue**, and when you set the property (**set**), it updates the value of **\_indexvalue** with the provided input value.

b. **Xvalues**: This is a public property that allows getting and setting the **\_dvalue**. It provides access to the numerical value stored in the **Mapping** object. When you read the property (**get**), it returns the current value of **\_dvalue**, and when you set the property (**set**), it updates the value of **\_dvalue** with the provided input value.

c. **TankName**: This is a public property that allows getting and setting the **\_tankname**. It provides access to the tank name or identifier stored in the **Mapping** object. When you read the property (**get**), it returns the current value of **\_tankname**, and when you set the property (**set**), it updates the value of **\_tankname** with the provided input value.

**Class Matrix:**

1. **Member Variables:**

a. **\_row**: A private integer variable that stores the number of rows in the matrix.

b. **\_column**: A private integer variable that stores the number of columns in the matrix.

c. **\_elements**: A private two-dimensional array of doubles (**double[,]**) that stores the matrix elements.

1. **Constructors:**

a. **Matrix(int row, int column, double[,] elements)**: This constructor is used to create a **Matrix** object with specified dimensions and elements. It takes the number of rows (**row**), the number of columns (**column**), and a 2D array of **double** (**elements**) representing the matrix elements. It initializes **\_row**, **\_column**, and **\_elements** with the provided values.

b. **Matrix(double[,] elements)**: This constructor is used to create a **Matrix** object from an existing 2D array of **double** (**elements**). It determines the dimensions of the matrix from the input array and initializes **\_row**, **\_column**, and **\_elements** accordingly.

c. **Matrix()**: This default constructor is used to create an empty **Matrix** object. It initializes **\_row** and **\_column** with their default integer values (0) and creates an empty **\_elements** array.

1. **Properties:**

a. **Row**: This is a public property that allows getting the number of rows in the matrix. b. **Column**: This is a public property that allows getting the number of columns in the matrix.

c. **Element**: This is a public property that allows getting the 2D array of **double** representing the matrix elements.

1. **Public Methods:**

a. **Matmul(Matrix m1, Matrix m2)**: This method performs matrix multiplication between two matrices **m1** and **m2** and returns the result as a new **Matrix** object. It checks if matrix multiplication is possible (the number of columns of **m1** is equal to the number of rows of **m2**) and throws a **MatrixException** if it is not possible.

b. **TransPose(Matrix m)**: This method returns the transpose of the input matrix **m** as a new **Matrix** object.

c. **Inverse(Matrix matToInverse)**: This method calculates the inverse of the input matrix **matToInverse** and returns the result as a new **Matrix** object. It first checks if the matrix is square (the number of rows is equal to the number of columns) and throws a **MatrixException** if it is not square or if the matrix is singular (determinant is zero).

1. **Private Methods:**

a. **SwapWithFirstNonzeroHeadingRow(Matrix m, int rowindex, ref int FirstNonZeroRow)**: This private method is used internally to swap the current row with the first non-zero heading row below it. It is used in the matrix inversion process to ensure that the diagonal element is non-zero.

b. **SwapBetweenTwoRows(Matrix m, int rowindex, int changewithRow)**: This private method is used internally to swap two rows of the matrix. It is used during the matrix inversion process to perform row operations.

1. **Nested Class:**

a. **MatrixException**: This is a private nested class that inherits from **Exception**. It is used to throw custom exceptions related to matrix operations.

**Class Section:**

1. Member Variables:
   1. **\_sectionNo**: An integer variable representing the section number.
   2. **\_xlocation**: A double variable representing the X-coordinate location of the section.
   3. **\_secdraftList**: A list of doubles representing the draft values (water depth) at different points of the section.
   4. **\_secCurve**: An instance of the "SectionCurve" class, which contains data related to the hull section's curve.
   5. **\_secArea**: A list of doubles representing the cross-sectional area of the hull section at different points.
   6. **\_secTmom**: A list of doubles representing the transverse (T) moments of the hull section at different points.
   7. **\_secVmom**: A list of doubles representing the vertical (V) moments of the hull section at different points.
   8. **\_secXmom**: A list of Point2D instances representing the X moments of the hull section at different points.
   9. **\_vcb**: A double variable representing the VCB (Vertical Center of Buoyancy) of the hull section.
   10. **\_secCurveLength**: A double variable representing the length of the hull section's curve.
   11. **\_secVcb**: A list of doubles representing the VCB values at different points of the hull section.
2. Constructor: The class has a parameterless constructor that initializes all member variables.
3. Properties:
   1. **SectionNo**: A property to get or set the section number.
   2. **Xlocation**: A property to get or set the X-coordinate location of the section.
   3. **SecCurve**: A property to get or set the instance of the "SectionCurve" class that contains data about the hull section's curve.
   4. **SecTMoment**: A read-only property to get the list of transverse (T) moments.
   5. **SecVMoment**: A read-only property to get the list of vertical (V) moments.
   6. **SecXMoment**: A read-only property to get the list of X moments.
   7. **SecArea**: A read-only property to get the list of cross-sectional areas.
   8. **SecDraftList**: A read-only property to get the list of draft values.
   9. **SecCurveLength**: A property to get or set the length of the hull section's curve.
   10. **VCb**: A property to get or set the VCB (Vertical Center of Buoyancy) of the hull section.
   11. **SecVcb**: A read-only property to get the list of VCB values at different points of the hull section.
4. Public Methods:
   1. **AddArea**: Adds a cross-sectional area value to the **\_secArea** list.
   2. **Adddraft**: Adds a draft value to the **\_secdraftList** list.
   3. **AddSecVCb**: Adds a VCB value to the **\_secVcb** list.
   4. **AddSecTMom**: Adds a transverse moment value to the **\_secTmom** list.
   5. **AddSecVMom**: Adds a vertical moment value to the **\_secVmom** list.
   6. **AddSecXMom**: Adds an X moment value (represented as a Point2D instance) to the **\_secXmom** list.

**Class SectionCurve:**

1. Member Variables:
   1. **\_xloc**: A double variable representing the X-coordinate location of the curve.
   2. **\_point2Dlist**: A list of Point2D instances representing the points (coordinates) of the curve.
2. Constructor: The class has a constructor that initializes the **\_xloc** and **\_point2Dlist** member variables.
3. Properties:
   1. **Xloc**: A property to get or set the X-coordinate location of the curve.
   2. **SectionPts**: A read-only property to get the list of Point2D instances representing the points of the curve.
4. Public Methods:
   1. **Addpoints**: Adds a Point2D instance to the **\_point2Dlist**, representing a point on the curve.
   2. **GetSecCurveLength**: Calculates and returns the length of the section curve using the points in the **\_point2Dlist**.
   3. **GetUpdatedSectionCurve**: Generates an updated section curve based on provided input points (**lbPt** and **rtPt**) and other data from a list of **Section** instances (**secList**) and a maximum draft value (**maxDraft**).
   4. **GetSectionalYvalue**: Calculates and returns the Y-coordinate value of the curve at a given **xloc** (X-coordinate location) based on interpolation and provided data.
   5. **getUpdatedDrat**: Returns an updated draft value based on a maximum draft value and the current draft value, ensuring it does not exceed the maximum or fall below zero.
   6. **GetModifiedCurve**: Returns a modified curve that represents a section of the original curve from the provided **ptList** of points to an **endPt** point on the curve.
5. Private Methods:
   1. **getCurveLength**: Calculates and returns the total length of the curve based on the distances between consecutive points in the **\_point2Dlist**.
   2. **getDistance**: Calculates and returns the Euclidean distance between two given **Point2D** instances (**pt1** and **pt2**).

**Class PublicHelper:**

1. **Constructor**: The class has a default constructor **PublicHelper()**.
2. **Public Methods**:
3. **GetCentroid**: This method takes a list of objects (**objList**) as input, where each object must have a property **CG** of type **Point** and a property **Weight** of type **double**. It calculates the centroid of the objects by finding the weighted average of their CG coordinates based on their weights. The method returns an object (**Object**) with the total weight and the centroid (**CG**) coordinates.
4. **GetIntValueforOdd**: This method takes a list of 2D points (**pt2Dlist**) as input. It calculates the integral for odd numbers of points using the trapezoidal rule and returns the result.
5. **GetIntValueforEven**: This method takes a list of 2D points (**ptList**) as input. It calculates the integral for even numbers of points using the trapezoidal rule and returns the result.
6. **Trapizoidal**: This method calculates the integral using the trapezoidal rule for a list of 2D points (**ptList**) and returns the result.
7. **GetShipVol**: This method takes a list of **SectionCurve** objects (**curveList**) as input, where each **SectionCurve** must have a property **SectionPts** of type **List<Point2D>**. It calculates the volume of the ship based on the area of sections and returns the result using the trapezoidal rule.
8. **GetSecArea**: This method calculates the area of a section defined by a list of 2D points (**ptList**). It uses **GetIntValueforEven** for even numbers of points and **GetIntValueforOdd** for odd numbers of points.
9. **GetSecAreaTrapz**: Similar to **GetSecArea**, this method calculates the area of a section using the trapezoidal rule.
10. **Getmoment**: This method calculates the moment of the ship's section based on the provided list of **Section** or **SectionCurve** objects. The moment is calculated about the given point (**stPt** and **endPt**).
11. **Getmoment**: Overloaded method, it calculates the moment of the ship's section based on the provided list of **SectionCurve** objects and the given draft (**d**).
12. **GetMaxMinElementInArray**: This method finds the maximum and minimum elements in a list of doubles (**ptList**) and returns them as a **Point2D** object.
13. **InterpolateData**: This method interpolates data for a given value (**d**) in the list of 2D points (**pt2Dlist**) using either X or Y interpolation based on the **nIndex** value.
14. **SolveLinearEquation**: This method calculates the y-value on a linear equation defined by two points for a given x-value.
15. **SolveLinearEquation**: This overloaded method computes the y-value of a linear equation in slope-intercept form for a given x-value, using the provided slope and y-intercept (c) values.
16. **GetDraftData**: This method calculates draft-wise data based on the provided lists of 2D points (**ls**) and weights (**fn**).
17. **SolveCoeffofQuadraticEquation**: This method calculates the coefficients of a quadratic equation (x = a0 + a1*Y + a2*Y^2) given three points (**pt1**, **pt2**, and **pt3**) using Sridharacharya Formula.
18. **QuadraticSolver**: This method solves the quadratic equation for a given **xvalue** using the provided coefficients (**coeff**).
19. **GetDensePointSet**: This method generates a dense point set based on the input **SectionCurve** object (**curve**). It uses **getDataset** to obtain dense data points between adjacent points of the curve.
20. **InverseSolveLinearEquation**: This function calculates the x-value corresponding to a given y-value on a linear equation defined by two points.
21. **InverseSolveLinearEquation**: This function determines the x-value for a given y-value on a linear equation in slope-intercept form, utilizing the inverse of the slope.
22. **LinearSolver**: This method solves two linear equations represented by **Vector2D** objects (**v1** and **v2**) to find their intersection point (**solutionPt**).
23. **GetCgfromPoint2DList**: This method calculates the centroid (**CG**) of a polygon defined by a list of 2D points (**ptList**). It uses **GetSecAreaTrapz** to calculate the area of the polygon.
24. **GetCgfromPoint2DVcg**: This method is similar to **GetCgfromPoint2DList**, but it calculates the centroid using a different formula based on the centroid of gravity of the polygon.
25. **GetAreaofPolygon**: This method calculates the area of a polygon defined by a list of 2D points (**pointList**) using the Shoelace formula.
26. **GetCentriodofPolygon**: This method calculates the centroid of a polygon defined by a list of 2D points (**ptlist**). It uses the method **GetArea** to calculate the area and then applies the formula for the centroid.
27. **Private Methods**:
28. **getIndex**: This private method is used to find the index of a given value (**d**) in a sorted list of 2D points (**dataList**). The **nIndex** determines whether to use X or Y values for comparison.
29. **getWtlist**: This private method calculates the weight list used in the odd-numbered integral calculation.
30. **Getdraftwisedata:** This private function calculates the accumulated draft-wise data based on points, function values, and an index, using a summation of products involving differences in X-coordinates, average Y-coordinates, and corresponding function values.
31. **getxWList**: This private method calculates the weights for trapezoidal integration.
32. **sympWtList**: This private method calculates the weights used for trapezoidal integration for even numbers of points.
33. **refinePtlist**: This private method removes the first element from a list of 2D points (**ptlist**) and returns the refined list.
34. **getSecAreaData**: This private function calculates and returns a list of **Point2D** data representing the x-coordinate and calculated sectional area based on a list of **SectionCurve** instances.
35. **GetArea**: This private function computes and returns the area of a quadrilateral defined by four **Point2D** vertices using the trapezoid rule, summing the areas of the four trapezoidal sections formed by consecutive points.
36. **getDataset**: This private method generates a dataset based on the given points and divisions for quadratic or linear fitting.

**Class FixedWeightModel:**

1. **Member Variables:**

**a. \_objectname:** A string variable representing the name of the object.

1. **Constructor:**

a. The class has a default constructor that does not take any parameters.

1. **Properties:**
2. **ObjectName:** A property to get or set the name of the object.

**Class Vector2D:  
  
Member Variables:**

1. **\_stPt:** A Point2D instance representing the starting point of the vector.
2. **\_endPt:** A Point2D instance representing the ending point of the vector.

**Constructors:**

1. **Vector2D():** Default constructor that initializes \_stPt and \_endPt with new instances of Point2D.
2. **Vector2D(Point2D stPt, Point2D endPt):** Parameterized constructor that sets \_stPt and \_endPt with the provided Point2D instances.

**Properties:**

1. **StPt:** A property to get or set the starting point of the vector.
2. **EndPt:** A property to get or set the ending point of the vector.

**Public Methods:**

1. **GetTankAreaforFixZ(SectionCurve secCurve, double z):** Calculates and returns the tank area at a fixed Z-coordinate for the given SectionCurve and Z-value.
2. **GetVectorfromPointList(List<Point2D> ptList):** Creates and returns a list of Vector2D instances from the given list of Point2D instances.
3. **GetPoint(Vector2D v, double param):** Calculates and returns the Point2D on the vector V at the specified parameter value (param).
4. **GetMaxTankVolume(List<SectionCurve> curveList, out Point cGLoc):** Calculates and returns the maximum tank volume and the centroid location (cGLoc) of the tank from a list of SectionCurve instances.
5. **GetTankSecPropertyforFixZ(List<SectionCurve> curveList, double z, ref List<Point2D> cg):** Calculates and returns tank sectional properties for a fixed Z-coordinate from a list of SectionCurve instances.

**Private Methods:**

1. **getCentroid(List<Point2D> ptList):** Calculates and returns the centroid of the given list of Point2D instances.
2. **getsecAreaFixZ(SectionCurve secCurve, double z, out Point cg):** Calculates and returns the tank sectional area at a fixed Z-coordinate for the given SectionCurve and Z-value, and also sets the centroid location (cg) of the tank.
3. **getTankYvalueforIxx(List<Point2D> pointList):** Not implemented, returns 0.
4. **getPolygon(List<Vector2D> vList, Point2D trimmingPt):** Calculates and returns a list of Point2D instances representing a polygon obtained by trimming the Vector2D instances from the given list (vList) with the provided trimming point (trimmingPt).
5. **checkvalue(Vector2D v, Point2D pt):** Determines whether a point (pt) lies above, below, or on the line formed by the Vector2D instance (v) and returns the result as an integer (0: above, 1: below, 2: on the line).
6. **checkOrientation(Vector2D v, double z):** Determines whether a point at a given Z-coordinate (z) is above or below the line formed by the Vector2D instance (v) and returns the result as a boolean (true: above, false: below).
7. **GetTankixxforfixedZ(double z,Tank tank):** This function calculates the cross-sectional area of a tank at a fixed vertical position (z-coordinate) by iteratively evaluating and summing the areas of individual sectional curves within the tank, taking into account specific conditions and using various helper methods for calculations.

**Class Object:**

**Member Variables:**

1. **\_wt**: Holds the weight of the object.
2. **\_cg**: Holds the center of gravity (CG) of the object, represented by a **Point**.

**Properties:**

1. **Weight**: Gets or sets the weight of the object.
2. **CG**: Gets or sets the center of gravity (CG) of the object, represented by a **Point**.

**Class WaterPlane:**

1. **Member Variables**:
   1. **\_nwaterLine**: An integer variable that represents the number of waterlines.
   2. **\_zlocation**: A double variable that stores the z-location (vertical position) of the waterplane.
   3. **\_waterline**: An object of the **SectionCurve** class representing the waterline's cross-sectional curve.
   4. **\_awp**: A double variable that stores the area of the waterplane.
   5. **\_lcf**: A double variable that stores the longitudinal center of flotation of the waterplane.
   6. **\_lengthWline**: A double variable that stores the length of the waterline.
2. **Constructor**:
   1. **WaterPlane()**: The default constructor initializes all member variables, setting numeric variables to 0 and creating an empty **SectionCurve** object.
3. **Properties**:
   1. **NwaterLine**: A property used to get or set the number of waterlines.
   2. **Lcf**: A property used to get or set the longitudinal center of flotation.
   3. **Awp**: A property used to get or set the area of the waterplane.
   4. **Zlocation**: A property used to get or set the z-location (vertical position) of the waterplane.
   5. **WlineLength**: A property used to get or set the length of the waterline.
   6. **Waterline**: A property used to get or set the **SectionCurve** object representing the waterline's cross-sectional curve.
4. **Public Methods**:
   1. **GetTrimedWaterLine(List<Section> secList, Point2D stlpt, Point2D endrtPt, double maxdraft)**: Returns a list of **Point2D** objects representing the trimmed waterline for a given section list, starting and ending points, and maximum draft.
   2. **GetWpAreaMoment(List<Point2D> ptList, double refxPt)**: Calculates and returns the waterplane area moment about the reference x-point.
   3. **GetTcf(List<Point2D> ptList)**: Calculates and returns the transverse center of flotation (TCF) of the waterplane.
   4. **GetHeeledWaterLine(List<SectionCurve> refineList, Point2D pt)**: Returns a list of **Point2D** objects representing the waterline when the ship is heeled (inclined) by a given angle.

**Class WriteOutput:**

**Constructor:**

1. **WriteOutput()**: Default constructor for the **WriteOutput** class.

**Public Methods:**

1. **WritetankInfo(List<Tank> tankList)**: Writes tank information to individual text files for each tank in the specified path.
2. **WritingSummary(List<Summary> sumList)**: Writes the summaries to the console, displaying the object name and value.
3. **WriteLongStData(List<Point2D> data, string filename)**: Writes data to a file with the specified filename in the output path.
4. **WriteLoadCurves(List<Point2D> data1, List<Point2D> data2, string filename)**: Writes load curves data to a file with the specified filename in the output path.
5. **WriteLiquidWtSummary(List<LiquidWeightModel> model)**: Writes liquid weight summary data to the "LiquidSummary.dat" file in the output path.
6. **MaxTankVolume(List<LiquidWeightModel> model)**: Writes maximum tank volume data to the "MaxTankVolume.dat" file in the output path.
7. **WriteLSCalculation**: Writes longitudinal calculation data to the console.

**Class LightWeightModel:**

1. **Member Variables:**
   1. **\_tankName**: Holds the name of the tank.
   2. **\_density**: Holds the density of the liquid in the tank.
2. **Constructor:**
   1. The constructor initializes the **\_density** variable to zero.
3. **Properties:**
   1. **Tankname**: Gets or sets the name of the tank.
   2. **Density**: Gets or sets the density of the liquid in the tank.
4. **Public Method:**
   1. **GetLiquidWt**: This method calculates the liquid weight for each tank based on the mapping list and tank list provided as input. It then interpolates the draft, LCG, TCG, and VCG values for each tank using helper methods. Finally, it creates a list of **LiquidWeightModel** objects representing the liquid weight details for each tank. The method also writes the liquid weight summary to an output file using the **WriteOutput** helper class.

**Class CrossCurve:**

**Member Variables:**

1. \_crosscurves: A list of dictionaries, where each dictionary represents a set of cross curves with the displacement as the key and a list of Point2D instances as the value.
2. \_dispVlcb: A list of dictionaries, similar to \_crosscurves, representing the displacement vs. LCB (Longitudinal Center of Buoyancy) data.
3. \_dispVtcb: A list of dictionaries, similar to \_crosscurves, representing the displacement vs. TCB (Transverse Center of Buoyancy) data.
4. \_dispVvcb: A list of dictionaries, similar to \_crosscurves, representing the displacement vs. VCB (Vertical Center of Buoyancy) data.

**Constructor:**

1. The class has a constructor that initializes the lists of cross curves (\_crosscurves, \_dispVlcb, \_dispVtcb, \_dispVvcb) with empty lists.

**Properties:**

1. **CrossCurves:** A read-only property to get the list of cross curves (displacement vs. curve points).
2. **DispVLcb:** A read-only property to get the list of displacement vs. LCB (Longitudinal Center of Buoyancy) data.
3. **DispVTcb:** A read-only property to get the list of displacement vs. TCB (Transverse Center of Buoyancy) data.
4. **DispVVcb:** A read-only property to get the list of displacement vs. VCB (Vertical Center of Buoyancy) data.

**Public Methods:**

1. **AddCrossCurve(Dictionary<double, List<Point2D>> crCurve):** Adds a dictionary representing a cross curve to the list of cross curves (\_crosscurves).
2. **AddLcbPoint(Dictionary<double, List<Point2D>> crCurve):** Adds a dictionary representing displacement vs. LCB data to the list (\_dispVlcb).
3. **AddTcbPoint(Dictionary<double, List<Point2D>> crCurve**): Adds a dictionary representing displacement vs. TCB data to the list (\_dispVtcb).
4. **AddVcbPoint(Dictionary<double, List<Point2D>> crCurve):** Adds a dictionary representing displacement vs. VCB data to the list (\_dispVvcb).
5. **GetKnCurve(List<SectionCurve> secCurveList, double thetamin, double thetamax, double thetaIncrement, double zMax, double rho):** Calculates and returns the Kn curve for zero trim based on input data.
6. **GetSecCrossCurveValue** **(List<SectionCurve> secCurveList, double theta, double dz, ref List<Point2D> tMomList, ref List<Point2D> zMomList, ref List<Point2D>yList):** This function computes cross-sectional curve values for torque, z-moment, and y-coordinate at different x-locations along SectionCurve instances, considering a given angle and vertical displacement, and populates the corresponding lists.
7. **GetSecCrossCurveValue** **(List<SectionCurve> secCurveList, double theta, double dz, ref List<Point2D> tMomList, ref List<Point2D> zMomList):** Similar to the above function, this version calculates cross-sectional curve values for torque and z-moment only, without considering y-coordinates, and updates the provided lists accordingly.

**Private Methods:**

1. **getYcrosscurvevalues(Point2D pt, double m, double c):** Calculates and returns the Y-coordinate value on the cross curve based on a given point (pt), slope (m), and constant (c).
2. **getKnforFixedtheta(List<SectionCurve> secCurveList, double theta, double maxZ, double rho, StreamWriter sr, ref List<Point2D> tcbList, ref List<Point2D> vcbList):** Calculates and returns the Kn curve for a fixed theta angle and displacement.
3. **writeData(double theta, Point2D pt, Point2D pt1, StreamWriter sr):** Writes the data for theta, displacement, TCB, and VCB to a StreamWriter.
4. **getKnforfixedthetaAnddisp(List<SectionCurve> secCurveList, double theta, double dz, double rho, out Point2D cgValue):** Calculates and returns the Kn and displacement values for a fixed theta and displacement.
5. **getValuesforfixZ(List<Point2D> draftVsY, double theta, double dz, out double ymax):** Calculates and returns the area, TCB, and VCB values for a fixed Z-coordinate.
6. **getValuesforfixZ(List<Point2D> draftVsY, double theta, double dz):** Calculates and returns the area, TCB, and VCB values for a fixed Z-coordinate.
7. **getYcrosscurvelist(List<Point2D> draftVsY, double m, double c, ref List<Point2D> tMomList, ref List<Point2D> zMomList):** Calculates and returns a list of Point2D instances representing the Y-coordinate values on the cross curve for given displacement and slopes.

**Class ShipGeom:  
 Member Variables:**

* + **\_lbp**: Length Between Perpendiculars (LBP) of the ship.
  + **\_loa**: Length Overall (LOA) of the ship.
  + **\_breadth**: Breadth of the ship.
  + **\_designdraft**: Design draft of the ship.
  + **\_depth**: Depth of the ship.
  + **\_fld**: Freeboard at the Load Draft (FLD).
  + **\_downfloodingangle**: Downflooding angle of the ship.
  + **\_draftAp:** A list containing points related to draft at the aft perpendicular of the ship.
  + **\_draftFp:** A list containing points related to draft at the forward perpendicular of the ship.
  + **\_draftVol:** A list containing points related to draft and volume of the ship.
  + **\_draftMass:** A list containing points related to draft and mass of the ship.
  + **\_draftWetarea:** A list containing points related to draft and wet area of the ship.
  + **\_draftLcb:** A list containing points related to draft and the longitudinal center of buoyancy of the ship.
  + **\_draftTcb:** A list containing points related to draft and the transverse center of buoyancy of the ship.
  + **\_draftVcb:** A list containing points related to draft and the vertical center of buoyancy of the ship.
  + **\_draftWpa:** A list containing points related to draft and the waterplane area of the ship.
  + **\_draftLcf:** A list containing points related to draft and the longitudinal center of flotation of the ship.
  + **\_draftIxx0:** A list containing points related to draft and the moment of inertia about the x-axis at the origin of the ship.
  + **\_draftIxxg:** A list containing points related to draft and the moment of inertia about the x-axis at the center of gravity of the ship.
  + **\_draftIyy0:** A list containing points related to draft and the moment of inertia about the y-axis at the origin of the ship.
  + **\_draftBml:** A list containing points related to draft and the longitudinal metacenter of the ship.
  + **\_draftBmt:** A list containing points related to draft and the transverse metacenter of the ship.
  + **\_draftKml:** A list containing points related to draft and the longitudinal radius of gyration of the ship.
  + **\_draftKmt:** A list containing points related to draft and the transverse radius of gyration of the ship.
  + **\_draftTpc:** A list containing points related to draft and the transverse projected center of the ship.
  + **\_draftMct:** A list containing points related to draft and the longitudinal center of the transverse metacenter of the ship.
  + **\_wtCurve:** A list containing points related to weight distribution of the ship.
  + **\_buoyancyCurve:** A list containing points related to buoyancy distribution of the ship.
  + **\_hulldistribution:** A list containing points related to hull distribution of the ship.
  + **\_waterLine:** A list containing points related to the waterline of the ship.

**Constructor:**

* + The default constructor initializes all member variables to **0**, and the parameterized constructor takes the Length Between Perpendiculars (LBP) as an argument.

**Properties:**

* + Properties to get/set various geometric dimensions of the ship such as **LBP**, **LOA**, **Breadth**, **DesignDraft**, **Depth**, **FLD**, and **DownFLDangle**.
  + Properties to get read-only access to the lists storing different draft-related data points and curves such as **DraftAp**, **DraftFp**, **DraftVol**, **DraftMass**, **DraftWetarea**, **DraftLcb**, **DraftTcb**, **DraftVcb**, **DraftWpa**, **DraftLcf**, **DraftIxx0**, **DraftIxxg**, **DraftIyy0**, **DraftBml**, **DraftBmt**, **DraftKml**, **DraftKmt**, **DraftTpc**, **DraftMct**, **WtCurve**, **BuoyancyCurve**, **HullWtDistribution**, and **WaterLine**.

**Public Methods:**

* + **AdddraftAp**: Adds a Point2D instance to the **\_draftAp** list.
  + **AdddraftFp**: Adds a Point2D instance to the **\_draftFp** list.
  + **AdddraftVol**: Adds a Point2D instance to the **\_draftVol** list.
  + **AdddraftMass:** Adds a Point2D instance to the **\_draftMass** list.
  + **AdddraftWetarea**: Adds a Point2D instance to the **\_draftWetArea** list.
  + **AdddraftLcb:** Adds a Point2D instance to the **\_draftLcb** list.
  + **AdddraftTcb**: Adds a Point2D instance to the **\_draftTcb** list.
  + **AdddraftVcb:** Adds a Point2D instance to the **\_draftVcb** list.
  + **AdddraftWpa**: Adds a Point2D instance to the **\_draftWpa** list.
  + **AdddraftLcf:** Adds a Point2D instance to the **\_draftLcf** list.
  + **AdddraftIxx0**: Adds a Point2D instance to the **\_draftIxx0** list.
  + **AdddraftIxxg:** Adds a Point2D instance to the **\_draftIxxg** list.
  + **AdddraftIyy0**: Adds a Point2D instance to the **\_draftIyy0** list.
  + **AdddraftBml:** Adds a Point2D instance to the **\_draftBml** list.
  + **AdddraftBmt**: Adds a Point2D instance to the **\_draftBmt** list
  + **AdddraftKml:** Adds a Point2D instance to the **\_draftKml** list.
  + **AdddraftKmt**: Adds a Point2D instance to the **\_draftKmt** list.
  + **AdddraftTpc:** Adds a Point2D instance to the **\_draftTpc** list.
  + **AdddraftMct**: Adds a Point2D instance to the **\_draftMct** list.
  + **AdddWtCurvePt:** Adds a Point2D instance to the **\_wtCurve** list.
  + **AdddBuoyancyCurvePt**: Adds a Point2D instance to the **\_buoyancyCurve** list
  + **AddWaterlinePt**. Adds a Point2D instance to the **\_waterLine** list.
  + **UpdateBuoyancyCurve**: Method to update the buoyancy curve.
  + **UpdateWaterline**: Method to update the waterline
  + **UpdateLoadDistribution**: Method to update the Load Distribution.
  + **GetHullWtDistribution**: Method to update the Hull weight Distribution.

**Class Wind:**

**Member Variables**:

* **\_x1**: A double variable representing the first coefficient used in wind criteria calculations.
* **\_x2**: A double variable representing the second coefficient used in wind criteria calculations.
* **\_k**: A double variable representing another coefficient used in wind criteria calculations.
* **\_s**: A double variable representing yet another coefficient used in wind criteria calculations.
* **\_maxheel**: A double variable representing the maximum heeling angle caused by wind.

**Constructor**:

* **Wind(double X1, double X2, double k, double s, double maxheel)**: Parameterized constructor with member variables as parameters.
* **Wind()**: Default constructor that initializes member variables with default values (zeros).

**Properties**:

* **X1**: A public double property to get or set the value of **\_x1**.
* **X2**: A public double property to get or set the value of **\_x2**.
* **s**: A public double property to get or set the value of **\_s**.
* **k**: A public double property to get or set the value of **\_k**.
* **Maxheel**: A public double property to get or set the value of **\_maxheel**.

**Private Methods**:

* **getMintheta(Wind criteria, double meandraft, double vcg, double depth):**  Calculates and returns the minimum roll angle (heeling angle) required for a vessel based on specified criteria, mean draft, vertical center of gravity (vcg), and depth.
* **areaRatio(Dictionary<int,List<Point2D>> areaInfo):** Computes and returns a list of area ratios based on a dictionary of section information, utilizing helper methods to calculate sectional areas.
* **getWgz (Object shp, CrossCurve curve, double minAngle, double downfloodingAngle, double maxheel, double wharm, ref Dictionary<int, List<Point2D>> areaInfo) :** Computes a list of GZ values (righting arm) for various angles of heel, considering specific conditions, and updates an area information dictionary.
* **getsplitPoint (List<Point2D> windGz, double wharm) :** Determines and returns a splitting point for a given list of GZ values, used for splitting curves in later calculations.
* **getsplitedCurve(List<Point2D> windGz,Point2D splitPt):** Splits a list of GZ values based on a given splitting point and returns a dictionary containing the divided curves.
* **calculateHeelingarm (double area, double z, double disp):** Calculates and returns the heeling arm (righting arm) for a given area, vertical location (z-coordinate), and displacement.
* **calculateDownheelingangle(double height,double breadth):** Computes and returns the angle of heel at which downflooding occurs, based on given dimensions.
* **calculateForce:** Calculates and returns the hydrodynamic force exerted on an object based on its area and velocity.
* **calculateForceIMO:** Computes and returns the IMO-standardized hydrodynamic force on an object, considering its lateral area and vertical location.
* **findHeelangle(double force, double disp, double gm):** Determines and returns the angle of heel for a vessel based on the applied force, displacement, and metacentric height.
* **getsuperStructureData(List<FixedWeightModel> superstructureList, FixedWeightModel aboveWater):** Combines a list of superstructure models and calculates their centroid based on weight and position.
* **areaCalculation (first version):** Computes the area and centroid for fixed weight models based on given sectional information, a reference point, and vessel dimensions.
* **areaCalculation (second version):** Calculates the area and centroid for fixed weight models considering lateral wind pressure distribution.
* **getDividingLine:** Constructs and returns a dividing line defined by two points for a given point and length.
* **getLateralAreaInfo:** Computes and returns a dictionary of lateral area information, split into upper and lower sections, based on a reference point, length, and depth.

**Public Method**:

* **GetWindCriteria**: Calculates various wind criteria for ship stability given wind velocity and other ship-related parameters. It returns a list of **Summary** objects containing the results.

**Class Summary:**

1. **Member Variables**:
   * **\_objectname**: A string variable that stores the name of the object or parameter.
   * **\_objectvalue**: A double variable that stores the corresponding value of the object or parameter.
2. **Constructors**:
   * **Summary()**: The default constructor initializes **\_objectvalue** to 0.
   * **Summary(string str, double objValue)**: This parameterized constructor takes a string and a double as arguments and assigns them to **\_objectname** and **\_objectvalue**, respectively.
3. **Properties**:
   * **ObjectName**: A string property used to get or set the name of the object or parameter.
   * **ObjectValue**: A double property used to get or set the value of the object or parameter.
4. **Public Methods**:

* **GetGzCurveInfo**: Computes and returns a list of GZ curve points based on the intact weight distribution of fixed and liquid weight models, a cross curve, and ship displacement details.
* **GetLoadingSummary**: Generates and returns a loading summary containing information about liquid weight distribution based on categorized liquid weight models and fixed weight models.
* **GetIntactCriteriaValue**: Calculates and returns a list of summaries detailing intact stability criteria values using fixed and liquid weight models, a cross curve, ship geometry, and other parameters.
* **GetEquilibriumConditionDetails**: Computes and returns a list of summaries describing equilibrium conditions for a vessel, considering its geometry, cross curve data, weight models, sections, tanks, and other parameters.
* **GetGridvalue**: Retrieves and returns mapping values for a tank's grid points based on tank properties, density, and mapping details.
* **GetLongitudinalStrengthSummaryInfo**: Computes and returns a list of summaries containing longitudinal strength information, considering ship geometry, fixed weight models, tanks, mappings, sections, density, and more.
* **GetLSCalculationInfo**: Calculates and returns a list of longitudinal calculation details based on previously generated longitudinal strength summaries, mapping information, shear force, and bending moment data.
* **GetDamagedHydrostaticFull**: Computes and returns a dictionary containing details of hydrostatic damage conditions for different tanks, considering ship geometry, fixed and liquid weight models, sections, damage tank names, cross curve, density, and tank list.
  + **GetWindCriteriaInfo**: Computes wind criteria information and returns a list of summaries based on specified velocity, superstructure models, ship geometry, sections, fixed and liquid weight models, cross curve, density, wind criteria, and also updates a list of wind-induced GZ values.

1. **Private Methods**:

* **convertObjectType (List<FixedWeightModel> objList):** Converts a list of FixedWeightModel objects into a list of Point2D objects, excluding those with the "Lightship" object name. Each Point2D consists of the X-coordinate as the CG's X value and the Y-coordinate as the weight.
* **getCategoryWiseLiquidWt:** **(Dictionary<string, List<LiquidWeightModel>> liquidWeight)** Categorizes and returns a list of LiquidWeightModel objects based on liquid weight data provided in a dictionary format. Centroids are calculated for each category and added to the resulting list.
* **getdamageTanks(List<String> tankname, List<Tank> tankList):** Retrieves a list of damaged tanks from a given list of tank names and a list of all available tanks, matching the tank names to create a list of affected tanks.
* **getmaxBendingShearInfo(List<Summary> longsumList):** Retrieves and returns the maximum shear force and maximum bending moment values from a list of longitudinal strength summaries as a Point2D object.

**Class Tank:**

1. **Member Variables**:
   * **\_tankCurvedata**: A list of **SectionCurve** objects that represents the tank's cross-sectional curves.
   * **\_liquidrho**: A double variable that stores the density of the liquid inside the tank.
   * **\_voldata**: A list of **Point2D** objects that stores the tank's volume data at different vertical levels.
   * **\_massdata**: A list of **Point2D** objects that stores the tank's mass data at different vertical levels.
   * **\_percentageOffilling**: A list of **Point2D** objects that stores the percentage filling of the tank at different vertical levels.
   * **\_tankLcg**: A list of **Point2D** objects that stores the longitudinal center of gravity (LCG) of the tank at different vertical levels.
   * **\_tankTcg**: A list of **Point2D** objects that stores the transverse center of gravity (TCG) of the tank at different vertical levels.
   * **\_tankVcg**: A list of **Point2D** objects that stores the vertical center of gravity (VCG) of the tank at different vertical levels.
   * **\_fsmt**: A list of **Point2D** objects that stores the longitudinal static stability (FSMT) of the tank at different vertical levels.
   * **\_fsml**: A list of **Point2D** objects that stores the lateral static stability (FSML) of the tank at different vertical levels.
   * **\_sectankDraftVarea**: A list of **SectionCurve** objects that stores the draft and corresponding cross-sectional area of the tank.
   * **\_maxWeight**: A double variable that stores the maximum weight that the tank can carry.
   * **\_permiability**: A double variable that represents the permeability factor of the tank.
   * **\_tankName**: A string variable that stores the name of the tank.
   * **\_tankCode**: A string variable that stores the code of the tank.
2. **Constructor**:
   * **Tank()**: The default constructor initializes all member variables, creating empty lists and setting numeric variables to 0.
3. **Properties**:
   * **LiquidRho**: A double property used to get or set the density of the liquid inside the tank.
   * **TankCurvedata**: A read-only property that returns the list of **SectionCurve** objects representing the tank's cross-sectional curves.
   * **VolumeData**: A read-only property that returns the list of **Point2D** objects representing the tank's volume data.
   * **MassData**: A read-only property that returns the list of **Point2D** objects representing the tank's mass data.
   * **PerOfFilling**: A read-only property that returns the list of **Point2D** objects representing the percentage filling of the tank.
   * **TankLcg**: A read-only property that returns the list of **Point2D** objects representing the tank's longitudinal center of gravity.
   * **TankTcg**: A read-only property that returns the list of **Point2D** objects representing the tank's transverse center of gravity.
   * **TankVcg**: A read-only property that returns the list of **Point2D** objects representing the tank's vertical center of gravity.
   * **Fsmt**: A read-only property that returns the list of **Point2D** objects representing the tank's longitudinal static stability.
   * **Fsml**: A read-only property that returns the list of **Point2D** objects representing the tank's lateral static stability.
   * **MaxWeight**: A property used to get or set the maximum weight that the tank can carry.
   * **SecTankDraftVArea**: A read-only property that returns the list of **SectionCurve** objects representing the draft and cross-sectional area of the tank.
   * **TankName**: A property used to get or set the name of the tank.
   * **TankCode**: A property used to get or set the code of the tank.
   * **Permiability**: A double property used to get or set the permeability factor of the tank.
4. **Public Methods**:
   * **AddCurvedata(SectionCurve curve)**: Adds a **SectionCurve** object representing a cross-sectional curve to the tank's **\_tankCurvedata** list.
   * **AddVolData(Point2D pt)**: Adds a **Point2D** object representing volume data at a specific vertical level to the tank's **\_voldata** list.
   * **AddMassData(Point2D pt)**: Adds a **Point2D** object representing mass data at a specific vertical level to the tank's **\_massdata** list.
   * **AddPerOfFillData(Point2D pt)**: Adds a **Point2D** object representing the percentage filling at a specific vertical level to the tank's **\_percentageOffilling** list.
   * **AddTankLcg(Point2D pt)**: Adds a **Point2D** object representing the longitudinal center of gravity at a specific vertical level to the tank's **\_tankLcg** list.
   * **AddTankVcg(Point2D pt)**: Adds a **Point2D** object representing the vertical center of gravity at a specific vertical level to the tank's **\_tankVcg** list.
   * **AddTankTcg(Point2D pt)**: Adds a **Point2D** object representing the transverse center of gravity at a specific vertical level to the tank's **\_tankTcg** list.
   * **AddFsmt(Point2D pt)**: Adds a **Point2D** object representing longitudinal static stability at a specific vertical level to the tank's **\_fsmt** list.
   * **AddFsml(Point2D pt)**: Adds a **Point2D** object representing lateral static stability at a specific vertical level to the tank's **\_fsml** list.
   * **GetSecTankAreaSounding(Tank tank)**: Updates the **\_sectankDraftVarea** list with modified cross-sectional curves for sounding calculation based on the tank type.
   * **GetTankSoundings(Tank tank)**: Updates various tank properties like **\_voldata**, **\_massdata**, **\_percentageOffilling**, **\_tankLcg**, **\_tankTcg**, **\_tankVcg**, **\_fsmt**, and **\_fsml** based on the tank type and sounding data.
   * **GetTankFsm(Tank tank, ShipGeom ship)**: Calculates and updates **\_fsml** and **\_fsmt** values based on the tank's geometry and ship properties.
   * **UpdateDensity(double rho)**: Updates the density of the liquid inside the tank.
   * **UpdateMaxWt(double maxWt)**: Updates the maximum weight that the tank can carry.
   * **GetDamageTankDetails(List<string> tankNameList, List<Tank> tankList)**: Returns a list of tanks with matching names from the input list of tank names.
5. **Private Methods**:
   * **fillsoundingForStank(List<SectionCurve> curveList, Tank tank)**: Calculates and fills the tank sounding data for tanks of type "S" (storage tanks).
   * **fillsoundingForPtank(List<SectionCurve> curveList, Tank tank)**: Calculates and fills the tank sounding data for tanks of type "P" (port tanks).
   * **fillsoundingForCtank(List<SectionCurve> curveList, Tank tank)**: Calculates and fills the tank sounding data for tanks of type "C" (center tanks).
   * **modifyTankDataforCtank(Tank tank)**: Modifies the tank data for tanks of type "C" to handle symmetric data.
   * **modifyTankdataForPtank(Tank tank)**: Modifies the tank data for tanks of type "P" to handle symmetric data.
   * **getstring(string s)**: Extracts the last part of a string separated by periods and returns it.
   * **getMaxmin(Tank tank)**: Calculates and returns the maximum and minimum vertical levels of the tank based on the cross-sectional curves.
   * **maximumMinimumElementArray(List<double> ptList)**: Calculates and returns the maximum and minimum values from a list of double values.

**Class Damage:**

1. **Constructor:**
   * The class has a default constructor with no parameters.
2. **Public Methods:**
   * **GetDamagedHydrostatic**: This method calculates the damaged hydrostatic properties of the ship after sustaining damage. It takes various inputs, such as ship geometry (**ship**), lists of fixed weights (**fixedWt**), liquid weights (**liquidWt**), sections (**secList**), tanks (**tankList**), a cross curve (**curve**), and fluid density (**rho**). This method returns a list of **Summary** objects containing various stability and hydrostatic properties, and it also populates the **criteria** list with additional criteria data.
3. **Private Methods:**
   * **getdownFLDinfo**: A helper method that calculates and returns information related to the downflooding angle.
   * **trimValue**: A helper method that calculates the trim of the ship based on various inputs.
   * **damagedWt**: A helper method that calculates the damaged weight and center of gravity of the ship.
   * **addedwtList**: A helper method that modifies the liquid weight models of the damaged tanks based on the draft line (draftLine).
   * **modifytankValue**: A helper method that modifies the weight and center of gravity of a damaged tank based on the draft line.

**Class InputParamHelper:**  **Constructor:**

* The class has a default constructor with no parameters.

**Public Methods:**

* **ReaddShipgeomdata**: This method reads ship geometry data from a file named "Hull Hydrostatics.dat." It parses the data and populates a **ShipGeom** object with the relevant information.

reading data from a file, organizing it into instances of the **Point2D** class, and then adding these instances to a **ShipGeom** object and return shgeom;

* **ReadSectionData**: This method reads section data from a file named "hullSection.dat." It parses the data and returns a list of **Section** objects, each containing information about sections and their curves.

reads data from two files, creates instances of the **Section** class, populates them with data, and then stores them in a list. Additionally, it reads X coordinates from another file and assigns them to each **Section** object.

* **ReadFrameInput**: Reads and processes input data from a file named "Frames.dat" located at a specific path. It creates and populates a list of **Mapping** objects by parsing each line of the input file. Each line is split by a comma, where the first part represents an integer index value and the second part represents a double value.

explain this method and its implementation in detail for a beginner:

* **ReadHullDistData**: This method reads hull distribution data from a file named "Hull\_Dist.dat." It parses the data and returns a list of **Vector2D** objects, each representing a hull distribution vector.
* **ReadLoadingData**: This method reads loading data from a file named "Loading.dat." It parses the data and returns a list of **Mapping** objects, each containing tank names and associated loading data.
* **GetPermiability**: This method reads permiability data from a file named "permiability.dat." It parses the data and returns a list of **Tank** objects, each containing permiability and liquid density information.
* **ReadFixedWeight**: This method reads fixed weight data from a file named "solidWt.dat." It parses the data and returns a list of **FixedWeightModel** objects, each containing fixed weight information and center of gravity.
* **ReadDamagedTank**: This method reads information about damaged tanks from a file named "damageTank.dat." It parses the data and returns a list of strings, each representing a damaged tank name.
* **ReadLateralAreaInfo**: This method reads lateral area information from a file named "windheel.dat." It parses the data and returns a list of **FixedWeightModel** objects, each containing lateral area information and center of gravity.
* **ReadShipTankData**: Reads and processes data from a file named "tank.txt" to extract tank geometry information. It separates tank data sections marked by "*PART*" lines, creates a list of **Tank** objects with extracted data, and returns the list. If the file or data is invalid, it returns null.

**Private Methods:**

* **getTankGeo**: This method extracts and processes the tank geometry information from a list of strings and returns a **Tank** object with the relevant data.

**Class IntactStability**:

**Public Method:**

* **GetIntactSummaryInfo**: This method takes various input parameters related to ship geometry, cross curves, fixed weight, liquid weight, sections, tanks, and density. It calculates and returns a list of **Summary** objects, each containing essential intact stability information such as trim, heel, displacement, LCG, TCG, VCG, draft values (aft, midship, and forward), GM (both fluid and solid), FS correction, and VCG correction.

**Private Method:**

* **getfsmcorrection**: This method calculates the free surface correction (FS correction) based on liquid weight data, mean draft, and tank information. It iterates through the liquid weight and tank lists to find the corresponding FS correction values and aggregates them to calculate the total FS correction.

**Class LongitudinalCalculations:**

**Member Variables:**

* **\_frameno**: Holds the frame number.
* **\_framespacing**: Holds the distance of the frame.
* **\_shearforceval**: Holds the shear force at the frame.
* **\_shearforceper**: Holds the percentage of shear force at the frame.
* **\_bendingmomentval**: Holds the bending moment at the frame.
* **\_bendingmomentper**: Holds the percentage of bending moment at the frame.

**Constructor:**

* **Default constructor:** Initializes all member variables with default values of **0**.
* **Parameterized constructor:** Accepts values for frame number, frame spacing, shear force value, shear force percentage, bending moment value, and bending moment percentage, and initializes the corresponding member variables.

**Properties:**

* **FrameNo**: Gets or sets the frame number.
* **FrameSpacing**: Gets or sets the distance of the frame.
* **ShearAtFrame**: Gets or sets the shear force at the frame.
* **ShearPerAtFrame**: Gets or sets the percentage of shear force at the frame.
* **BendingAtFrame**: Gets or sets the bending moment at the frame.
* **BendingPerAtFrame**: Gets or sets the percentage of bending moment at the frame.

**Public Method:**

* **GetBendingMomentInfo**: This method calculates the bending moment and shear force information for the given ship geometry, solid weight distribution, tank geometry information, mapping list, section list, and fluid density. It first updates the load distribution based on the given inputs, then calculates the ship's center of gravity (CG). After that, it calculates the load curve and the shear force using helper methods. Finally, it calculates the bending moment using the shear force data and returns the list of bending moments.

**Class OutputParamHelper:**

**Constructor:**

* **OutputParamHelper()**: Default constructor for the **OutputParamHelper** class.

**Private Methods:**

* **updateTrim(double trim, double zchange, double lcg, double lcb)**: Updates the trim value based on the change in draft.
* **getSignature(double lcg, double lcb)**: Gets the signature (1 or -1) based on the comparison of LCG and LCB.
* **updateSinkage(double zchange, double wt, double disp)**: Updates the sinkage value based on the change in weight.
* **getmodifiedtrim(double lcg, double lcb, int sig)**: Gets the modified trim value based on LCG, LCB, and the signature.
* **getMoment(double lcb, double lcg, double wt)**: Calculates the moment based on LCB, LCG, and weight.
* **getModifiedSectionCurve(List<Section> secList)**: Modifies the section curves by reducing the Y values to half.
* **modifyPtList(List<Point2D> ptList)**: Modifies the point list by reducing the Y values to half.
* **getDensePtCurve(List<SectionCurve> curves)**: Gets dense point curves by adding more points to the existing curves.
* **getKNcurve(CrossCurve curveData, double disp)**: Gets the KN curve based on the cross curve data and displacement.
* **getDataFromCrossCurve(List<Dictionary<double, List<Point2D>>> dataList, double disp)**: Gets the data from the cross curve for the specified displacement.
* **getBmt(ShipGeom shp, double rho)**: Calculates the BMT (Block Moment of Transverse) based on ship geometry and density.
* **getIxx(ShipGeom shp)**: Calculates the Ixx value based on ship geometry.
* **ptListforIxxCalculation(List<Point2D> waterline)**: Prepares a list of points for Ixx calculation based on the waterline points.
* **getWt(Tank tank, double x, double xvalue)**: Gets the weight value for the specified tank and x location.
* **getLiquidWt(double x, List<Tank> tankData, List<Mapping> mp)**: Gets the total liquid weight for the specified x location.
* **SolidWt(double x, double wtLoc, double solidWt, double dist)**: Calculates the solid weight based on x location, weight location, solid weight, and distance.
* **getSolidWt(double x, List<Point2D> solidWt, double distance)**: Gets the total solid weight for the specified x location.
* **GetDeadWeight(List<LiquidWeightModel> LiquidWeights, List<FixedWeightModel> FixedWeights)**: Calculates the total deadweight based on liquid and fixed weights.
* **GetDeadWeightLCG(List<LiquidWeightModel> LiquidWeights, List<FixedWeightModel> FixedWeights)**: Calculates the longitudinal center of gravity based on liquid and fixed weights.
* **GetDeadWeightVCG(List<LiquidWeightModel> LiquidWeights, List<FixedWeightModel> FixedWeights)**: Calculates the vertical center of gravity based on liquid and fixed weights.
* **GetDeadWeightTCG(List<LiquidWeightModel> LiquidWeights, List<FixedWeightModel> FixedWeights)**: Calculates the transverse center of gravity based on liquid and fixed weights.
* **IndexValue(double pt, List<Point2D> steelWt)**: Gets the index value based on the steel weight data and the specified point.

**Public Method**

* **GetShipHeelAngle**: This method calculates the heel angle of the ship using input parameters such as ship geometry, section data, weight, center of gravity, and density. It uses iterative calculations to determine the heel angle that balances the moments and weight distribution.
* **SracpGetHeelAngleFromIteration**: This method is very similar to **GetShipHeelAngle**, but the name and a few variables have been changed.
* **GetHeelangle**: This method calculates the heel angle of the ship using the input parameters, including weight, center of gravity, and cross-curve data. It performs interpolation to determine the heel angle based on the cross-curve data.
* **GetWtDistribution**: This method calculates the weight distribution in the ship by considering steel weight, liquid weight, and solid weight. It returns a list of points representing the weight distribution.
* **IntactWtDetails**: This method returns details about the intact weight of the ship, considering fixed weights and liquid weights.
* **GetGm**: This method calculates the metacentric height (GM) of the ship using input parameters such as draft, cross-curve data, and displacement.
* **GetGmfromSmalAngle**: This method calculates GM based on small-angle stability considerations using input parameters such as the ship's geometry, density, and center of gravity.
* **GetGZcurve**: This method calculates the righting arm (GZ curve) of the ship using cross-curve data, the center of gravity, and displacement.
* **GetCrossCurveData**: This method retrieves the cross-curve data using input parameters such as sections, angles, maximum draft, and density.
* **GetTrimValue**: This method calculates the trim of the ship based on input parameters like ship geometry, sections, weight, longitudinal center of gravity, and density. It performs iterative calculations to determine the trim.
* **GetTrim**: This method is very similar to **GetTrimValue** but uses a different approach to calculate trim.
* **GetTankData**: This method calculates and returns data related to the tanks on the ship using input parameters like tank type, density, and mapping data.
* **GetLoadCurve**: This method calculates the load curve of the ship using the weight and buoyancy data.
* **GetShearForce**: This method calculates the shear force on the ship using the load data.
* **GetBendingMoment(List<Point2D> shearForce):** This method calculates the bending moments at various points on the ship's hull based on the given shear forces.
* **GetSummary(List<LiquidWeightModel> LiquidWeights, List<FixedWeightModel> FixedWeights):** This method generates a summary of different loads acting on the ship, including liquid weights and fixed weights.
* **GetAreafromZeroTotheta(List<Point2D> ptList, double maxtheta):** This method calculates the area under a curve up to a specified angle (maxtheta) using the trapezoidal rule.
* **GetmaxGztilltheta(List<Point2D> ptList, double theta):** This method calculates the maximum GZ value until a specified angle (theta) from a list of GZ points.
* **GetIntactCriteria(Object shipDisp, CrossCurve curve, ShipGeom ship, double rho**): This method computes various intact stability criteria such as the area under the GZ curve up to 30 and 40 degrees, the maximum GZ value, and the metacentric height.
* **GetLongitudinalCalculation(List<Point2D> shearForce, List<Point2D> bendingMoment):** This method calculates longitudinal strength calculations for the ship based on shear forces and bending moments.
* **GetIntactSummary():** This method generates a summary of intact stability parameters such as trim, heel, displacement, LCG, TCG, VCG, drafts, GM (fluid), GM (solid), and various corrections.
* **GetLongitudinalStrengthSummary(List<Point2D> shearForce, List<Point2D> bendingMoment):** This method generates a summary of longitudinal strength parameters, including minimum and maximum shear forces and bending moments and their locations.
* **GetDisplacement(List<LiquidWeightModel> LiquidWeights, List<FixedWeightModel> FixedWeights):** This method calculates the total displacement of the ship by summing up the weights of liquid and fixed components. If there is a specific component called "Lightship," its weight is added to the displacement.
* **GetDisplacementLCG(List<LiquidWeightModel> LiquidWeights, List<FixedWeightModel> FixedWeights):** This method calculates the longitudinal center of gravity (LCG) of the ship based on the weights and LCG positions of liquid and fixed components.
* **GetDisplacementVCG(List<LiquidWeightModel> LiquidWeights, List<FixedWeightModel> FixedWeights):** This method calculates the vertical center of gravity (VCG) of the ship based on the weights and VCG positions of liquid and fixed components.
* **GetDisplacementTCG(List<LiquidWeightModel> LiquidWeights, List<FixedWeightModel> FixedWeights):** This method calculates the transverse center of gravity (TCG) of the ship based on the weights and TCG positions of liquid and fixed components.
* **updateheelangle(double tcg, double tcb, double increment, double angle**): This method updates the heel angle of the ship based on the difference between the transverse center of gravity (TCG) and the transverse center of buoyancy (TCB). The method takes the current angle and increments it based on the direction of the change in the TCG-TCB difference.