**Area.java:**

Area.java is a class that defines a shape consisting of a border and may contain holes. (necessarily a PCB board).

This class consists of functions like (essentially functions which define or support the area):

* **contains:** checks if a point exists in the area.
* **turn\_90\_degree:** Helps turn the area 90 degrees based on a defined pole.
* **mirror\_horizontal:** Mirrors the area horizontally through a defined pole.
* **mirror\_vertical:** Mirrors the area vertically through a defined pole.
* **Is\_empty:** checks if area is empty(no points or holes).
* **Is\_bounded:** checks if area is contained in a sufficiently large box.
* **Dimension:** returns 2, if the area contains 2 dimensional shapes , 1, if it contains curves, 0, if it is reduced to a point and -1, if it is empty.

**Shape.java:**

An interface (contains functions which are used in subclasses) which describes the functionality of 2-dimensional shapes in the plane(does not contain holes).

**TileShape.java:**

Abstract class defining functionality for convex shapes, whose borders consists of straight lines.

**PolygonShape.java:**

Shape described by a closed polygon of corner points.

* **is\_convex():** Checks, if every line segment between 2 points of the shape is contained completely in the shape.
* **split\_to\_convex():** Splits this polygon shape into convex pieces. The result is not exact, because rounded intersections of lines are used in the result pieces. It can be made exact, if Polylines are returned instead of Polygons, so that no intersection points are needed in the result.
* **split\_to\_convex\_recu():** Private recursive part of split\_to\_convex. Returns a collection of polygon shape pieces.

**PolylineShape.java:**

Abstract class with functions for shapes, whose borders consist of straight lines.

* **corner\_is\_bounded():** checks if corner is bounded.
* **border\_line\_count():** counts the number of borderlines of the shape
* **bounded\_corners():** Return all unbounded corners of this shape.
* **equals\_corner():** checks if a defined point is a corner in the shape.
* **centre\_of\_gravity():** returns arithmetic middle of the corners in the shape.
* **circumference():** Returns the cumulative border line length of the shape.

**PolylineArea.java:**

A PolylineArea is an Area, where the outside border curve and the hole borders consist of straight lines. Overrides functions from the Area.java file.

**ExpendableObject.java:**

An object, which can be expanded by the maze expansion algorithm.

* **get\_shape():** Calculates the intersection of the shapes of the 2 objects belonging to this door.
* **get\_dimension():** Returns the dimension ot the intersection of the shapes of the 2 objects belonging to this door.
* **other\_room():** Returns the other room to p\_room if this is a door and the other room is a CompleteExpansionRoom. Else null is returned.
* **maze\_search\_element\_count():** Returns the count of MazeSearchElements in this expandable object
* **get\_maze\_search\_element():** Returns the p\_no-th MazeSearchElements in this expandable object
* **reset():** Resets this ExpandableObject for autorouting the next connection.

**ExpansionDrill.java:**

Layer change expansion object in the maze search algorithm.

* **calculate\_expansion\_rooms():** Looks for the expansion room of this drill on each layer. Creates a CompleteFreeSpaceExpansionRoom, if no expansion room is found. Returns false, if that was not possible because of an obstacle at this.location on some layer in the compensated search tree.
* **draw():** Test draw of the shape of this drill.

**ExpansionDoor.java:**

An ExpansionDoor is a common edge between two ExpansionRooms.

* **get\_shape():** Calculates the intersection of the shapes of the 2 rooms belonging to this door.
* **get\_dimension():** The dimension of a door may be 1 or 2. 2-dimensional doors can only exist between ObstacleExpansionRooms
* **other\_room():** Returns one of the 2 doors, or returns null.
* **get\_section\_segments():** Calculates the Line segments of the sections of this door.
* **calc\_door\_line\_segment():** Calculates a diagonal line of the 2-dimensional p\_door\_shape which represents the restraint line between the shapes of this.first\_room and this.second\_room.
* **reset():** Resets this ExpandableObject for autorouting the next connection.
* **allocate\_sections():** allocates and initialises p\_section\_count sections

**FreeSpaceExpansionRoom.java:**

Expansion Areas used by the maze search algorithm.

* **add\_door():** Adds given door to the list of doors of the defined room.
* **get\_doors():** Returns the list of doors of this room to neighbour expansion rooms
* **clear\_doors():** Removes all doors from this room.
* **door\_exists():** Checks, if this room has already a door

**IncompleteFreeSpaceExpansionRoom.java:**

An expansion room, whose shape is not yet completely calculated.

**InsertFoundConnectionAlgo.java:**

Inserts the traces and vias of the connection found by the autoroute algorithm.

**InsertFoundConnection.java:**

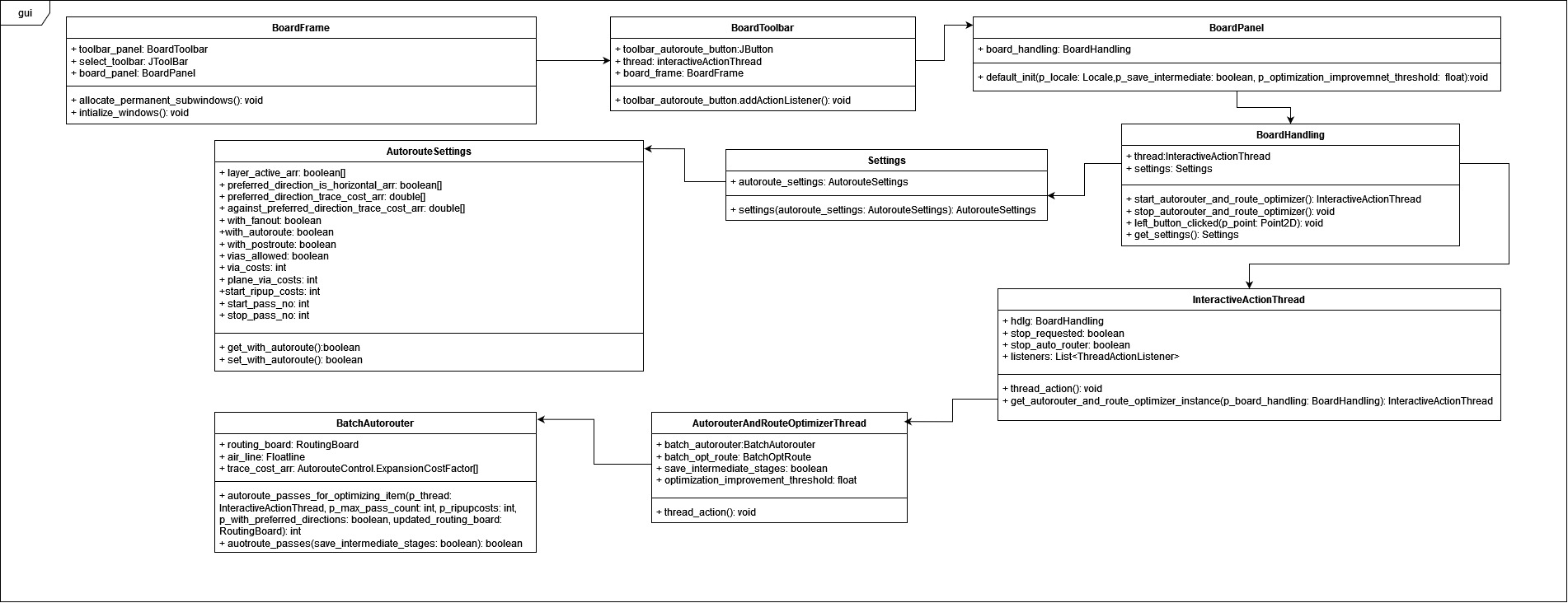
Inserts the traces and vias of the connection found by the autoroute algorithm.

* **insert\_trace():** Inserts the trace by shoving aside obstacle traces and vias. Returns false, that was not possible for the whole trace.
* **insert\_via():** Searches the cheapest via masks containing from one layer to another layer, so that a forced via is possible at a given location with this mask and inserts the via. Returns false, if no suitable via mask was found or if the algorithm failed.

**ItemSelectionStrategy.java:**

This defines various methods possible for ItemSelectionStrategy. Like Sequention, Random or Prioritized.

**Auto-router Backend Classes:**

****

The class diagram presented above is derived from the current application code and outlines the structural framework for implementing the auto-routing feature. This diagram will serve as a valuable reference for developing the auto-router in our new application.

It consists of the methods and classes that are behind the implementation of the auto-router in the application. Most of the classes relate to the supporting classes related to the auto-router, like auto-route settings, which help define parameters for the auto-routing process; board handling, which controls the board with which the user interacts and more.

**BoardFrame:**

Contains components of the auto-router application, like toolbar, MenuBar, File handling, controls and so on. This class defines the structure “Toolbar“ from where auto-router can be toggled to start auto-routing in the application.

**BoardToolbar:**

Contains components in the toolbar like Undo, Autoruter, Ratsnests, Zoom in, Zoom all etc. In this class we are interested in how the auto-rute is designed and what is called when it is toggled.

**BoardPanel:**

Contains the graphical representation of a routing board. This is essential for the app to make it user friendly. THe user is able to understand and visually look at the board due to this class.

**BoardHandling:**

This helps propagate changes on the board from the backend. Given the auto-router is toggled, the function start\_autoruter\_and\_route\_optimizer() is called. This class is one of the main part of the project which enables control of the auto-routing process

**Settings:**

THis class contains all the settings required for routing to be enabled in the application. This class extends autoroute settings. It also defines routing settings.

**AutorouteSettings:**

This class contains the autoroute settings parameter and detailed autoroute settings parameter. This defines a series of variables which are essential to enable auto-routing in the application like it controls fanout, post\_route, via\_costs, ripup\_costs, etc.

**InteractiveActionThread:**

This controls the process of execution for the auto-router, it helps connect a thread ot the auto-router(in case of single threaded auto-routing).

**BatchAutorouter:**

This contains the main code for batch autorouting, required which enables auto-routing, The autorouter main code, that is the maze expansions algorithm, control over fanout, autoroute engineer are called in this class.