**Exercise 1:**

A screenshot of a computer

Description automatically generated

**Drawing**

Attributes:

* **lines**: A set of lines, represented as a **LinkedHashSet** of **Line** objects.

Methods:

* **drawPolygon(LinkedHashSet<Line> lines):** Takes in a set of lines, creates a polygon, and prints out the polygon as well as its perimeter.
* **perimeter(Polygon p):** Takes in a polygon and calculates its perimeter by iterating over its constituent lines and summing their distances.
* **main(String[] args):** The main method which instantiates a **Drawing** object, some **Point** objects, and **Line** objects. It then forms drawings using the **drawPolygon** method.

**Polygon**

Attributes:

* **polygon:** A **LinkedHashSet** of **Line** objects that represent the lines that make up the polygon.
* **objID**: An integer ID assigned to each individual **Polygon** object. This ID is unique and increments for each **Polygon** instance.
* **it**: An **Iterator** over the set of lines in the polygon.
* **classID**: A static integer that keeps track of the number of **Polygon** objects created and assigns a unique ID to each.

Methods:

* **getLine():** Returns an iterator over the set of lines in the polygon.
* **classID():** A static method that returns the current value of **classID**.
* **toString():** Returns a string representation of the polygon and its constituent lines.

**Line**

Attributes:

* **start**: A **Point** object representing the start of the line.
* **end**: A **Point** object representing the end of the line.
* **objID**: An integer ID assigned to each individual **Line** object. This ID is unique and increments for each Line instance.
* **classID**: A static integer that keeps track of the number of Line objects created and assigns a unique ID to each.

Methods:

* **Line(Point a, Point b):** Constructor that initializes the line with a starting and ending point. The constructor also assigns a unique ID to the line.
* **distance():** Returns the distance between the start and end points of the line.
* **toString():** Returns a string representation of the line, its unique ID, and its start and end points.

**Point**

Attributes:

* **x**: An integer representing the x-coordinate of the point.
* **y**: An integer representing the y-coordinate of the point.

Methods:

* **Point(int x, int y):** Constructor that initializes the point with given x and y coordinates.
* **distance(Point a, Point b):** A static method that takes in two Point objects and returns the Euclidean distance between them.
* **toString():** Returns a string representation of the point, indicating its x and y coordinates.

**Relationship:**

1. The **Drawing** class has dependencies on the **Line, Polygon,** and **Point** classes.
2. Both **Line** and **Polygon** classes exhibit composition relationships, with **Line** being composed of **Point** objects and **Polygon** being composed of **Line** objects, indicating strong ownership and lifecycle dependencies between them.

**Exercise 2:**

A screenshot of a computer

Description automatically generated

**Exercise 3:**

A diagram of a computer

Description automatically generated with medium confidence

**InventoryManager**

Attributes:

* **tools:** A list that holds all tools in the inventory.
* **suppliers:** A list that holds all suppliers.
* **orders:** A list containing all the order lines made.

Methods:

* **addTools(tool: Tool):** Adds a new tool to the list of tools in the inventory.
* **deleteTool(toolId: int):** Deletes a tool from the inventory based on its ID.
* **searchByName(name: String):** Searches for tools in the inventory based on their names.
* **searchById(toolId: int):** Searches for a tool in the inventory based on its ID.
* **checkStock():** Checks the stock of each tool in the inventory. If stock goes below 40, it triggers the order process.
* **makeOrder():** Creates an order for tools with low stock.
* **loadToolsFromFile():** Loads the tools' data from the file into the tools list.
* **loadSuppliersFromFile():** Loads the suppliers' data from the file into the suppliers list.
* **exportOrder(orderData: HashMap):** Exports the order data to the orders.txt file.

**Tool**

Attributes:

* **toolId:** The unique ID of the tool.
* **name:** Name of the tool.
* **stock:** Number of units of the tool in stock.
* **price:** Price of the tool.
* **supplierId:** The ID of the supplier providing this tool.

Methods:

* **decreaseStock():** Decreases the stock of the tool.
* **increaseStock():** Increases the stock of the tool.

**Supplier**

Attributes:

* **supplierId:** The unique ID of the supplier.
* **company:** The name of the supplier company.
* **address:** The address of the supplier company.
* **contact:** The contact information for the supplier.

**OrderLine**

Attributes:

* **tools:** A list containing tools related to the order line.
* **suppliers:** A list containing suppliers related to the order line.
* **orders:** A list containing orders associated with the order line.

Methods:

* **runOrderLine():** Initiates the order line process.
* **makeOrder():** Creates an order.
* **makeItem():** Creates an item for the order.
* **searchTool():** Searches for a tool.
* **searchSupplier():** Searches for a supplier.

**Order**

Attributes:

* **orderId:** The unique ID of the order.
* **date:** The date when the order was placed.
* **items:** A list containing items in the order.

**Item**

Attributes:

* **name:** Name of the item.
* **supplier:** The supplier of the item.
* **quantity:** Number of units of the item.

**FileHandler**

Methods:

* {static} readToolsFromFile(): Reads tool data from a file and returns a list of tools.
* {static} readSuppliersFromFile(): Reads supplier data from a file and returns a list of suppliers.
* {static} writeOrderToFile(order: Order): Writes the given order data to a file.

Class Relationships:

1. **InventoryManager & Tool, Supplier, OrderLine**:

- The **InventoryManager** class serves as the main orchestrator for the inventory management system. It holds aggregations to the **Tool**, **Supplier**, and **OrderLine** classes.

- Aggregation is used here to denote that while the **InventoryManager** contains these entities, their lifetimes are not strictly bound to the manager.

1. **Tool & Supplier:**

- Each **Tool** has a direct association with a single **Supplier**. This implies that while each tool is sourced from a specific supplier, a single supplier can supply a variety of tools to the shop.

1. **OrderLine & Tool, Supplier, Order:**

- The **OrderLine** class aggregates **Tool** and **Supplier**. It also has a direct association with the **Order** class.

- An order line references multiple tools and suppliers, but they can exist independently of any order line.

1. **Order & Item:**

- The **Order** class is composed of multiple **Items**. This composition indicates a stronger relationship where an Order is fundamentally made up of items, and these items don't have meaningful existence outside of an order.

1. **InventoryManager & FileHandler:**

- The **InventoryManager** interacts with the **FileHandler** class for various file operations like loading tools, suppliers, and exporting orders.

**Exercise 4:**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**