Student: Leila Erbay
ID: 260672158
Assignment: 4

My class diagram does not create widget as its own object. I was considering doing this, but for the purpose of the case study, the company only cared about numbers thus I considered a widget to be an int. The class diagram does not show how the process would run as either an Employee wanting to use the system or a Manager wanting to use the system. The case study also does not go into further detail of how the customer would use the system. The class diagram only shows the methods and objects that would be implemented for the case study presented. If additional methods or objects are needed the class diagram would expand. But again, the class diagram does not show a sequence or the process of which users would interact with the system, such as the steps that proceed after an Employee chooses an option from the menu or chooses an option that is meant only for Managers, and similarly the class diagram does not show the process for the Manager after choosing to see the widget production history or if he chooses an option meant only for employees. Another type of diagram would be needed to show these steps.

For Q1, the optimal UML Diagram, in my opinion, is an activity diagram. An activity diagram is used to demonstrate different case uses. In this problem, there are many players using the system, specifically Employees and Manager(s), and if the problem were to be further developed, Customer(s) could possibly interact within their appropriate parts of the company's system. Employee, Manager and Customer would each have a starting point within their swim lane and they would each have different processes that could proceed.

Below is a rough description of how the activity diagram may appear.
The activity diagram would have three main categories: Person, System, Record.
Within Person, there would be sub categories labelled: Employee, Manager and Customer.
Since the System is meant to work for different type of Employee or Manager (or even further developed, Customer), there would be a starting point in Employee, Manager, and Customer.

With a starting point in **Employee**, the next actions that would occur:

- arrow pointing to system with the action of *menu appearing*
- Employee chooses an option, one of the following: punchIn, punchOut, employee Punch History, widget production history quit

- *punchIn* is chosen from the menu:

1	- activity: work in System is called	Loc: System
2	- activity: validateEmployeeID called inside work	Loc: System
3	- activity: validateID	Loc: Employee
4	-depending on response of validation ask for ID again, or:	Loc: System
5	-activity: punchIn	Loc: Employee
6	- activity: ask user if multiple widgets will be made else assume 1	Loc: Employee

Student: Leila Erbay
ID: 260672158
Assignment: 4

7	- activity: getRawMaterial	Loc: Employee
8	-activity: incrRawMaterial	Loc: System
9	- activity: createWidgets	Loc: Employee
10	- activity: shipWidgets(:Customer)	Loc: Employee
11	- activity: punchOut appears from menu	Loc: System
12	- activity: punchOut()	Loc: Employee
13	- activity: createRecord	Loc: Employee
14	-activity: record being created	Loc: Record
15	-activity: addRecord	Loc: Employee
16	- activity: scanner asks user if they would like to create more widgets	Loc: Employee
17	- activity: if more widgets are to be created, start at 5 and do it all again	Loc: Employee
18	- activity: if no more widgets to be created, go to menu	Loc: System

- *punchOut* is chosen from the menu:

1	- act: validateEmployeeID is called	Loc: System
2	- act: validateID is called	Loc: Employee
3	-depending on response of validation, ask for ID again or	Loc:System
4	-act: check if User has already punchedOut if has not yet punched out, punchOut() is called if user already punchedOut, tell user he has to punch in before punching out	Loc: Employee
5	-act: menu appears	Loc: System

- *Employee Punch History* is chosen from the menu:

1	- act: validateEmployeeID	Loc: System
2	- act: validateID	Loc: Employee
3	-depending on response of validation ask for ID again, or:	Loc: System

Student: Leila Erbay
ID: 260672158
Assignment: 4

4 - act: exportRecords Loc: Employee
5 - act: ask for a date the Employee would like to see Loc: System
6 - act: based on range, display punch times from records from HashTable of single Employee Loc: System
7 -act: display menu again Loc: System

- *widget production history* is chosen from the menu
1 - act: validateManagerID Loc: System
2 - act: validateID Loc: Manager
3 - because only managers seem to have the right to this option, the system would tell the employee his/her error Loc: System
4 - act: menu appears again Loc: System

- *quit* is chosen from the menu
1 -act: program ends Loc: System

With a starting point in **Manager**:

- menu appears: punchIn, punchOut, employee punch history, widget production history, quit

- *punchIn, punchOut, employee punch history* is chosen
1 act: validateEmployeeID Loc: System
2 act: validateID Loc: Employee
3 assuming these are only actions for employees, Manager will be told his/her error Loc: System
4 act: menu appears again Loc: System

- *widget production history* is chosen
1 act: validateManagerID Loc: System
2 act: validateID Loc: Manager
3. depending on the input by the user, ask for idea again or Loc: System
4. ask user for date range Loc: System
5. act : if databaseEmployeeRecords does not have 20 key values then loop through all Employees and generate all sets of records and fill databaseEmployeeRecords Loc: System
6. act: Loc: System

Student: Leila Erbay
ID: 260672158
Assignment: 4

	loop through each employee and determine the total number of widgets they created within date range	
6b	act: print each employee's id and # of widgets created within date range [occurs with step 6]	Loc: System
7	act: when loop is done, return to menu	Loc: System

- *quit* is chosen

1	-act: program ends	Loc: System