**Clock System - Tasks**

The Clock system models a timer and multiple clocks. This system is for training purposes only.

* **Check implemented features**

Select each implemented feature for the move time functionality:

|  |  |
| --- | --- |
|  | Sets hour to zero if hour is 24. |
|  | Shows milliseconds. |
|  | Represent a second passing time by sleeping the thread a second. |
|  | Clocks states are updated in every new second. |
|  | Move time functionality increments timer. |
|  | Option to reset time. |

* **Sort execution flow**

For the move time functionality, we ask you to sort the given list of steps to reflect the execution flow written in the source code. Fill in the first column with the appropriate order number when applicable.

**Note:** have in mind that there might be some incoherent step order purposely introduced in the code to avoid you finding a correct answer without reading the source code and just following your common sense.

|  |  |
| --- | --- |
| Order | Steps involved in spell checker system functionality. |
|  | Create a analog clock. |
|  | Adds clock a a list. |
|  | Each clock state gets updated. |
|  | Create a digital clock. |
|  | Increments timer one second. |
|  | Sleeps one second. |

* **Look for changed and unchanged objects**

Fill in the table using one row per participant object’s type, also add the number of objects for such type, and write yes or no whether those objects change state during the execution of the system functionality that moves time.

|  |  |  |
| --- | --- | --- |
| Object’s type | Number of objects | Change |
| Clocktimer |  |  |
| Clock |  |  |

* Describe objects interactions

Describe the interaction between objects/classes by filling in the following matrix. Write a cross when there exists an interaction initiated by object/class named in row requesting service from object/class named in column for the functionality that model a food shortage situation (executed in Main.java at line 6). Between brackets is the file and line number for the objects/classes of interest, just follow each specific object/class to discover its interactions.

|  |  |  |  |
| --- | --- | --- | --- |
|  | [Main.java line 3]  Clocktimer | [Main.java line 4]  DigitalClock | [Main.java line 5]  AnalogClock |
| [Main.java line 3]  Clocktimer | n/a |  |  |
| [Main.java line 4]  DigitalClock |  | n/a |  |
| [Main.java line 5]  AnalogClock |  |  | n/a |

**Product sales system - Tasks**

The Product sales system models an hypothetical situation of a food shortage. This system is for training purposes only.

* **Check implemented features**

The functionality to acquire all products (executed in Main.java at line 10) models a hypothetical situation of food shortage. Select each feature the functionality for acquiring all products implements:

|  |  |
| --- | --- |
|  | Person acquires a product. |
|  | Store sells products. |
|  | Get name of a product. |
|  | Person to string. |
|  | Person acquires all products. |
|  | Store raises product price. |

* **Sort execution flow**

The functionality to acquire all products (executed in Main.java at line 10) models a hypothetical situation of food shortage. Sort the given list of steps to reflect the execution flow written in the source code for the acquire all products functionality. Fill in the first column with the appropriate order number when applicable:

**Note:** have in mind that there might be some incoherent step order purposely introduced in the code to avoid you finding a correct answer without reading the source code and just following your common sense.

|  |  |
| --- | --- |
| Order | Steps involved in food shortage system functionality. |
|  | The store removes a product. |
|  | The store adds a product. |
|  | The person acquires a product. |
|  | The store gives the person the number of available products in the store. |
|  | The system prints store’s state. |
|  | The store gets a product from stock. |
|  | The system prints person’s state. |

* **Look for changed and unchanged objects**

Fill in the table using one row per participant class or object’s type for the acquire-all-products functionality (executed in Main.java at line 10), also add the number of objects for such type and write whether those objects change state during the execution of the functionality:

|  |  |  |
| --- | --- | --- |
| Object’s type | Number of objects | Change |
| Profiteer |  |  |
| Store |  |  |
| Product |  |  |

* **Describe objects interactions**

Describe the interaction between objects/classes by filling in the following matrix. Write a cross when there exists an interaction initiated by object/class named in row requesting service from object/class named in column for the functionality that model a food shortage situation (executed in Main.java at line 10). Between brackets is the file and line number for the objects/classes of interest, just follow each specific object/class to discover its interactions.

|  |  |  |
| --- | --- | --- |
| OO | [Main.java line 3]  **new Profiteer();** | [Main.java line 4]  **new Store();** |
| [Main.java line 3]  **new Profiteer();** | n/a |  |
| [Main.java line 4]  **new Store();** |  | n/a |