SWT Handin 3 - Microwave Oven

Group 4 handin information

	Name	Student Number
1	Atren Amanoel Darvesh	201405993
2	Simon Hjortgaard Bos	201910459
3	Mathias Birk Olsen	202008722
4	Oliver Vestergaard Schousboe	202008211

Table of contents

Git

GitHub link

Git Structure

We have decided to use `Feature branching` for our git workflow.

Git branches

There are a total of 4 branches. The different brances are:

1. main
2. feature/buzzer
3. feature/powerChange
4. feature/timeChange

Tags

Add tags on features for milestones such as:

1. When the solution can compile
2. Partial features
3. Addition of tests WHERE IT MAKES SENSE 😀

Remember to commit before tag

1. `git add -A`
2. `git commit -m "{Commit Message}"`
3. `git tag -a {tagID} -m "{tagMessage}"`

Rebase features before merge.

Commands for rebasing:

1. `git switch feature/[featureName]`
2. `git rebase main`
3. `git switch main`
4. `git merge feature/[featureName]`

Jenkins jobs

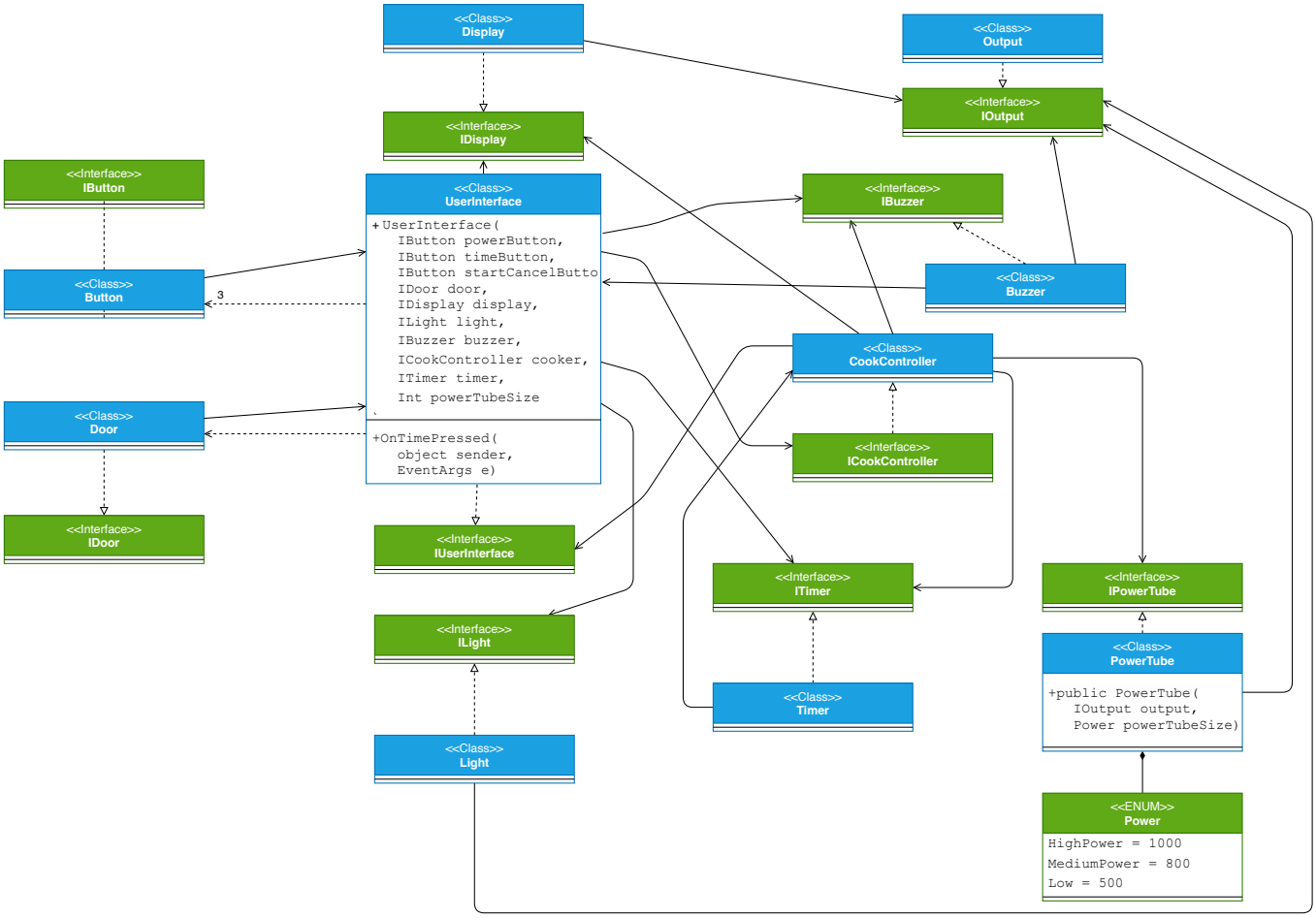
There are a total of 4 jobs - one for each branch. Links to the different jobs:

1. team04E22-microwave-main

2. team04E22-microwave-feature-buzzer
3. team04E22-microwave-feature-powerChange
4. team04E22-microwave-feature-timeChange

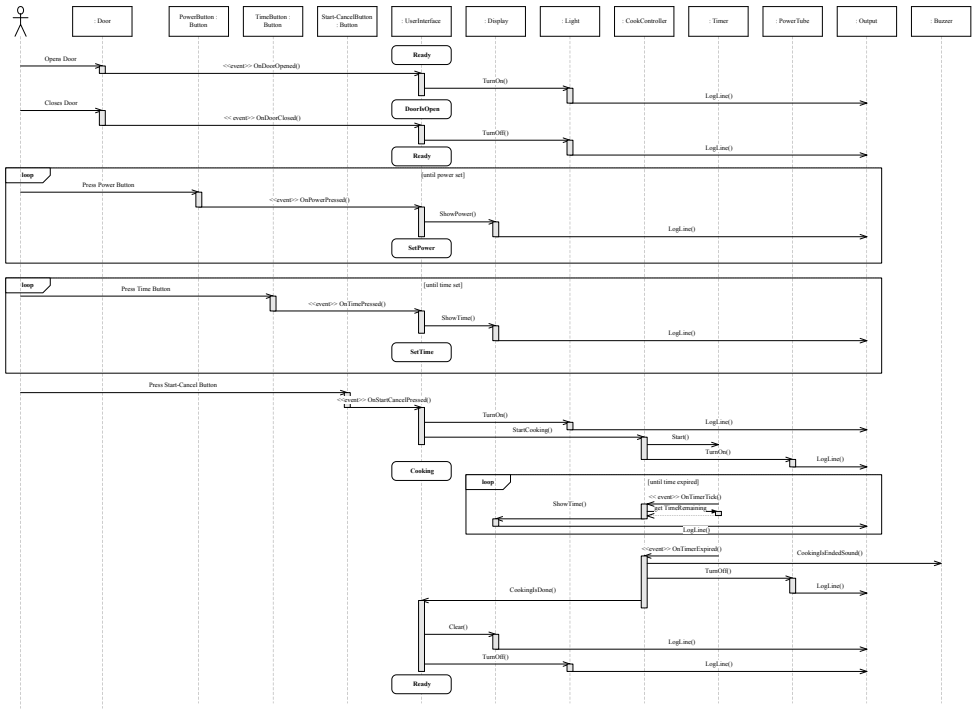
Class diagram

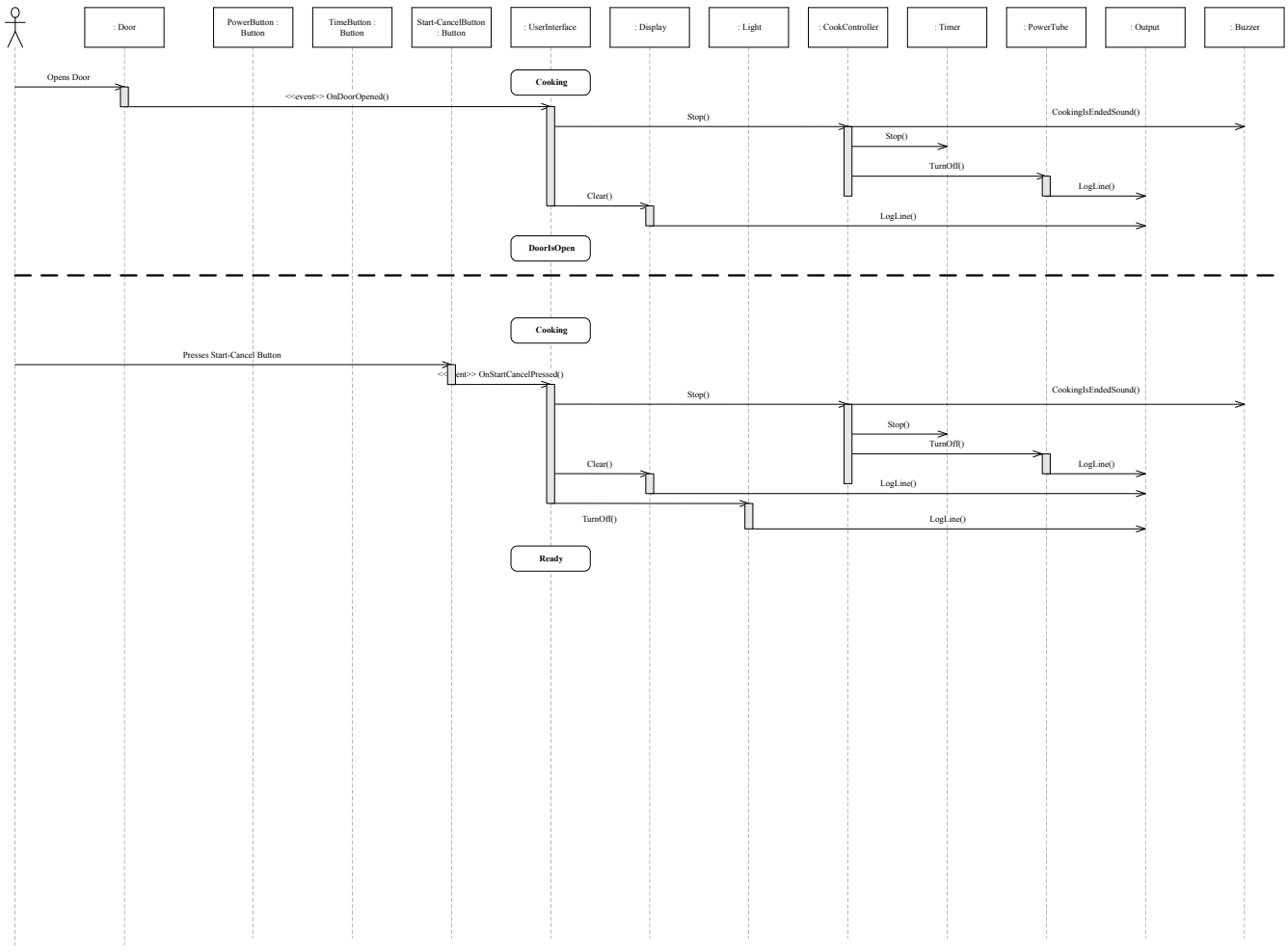
It was decided to use the existing class diagram as a base and only add any changes we made to it.



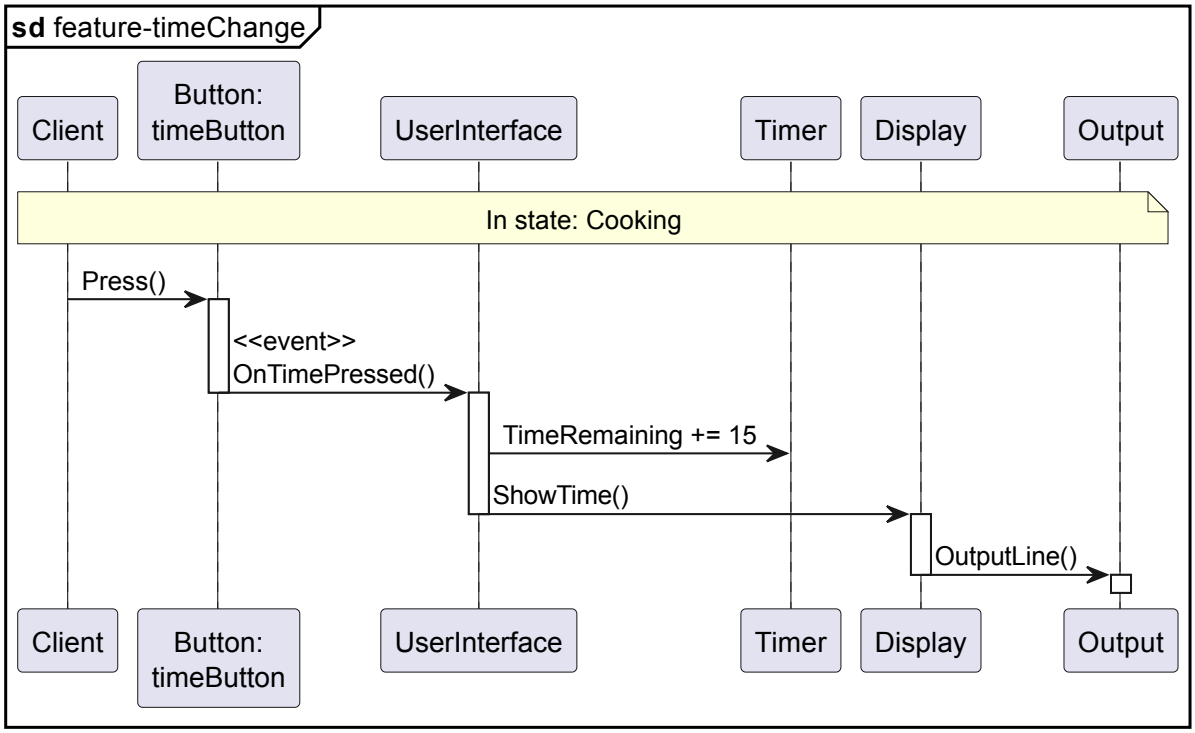
Sequence diagrams

Buzzer


: Door
PowerButton : Button
TimeButton : Button
Start-CancelButton : Button
: UserInterface
: Display
: Light
: CookController
: Timer
: PowerTube
: Output
: Buzzer
Opens Door
<<event>> OnDoorOpened()
Ready
TurnOn()
LogLine()
Closes Door
<< event>> OnDoorClosed()
DoorIsOpen
TurnOff()
Ready
LogLine()
loop
[until power set]
Press Power Button
<<event>> OnPowerPressed()
ShowPower()
LogLine()
SetPower
loop
[until time set]
Press Time Button
<<event>> OnTimePressed()
ShowTime()
LogLine()
SetTime
Press Start-Cancel Button
<<event>> OnStartCancelPressed()
TurnOn()
LogLine()
StartCooking()
Start()
TurnOn()
LogLine()
Cooking
loop
[until time expired]
<< event>> OnTimerTick()
ShowTime()
Get TimeRemaining
LogLine()
<<event>> OnTimerExpired()
CookingIsEndedSound()
TurnOff()
LogLine()
CookingIsDone()
Clear()
LogLine()
TurnOff()
LogLine()
Ready


: Door
PowerButton : Button
TimeButton : Button
Start-CancelButton : Button
: UserInterface
: Display
: Light
: CookController
: Timer
: PowerTube
: Output
: Buzzer
Opens Door
Cooking
<<event>> OnDoorOpened()
Stop()
CookingIsEndedSound()
Stop()
TurnOff()
LogLine()
Clear()
LogLine()
DoorIsOpen
Cooking
Presses Start-Cancel Button
<<event>> OnStartCancelPressed()
Stop()
CookingIsEndedSound()
Stop()
TurnOff()
Clear()
LogLine()
LogLine()
TurnOff()
LogLine()
Ready

Time change

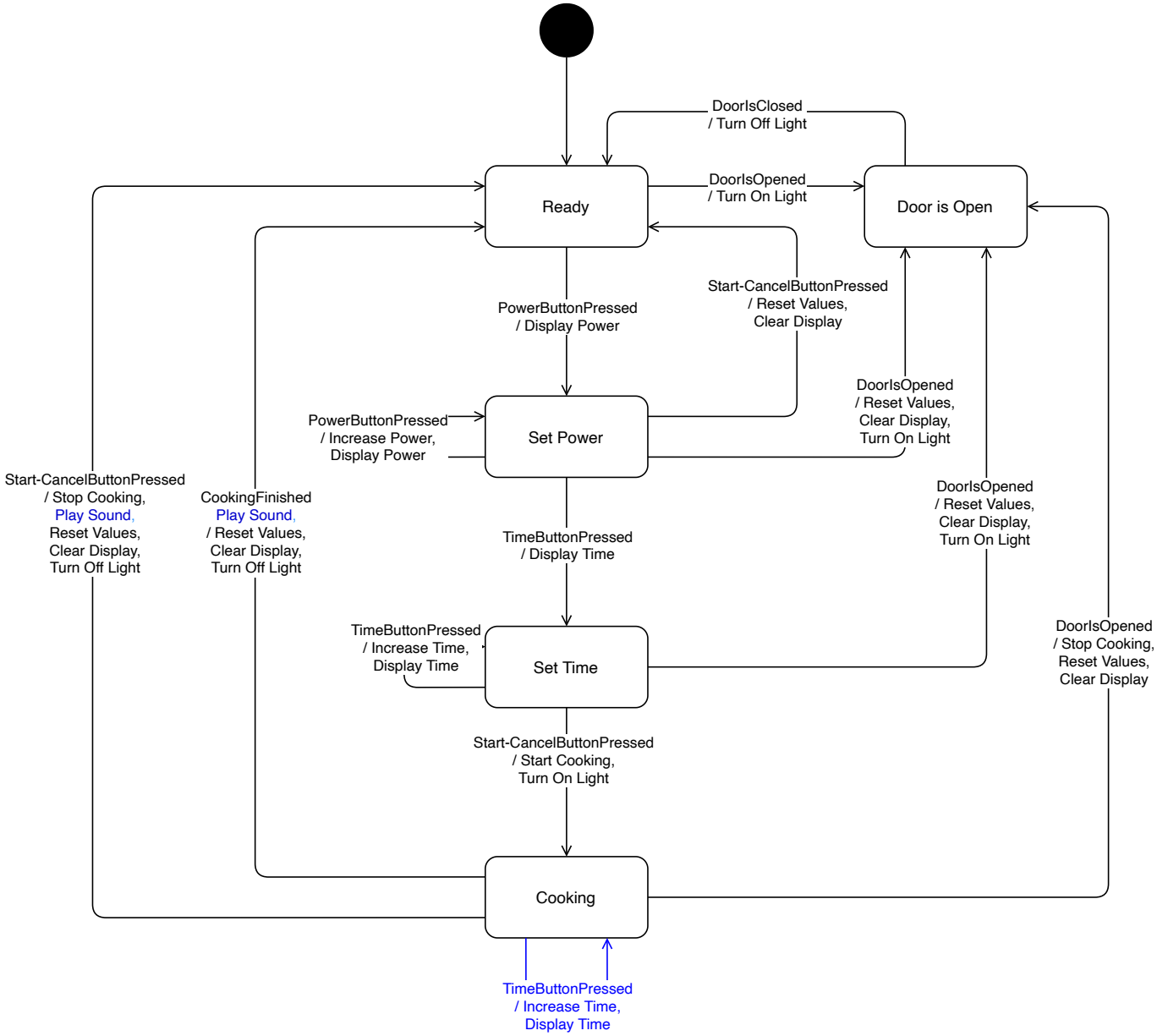


Power change

there is no behavioral change added with the power tube Refractoring

Updated state machine diagram

It was decided to copy the diagram from the handout and extend it with the new features. Additions are colored blue.



Description of changes and additions

Power Change

The power change feature requirements where that the microwave should be able to be buildt with different Watt sizes of powertubes. The first change was then simply to add a property to the powerTube class. This property was restricted to a fixed amout of enums, to ensure the power level fittet with real power tubes. The checks involving the max watt level where changed to depent on this property. This entralied enjecting the value in the user interface aswell.

Since there now is two dependentsies involving the same configuration value, an argument for a dedicted factory methoed can be made. This would ensure that the oven would be configured correctly. This is an inprovement that can be made in later iterations.

Updating tests

The power change feature demanded change to two classes, the UI and the powerTupe its self. The UI has a lot of tests, where only some of them are dependt on the configuration.

Two aprroces can be taken, with in the scope of this exercise. The first being that each test gets the resposibility of arrange the uut. The second being that the set still is respossibel for creating the uut, but the test dependent on the configuration reassing the uut it needs.

The second approce creats a redundent piese of code, but it ensure that the tests are a bit more maintaineble. Since there is fewer changes needed to be made, in case the config is changed agian.

Both cases however, are not ideal. This is a brilliant exsample of the importences of a good design from the get go. Becuase a late change to the inital design, demands an aboundece of change.

Funcy test errors

The chosen approce of overriding the uut in the test that need varibal configuration, introduces a very strange error. evt. it was discovered that the uut is was replaced, but its dependencties had registred with the different event handlers. Now, once an event was raised, both the onld and the new event handlers reacted, causing each call to be called twice.

To deal with this problem, a uut factory methoed was created. This factory created all the dependentsies and the uut as new object, ensuring all the references to the old object was broken.

Buzzer

There has been implemented a buzzer class that has the responsibility of making a 3 burst sound whenever Cooking is done. This is obtained by making an interface and a class that implements this interface which has 1 function "CookingIsEndedSound()". CookingController.cs and UserInterface.cs have been altered so as it also takes a buzzer object in its constructor parameter. The function CookingIsEndedSound() is then inserted in CookController.OnTimerExpired(), CookController.Stop() and UserInterface.CookingIsDone().

In the tests the test.integration project was unloaded so to avoid testings conflicts. The BuzzerTest.cs tests the simple functionality of the buzzer class. It creates a Trace object which can catch console outputs when running the tests. This output is then asserted with the expected string value "Ding, Ding, Ding! Cooking Done\a".

There has also been written a functional test in the CookingControllerTest.cs. Cooking_stopsound_CookingDone() tests if the CookingIsEndedSound() through the uut (CookController.StartCooking() function).

Time change

The only things added to the code are

- `UserInterface` has a reference to the `Timer` to make it possible to add time while cooking.
- In the `UserInterface` state machine (switch case), the logic for adding time while in state `cooking` was addded.
- The `Timer` classes property TimeRemaining was changed to have a public setter instead of the private in the original.