CSE 443 MIDTERM PROJECT

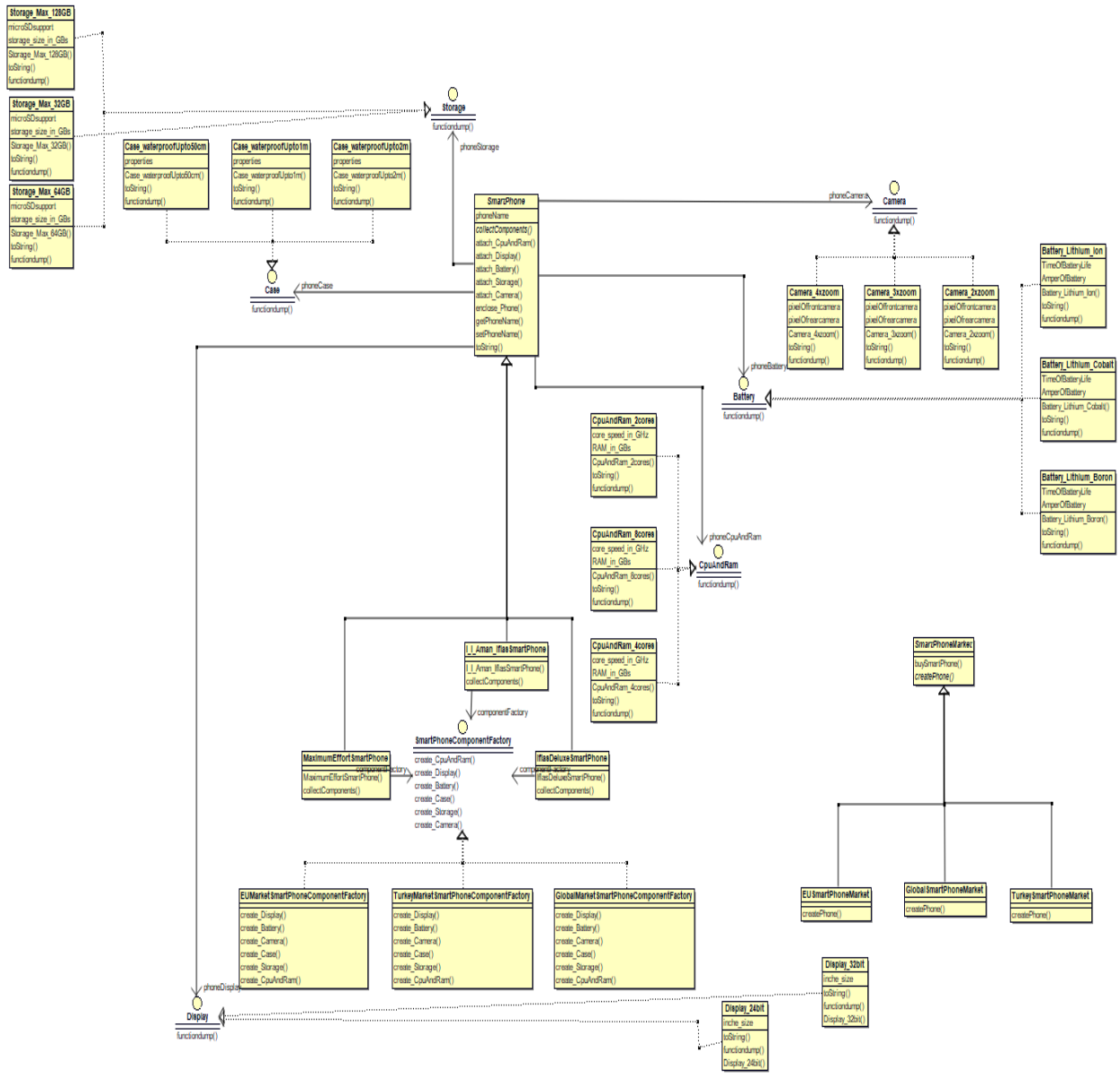
REPORT

PART-1

First, As mentioned in the homework pdf, I used the abstract factory design pattern. I have SmartPhoneComponentFactory interface like PizzaIngredientsFactory in the book. I have created regional ComponentFactory classes according to each regional market and these classes are implementing SmartPhoneComponentFactory interface. I have created an abstract SmartPhone class. It holds objects for phone parts in this class. This class have method named as collectComponents(). This method takes the necessary parts of phone from the relevant regional component factory. I have an abstract SmartPhoneMarket class like pizza store in the book. This class has buySmartPhone() and abstract createPhone() methods. Every smart phone model must extends from this SmartPhone class. And every regional market class (regional pizza store) must extend from SmartPhoneMarket class. Every phone model gives his regional smartPhoneComponentFactory. I have display, storage, case vs interfaces. These interfaces are implemented by concrete material classes that will have some different properties. Thus, regional component factories will be able to choose the material they want.

Each phone model gives the properties that do not change according to the region to the constructur of the ComponentFactory class of the region where it will be sold. In the regional component Factory class, it returns the materials together with the features that vary according to the region.

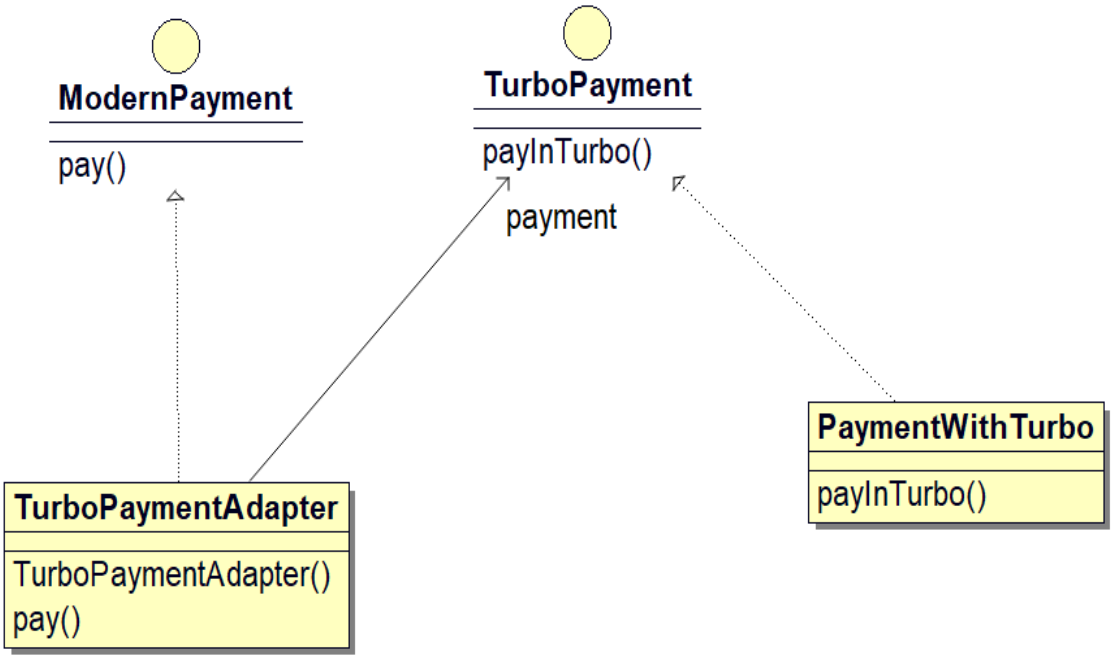
Uml Diagram:


Storage_Max_128GB
microSDsupport
storage_size_in_GBs
Storage_Max_128GB()
toString()
functiondump()
Storage_Max_32GB
microSDsupport
storage_size_in_GBs
Storage_Max_32GB()
toString()
functiondump()
Storage_Max_64GB
microSDsupport
storage_size_in_GBs
Storage_Max_64GB()
toString()
functiondump()
Storage
functiondump()
phoneStorage
Case_waterproofUpto50cm
properties
Case_waterproofUpto50cm()
toString()
functiondump()
Case_waterproofUpto1m
properties
Case_waterproofUpto1m()
toString()
functiondump()
Case_waterproofUpto2m
properties
Case_waterproofUpto2m()
toString()
functiondump()
Case
functiondump()
phoneCase
SmartPhone
phoneName
collectComponents()
attach_CpuAndRam()
attach_Display()
attach_Battery()
attach_Storage()
attach_Camera()
enclose_Phone()
getPhoneName()
setPhoneName()
toString()
phoneCamera
Camera
functiondump()
Camera_4xzoom
pixelOffrontcamera
pixelOfrearcamera
Camera_4xzoom()
toString()
functiondump()
Camera_3xzoom
pixelOffrontcamera
pixelOfrearcamera
Camera_3xzoom()
toString()
functiondump()
Camera_2xzoom
pixelOffrontcamera
pixelOfrearcamera
Camera_2xzoom()
toString()
functiondump()
Battery_Lithium_Ion
TimeOfBatteryLife
AmperOfBattery
Battery_Lithium_Ion()
toString()
functiondump()
Battery_Lithium_Cobalt
TimeOfBatteryLife
AmperOfBattery
Battery_Lithium_Cobalt()
toString()
functiondump()
Battery_Lithium_Boron
TimeOfBatteryLife
AmperOfBattery
Battery_Lithium_Boron()
toString()
functiondump()
phoneBattery
Battery
functiondump()
CpuAndRam_2cores
core_speed_in_GHz
RAM_in_GBs
CpuAndRam_2cores()
toString()
functiondump()
CpuAndRam_8cores
core_speed_in_GHz
RAM_in_GBs
CpuAndRam_8cores()
toString()
functiondump()
phoneCpuAndRam
CpuAndRam
functiondump()
CpuAndRam_4cores
core_speed_in_GHz
RAM_in_GBs
CpuAndRam_4cores()
toString()
functiondump()
I_I_Aman_IflasSmartPhone
I_I_Aman_IflasSmartPhone()
collectComponents()
componentFactory
SmartPhoneComponentFactory
create_CpuAndRam()
create_Display()
create_Battery()
create_Case()
create_Storage()
create_Camera()
MaximumEffortSmartPhone
MaximumEffortSmartPhone()
collectComponents()
componentFactory
IflasDeluxeSmartPhone
IflasDeluxeSmartPhone()
collectComponents()
componentFactory
SmartPhoneMarket
buySmartPhone()
createPhone()
EUSmartPhoneMarket
createPhone()
GlobalSmartPhoneMarket
createPhone()
TurkeySmartPhoneMarket
createPhone()
EUMarketSmartPhoneComponentFactory
create_Display()
create_Battery()
create_Camera()
create_Case()
create_Storage()
create_CpuAndRam()
TurkeyMarketSmartPhoneComponentFactory
create_Display()
create_Battery()
create_Camera()
create_Case()
create_Storage()
create_CpuAndRam()
GlobalMarketSmartPhoneComponentFactory
create_Display()
create_Battery()
create_Camera()
create_Case()
create_Storage()
create_CpuAndRam()
phoneDisplay
Display
functiondump()
Display_32bit
inche_size
toString()
functiondump()
Display_32bit()
Display_24bit
inche_size
toString()
functiondump()
Display_24bit()

PART-2

I have created a TurboPaymentAdapter class that implements ModernPayment for Turbo payment. Thus, the user can use class objects that implement the old turboPayment interface as if they were ModernPayment objects with turboPaymentAdapter.

Uml Diagram:



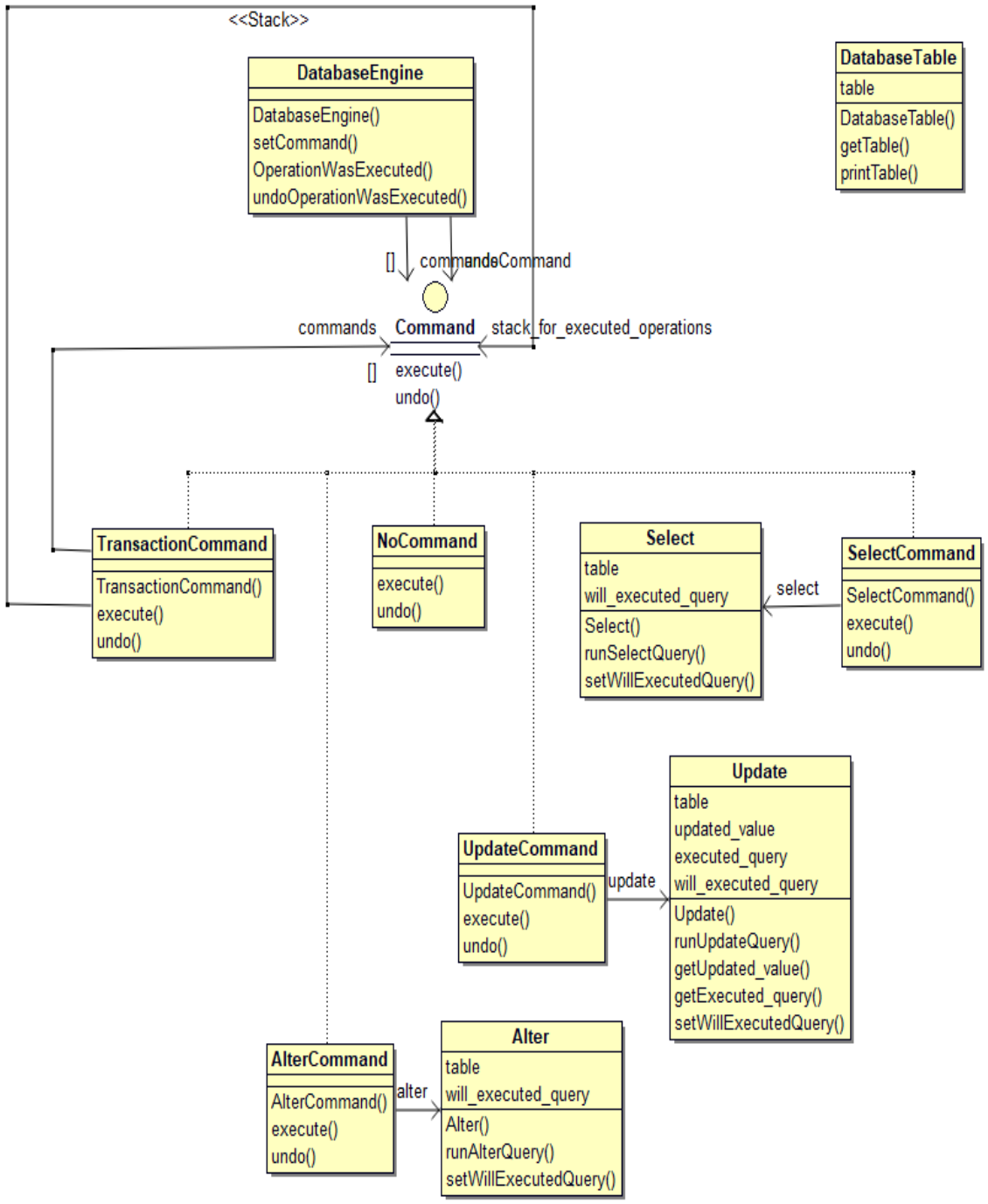
PART-3

First of all, i used Command Pattern because we need to assign operations to the database engine slots without knowing the behaviour of the operation and the invoker must have more than one operation holding capacity.

I just implemented the update operation fully from the operations. In other operations, I only performed printing. Because I could not establish the exact logic of the other operations working alone. So I designed them to do symbolic processing. To keep the database simple, I set it as a 2-dimensional string array. I created select, alter, and update classes. This classes hold an database table (2 dimesional string array) in themselves. And they have related queryRun methods. These methods make changes on the database. But again, I am saying that the classes related to update operation have the capacity to make changes on the database. Others are just in this case structurally. And i have AlterCommand, SelectCommand, UpdateCommand classes that implements the Command interface. In main, an update object created with parameter “String Query”. Then a updateCommand object is created with update object. So it keep update object in itselfs. Then this updateCommand object is placed in the slot with the setCommand function of the databaseEngine (Invoker) object. Then the transaction is done by operating the slot button. Again, the process can be undone by calling undo command over the invoker.

I created a class called TransactionCommand. This class was written to be able to run multiple commands, similar to the MacroCommand class in the book. This is necessary given that transactions occur from more than one operation. Inside the TransactionCommand class, there is a stack that holds Commands that executed properly. If all Commands in a TransactionCommand do not run and terminate correctly, the undo () method of all Commands in the stack will be executed in the method undo() of TransactionCommand. Thus, the answer to the problem asked in the part b in the homework pdf was given.

Uml Diagram:


<<Stack>>
DatabaseEngine
DatabaseEngine()
setCommand()
OperationWasExecuted()
undoOperationWasExecuted()
DatabaseTable
table
DatabaseTable()
getTable()
printTable()
[]
commands
Command
stack_for_executed_operations
execute()
undo()
TransactionCommand
TransactionCommand()
execute()
undo()
NoCommand
execute()
undo()
Select
table
will_executed_query
Select()
runSelectQuery()
setWillExecutedQuery()
select
SelectCommand
SelectCommand()
execute()
undo()
Update
table
updated_value
executed_query
will_executed_query
Update()
runUpdateQuery()
getUpdated_value()
getExecuted_query()
setWillExecutedQuery()
UpdateCommand
UpdateCommand()
execute()
undo()
update
AlterCommand
AlterCommand()
execute()
undo()
alter
Alter
table
will_executed_query
Alter()
runAlterQuery()
setWillExecutedQuery()

PART-4

Since both transform have almost the same algorithm structure, an algorithm is written in which the main operation is fixed. The function of reading inputs in this algorithm is implemented in the abstract Transform1D class, since both transform can work with that. Because both transform read inputs as real numbers. Other methods cant work for both two transform. So they must be implemented in concrete transform classes.

In addition, DFT is also said that sometimes the user may want to suppress the execution time. For this, there will be a hook method at the end of the main stream algorithm. By default, this method will return "false". However, DFT will implement this hook method in the concrete block and enable it to return according to the result received from the user.

Uml Dıagram: