Object oriented analysis and design

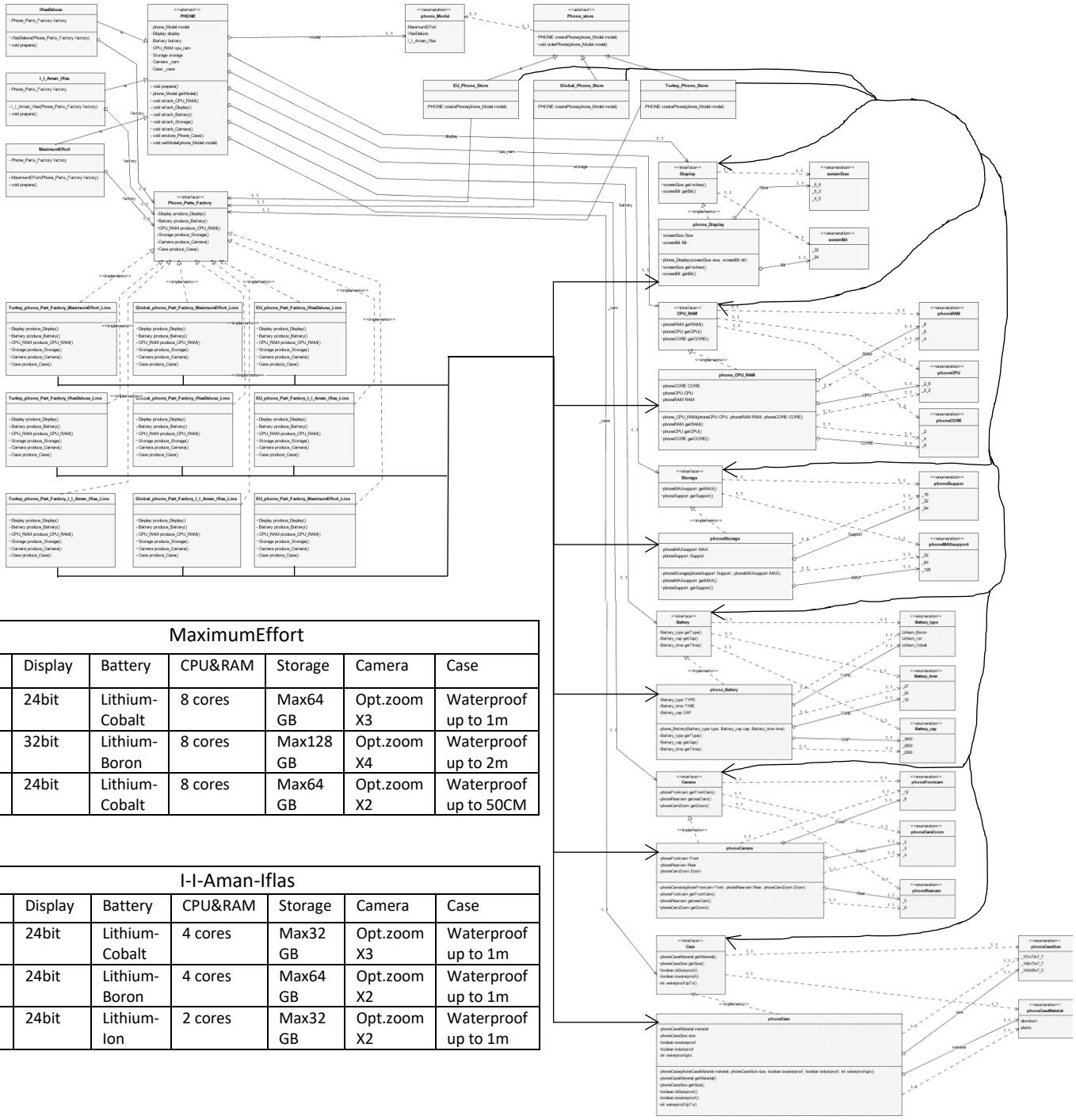
CSE443

#MIDTERM

Jwan hussein

151044078

Q1)



MaximumEffort						
Market	Display	Battery	CPU&RAM	Storage	Camera	Case
Turkye	24bit	Lithium-Cobalt	8 cores	Max64 GB	Opt.zoom X3	Waterproof up to 1m
EU	32bit	Lithium-Boron	8 cores	Max128 GB	Opt.zoom X4	Waterproof up to 2m
Global	24bit	Lithium-Cobalt	8 cores	Max64 GB	Opt.zoom X2	Waterproof up to 50CM

I-I-Aman-Iflas						
Market	Display	Battery	CPU&RAM	Storage	Camera	Case
Turkye	24bit	Lithium-Cobalt	4 cores	Max32 GB	Opt.zoom X3	Waterproof up to 1m
EU	24bit	Lithium-Boron	4 cores	Max64 GB	Opt.zoom X2	Waterproof up to 1m
Global	24bit	Lithium-Ion	2 cores	Max32 GB	Opt.zoom X2	Waterproof up to 1m

The creator classes :

```
public abstract class Phone_store
public class EU_Phone_Store extends Phone_store
public class Turkey_Phone_Store extends Phone_store
public class Global_Phone_Store extends Phone_store
```

the product classes :

```
public abstract class PHONE
public class MaximumEffort extends PHONE
public class I_I_Aman_Iflas extends PHONE
public class IflasDeluxe extends PHONE
```

Abstract Factory Interface :

```
public interface Phone_Parts_Factory
```

Concrete Factory classes :

```
public class EU_phone_Part_Factory_I_I_Aman_Iflas_Line implements Phone_Parts_Factory
public class EU_phone_Part_Factory_IflasDeluxe_Line implements Phone_Parts_Factory
public class EU_phone_Part_Factory_MaximumEffort_Line implements Phone_Parts_Factory
public class Global_phone_Part_Factory_MaximumEffort_Line implements Phone_Parts_Factory
public class Global_phone_Part_Factory_I_I_Aman_Iflas_Line implements Phone_Parts_Factory
public class Global_phone_Part_Factory_IflasDeluxe_Line implements Phone_Parts_Factory
public class Turkey_phone_Part_Factory_I_I_Aman_Iflas_Line implements Phone_Parts_Factory
public class Turkey_phone_Part_Factory_IflasDeluxe_Line implements Phone_Parts_Factory
public class Turkey_phone_Part_Factory_MaximumEffort_Line implements Phone_Parts_Factory
```

each class represent phone components production line for a region and a phone model , for example if `Turkey_Phone_Store` order a phone of model `MaximumEffort` then the component of the phone must be provided by `Turkey_phone_Part_Factory_MaximumEffort_Line` .

Abstract Components interfaceses :

```
public interface Battery
public interface Camera
public interface Case
public interface CPU_RAM
public interface Display
public interface Storage
```

Concrete Components classes

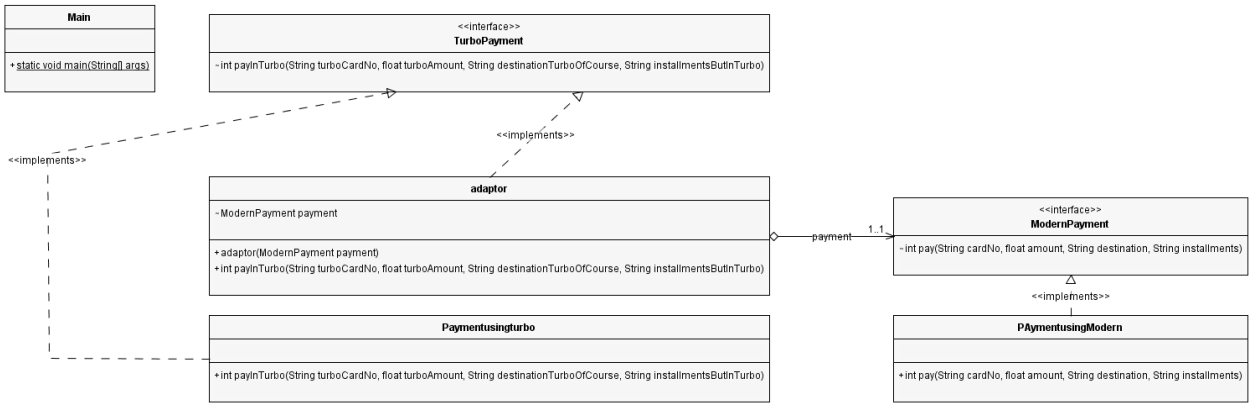
```
public class phoneStorage implements Storage
public class phone_Display implements Display
public class phone_CPU_RAM implements CPU_RAM
public class phoneCase implements Case
public class phone_Battery implements Battery
public class phoneCamera implements Camera
```

following an order from Turkey_Phone_Store for a MaximumEffort model .

1) First we need a Turkey_phone_store :
   Phone_store trstore = new Turkey_Phone_Store();
2) Taking the order
   trstore.orderPhone(phone_Model.MaximumEffort);
3) The orderPhone() method first calls the createPhone() method
   PHONE phone = createPhone(phone_Model.MaximumEffort);
4) When the createPhone() method is called , that is when the abstract factory gets involved
   PHONE ph = new MaximumEffort(new Turkey_phone_Part_Factory_MaximumEffort_Line());
5) Once the prepare() method is called the factory is asked to prepare the components
6) Finally
   a) attach cpu & ram to theboard
   b) attach display
   c) attach battery
   d) attach storage
   e) attach camera and
   f) enclose the phone case.

Any new model can be added to the system by adding the name of the model to phone_Model enumerator and building Concrete Factory classes related to the new model .

Q2)



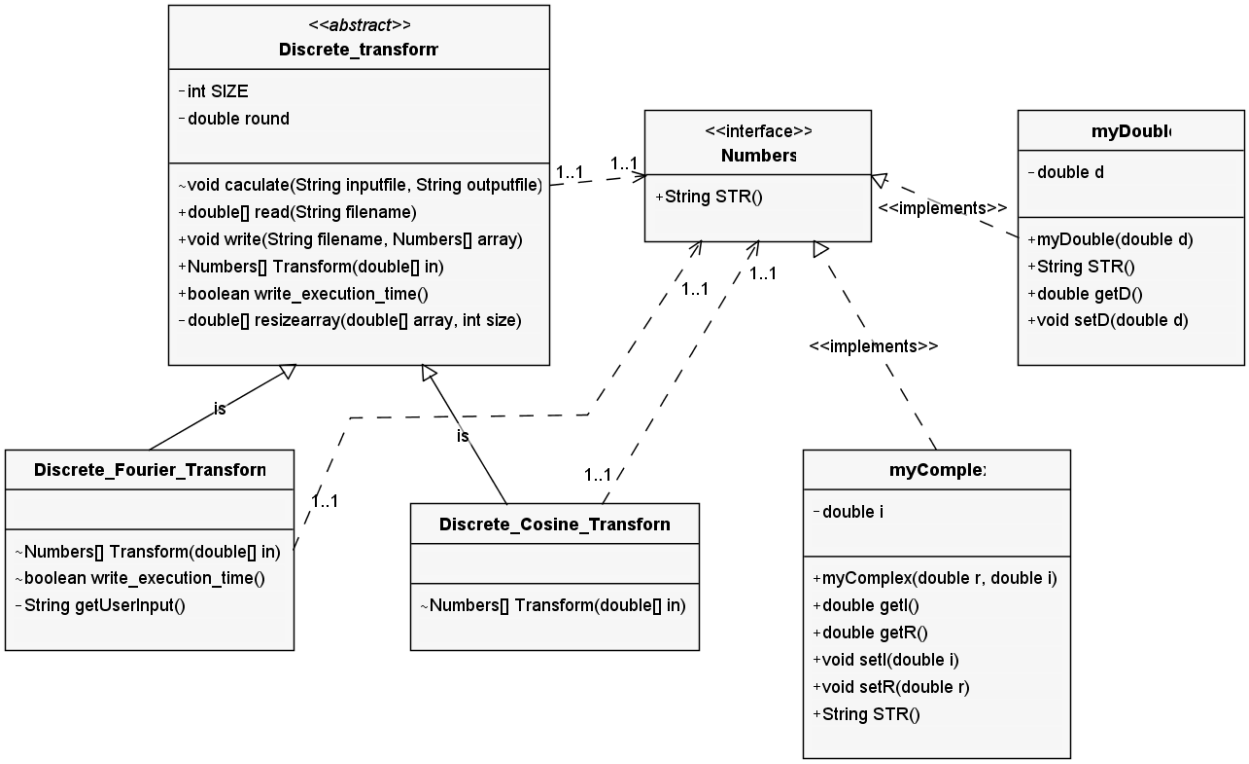
Since we can not change the old library we will need to create a new class that act like an adaptor between the old library and the new one .

the adaptor class will implement the old library .

the adaptor calss contains an object of the new library type , and the adaptor class will implement the method payInTurbo(4) .

the payInTurbo(4) method in the adaptor class will delegate the payment to the ModernPayment object by calling payment.pay(4) , so whenever adaptor.payInTurbo(4) is called the method ModernPayment.pay(4) will be executen in the background .

Q4)



The Discrete Fourier Transform and the Discrete Cosine Transform are following pretty much the same main procedure.
first step read the numbers from the file
second step transform the numbers into N outputs (DFT or DCT)
third step write the output to the file
abstract class discrete_transform

```
final void caculate(String inputfile , String outputfile)
```

the first parameter is the path to the input file
the second parameter is the path to the output file
the method is final so no child of the class can overwrite it
the method follows the following steps
1) check if the user want to print the time of the execution on the screen
If true then start the timer , if false then skip to the next step
2) reads the number from the input file .
3) calls Transform method .
4) write the output to the file
5) print the time of the execution if the user asked for .

```
public double[] read(String filename)
```

read the number from the file to an array

```
public void write(String filename , Numbers[] array)
```

write the array of numbers to the output file

```
public boolean write_execution_time()
```

this method returns false by default means that the user don't want to print the execution time on the screen

any child class can overwrite this method .

```
public abstract Numbers[] Transform(double[] in);
```

this is the only step that differs so it has been declared as abstract

returns an array of Numbers

The Discrete Fourier Transform and the Discrete Cosine Transform classes are overwriting the method Transform() .

The Discrete Fourier Transform class overwrite the method write_execution_time() and ask the user to print the execution time or not .