Gebze Technical University
Computer Engineering

Object Oriented Analysis and Design

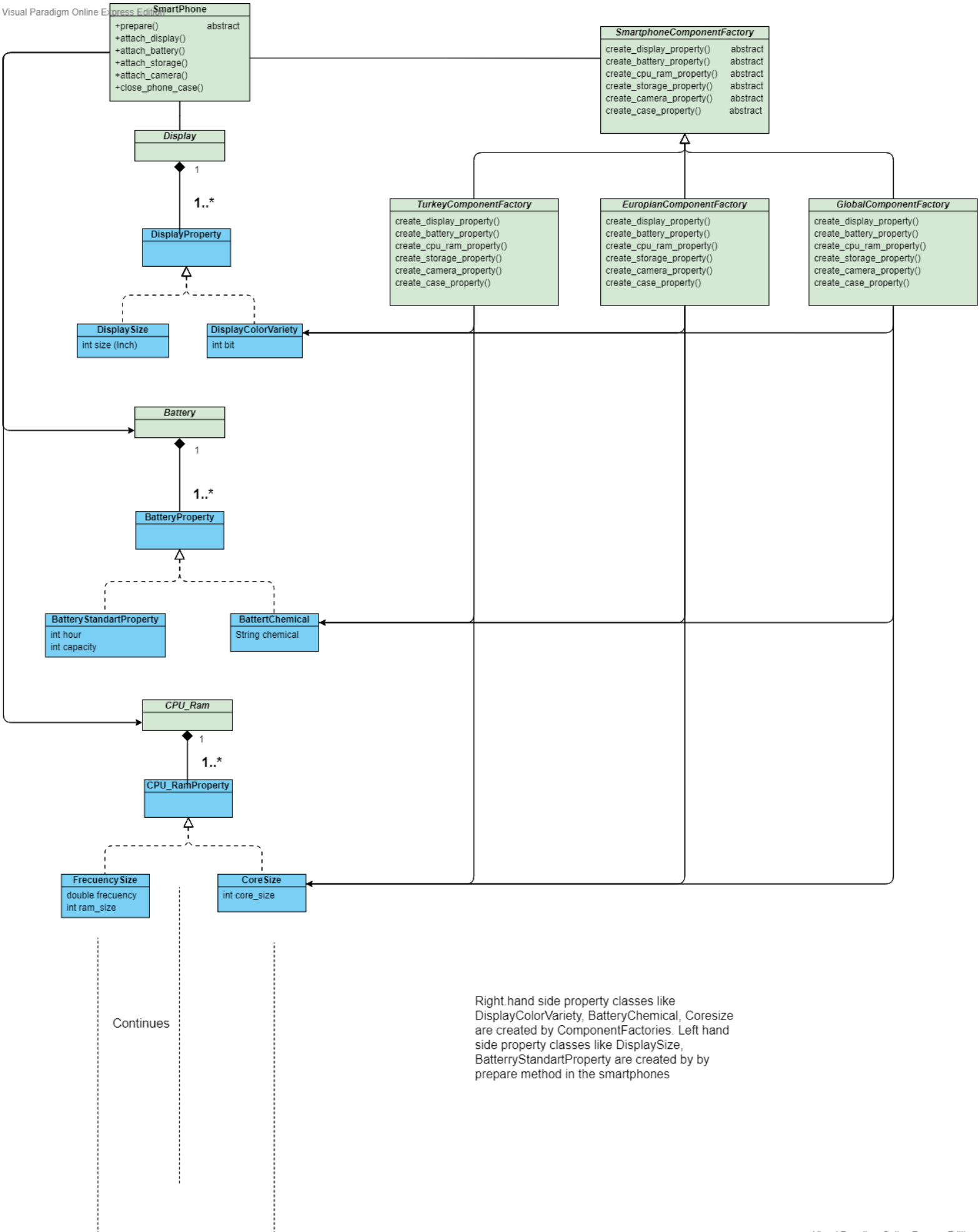
CSE443 – 2020

MIDTERM-REPORT

Yusuf Abdullah ARSLANALP

151044046

Question 1


Visual Paradigm Online Express Edition
SmartPhone
+prepare() abstract
+attach_display()
+attach_battery()
+attach_storage()
+attach_camera()
+close_phone_case()
SmartphoneComponentFactory
create_display_property() abstract
create_battery_property() abstract
create_cpu_ram_property() abstract
create_storage_property() abstract
create_camera_property() abstract
create_case_property() abstract
Display
1
1..*
DisplayProperty
TurkeyComponentFactory
create_display_property()
create_battery_property()
create_cpu_ram_property()
create_storage_property()
create_camera_property()
create_case_property()
EuropianComponentFactory
create_display_property()
create_battery_property()
create_cpu_ram_property()
create_storage_property()
create_camera_property()
create_case_property()
GlobalComponentFactory
create_display_property()
create_battery_property()
create_cpu_ram_property()
create_storage_property()
create_camera_property()
create_case_property()
DisplaySize
int size (Inch)
DisplayColorVariety
int bit
Battery
1
1..*
BatteryProperty
BatteryStandartProperty
int hour
int capacity
BattertChemical
String chemical
CPU_Ram
1
1..*
CPU_RamProperty
FrecuencySize
double frecuency
int ram_size
CoreSize
int core_size
Continues
Right.hand side property classes like DisplayColorVariety, BatteryChemical, Coresize are created by ComponentFactories. Left hand side property classes like DisplaySize, BatterryStandartProperty are created by by prepare method in the smartphones
Visual Paradigm Online Express Edition

```
ProductionLine PL_Turkey = new TurkeyProducionLine();
Smartphone ph1 = PL_Turkey.order_phone( model: "MaximumEffort" );
```

```
ProductionLine PL_eurpe = new EuropeProductionLine();
Smartphone ph4 = PL_eurpe.order_phone( model: "MaximumEffort" );
```

I used abstract factory design pattern for creating smartphone objects. In the above I showed how I created Smartphone objects in main. To create smartphone object we need ProductionLine object.

In below I showed ProductionLine abstract class.

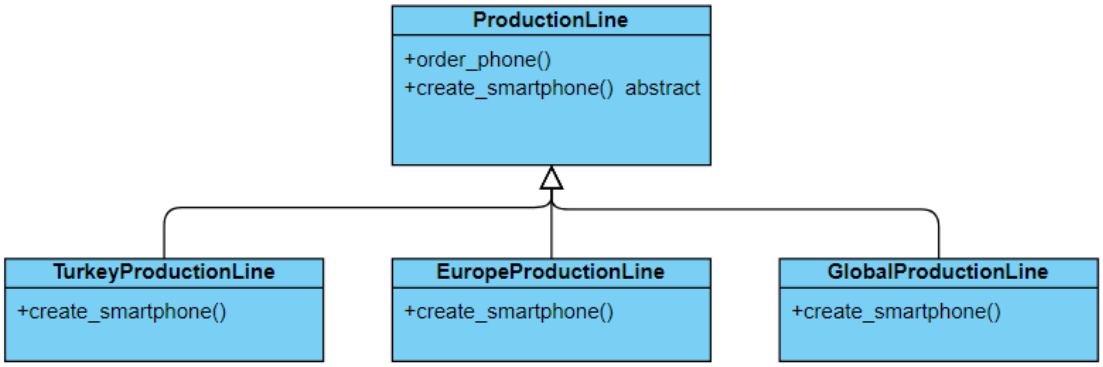
```
public abstract class ProductionLine {
    Smartphone order_phone( String model )
    {
        Smartphone phone = create_smartphone( model );

        phone.attach_cpu_ram();
        phone.attach_display();
        phone.attach_battery();
        phone.attach_storage();
        phone.attach_camera();
        phone.close_phone_case();

        return phone;
    }

    abstract Smartphone create_smartphone( String model );
}
```

Three production line class extends abstract class ProductionLine. Class diagram given below.



Every production line class implements their own create function. Create function of TurkeyProductionLine class given below.

```
public class TurkeyProducionLine extends ProductionLine {
    @Override
    Smartphone create_smartphone(String model) {
        Smartphone phone = null;
        SmartphoneComponentFactory  SCompoent_factory = new TurkeyComponentFactory();

        if( model.equalsIgnoreCase( anotherString: "MaximumEffort" ) )
        {
            phone = new MaximumEffort( SCompoent_factory );
        }
        else if( model.equalsIgnoreCase( anotherString: "IflasDeluxe" ) )
        {
            phone = new IflasDeluxe( SCompoent_factory );
        }
        else if( model.equalsIgnoreCase( anotherString: "I-I-Aman-Iflas" ) )
        {
            phone = new IIAmanIflas( SCompoent_factory );
        }
        return phone;
    }
}
```

We can see from the above MaximumEffort class takes factory object as parameter. And it uses factory object to create smartphone components.

We can see from the below how battery object in a phone class created. The below code snippet is belongs to the prepare function of MaximumEffort class.

```
BatteryProperty battery_market_p;
BatteryProperty battery_model_p;
battery_market_p = componentFactory.create_battery_property();
battery_model_p = new BatteryStandartProperty( hour: 27,  capacity: 3600 );
battery.add_property( battery_model_p );
battery.add_property( battery_market_p );
```

Evaluating Maintainability

Suppose that we want to make a new smartphone with wireless charge property. The phone name is ExclusiveBattery. To do that we just need to write two extra class. One of them is ExclusiveBattery class. And other of them is BatteryWirelessProperty class. We add two line code to prepare function of ExclusiveBattery class. And we have a new brand smartphone with wireless charge property.

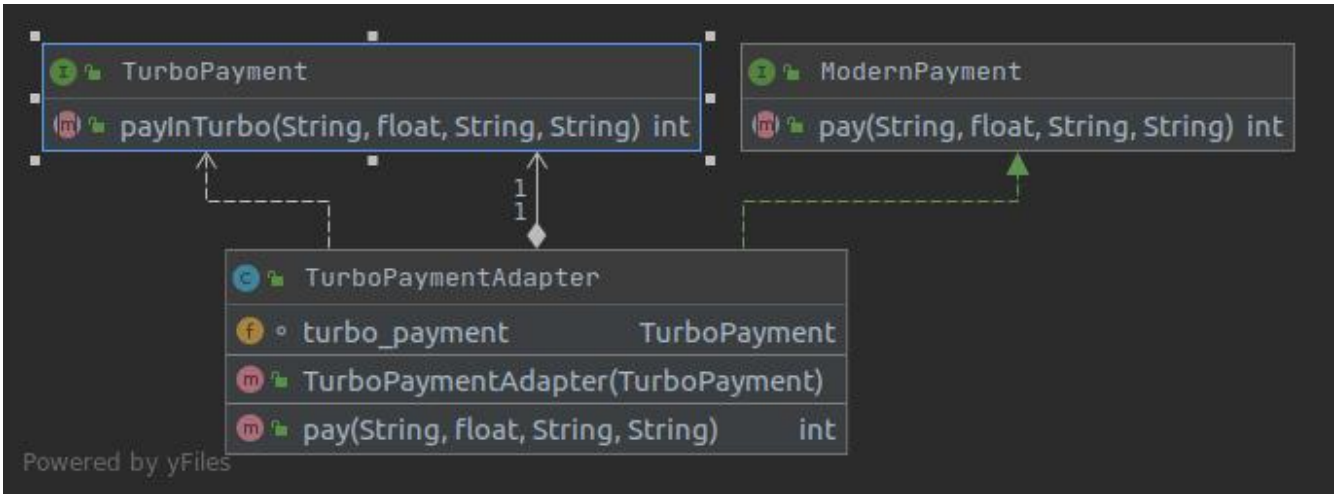
```
property()
{
        .
        .
        .
        .

    BatteryProperty wireless_charge = new BatteryWirelesspProperty();
    battery.add_property( wireless_charge );
        .
        .
        .
        .
}
```

Question 2

I used adapter design pattern to solve this problem. In this design I have a TurboPaymentAdapter class that implements ModernPayment. TurboPaymentAdapter class has a TurboPayment reference. So that with TurboPaymentAdapter class, all classes that implement TurboPayment can be used by ModernPayment interface.

The class diagram is in below



Adapter Class:

```java
public class TurboPaymentAdapter implements ModernPayment {
    TurboPayment turbo_payment;

    public TurboPaymentAdapter(TurboPayment turbo_payment) {
        this.turbo_payment = turbo_payment;
    }

    @Override
    public int pay(String cardNo, float amount, String destination, String installments) {
        return turbo_payment.payInTurbo( cardNo, amount, destination, installments );
    }
}
```

```java
public interface ModernPayment {
    int pay(String cardNo, float amount, String destination, String installments);
}
```

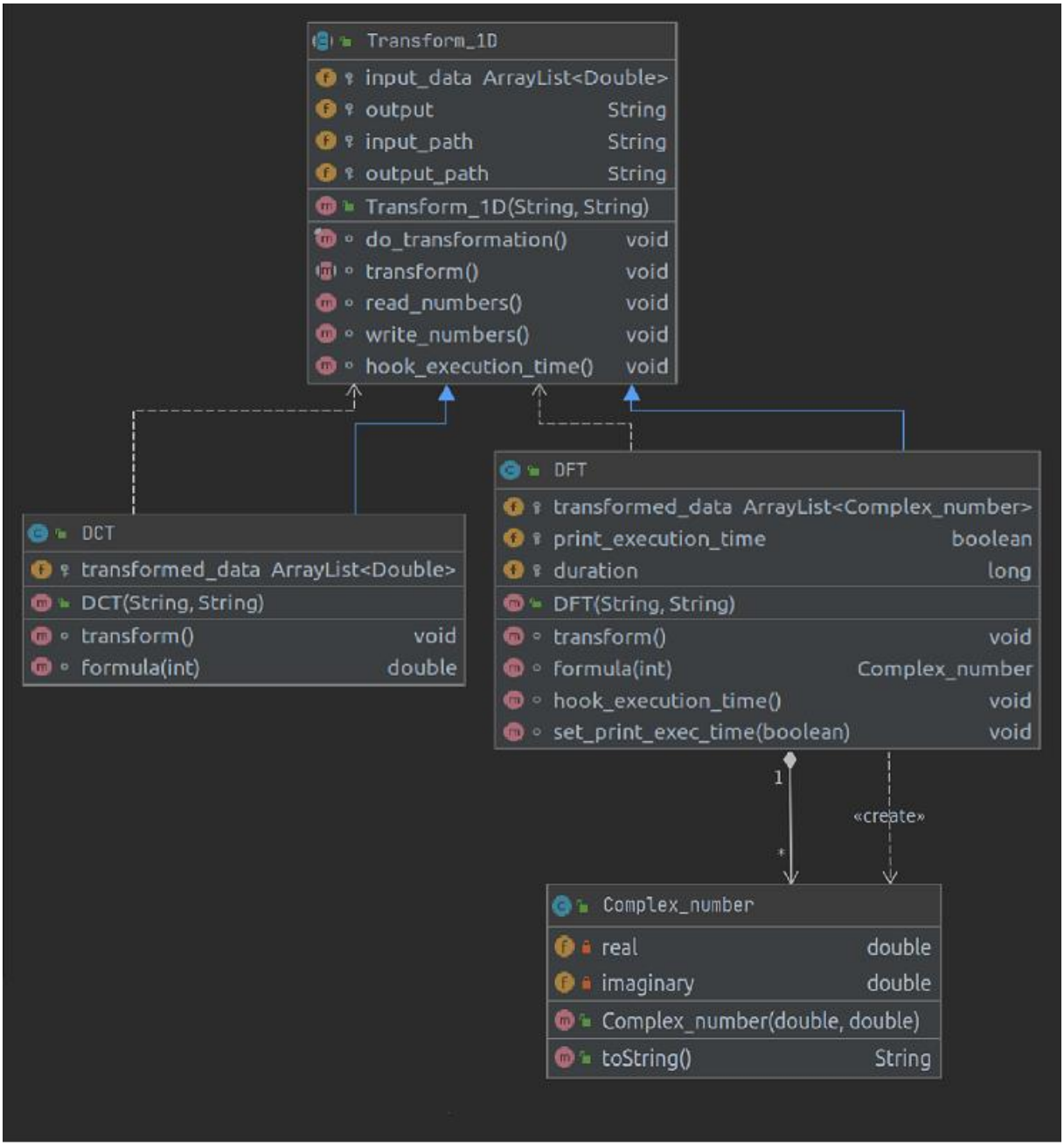
```java
public interface TurboPayment {
    int payInTurbo(String turboCardNo, float turboAmount,
                   String destinationTurboOfCourse, String installmentsButInTurbo);
}
```

```java
public class ChildOfTurboPayment implements TurboPayment {
    @Override
    public int payInTurbo(String turboCardNo, float turboAmount,
                          String destinationTurboOfCourse, String installmentsButInTurbo)
    {
        System.out.println( "I imlemented TurboPayment Interface" );
        System.out.println( "ModernPayment interface can use me with adapter design patern" );
        return 0;
    }
}
```

An example for use of adapter class is in below. Using ModernPayment reference we used an object that implement TurboPayment.

```java
public static void main(String[] args) {
    ChildOfTurboPayment child_of_turbo = new ChildOfTurboPayment();
    TurboPaymentAdapter convert_to_modern = new TurboPaymentAdapter( child_of_turbo );
    ModernPayment modern_payment_reference = convert_to_modern;
    modern_payment_reference.pay( cardNo: "string", amount: 100, destination: "string", installments: "string" );
}
```

Question 4



DFT class implements Discrete Fourier Transform and DCT class implement Discrete Cosine Transform. Both DFT and DCT extends abstract class Transform_1D. Both DFT and DCT classes inherit read_numbers and write_numbers functions from Transform_1D parent class. transform function implemented as abstract in Transform_1D class. Every concrete class that implements Transform_1D must implement transform function.

Both DFT and DCT consists of three steps. In parent class Transform_1D do_transformation function calls these steps. Implementation of do_transformation function are given below. **do_transformation function defined as final. So child classes can't override it.**

```
final void do_transformation()
{
    read_numbers();
    transform();
    write_numbers( );
    hook_execution_time();
}
```

In the Transform_1D function hook_execution_time do nothing. DCT class does not override hook_execution_time function. But DFT class override it.

DFT class has a boolean field print_execution_time. By default, it is false. When user wants to print execution time it sets print_execution_time field true. hook_execution_time function prints execution time only when print_execution_time field is true.

```
@Override
void hook_execution_time() {
    if( print_execution_time )
    {   //convert nanosecond to milisecond
        duration = duration / 1000000;
        System.out.println( "Time of execution: " + duration + " milliseconds" );
    }
}
```