Solver: Dilion Amadeo

Email Address: DILL0002@e.ntu.edu.sg

1. (a)
   (i) Same message can be sent to different objects
   (ii) Sending object does not need to know class of receiving object or how object will respond

   (b)
   (i) Attribute: When an attribute is declared as final, it's value cannot be changed (constant)
   Example: final int number = 1;
   (ii) Method: When a method is declared as final, the method cannot be overridden in subclasses.
   Example: final int strictMethod() { return 1; }
   (iii) Class: When a class is declared as final, the class cannot be extended.
   Example: final class soloClass { int solo; }

   (c)

Number	Line	Class Method	Type of Casting	Outcome
(i)	1	-	Upcasting	RUNTIME OK
	2	-	Upcasting	RUNTIME OK
	3	ClassC	-	RUNTIME OK
	4	ClassB	-	RUNTIME OK
(ii)	1	-	Upcasting	RUNTIME OK
	2	-	Upcasting	RUNTIME OK
	3	-	Downcasting	RUNTIME OK
	4	-	-	Compile Error
(iii)	1	-	Upcasting	RUNTIME OK
	2	-	Downcasting	RUNTIME OK
	3	-	Upcasting	RUNTIME OK
	4	-	Downcasting	RUNTIME OK
	5	ClassG	-	RUNTIME OK

2.

```
import java.util.ArrayList;
import java.util.Date;

public class Order {
    private String cashierName = NULL;
    private Date date;
    private int maxAllowedItems;
    private int totalItems = 0;
    private ArrayList lineItemList = NULL;

    public Order(String cname, int mcap, Date date)
    {
        cashierName = cname;
        maxAllowedItems = mcap;
        this.date = date;
        lineItemList = new ArrayList<LineItem>();
    }

    public String getCashierName()
    {
        return cashierName;
    }

    public Date getDate()
    {
        return date;
    }

    public int getMaxAllowedItems()
    {
        return maxAllowedItems;
    }

    public int getMaxAllowedItems()
    {
        return maxAllowedItems;
    }
```

```
public Boolean addLineItem(LineItem t)
    {
        if (totalItems == maxAllowedItems || lineItemList.size()
== maxAllowedItems || t == NULL) {
            return false;
        }
        int indexOfItem = lineItemList.indexOf(t);
        if (indexOfItem == -1) {
            return NULL;
        }
        LineItem temp = lineItemList.get(indexOfItem);
        int newQty = temp.getQty() + t.getQty();
        temp.setQty(newQty);
        lineItemList.remove(temp);
        lineItemList.add(temp);
        totalItems++;
        return true;
    }
    public Boolean removeLineItem(LineItem t)
    {
        if (lineItemList == NULL || lineItemList.size() == 0 ||
totalItems == 0 || t == NULL) {
            return false;
        }

        if (!lineItemList.contains(t)) {
            return false;
        }

        lineItemList.remove(t);
        totalItems--;
        return true;
    }
```

```
public LineItem findLineItem(LineItem t)
    {
        if (lineItemList == NULL || lineItemList.size() == 0 ||
totalItems == 0 || t == NULL) {
            return NULL;
        }

        int indexOfItem = lineItemList.indexOf(t);

        if (indexOfItem == -1) {
            return NULL;
        }

        return lineItemList.get(indexOfItem);
    }

    public double calcTotalPrice()
    {
        if (lineItemList == NULL || totalItems == 0) {
            return 0.0;
        }

        double totalPrice = 0.0;
        for (LineItem t : lineItemList) {
            totalPrice += t.getPrice();
            t.toString();
        }

        return totalPrice;
    }
}
```

3. A.

```
#include <iostream>
#include <string>

using namespace std;

class ClassA
{
public:
    ClassA();
    ~ClassA();
    virtual void processA() = 0;
    virtual void load(string f){};

};

class ClassC : public InterfaceB, ClassA
{
public:
    ClassC();
    ~ClassC();
    void send(string s){};
    void processA(){};
};
```

B.

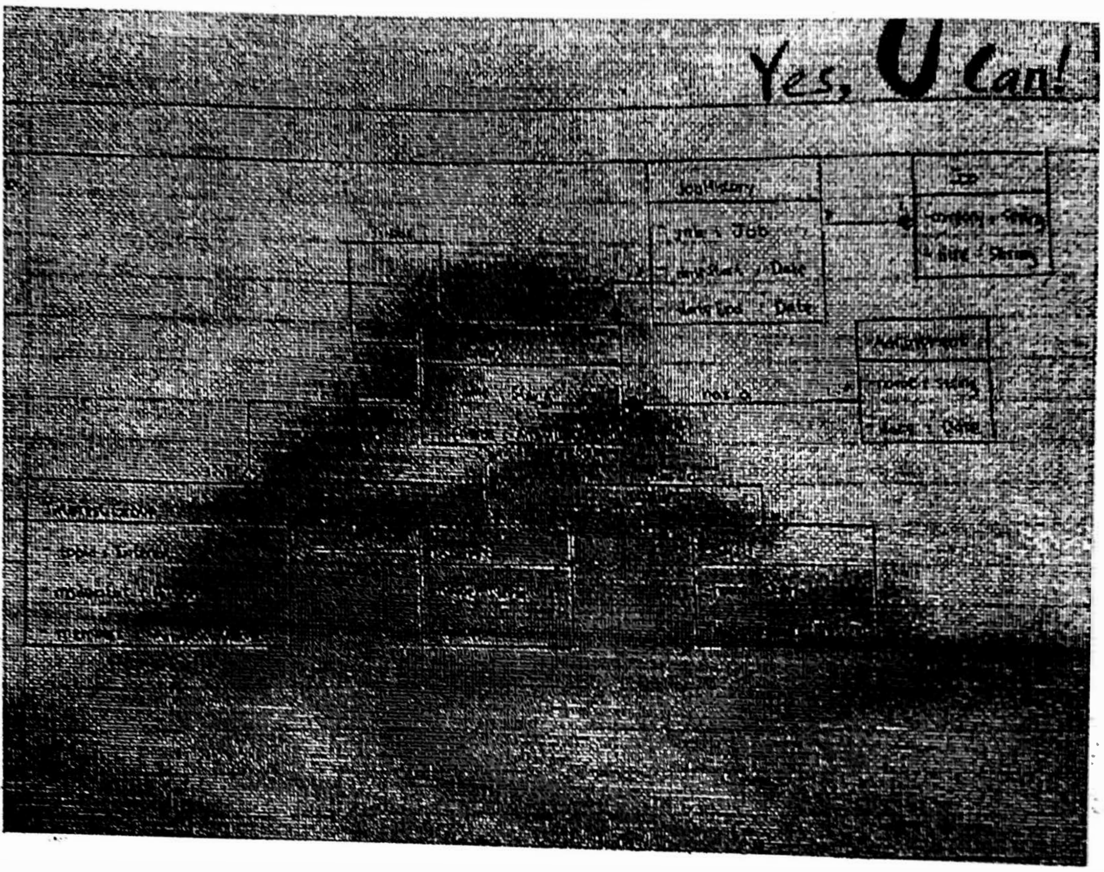
```
#include <iostream>
#include <string>

using namespace std;

class Fraction
{
public:
    Fraction(int num, int deno = 1)
    {
        _num = num;
        _deno = deno;
    };
    ~Fraction();
    friend Fraction operator *(const Fraction, const Fraction);
private:
    int _num;
    int _deno;
};

Fraction operator *(Fraction f1, Fraction f2)
{
    int num = f1._num * f2._num;
    int deno = f1._deno * f2._deno;
    return (Fraction(num, deno));
}
```

C.

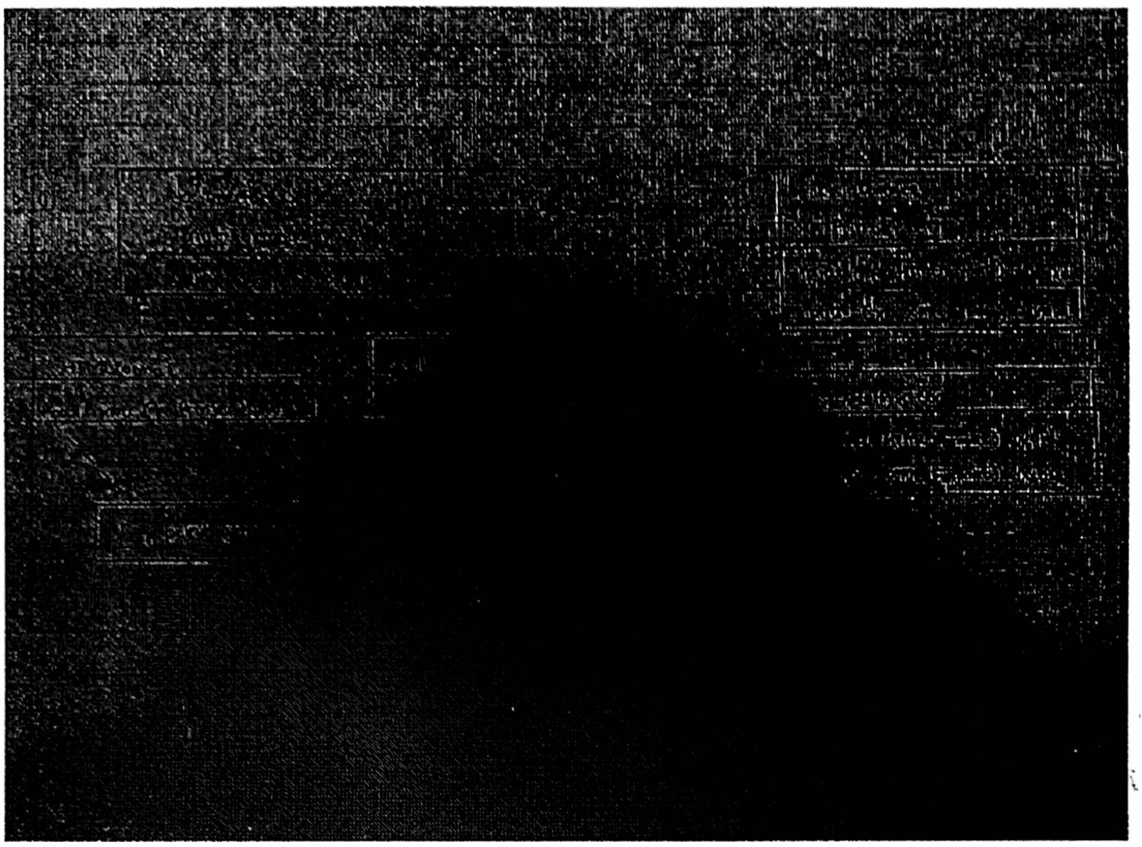
```
public class Task {

    public Store doCheck(Manager mgr)
    {
        Store st = new Store();

        ArrayList al = mgr.getSubTasks();
        for (Task subtask : al) {
            boolean result = subtask.status();
            if (result) {
                st.add(subtask);
                startTask(subtask);
            }
        }

        return st;
    }

    public void startTask(Task subtask)
    {
        subtask.start();
    }
}
```

4. Note: There can be a lot of variation for the answers to these following questions. Do not worry if your answer is different. As long as it is clear and understandable, it should be fine ☺

a.


Yes, U Can!
JobHistory
job : Job
dateStart : Date
dateEnd : Date
Job
company : String
title : String
Achievement
name : String
date : Date
has a

b.



(ii)

- Low Coupling: there is low dependencies between classes
- High Cohesion: the classes each has single responsibility
- Single Responsibility: each class only has a single responsibility. The PrintFormat Interface and its descendant's job are to prepare the data. The Printer Interface and its descendant's job are to print the data according to its format.
- Interface Segregation: the classes depend only on interfaces that they use. Both Printer and PrintFormat are client specific interfaces.
- Open-Closed Principle: the printer interface and its descendants are closed for modification but open for extension (you can extend to add new printing methods for more data formats)