

**Coursework**  
**Object-Oriented Specification and Design**  
**5CCS2OSD**

**Team 10:**

Van Vuuren, Kelvin Charl Student No.: 1544576

Papatheodoulou, Dimitris Student No.: 1545299

Kermani Nejad, Mohammadreza Student No.: 1546710

Ghebache, Rihem Student No.: 1556181

## **1.1 Ambiguous and incomplete requirements**

The brief for the coursework contained many ambiguities which meant we had to make several assumptions about the vague requirements.

The first assumption we made was that bonds could be owned by more than one investor as this detail was omitted in the brief. This meant the system did not need to contain any checks to see if a particular bond had already been purchased. Despite this each bond was still unique, containing its own ID number.

Another important detail we had to decide upon was the type of client using the system. This could be large institutions managing portfolios of large client base, right down to singular private investors. The assumption we made was that the system was to be used by private investors to gain a more informed decision on the details and effects buying particular bonds would have on their portfolio.

As we have assumed the system is being used by private investors, we also assumed that each investor only has one portfolio and that each portfolio can only be owned by this investor. The system user assumption also impacted our decision on the what type we should give the attributes inside the program. As the system was to be used by private investors, we assumed that the size of their balance, sum of investment, and corresponding bond payouts would be relatively small so the use of Ints and Doubles would suffice. However, if we had assumed the system was to be used by large financial institutions then the use of Longs rather than Ints would have been necessary.

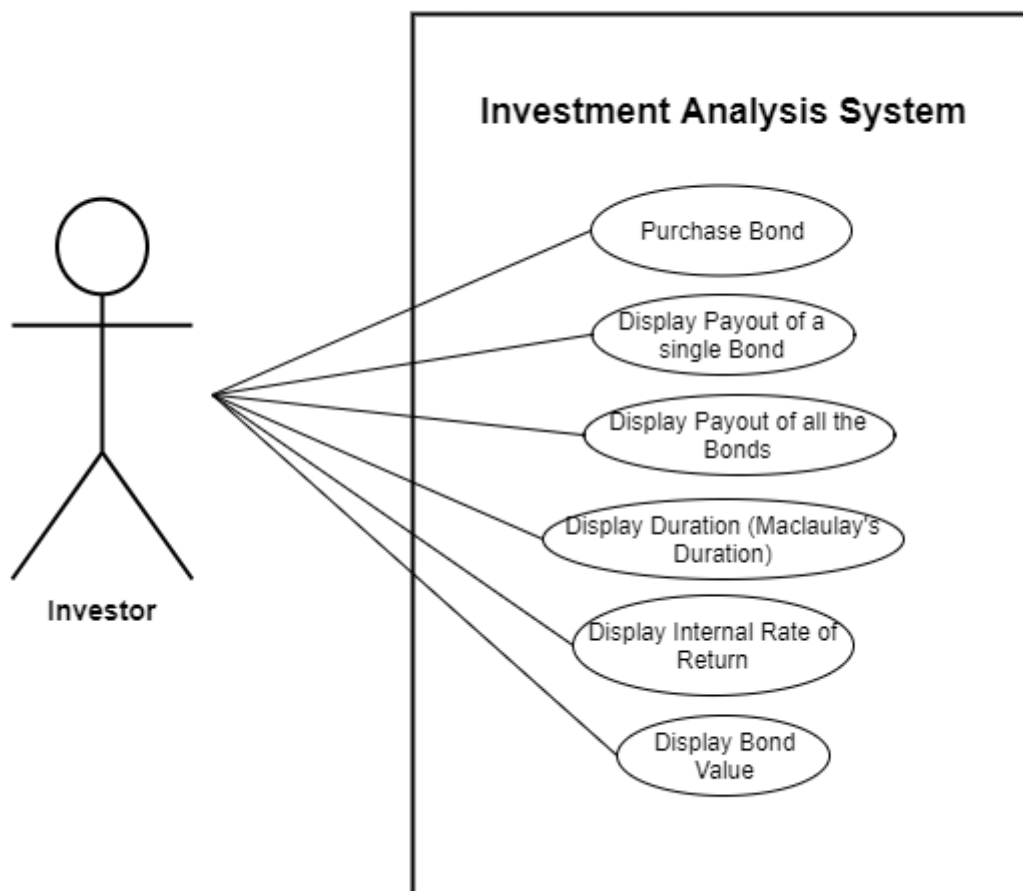
Additionally, because the brief calls the system “Investment Analysis”, we assumed that the system would not need to contain any kind of time frame where it regulates a user’s ability to buy according to the timeline of payouts from bonds purchased and this effect on their balance. Instead, a user can simply add to their

portfolio however many bonds their inputted balance allows, where they can then view details on these bonds and their eventual return.

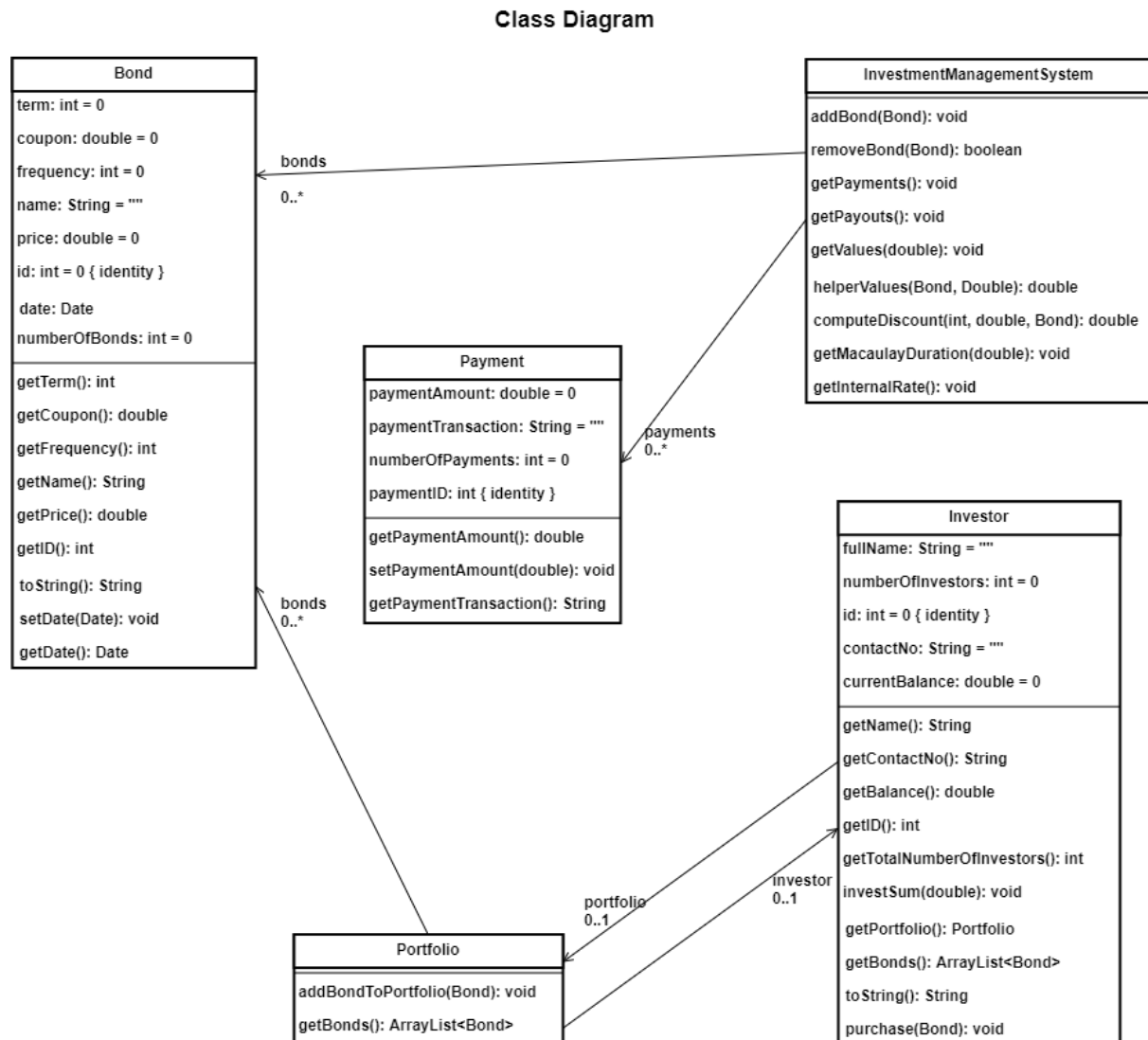
Finally, for the sake of simplicity and the fact we have omitted a form of a timeline from the program, we also assumed that all bonds on the system had not expired and were still valid. Therefore, we have not included any kind of check for this even though bonds would eventually become invalid.

After identifying these ambiguities and how we were going to deal with them, we drew the use case diagram below to show an overview of the functionality the system would have.

### Use Case Diagram



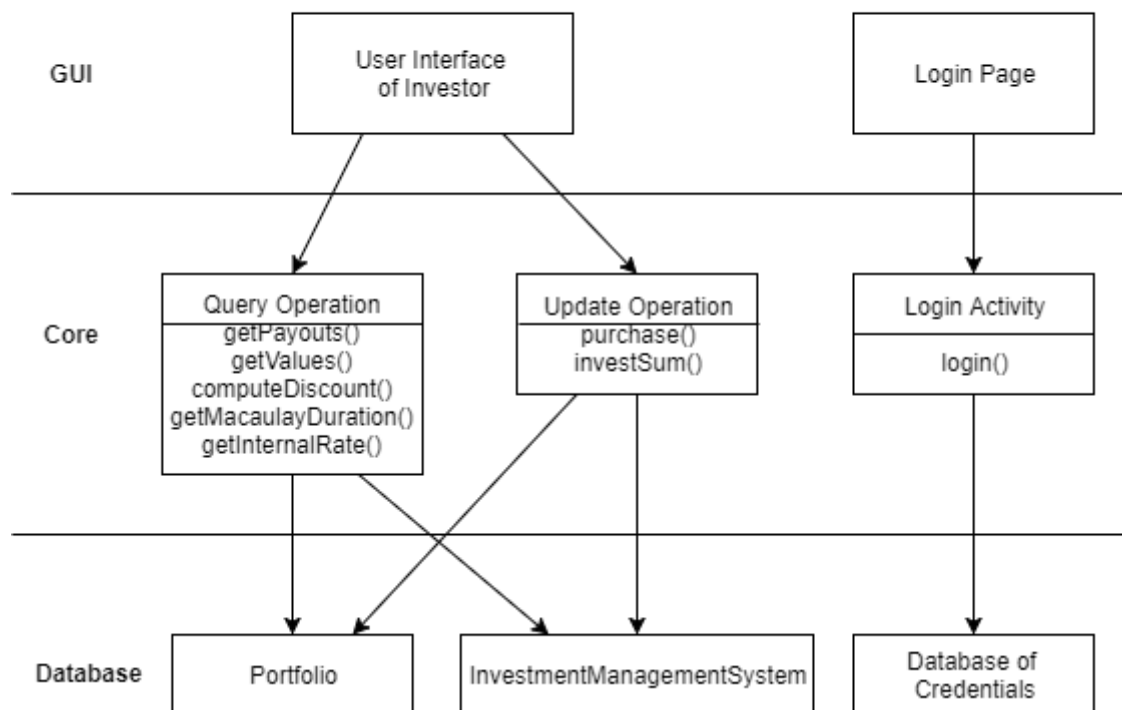
After creating the use case diagram, our next task was to create a class diagram in order to give a visual representation of how we would go about implementing the program with each class and its functionality broken down into separate functions, as well as the relationships between these classes.



The architecture diagram below shows a conceptual model of how the system would be structured and its behaviour to provide a graphical model of its applications and its method of data processing and storage, as well as its external interface.

## 1.2

### Architecture Diagram



After completing initial class diagrams, these classes were then expanded on in more detail through pseudocode by defining pre and post conditions.

#### **Investor**

Update investSum(double sum): void

Pre: sum > 0

Post: currentBalance = currentBalance + sum

Update purchase(Bond bond): void

Pre: currentBalance >= bond.getPrice()

Post: balance = balance - bond.getPrice()

&& bond.setDate(new Date)

&& bond : portfolio

### **Portfolio**

Update **addBondToPortfolio(Investor investor): void**

Pre: true

Post: bond : bonds

### **InvestmentManagementSystem**

Update **addBond(Bond bond): void**

Pre: true

Post: bond : bonds

Update **removeBond(Bond bond): boolean**

Pre: bond : bonds

Post: bond : bonds

Result: true

Query **getPayments(): void**

Pre: bond : bonds

Post: i : Integer , i > 0 & i <= bond.getTerm()-1

&& new Payment.setPaymentAmount(bond.getCoupon()) : payments

Result: new Payment.setPaymentAmount(bond.getCoupon() + bond.getPrice()) :

payments

&& for (payment : payments) -> payment.getPaymentAmount()

Query **getPayouts(): void**

Pre: bond : bonds

Post: i : Integer , i > 0 & i <= bond.getTerm()

totalPayout = totalPayout + bond.getTerm()

Result: totalPayout = totalPayout + 100

Query **getValues(double inflationRate): void**

Pre: bond : bonds && totalValue = 0

Post: i : Integer , i > 0 & i <= bond.getTerm()

totalValue = totalValue + computeDiscount(i, inflationRate, bond)

```

Result:  totalValue = (((totalValue + computeDiscount(term, inflationRate,
bond))*1000.round()).div(1000)).oclAsType(double)

Query helperValues(Bond bond, double inflationRate): double
Pre: true
Post: i : Integer , i > 0 & i<= bond.getTerm()
        totalValue = totalValue + computeDiscount(i, inflationRate, bond)
Result: totalValue = (((totalValue + computeDiscount(term, inflationRate,
bond))*1000.round()).div(1000)).oclAsType(double)

Query computeDiscount(int yearOfPayment, double inflationRate, Bond bond):
double
Pre: yearOfPayment == bond.getTerm()
Post:   if yearOfPayment == bond.getTerm()
        then result = (bond.getCoupon() + 100).div(pow(( 1 + inflationRate
),term))
        else result = (bond.getCoupon()).div(pow(( 1 + inflationRate
),yearOfPayment))
        endif

Query getMacaulayDuration(double inflationRate): void
Pre:  bond : bonds
        && macaulayDuration=0, dividend=0 & helper=0
Post: i : Integer , i > 0 & i<= bond.getTerm()
        helper = helper + (i * bond.getCoupon()).div(pow((1 + inflationRate),
i))
        end loop
        dividend = (helper + bond.getTerm() * 100).div(pow(1 + inflationRate,
bond.getTerm()))
        totalValue: double = 0
        i : Integer , i > 0 & i<= bond.getTerm()-1
        totalValue = totalValue + computeDiscount(i, inflationRate, bond)
        end loop
        totalValue =totalValue+ computeDiscount(bond.getTerm(), inflationRate,
bond)
        macaulayDuration = dividend.div(totalValue)
Result: ((macaulayDuration*1000).round()).div(1000)).oclAsType(double)

```

```

Query getInternalRate(): void
Pre:  bond : bonds
        && (((helperValues(bond,irr)*10) .round().oclAsType(double)).div(10)) !=
        (bond.getPrice())
        && upperBound=1, lowerBound = 0 & irr =0
Post: irr = (upperBound + lowerBound).div(2);
        if ((helperValues(bond,irr)*100).round().oclAsType(double)).div(100) >
        bond.getPrice()
        then lowerBound = irr
        else upperBound = irr
        endif
Result: ((irr*10000).round()).div(10000)). oclAsType(double)

```

### 1.3

For the final part of the assignment, java implementations of the classes defined in the class diagrams were written following the structure planned out previously using pseudocode. The raw java code of the final implementation is listed below

Note: as the brief left out any information on GUI, we use the main function as a means to run test cases through the program, printing out results to the console. So, as the code in that class constantly changes, it has not been included below but will be included in the test cases.

#### Class: InvestmentManagementSystem

```

package investmentAnalysisSystem;
import java.util.ArrayList;

public class InvestmentManagementSystem {

    private ArrayList<Bond> bonds;
    private ArrayList<Payment> payments;

    public InvestmentManagementSystem(){
        bonds = new ArrayList<Bond>();
        payments = new ArrayList<Payment>();
        //setPayments();
    }

    public void addBond(Bond bond){
        bonds.add(bond);
    }

```

```

    public boolean removeBond(Bond b){
        for( Bond bond : bonds){
            if(bond.equals(b)){
                bonds.remove(b);
                return true;
            }
        }
        return false;
    }

    public void getPayments(){
        for(Bond bond : bonds){

            for (int i = 1; i <= bond.getTerm() - 1; i++) {

                Payment payment = new Payment();
                payment.setPaymentAmount(bond.getCoupon());
                payments.add(payment);

            }

            Payment payment = new Payment();
            payment.setPaymentAmount(bond.getPrice() + (bond.getCoupon()));
            payments.add(payment);

            System.out.println("The payments for the bond => " + bond + " are as
follows:");
            for (Payment p : payments) {
                System.out.print("The payment amount is: " + p.getPaymentAmount() +
" | The transaction number is: " + p.getPaymentTransaction() + "\n");
            }
        }
    }

    public void getPayouts(){
        for(Bond bond : bonds){

            double totalPayout = 0.0;

            for (int i = 1; i <= bond.getTerm(); i++) {
                totalPayout += bond.getCoupon();
            }

            totalPayout += 100;

            System.out.println("The total payout is for the bond : " + bond + " is : "
+ totalPayout);
        }
    }

    public void getValues(double inflationRate){
        for(Bond bond : bonds){

            double totalValue = 0;
            for(int i = 1; i <= bond.getTerm() - 1; i++){
                totalValue += computeDiscount(i, inflationRate, bond);
            }
            totalValue += computeDiscount(bond.getTerm(), inflationRate, bond);
        }
    }

```



```

        System.out.println("The value for the bond: " + bond + " is:" +
(double)Math.round(totalValue*1000)/1000);

    }

}

private double helperValues(Bond bond,double inflationRate){
    double totalValue = 0;
    for(int i = 1; i <= bond.getTerm() - 1; i++){
        totalValue += computeDiscount(i, inflationRate, bond);
    }
    totalValue += computeDiscount(bond.getTerm(), inflationRate, bond);

    return (double)Math.round(totalValue*1000)/1000 ;
}

public double computeDiscount(int yearOfPayment,double inflationRate, Bond bond){
    if(yearOfPayment == bond.getTerm()) return (bond.getCoupon() + 100) / Math.pow(( 1
+ inflationRate ), bond.getTerm());

    else return (bond.getCoupon()) / Math.pow(( 1 + inflationRate ), yearOfPayment);
}

public void getMacaulayDuration(double inflationRate){
    for(Bond bond : bonds){
        double macaulayDuration = 0;
        double dividend = 0;
        double helper = 0;

        for(int i = 1; i <= bond.getTerm(); i++){
            helper += (i * bond.getCoupon()) / Math.pow((1 + inflationRate),
i);
        }

        dividend = helper + bond.getTerm() * 100 / Math.pow(1 + inflationRate,
bond.getTerm());

        double totalValue = 0;
        for(int i = 1; i <= bond.getTerm() - 1; i++){
            totalValue += computeDiscount(i, inflationRate, bond);
        }
        totalValue += computeDiscount(bond.getTerm(), inflationRate, bond);

        macaulayDuration = dividend / totalValue;

        System.out.println("The Macaulay Duration for the bond: " + bond + " is: "
+(double)Math.round(macaulayDuration*1000)/1000);
    }

}

public void getInternalRate(){
    for(Bond bond: bonds){

        double irr = 0;
        double upperBound = 1;
        double lowerBound = 0;

        while((double)Math.round(helperValues(bond,irr)*10)/10 != bond.getPrice()){
            irr = (upperBound + lowerBound) / 2;

```

```

        if((double)Math.round(helperValues(bond,irr)*100)/100 >
bond.getPrice() ) lowerBound = irr;
        else upperBound = irr;
    }

    System.out.println("The Internal Rate of Return of the bond " + bond + "
is: " + (double)Math.round(irr*10000)/10000);

    }

}

}

```

## Class: Bond

```

package investmentAnalysisSystem;

import java.util.Date;

public class Bond {

    private final int term; // number of years to expire
    private final double coupon; // percentage of investment
    private final int frequency; // frequency of payment per year
    private final String name; // name of bond
    private final double price; // price of bond

    private Date date;

    public final int id;
    private static int numberOfBonds = 0;

    // Based on the test file, we assume the coupon value is not in percentage and its the
    actual value

    Bond(int term, double coupon, int frequency, String name, double price){
        this.term = term;
        this.coupon = coupon;
        this.frequency = frequency;
        this.name = name;
        this.price = price;
        id = ++numberOfBonds;
    }

    public int getTerm(){
        return term;
    }

    public double getCoupon(){
        return coupon;
    }

    public int getFrequency(){
        return frequency;
    }

    public String getName(){
        return name;
    }

    public double getPrice(){
        return price;
    }

    public int getID(){

```

```
        return id;
    }

    public String toString(){
        return "Name: " + name + " | Term: " + term + " | Coupon: " + coupon +
            " | Price: " + price + " | Frequency: " + frequency + " | Purchase
date : " + date ;
    }

    public void setDate(Date date){
        this.date = date;
    }

    public Date getDate(){
        return date;
    }
}
```

### Class: Investor

```
package investmentAnalysisSystem;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;

public class Investor {

    private final String fullName; // Fullname of investor
    private static int numberOfInvestors; // Number of investors
    private final int id; // unique ID for each investors
    private String contactNo;

    private double currentBalance; // amount of money in their account
    private Portfolio portfolio; //

    public Investor(String fullName, String contactNo){
        this.fullName = fullName;
        this.contactNo = contactNo;
        id = ++numberOfInvestors ;
        portfolio = new Portfolio(this);
    }

    public String getName(){
        return fullName;
    }

    public String getContactNo(){
        return contactNo;
    }

    public double getBalace(){
        return currentBalance;
    }

    public Portfolio getPortfolio(){
        return portfolio;
    }

    public int getID(){
        return id;
    }
}
```

```

    public int getTotalNumberOfInvestors(){
        return numberOfInvestors;
    }

    ArrayList<Bond> getBonds(){
        return portfolio.getBonds();
    }

    public void investSum(double sum){
        currentBalance += sum;
    }

    public void purchase(Bond bond){

        if(currentBalance >= bond.getPrice()){
            currentBalance -= bond.getPrice();
            DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
            Date date = new Date();
            bond.setDate(date);
            portfolio.addBondToPortfolio(bond);
            System.out.println(fullName + " with the ID:" + id + " just purchased the
bond with the ID: "
                                + bond.getID() + " at: " + bond.getDate());

            System.out.println("The Current Balance: " + currentBalance);
        }
        else{
            System.out.println("Purchase failed because of insufficient fund");
        }
    }

    public String toString(){
        return "Fullname: " + fullName + " | Investor ID:" + id;
    }
}

```

### Class: Payment

```

package investmentAnalysisSystem;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

public class Payment {

    private double paymentAmount = 0;
    private final String paymentTransaction;

    public final int paymentID;
    private static int numberOfPayments = 0;

    public Payment(){
        paymentID = ++numberOfPayments;
        DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
        Date date = new Date();
        String transaction = dateFormat.format(date) + paymentID;
        transaction = transaction.replaceAll("\\s+", "");
        transaction = transaction.replaceAll("\\\\/", "");
        transaction = transaction.replaceAll("\\\\:", "");
        paymentTransaction = transaction;
    }

    public double getPaymentAmount() {
        return paymentAmount;
    }
}

```

```

    public void setPaymentAmount(double price) {
        paymentAmount = price;
    }

    public String getPaymentTransaction(){
        return paymentTransaction;
    }

}

```

## Class: Portfolio

```

package investmentAnalysisSystem;
import java.util.ArrayList;

public class Portfolio {

    ArrayList<Bond> bonds;
    Investor investor;

    public Portfolio(Investor investor){
        this.investor = investor;
        bonds = new ArrayList<Bond>();
    }

    void addBondToPorfolio(Bond bond){
        bonds.add(bond);
    }

    ArrayList<Bond> getBonds(){
        return bonds;
    }

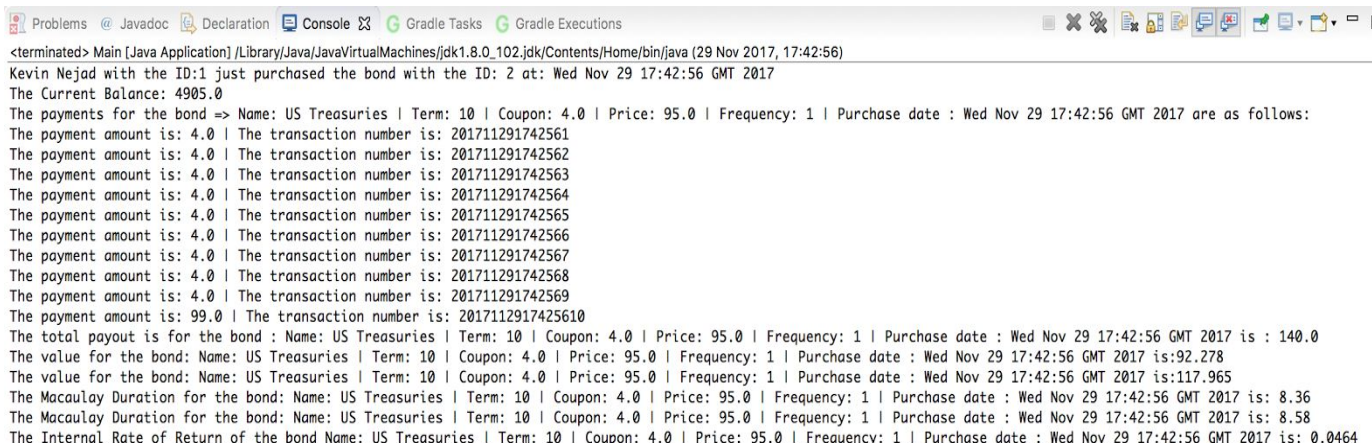
}

```

## Test Cases

### Test Case 1:

The first test case uses an initial investment of £103 into a bond with a 5% coupon, where payments are made each year for the bond's full term of 5 years.



The screenshot shows the console output of a Java application. The output starts with a terminated message and then displays the purchase of a bond by Kevin Nejad. It lists the bond's details: Name: US Treasuries, Term: 10, Coupon: 4.0, Price: 95.0, Frequency: 1, and Purchase date: Wed Nov 29 17:42:56 GMT 2017. The current balance is 4905.0. The output then shows a series of payments for the bond, each with a payment amount of 4.0 and a transaction number. The final payment is 99.0. The output also shows the total payout for the bond, which is 140.0, and the value for the bond, which is 92.278. The Macaulay Duration for the bond is 8.36, and the Internal Rate of Return is 0.0464.

```

<terminated> Main [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_102.jdk/Contents/Home/bin/java (29 Nov 2017, 17:42:56)
Kevin Nejad with the ID:1 just purchased the bond with the ID: 2 at: Wed Nov 29 17:42:56 GMT 2017
The Current Balance: 4905.0
The payments for the bond => Name: US Treasuries | Term: 10 | Coupon: 4.0 | Price: 95.0 | Frequency: 1 | Purchase date : Wed Nov 29 17:42:56 GMT 2017 are as follows:
The payment amount is: 4.0 | The transaction number is: 201711291742561
The payment amount is: 4.0 | The transaction number is: 201711291742562
The payment amount is: 4.0 | The transaction number is: 201711291742563
The payment amount is: 4.0 | The transaction number is: 201711291742564
The payment amount is: 4.0 | The transaction number is: 201711291742565
The payment amount is: 4.0 | The transaction number is: 201711291742566
The payment amount is: 4.0 | The transaction number is: 201711291742567
The payment amount is: 4.0 | The transaction number is: 201711291742568
The payment amount is: 4.0 | The transaction number is: 201711291742569
The payment amount is: 99.0 | The transaction number is: 2017112917425610
The total payout is for the bond : Name: US Treasuries | Term: 10 | Coupon: 4.0 | Price: 95.0 | Frequency: 1 | Purchase date : Wed Nov 29 17:42:56 GMT 2017 is : 140.0
The value for the bond: Name: US Treasuries | Term: 10 | Coupon: 4.0 | Price: 95.0 | Frequency: 1 | Purchase date : Wed Nov 29 17:42:56 GMT 2017 is:92.278
The value for the bond: Name: US Treasuries | Term: 10 | Coupon: 4.0 | Price: 95.0 | Frequency: 1 | Purchase date : Wed Nov 29 17:42:56 GMT 2017 is:117.965
The Macaulay Duration for the bond: Name: US Treasuries | Term: 10 | Coupon: 4.0 | Price: 95.0 | Frequency: 1 | Purchase date : Wed Nov 29 17:42:56 GMT 2017 is: 8.36
The Macaulay Duration for the bond: Name: US Treasuries | Term: 10 | Coupon: 4.0 | Price: 95.0 | Frequency: 1 | Purchase date : Wed Nov 29 17:42:56 GMT 2017 is: 8.58
The Internal Rate of Return of the bond Name: US Treasuries | Term: 10 | Coupon: 4.0 | Price: 95.0 | Frequency: 1 | Purchase date : Wed Nov 29 17:42:56 GMT 2017 is: 0.0464

```

```

import java.text.DateFormat;

public class Main {

    public static void main(String[] args){

        InvestmentManagementSystem bs = new InvestmentManagementSystem();

        Bond b1 = new Bond(5,5,1,"UK Government",103);
        Bond b2 = new Bond(10, 4,1, "US Treasuries",95);
        Bond b3 = new Bond(20,3, 1,"JGBs",92);
        Bond b4 = new Bond(15,2,1,"OATs",120);

        Investor investor1 = new Investor("Kevin Nejad", "+44(0)7654140143");
        investor1.investSum(5000);

        investor1.purchase(b2);

        bs.addBond(b2);

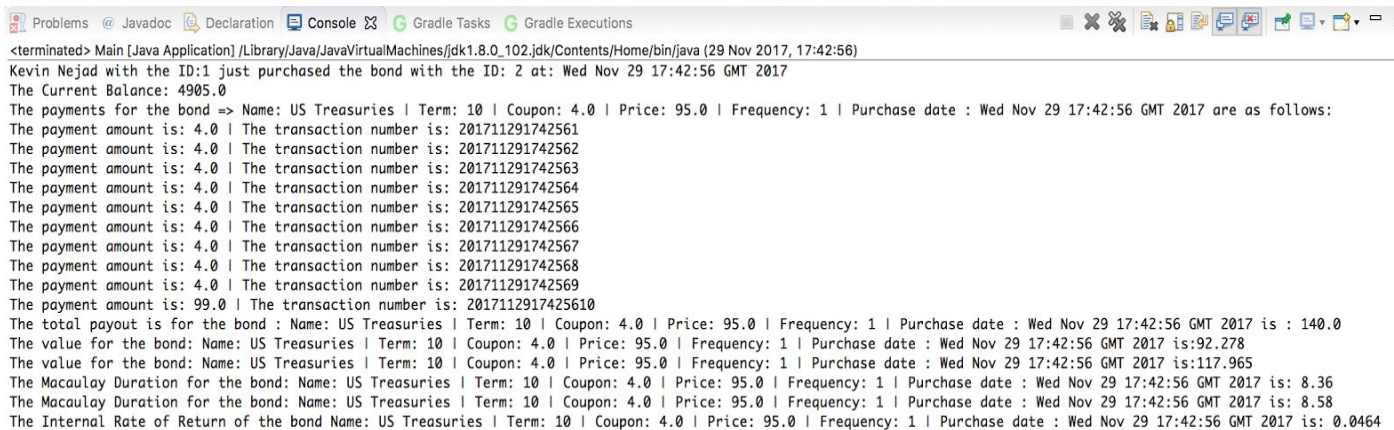
        bs.getPayments();
        bs.getPayouts();
        bs.getValues(0.05);
        bs.getValues(0.02);
        bs.getMacaulayDuration(0.05);
        bs.getMacaulayDuration(0.02);
        bs.getInternalRate();

    }
}

```

## Test Case 2:

The second test case uses an initial investment of £95 into a bond with a 4% coupon, where payments are made each year for the bond's full term of 10 years.



```

<terminated> Main [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_102.jdk/Contents/Home/bin/java (29 Nov 2017, 17:42:56)
Kevin Nejad with the ID:1 just purchased the bond with the ID: 2 at: Wed Nov 29 17:42:56 GMT 2017
The Current Balance: 4905.0
The payments for the bond => Name: US Treasuries | Term: 10 | Coupon: 4.0 | Price: 95.0 | Frequency: 1 | Purchase date : Wed Nov 29 17:42:56 GMT 2017 are as follows:
The payment amount is: 4.0 | The transaction number is: 201711291742561
The payment amount is: 4.0 | The transaction number is: 201711291742563
The payment amount is: 4.0 | The transaction number is: 201711291742564
The payment amount is: 4.0 | The transaction number is: 201711291742565
The payment amount is: 4.0 | The transaction number is: 201711291742566
The payment amount is: 4.0 | The transaction number is: 201711291742567
The payment amount is: 4.0 | The transaction number is: 201711291742568
The payment amount is: 4.0 | The transaction number is: 201711291742569
The payment amount is: 99.0 | The transaction number is: 2017112917425610
The total payout is for the bond : Name: US Treasuries | Term: 10 | Coupon: 4.0 | Price: 95.0 | Frequency: 1 | Purchase date : Wed Nov 29 17:42:56 GMT 2017 is : 140.0
The value for the bond: Name: US Treasuries | Term: 10 | Coupon: 4.0 | Price: 95.0 | Frequency: 1 | Purchase date : Wed Nov 29 17:42:56 GMT 2017 is:92.278
The value for the bond: Name: US Treasuries | Term: 10 | Coupon: 4.0 | Price: 95.0 | Frequency: 1 | Purchase date : Wed Nov 29 17:42:56 GMT 2017 is:117.965
The Macaulay Duration for the bond: Name: US Treasuries | Term: 10 | Coupon: 4.0 | Price: 95.0 | Frequency: 1 | Purchase date : Wed Nov 29 17:42:56 GMT 2017 is: 8.36
The Macaulay Duration for the bond: Name: US Treasuries | Term: 10 | Coupon: 4.0 | Price: 95.0 | Frequency: 1 | Purchase date : Wed Nov 29 17:42:56 GMT 2017 is: 8.58
The Internal Rate of Return of the bond Name: US Treasuries | Term: 10 | Coupon: 4.0 | Price: 95.0 | Frequency: 1 | Purchase date : Wed Nov 29 17:42:56 GMT 2017 is: 0.0464

```

```

import java.text.DateFormat;

public class Main {

    public static void main(String[] args){

        InvestmentManagementSystem bs = new InvestmentManagementSystem();

        Bond b1 = new Bond(5,5,1,"UK Government",103);
        Bond b2 = new Bond(10, 4,1, "US Treasuries",95);
        Bond b3 = new Bond(20,3, 1,"JGBs",92);
        Bond b4 = new Bond(15,2,1,"OATs",120);

        Investor investor1 = new Investor("Kevin Nejad","+44(0)7654140143");
        investor1.investSum(5000);

        investor1.purchase(b2);

        bs.addBond(b2);

        bs.getPayments();
        bs.getPayouts();
        bs.getValues(0.05);
        bs.getValues(0.02);
        bs.getMacaulayDuration(0.05);
        bs.getMacaulayDuration(0.02);
        bs.getInternalRate();

    }
}

```

## Roles:

Team Members:	Main Roles:	Support:
Kelvin Charl Van Vuuren	Report	Pseudocode, Java implementation
Dimitris Papatheodoulou	Use Case diagram, Class diagram	Report, Java implementation
Mohammadreza Kermani Nejad	Java implementation, Class diagram, Test cases	Use Case diagram, Pseudocode
Rihem Ghebache	Pseudocode, Architecture diagram	Report, Test cases