

## **SUTD 50.001 Introduction to Information Systems and Programming Problem Set 3B**

**For all questions, please access the vocareum link found at eDimension for the starter code and to submit.**

**The Vocareum link is for submission only. Please work on the problems in Android studio, and this includes writing code for the test cases.**

**To prevent hard-coding, test cases used in Vocareum *may* be different from those provided here and will not be given to you.**

**There is only one question in this Problem Set.**

This problem set is written to help you have an idea of what to expect for **Final Exam Part 1**.

In the **Final Exam Part 1**, there will be parts.

- **Part A – write Java class(es).** You will submit your answer to Vocareum (through the Schoolyear Exam browser during the exam).
- **Part B – write the code for MainActivity.java.** You will write your code on the question paper.
- **Part C – conceptual questions.** You will write your answer on the question paper.

## Q1. [Programming Question] Java Programming and Android App

### Background

Financial institutions sell instruments called “bonds” to borrow money from investors.

Bonds are sold at a **selling price** in dollars. In return, the institution promises two things:

- To buy the bond back from you at the **face value** (in dollars) a certain period of time later (called the “**duration**”, expressed in number of years)
- To pay you an **interest payment** (this is paid annually in dollars, i.e. once every year)

Investors evaluate the bonds by calculating the **yield to maturity**  $r$ , which is a function of the **duration**, the **face value**, the **selling price** and the **interest payment**.

There are two formulas for the **yield to maturity**  $r$  based on the type of bond. The formulas below produce a decimal value of  $r$  between 0 and 1.

“Zero Coupon Bonds” - If a bond has **zero annual interest payment**, the yield to maturity  $r$  is calculated using the following formula:

$$r = \left( \frac{\text{Face value}}{\text{Selling price}} \right)^{1/\text{duration}} - 1$$

“Coupon-Paying Bonds” - If a bond has **positive annual interest payment**, the yield to maturity  $r$  satisfies the following formula:

$$\text{selling price} = (\text{interest payment}) \times \frac{1 - \left( \frac{1}{1+r} \right)^{\text{duration}}}{r} + \frac{\text{face value}}{(1+r)^{\text{duration}}}$$

For this formula,  $r$  must be obtained by using an iterative numerical method.

Details of this numerical method are described here. First, we define

$$f(r) = \text{selling price} - (\text{interest payment}) \times \frac{1 - \left( \frac{1}{1+r} \right)^{\text{duration}}}{r} - \frac{\text{face value}}{(1+r)^{\text{duration}}}$$

In this test question, the interval bisection method is used to solve for the value of  $r$  that makes  $f(r) = 0$ . The pseudocode is given below. Note that in step 1 below, the choice of  $r_{up}$  and  $r_{down}$  guarantees that the values of  $f(r_{up}), f(r_{down})$  have opposite signs.

- 1) Choose starting values of  $r_{up} = 1$  and  $r_{down} = 1 \times 10^{-10}$
- 2) Calculate  $\text{delta} = r_{up} - r_{down}$
- 3) While  $\text{delta} > 1 \times 10^{-5}$ 
  - a. Assign  $r_{middle} = \frac{1}{2}(r_{up} + r_{down})$
  - b. Calculate  $f(r_{middle}), f(r_{up}), f(r_{down})$
  - c. If  $f(r_{middle})$  and  $f(r_{up})$  have the same sign, assign  $r_{up} = r_{middle}$
  - d. If  $f(r_{middle})$  and  $f(r_{down})$  have the same sign, assign  $r_{down} = r_{middle}$
  - e. Calculate  $\text{delta}$  as defined in step 2
- 4) The result  $r = \frac{1}{2}(r_{up} + r_{down})$

The Yield to Maturity for four different bonds are shown below.

Test Case	Selling Price	Face Value	Duration <sup>1</sup>	Interest Payment (annual)	Yield To Maturity (4 d.p.)
1	900	1000	4	10	0.0374
2	1000	1000	1	10	0.0100
3	900	1000	1	0	0.1111
4	1000	1000	1	0	0.0000

Note 1. In actual finance terminology, this is called “tenor” or “term to maturity”. “Duration” has a different meaning for bonds, which we will not discuss here.

**Part A – Writing the Bond Class (24 points)**

Submit your answer to Vocareum to collect points to fulfil the coursework requirements
--

The starter code of **Bond.java** should contain the following statements.

```
public class Bond {

    private double sellingPrice;
    private double faceValue;
    private double interestPayment;
    private double duration;
    private YieldCalculation yieldCalculation;

    private Bond(double sellingPrice, double faceValue, double interestPayment,
double duration){
        this.sellingPrice = sellingPrice;
        this.faceValue = faceValue;
        this.interestPayment = interestPayment;
        this.duration = duration;
    }
}
```

**YieldCalculation.java** should contain the following statements, which you do not need to modify.

```
public interface YieldCalculation{

    double yieldToMaturity (Bond bond);

}
```

**TestBond.java** contains example of how the requirements of this questions are met. You can determine what the output should be by reading the background section.

```
public class TestBond {
    public static void main(String[] args){
        Bond.BondBuilder bondBuilder = new Bond.BondBuilder();
        Bond bond1 = bondBuilder.createBond();
        bond1.setYieldCalculation(new WithCouponYield());
        System.out.println("" + bond1.calculateYTM());
        Bond bond2 = bondBuilder
            .setDuration(1)
            .setFaceValue(1000)
            .setSellingPrice(900)
            .setInterestPayment(0).createBond();
        bond2.setYieldCalculation(new ZeroCouponYield());
        System.out.println("" + bond2.calculateYTM());
    }
}
```

The requirements are as follows.

- a. As the constructor is private, one way to instantiate the class **Bond** is to implement a static inner class **BondBuilder** that implements the builder design pattern. The method in **BondBuilder** that creates an instance of **Bond** using the constructor shown above is called **createBond()**.

If none of the setters in **BondBuilder** are called, i.e.

```
Bond bond = new Bond.BondBuilder().createBond();
```

the instance of the **Bond** class that is returned has the following default values: The default value of the selling price and face value is 1000, the default interest payment is 10 and the default duration is 1.

- b. The class **Bond** should have getter methods for the instance variables **sellingPrice**, **faceValue**, **interestPayment** and **duration**.
- c. The constructor for the **Bond** class shown above should be modified to throw an **IllegalArgumentException** if **sellingPrice**, **faceValue** and **duration** are zero or negative, and when **interestPayment** is negative. This is the only modification that you need to make to the constructor shown above.
- d. The class **Bond** should have a setter method for the instance variable **yieldCalculation**, which is for objects that implement the **YieldCalculation** interface.
- e. The class **Bond** should have a method **calculateYTM()** that takes no parameters and executes the **yieldToMaturity()** method in the **YieldCalculation** interface and returns its result as a **double**. You do not need to round the result.
- f. Two classes that implement the **YieldCalculation** interface
  - a. **ZeroCouponYield** – this is for bonds with zero annual interest rate payment
  - b. **WithCouponYield** – this is for bonds with non-zero annual interest rate payment

Please refer to the formulas described in this question. If you are unable to implement these calculations, the method may simply return a default value e.g. 0.01. This is so that you can use these objects in part 2.

- g. Note. Since the double data type is used in this question, you do not need to take care of floating-point errors. The autograder will accept small differences in results due to floating-point errors.

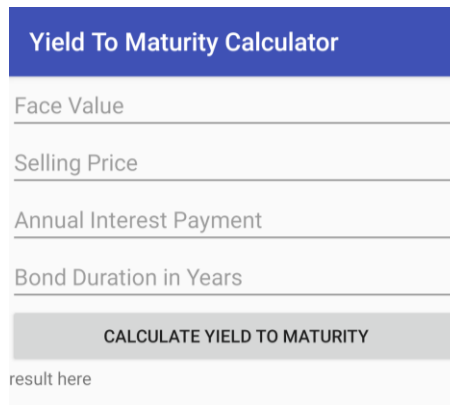
Submit the contents of  
**Bond.java**, **ZeroCouponYield.java**, **WithCouponYield.java** to Vocareum

**Part 2. User Interface For The Bond Class (26 points)**

**This part is included to give you an idea of what to expect in Final Exam Part 1 and is for your self-study. You do not need to submit this for marking.**

The **Bond** class is now going to be incorporated into a simple android app with the following user interface.

There are four **EditText** widgets with hints displayed. The default text that these widgets contain is an empty string.



- a. When the button **CALCULATE YIELD TO MATURITY** is clicked, values entered into the app are used to calculate the yield to maturity. The yield to maturity is displayed in percentage terms in two decimal places in the widget that currently shows **result here**.

Use the **Bond** class written in Part 1 to carry out the calculation.

- b. If any of the **EditText** widgets are left blank, a toast should be displayed with an appropriate message. The exact text of the message is left to you to decide.

Write the code in the space given to you in **MainActivity.java**.

The XML file for the layout above is given below.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/editTextFaceValue"
        android:inputType="number"
        android:hint="Face Value"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <EditText
        android:id="@+id/editTextSellingPrice"
        android:inputType="number"
        android:hint="Selling Price"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <EditText
        android:id="@+id/editTextAnnualInterest"
        android:inputType="number"
        android:hint="Annual Interest Payment"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <EditText
        android:id="@+id/editTextDuration"
        android:inputType="number"
        android:hint="Bond Duration in Years"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <Button
        android:id="@+id/buttonCalculateYield"
        android:text="Calculate Yield To Maturity"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <TextView
        android:id="@+id/textViewResult"
        android:text="result here"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

```
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    EditText editTextFaceValue;
    EditText editTextSellingPrice;
    EditText editTextAnnualInterest;
    EditText editTextDuration;
    Button buttonCalculateYield;
    TextView textViewResult;
    final String EMPTY_STRING = "";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

    }
}
```



**Part C - Your code in Q1**

**This part is included to give you an idea of what to expect in Final Exam Part 1 and is for your self-study. You do not need to submit this for marking.**

- a. Apart from the builder design pattern, state one other design pattern used in the **Bond** class in Q1. Justify your answer.
- b. A customer using your **Bond** class complains that the interval bisection method requires too many iterations. You are asked to use another numerical method to obtain the solution.

Describe how your program in Q1 can be modified to satisfy the customer's requirements.

Your answer should describe

- which parts of **MainActivity.java** should be modified and how (you may give code snippets)
- which of the existing classes should be modified (if any)
- what new classes should be added (if any)

*Note: You do not need to give details of the numerical method to be implemented.*