

2018 Final Exam Programming Part

concept questions not included

Name:

Student ID:

Cohort:

ISTD 50.001 Final Exam

Deleted

- Write your name and ID at the top of this page.
- Answer all six questions. Total 100 points.
- For Question 1 (Parts 1 to 3) submit your solutions to Vocareum.
- For Questions 2 - 6, write your answers on this exam paper.
- All answers will be manually graded. You may be able to earn partial credit for questions.
- **You are not allowed to use any Internet accessing or communicating device during the quiz.**
- **If you need your phone to test your app in question 1, it must be placed in FLIGHT MODE and SILENT MODE during the test. Not doing so amounts to cheating. It should be switched off, placed face down when not in use. The invigilator may request to inspect your phone at any time during the test.**
- **You are not allowed to consult anyone inside or outside of the classroom other than the course instructors of 50.001.**
- This exam is open book. You may refer to the course slides / notes and your personal notes.
- You can use Android Studio / other IDE to test your programs.
- Good luck!

2	
3	
4	
5	
6	
SubTotal	

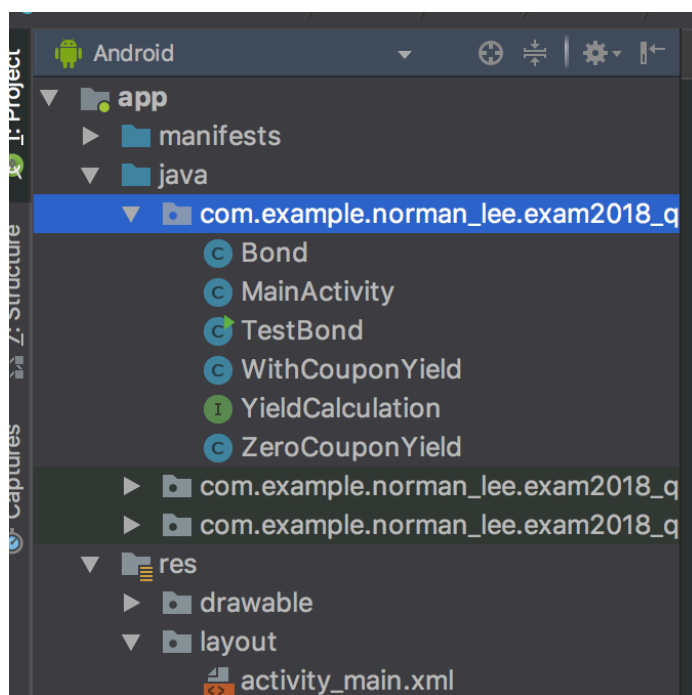
Q1. [Programming Question] Java Programming and Android App [56 points]

This question has three parts

- Part 1 involves plain java programming (24 points)
- Part 2 involves making use of your code in Part 1 in an Android App (26 points)
- Part 3 involves writing unit tests for your code in Part 1 (6 points)

Setting Up

- Open a new android app project with Empty Activity.
- Open the starter codes in Vocareum and include them in your android app project. An easy way is to create new java class files and copy-and-paste from Vocareum to those files.
- You should ensure that the **package** statement at the top of each file is the same one as your android app project.
- You have to create **WithCouponYield.java** and **ZeroCouponYield.java** yourself
- Copy and paste the contents of **MainActivity.java** file into your own project.
- Copy and paste the contents of the **activity_main.xml** file into your own project.
- By the end of Q1, your android App project should contain the following files.
- To execute the main() method in **TestBond**, you may right-click and select **Run TestBond.main()** .



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 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	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

Part 1 – Writing the Bond Class (24 points)

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;  
    }  
}
```

- 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()**.

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 and 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.

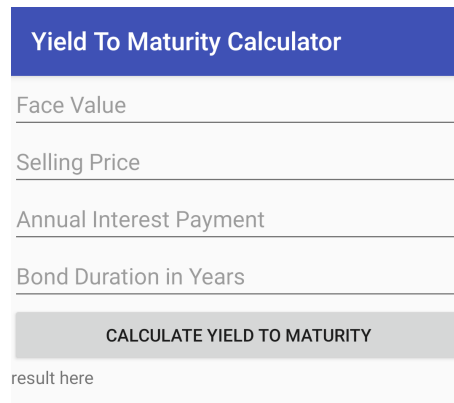
Submit the contents of
Bond.java, **ZeroCouponYield.java**, **WithCouponYield.java** to Vocareum

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

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

Retrieve the xml layout file from Vocareum and paste it into your project.

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



The screenshot shows a mobile app interface titled "Yield To Maturity Calculator". It features four input fields with placeholder text: "Face Value", "Selling Price", "Annual Interest Payment", and "Bond Duration in Years". Below these fields is a button labeled "CALCULATE YIELD TO MATURITY". At the bottom, there is a text area with the placeholder "result here".

The requirements are as follows

- 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.

- The calculation of the yield to maturity must be calculated in a background thread, as the calculation for non-zero interest payments is an iterative method. Hence you must use **AsyncTask** or any other suitable class, and any classes you write must be a nested class within **MainActivity.java**.
- 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.
- If any of the EditText widgets contain unacceptable values, you should display a toast with an appropriate message. The exact text of the message is left to you to decide.
- For the convenience of the user, data stored in the EditText widgets must be saved and reloaded when the app restarts.

Submit the contents of MainActivity.java to Vocareum

Part 3. Unit Tests for Bond Class (6 points)

Write unit tests in **ExampleUnitTest.java** using the Junit4 framework to verify the following:

- That the calculation of the yield to maturity for one of the types of bond is correct
- That unacceptable inputs to the **Bond** class are handled correctly.

You may write as many test methods as you wish to achieve these outcomes.

Submit the contents of ExampleUnitTest.java to Vocareum
--

Q2. [Written Question] Your code in Q1 [8 points]

- 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.