1.00/1.001 Introduction to Computers and Engineering Problem Solving

Fall 2002

Problem Set 4

Due: Day 15

Problem 1. Stocks and Bonds (100 %)

In this problem set, you will learn how to build a hierarchy of classes that represents various financial instruments. Everything you need to know about finance to complete this problem set is included in the explanation below.

Problem Description

It is 10 years from now. The economy is booming, you have paid off your student loans, and you are earning more money than you can spend. You decide to take your excess money out of your checking account and invest it. To evaluate your options, you will simulate what might happen if you invested your money in different kinds of financial instruments.

You consider stocks and bonds only. Your main concerns are how much money you can make per year from your investment (expected return) and the risk that you will take on. In this simple model, risk is modeled only using the standard deviation of the return on an investment. We assume that returns on investments of all securities obey a normal distribution. Thus, they can completely characterized by their mean (expected value) and standard deviation.

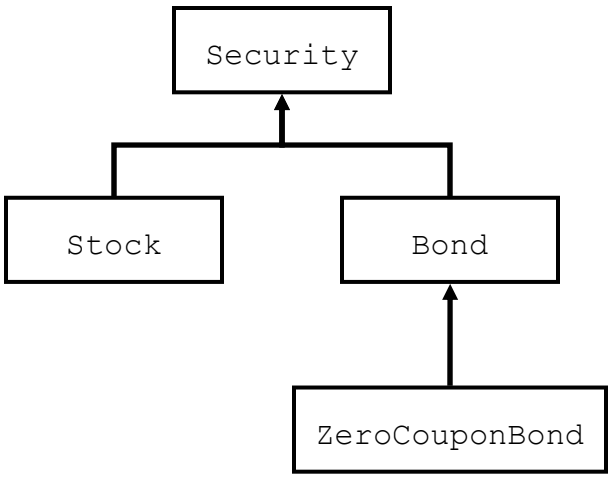

Figure 1. Security class hierarchy

Figure 1 above shows the hierarchy of classes that you will implement for this problem set. "Security" is the general term for debt or equity investments that you are considering. The Security class must be abstract, because you cannot invest in a "Security"---and therefore you cannot instantiate a Security object.

1. Implement the `Security` class. This class should be *abstract.* It must include the following:
   a. A constructor that will be called by subclasses. The constructor must initialize any fields that need to be initialized.

   b. A *name* attribute. (An "attribute" of a class simply means that the class has defined a getter and setter for some aspect of the class. For example, an integer attribute *X* means that the class would contain the following methods:

      ```
      public int getX() {...};
      public void setX(int x) {...};
      ```

      Note that the class may or may not have an integer field "x" to hold the value. It might instead calculate the value of X based on some other information.)

   c. A read-only attribute *expectedReturn* (Read-only attributes do not include a setter.) Since the computation of expected return of a security will be different for different classes of securities, be sure to declare *getExpectedReturn* as abstract. *getExpectedReturn* should return a *double* which is the percentage increase in the value of the security over one year. *E.g.,*, if the security is expected to return 10% over the next year, *getExpectedReturn* should return the value 0.1.

   d. An attribute *standardDeviation*. The values of most securities change unpredictably. Some change wildly over a short period of time, while others are almost completely stable. In this model, we use a standard deviation called historical volatility to represent the risk of an investment. This value represents the standard deviation of the actual return as compared to the expected return of a security. Assume that you are given the value of the standard deviation as an argument to the constructor.

   e. A read only-attribute *actualReturn*. Your method *getActualReturn* should simulate the marketplace by using the expected return and standard deviation of a security to return a sample value for the annual return of the security. You can do this using the following formulae:

$$r = \sqrt{-2\ln(\text{random}())}$$

$$\theta = 2\pi \times \text{random}()$$

$$\text{actual return} = \mu + \sigma \times r \times \cos(\theta)$$

where $\ln(x)$ is the natural logarithm of x,

random() returns a random number uniformly distributed between 0.0 and 1.0,

μ is the expected return of the security,

and σ is the standard deviation of the return of the security

Note that there are two independent calls to the random( ) function.

f. A *printMe* method that outputs the name, expected return, and standard deviation of a security. Expected return and standard deviation are percentages, so you should print them as such. Round off each number to the nearest 10000^{th}. In other words, if the expected return is 0.09482532841, your method should print it out as `9.48%`. (Hint: use `Math.rint` for rounding.)

2. Now create a concrete class `Stock` which extends `Security` and represents the common stock for a corporation. A stock represents ownership of a portion of a public company. Note that you don't care about what the price of the stock is, only the expected return and standard deviation, both of which will be provided to you. Define the following:

   a. An attribute *tickerSymbol* that represents the standard ticker symbol for the stock. (*E.g.,* "MSFT" for Microsoft.)

   b. A constructor, which, in addition to the parameters needed for `Security`, must take an expected return value and a ticker symbol.

   c. An *expectedReturn* attribute.

3. A `Bond` class that extends `Security`. When a company or government needs to raise funds, it can borrow money from investors by issuing a bond. The issuer promises to repay the bond with interest over a period of time, the *maturity* of the bond. An example of how this works is shown in figure 2.

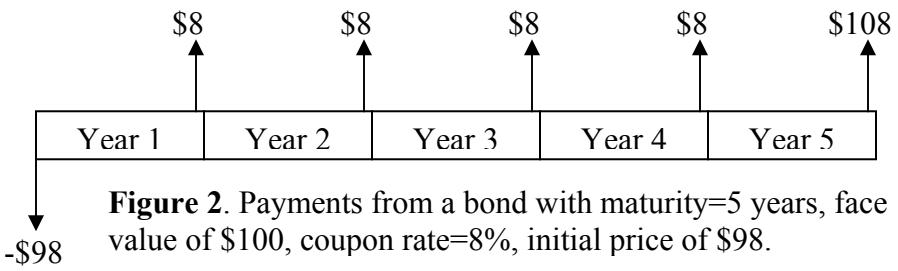

Figure 2. Payments from a bond with maturity=5 years, face value of $100, coupon rate=8%, initial price of $98.

Figure 2 is a timeline of the payments that a bondholder will receive after

purchasing a bond. Arrows going up indicate payments from the issuer of the bond to the bondholder. Arrows going down show payments from the bondholder to the issuer.

The bond in figure 2 has a maturity of 5 years. After 5 years, the purchaser of the bond is entitled to the *face value* of the bond, which is $100 in this example. A time t=0, the purchaser buys the bond by paying the initial price of the bond, $98 in the example. After that, the purchaser receives coupon payments at the end of each year, based on a *coupon rate* until the bond matures. In the example, the coupon rate is 8% of the face value. Thus, the final payment to the borrower at the end of year 5 is the sum of the face value ($100) and the last coupon payment ($8).

Now we consider how to calculate the expected return of a bond. The return on an asset in general is: (FinalValue – InitialValue)/InitialValue. If we summed all the coupon payments and final payment, we would get 8+8+8+8+8+100=$140 as our final value. However, since each of the coupon payments are paid out at different times, the final value would in fact be more than $140, as you could invest those coupon payments in a safe financial instrument such as a money market account.

To understand the final value of the bond, fast forward to the time of maturity of the bond. You receive the face value of the bond, $100 plus the final coupon of $8. One year ago you received an $8 coupon, which you invested in a money-market account that yielded 5% annual interest rate. That $8 is now worth $8 x 1.05 = $8.40. You do the same thing with the other payments, resulting in the following final value:

Year	*Cash*	*Compounded*
1	$8	$8x1.05^4 = $9.72405
2	$8	$8x1.05^3 = $9.261
3	$8	$8x1.05^2 = $8.82
4	$8	$8x1.05^1 = $8.40
5	$108	$108.00
total		$144.20505

The rate at which we can invest proceeds is called the *risk-free rate*, and it will be the same for all bonds. (Often, the yield on short-term government bond is used as the risk-free rate.)

We added up each of compounded coupon payments manually in the table above. In general, however, you can use the formula for the "future value of an ordinary annuity" as a shortcut. This will give you the final value of the coupon payments (but you still need to add in the face value of the bond). This formula is:

$FVoa = c \times \frac{(1+r_f)^n - 1}{r_f}$, where c is the coupon payment, and r_f is the risk-free rate.

We are now equipped to calculate the return on the bond in figure 2. Using our equation for return, we have:

$$r = \frac{FinalValue - InitialValue}{InitialValue} = \frac{144.20505 - 98}{98} = 0.4715 = 47.15\%$$

This is the return over five years. To get the average annual return (the expected return for Bond), we take the geometric average of the return:

$$\sqrt[n]{1+r} - 1 = \sqrt[5]{1.4715} - 1 = 0.0803 = 8.03\%$$

where n is the maturity of the bond. Note that if the risk-free rate never changed, then the *expected return* of a bond would also be the *actual return* of the bond. In real life, the risk-free rate changes all the time based on macroeconomic factors, so there is a non-zero standard deviation for the return on coupon bonds.

Write a `Bond` class that extends `Security`. It must include the following:

a. A constructor that takes as arguments the name of the bond, the standard deviation of the annual return, an integer number of years to maturity, the face value, the coupon rate, and the initial price of the bond.

b. An integer attribute *maturity* which represents the number of years to maturity.

c. An attribute *faceValue* that represents the face value of the bond.

d. An attribute *couponRate* that represents the coupon rate of the bond.

e. An attribute *initialValue* that is the initial price of the bond.

f. A static attribute *riskFreeRate* that represents the risk-free rate. Give it an initial value of 1.7%.

g. Override the `getExpectedReturn` method to calculate the expected return of a coupon bond. (You can do this in two lines.)

4. Implement a `ZeroCouponBond` class that inherits from your `Bond` class. All you need to write is the constructor that uses the following signature:

```
ZeroCouponBond(String name, int yearsToMaturity, double yield);
```

A zero-coupon bond does not pay coupons. The return to the investor is just derived from the final payment. For this problem set, we assume that the risk-free is the same for all maturities (a "flat yield curve").

We assume you cannot resell the zero-coupon bonds before maturity, which means that the standard deviation for zero-coupon bonds in this exercise is zero. The face value for all zero-coupon bonds is $100.

The initial price of a zero coupon P_0 can be computed from yield is as follows.

$P_0 = \frac{100}{(1+y)^n}$, where *y* is the yield, and *n* is the number of years to maturity.

5. In your `main` method, do the following:

   a. Create `Stock` objects for Microsoft and Red Hat, Inc:

company	*ticker*	*Expected return*	*Std dev*
Microsoft	MSFT	34%	19.49%
Red Hat, Inc.	RHAT	-7.9%	48.99%

   b. Create a `Bond` object representing a Treasury Note from the US government with a coupon rate of 2.875%, standard deviation of 1.6%, maturity of 2 years, face value of $100, and initial price of $100.

   c. Create a `ZeroCouponBond` object with maturity of 5 years and yield of 5%.

   d. Create an array of `Security` objects to hold your stock, bond and zero-coupon bond instances. Print out each security in the array by calling its `printMe` method.

   e. Prompt the user for the number of trials to run (*trialCount*). For each security in your array, call `getActualReturn` *trialCount* times. For each security, calculate the average expected return for the simulation and print it out (also rounding to 1/10000th).

      Sample output is shown below:

```
Name: Microsoft stock, expected return: 34.0%, standard deviation: 19.49%
Name: Red Hat stock, expected return: -7.9%, standard deviation: 48.99%
Name: 2-year Treasury Note, expected return: 2.86%, standard deviation: 1.6%
Name: 5-year Zero Coupon Bond, expected return: 5.0%, standard deviation: 0.0%

RESULTS OF RUNNING 10 TRIALS:
Microsoft stock average return: 26.24% (expected 34.0%)
Red Hat stock average return: -50.29% (expected -7.9%)
2-year Treasury Note average return: 3.32% (expected 2.86%)
5-year Zero Coupon Bond average return: 5.0% (expected 5.0%)
```

Run the simulation with 10, 100, and 10,000 trials and consider the results. What can you say about the results as the number of trials increases? Include your answer in the comments. (You do not need to turn in the output of your simulation run.)

Note to financial purists: we have made several assumptions to simplify the problems in this problem set; for example, we have made the (unreasonable) assumption that the returns on securities are completely independent, *i.e.*, that their co-variances are all zero. We have also assumed that historical volatility remains constant, which is not true in the real world.

Turnin

Turnin Requirements

- Hardcopy and electronic copy of ALL source code (all .java files).
- Place a comment with your name, username, section, TA's name, assignment number, and list of people with whom you have discussed the problem set on ALL files you submit.
- Do NOT turn in electronic or hardcopies of compiled byte code (.class files).

Electronic Turnin

Use *SecureFX* (or another secure ftp or secure shell program) to upload your problem set to your 1.00 homework locker.

Detailed instructions of how to upload are on the course website.

Since your problem set is due at the beginning of lecture, your uploaded problem should have a timestamp of no later than morning on the due date.

Penalties

- Missing Hardcopy: -10% off problem score if missing hardcopy.
- Missing Electronic Copy: -30% off problem score if missing electronic copy.
- Late Turnin: -30% off problem score if 1 day late. More than 1 day late = NO CREDIT.

If your problem set is late, or if a professor has granted you an extension in advance, do not submit a printed copy of your problem set.