

Problem Set 1

Due: Thursday, September 12

Submission:

- Problem 1 is a pencil-and-paper problem that only needs to appear in your hardcopy submission.
- Your hardcopy submission for Problem 2 should be (1) the file `Mortgage.java` and (2) a printout of the output requested in the problem.
- Your hardcopy submission for Problem 2 should be (1) the file `FindPrincipal.java` and (2) a printout of the output requested in the problem.

Remember to include a signed cover sheet (found at the end of this problem set description) at the beginning of your hardcopy submission.

Your softcopy submission should be your entire `Mortgage` directory.

Problem 1 [35]: Object Diagrams

In this problem, you will draw some object diagrams to illustrate your understanding of the Java Object Model and the important notion of sharing. Study the code for the `Pair` class in Fig. 1. You are asked to draw object diagrams that are “snapshots” of the state of the `Pair` application at two “snapshots” during its execution:

1. Draw an object diagram that depicts the state of the program immediately *after* the statement

```
boolean [ ] bools1 = comparisons(pairss);
```

is executed.

2. Draw an object diagram that depicts the state of the program immediately *after* the statement

```
boolean [ ] bools2 = comparisons(pairss);
```

is executed.

Recall that an object diagram shows the state of the objects in `ObjectLand`. There are two kinds of objects: (1) class instances, which are drawn as boxes labelled by their class name that contain instance variables labelled by their names and (2) array objects, which are drawn as boxes labelled by their array type that contain variables labelled by their index. Variables themselves are drawn as boxes that contain either primitive data (e.g., numbers, booleans, characters) or references (pointers) to other objects. Sharing is indicated by having two pointers that point to the same object.

Your object diagrams should *not* show execution frames, but they should include the following local variables of the `test` method: `pairss`, `a`, `bools1`, and `bools2`. These variables may be shown “floating” in the object diagrams. The diagrams should also show all objects that are reachable from these four variables. All such objects (both instances and arrays) should be labelled with their type, and should be drawn as boxes that contain labelled variables.

Notes:

```

public class Pair {

    // Instance variables:
    public int left;
    public int right;

    // Constructor method:
    public Pair (int l, int r) {
        left = l;
        right = r;
    }

    // Instance method:
    public boolean equals (Pair p) {
        return (left == p.left) && (right == p.right);
    }

    // Class methods:
    public static boolean [ ] comparisons (Pair [ ] [ ] pairArrays) {
        boolean [ ] bools = new boolean [8];
        for (int i = 0; i < 4; i++) {
            bools[i] = (pairArrays[i][0] == pairArrays[i][1]);
        }
        for (int j = 0; j < 4; j++) {
            bools[j+4] = (pairArrays[j][0].equals(pairArrays[j][1]));
        }
        return bools;
    }

    public static void test() {
        // Note: new Pair [4] [ ] creates a 4-slot array
        // whose slots hold objects of type Pair [ ].
        Pair [ ] [ ] pairss = new Pair [4] [ ];
        Pair [ ] a = new Pair [3];
        a[0] = new Pair (1,2);
        a[1] = a[0];
        a[2] = new Pair (3,4);
        pairss[0] = a;
        pairss[1] = a;
        pairss[2] = new Pair [3];
        pairss[2][0] = new Pair(1,2);
        pairss[2][1] = pairss[0][1];
        pairss[2][2] = pairss[1][2];
        pairss[3] = pairss[2];
        boolean [ ] bools1 = comparisons(pairss);
        // *** Object Diagram #1 should depict this point ***
        pairss[1][1] = a[2];
        pairss[3][0] = pairss[3][1];
        boolean [ ] bools2 = comparisons(pairss);
        // *** Object Diagram #2 should depict this point ***
    }

    public static void main (String [ ] args) {
        test();
    }
}

```

Figure 1: A sample Java class.

- The equality operator `a==b` is true for objects `a` and `b` if and only if `a` and `b` are the exact same object – i.e., they were created by the same call to `new`. In an object diagram, two object references are `==` if and only if they point to the same instance or array on the page.
- The `equals` method for pairs returns true if and only if they are structurally equivalent – i.e., they have the same `left` and `right` components. Two pairs may be `equals` without being `==`, but two pairs that are `==` are guaranteed to be `equals`.
- For this problem, you should submit only a harcopy version of your two object diagrams. You can use pencil and paper for your diagrams – you need not format them on a computer! (Unless you are very adept with drawing software, formatting the diagrams on the computer will take you much longer than drawing them by hand.)

Problem 2 [35]: Mortgage Payments

You're thinking of buying a house! You've decided to use Java to see what sort of monthly mortgage payments you can afford.

There are three relevant parameters in determining a mortgage:

1. The *principal*, which is the cost of the house after the downpayment.
2. The *annual interest rate*, measured in percent. These are hovering around the unusually low value of 6% these days.
3. The *number of years* in the mortgage, typically 30.

Let p be the principal, i be the annual interest rate, and y be the number of years in the mortgage. Then the monthly mortgage rate can be calculated according to the following formula:

$$\frac{p \cdot \left(\frac{i}{1200}\right)}{1 - \left(\frac{1}{1 + \frac{i}{1200}}\right)^{12 \cdot y}}$$

In this formula, the interest parameter i should be a number in the range 0 to 100 rather than 0 to 1. That is, 7.25% is expressed as 7.25, not 0.0725.

For example, the monthly mortgage payment on a \$500,000 house (after downpayment) with 8% interest and a 30-year mortgage is \$3668.82. After 30 years, the total mortgage payments amount to \$1,320,776 = 2.64 times the listed cost of the house. This is how banks make money!

In this problem you should do the following:

1. Create a directory named `Mortgage`.
2. In the `Mortgage` directory, create a file name `Mortgage.java` that contains a public class named `Mortgage`.
3. In the `Mortgage` class, flesh out the following class methods (a.k.a. *static* methods, a.k.a. functions):
 - A class method named `mortgage` that calculates and returns the monthly mortgage payment according to the above formula. The method should take as its parameters the three parameters that determine a mortgage. Use `Math.pow(base, exp)` to calculate the result of raising the number `base` to the exponent `exp`. Floating point parameters should have the type `double`, *not* `float`.

- A class method named `print` that takes the same parameters as `mortgage`, but prints out the parameter information, monthly mortgage payment, and 30-year total mortgage payment in the following format:

```
principal = 500000.0; interest = 7.25; years = 30.0;
mortgage = 3410.881400280954; total = 1227917.3041011435
```

- A class method `main` that for a principal of \$500,000 and 30 year mortgage, prints out one line in the above format for every interest rate between 5% and 10% in increments of 0.25. The `main` method should take a single argument of type `String []` (which is ignored in this and many other `main` methods).

For this problem, your hardcopy submission should contain a code listing for the `Mortgage` class and a printout of the output of executing the `main` method of this class. Your softcopy submission for this problem (and Problem 3 as well) should be your `Mortgage` directory.

Notes:

- Your `Mortgage` class should contain only class methods. It should not contain class variables, constructor methods, instance variables, or instance methods.
- To print out `Mortgage.java`, do one of the following:
 - Go to an Emacs editor buffer displaying `Mortgage.java` and select one of the menu options `File>Print Buffer` or `File>Postscript Print Buffer`.
 - Go to a shell, `cd` to the `Mortgage` directory, and execute the command `lpr Mortgage.java`.
- To print out output of executing the `main` method:
 - If you have executed the `main` method within a shell running within Emacs, you simply select one of the menu options `File>Print Buffer` or `File>Postscript Print Buffer` for the shell buffer.
 - If you are in a shell *outside* of Emacs, then you first need to save the output of the command `java Mortgage` to a file. You can do this using Unix file redirection as follows:

```
java Mortgage > out.txt
```

The above command sends the printed output of `java Mortgage` to the file `out.txt` rather than to the console. Once the data is in a file, use `lpr out.txt` to print it.

Problem 3 [30]: It's the Principal of the Thing

The mortgage method in Problem 2 tells you the monthly mortgage given the principal, interest rate, and the number of years. But suppose instead that you have determined the monthly mortgage M that best fits your budget and want to determine the principal P for the most expensive house you can afford for that mortgage (for a given interest rate and number of years).

One approach is to algebraically manipulate the formula from Problem 1 to express P as a function of M , interest, and the number of years. But you are rusty on your algebra, so you decide not to take that tack.

An alternative approach is to use the `mortgage` method from Problem 1 as a way of evaluating a sequence of guesses at the desired principal P . The principal P you seek is the one for which

`mortgage` returns M . You can use the binary search idea from the HiLo game as a way of effectively guessing P . In particular, suppose you know that P is in the closed interval $[lo, hi]$. Then because `mortgage` is a monotonically increasing function, you can use the result of applying `mortgage` to the midpoint of lo and hi to narrow P to be in one half of this interval. By successively halving the interval, you can quickly converge to a result P' whose mortgage M' is within one penny of M . At that point, you can declare P' as the desired principal.

```

public class FindPrincipal {

    public static double find (double mortgage, double interest, double years) {
        return findBetween(mortgage, interest, years, 0,
                           upperBound(mortgage, interest, years));
    }

    public static double findBetween (double mortgage,
                                      double interest,
                                      double years,
                                      double lo,
                                      double hi) {

        // Flesh this out.
    }

    public static double upperBound (double mortgage,
                                      double interest,
                                      double years) {

        // Flesh this out.
    }

    public static void testFind (double m, double i, double y) {
        System.out.println("find(" + m + ", " + i + ", " + y + ") = " + find(m,i,y));
    }

    public static void main (String [] args) {
        // Flesh this out.
    }
}

```

Figure 2: A sample Java class.

Implement this idea by fleshing out the missing parts of the `FindPrincipal` class in Fig. 2, which you should implement in a file `FindPrincipal.java` within your `Mortgage` folder. The `find` method defers to `findBetween` to find a principal in the interval $[0, u]$, where u is an upper bound on the desired principal P – i.e., u is guaranteed to be greater than or equal to P . The `find` method should use binary search to converge to a principal P whose mortgage is within one dollar of the mortgage parameter to `findBetween`. (Use `Math.abs` to calculate absolute values.) The `upperBound` method determines an upper bound u for P . It should implement the following strategy: starting with the principal 1, successively double it until reaching a principal Q whose mortgage is greater than the given mortgage limit. Q is clearly an upper bound for P . The `main` method should test your code by determining the maximum house price that can be purchased

for a \$1500.00 per month mortgage in a 30-year mortgage at all interest rates between 5.0% and 10.0%, at 0.25% increments. The `main` method should print out lines of the form:

```
find(1500, 7.75, 30.0) = 209376.0
```

with one line for each interest rate between 5.0% and 10.0%, using 0.25% increments.

For this problem, your hardcopy submission should contain a code listing for the `FindPrincipal` class and a printout of the output of executing the `main` method of this class. Your softcopy submission for this problem (and Problem 2) should be your `Mortgage` directory.

Problem Set Header Page
Please make this the first page of your hardcopy submission.

CS230 Problem Set 1

Due Thursday September 12

Name:

Date & Time Submitted:

Collaborators (*anyone you worked with on the problem set*):

By signing below, I attest that I have followed the collaboration policy as specified in the Course Information handout.

Signature:

*In the **Time** column, please estimate the time you spend on the parts of this problem set. Please try to be as accurate as possible; this information will help me design future problem sets. I will fill out the **Score** column when grading your problem set.*

Part	Time	Score
General Reading		
Problem 1 [35]		
Problem 2 [35]		
Problem 3 [30]		
Total		