1.00/1.001 Introduction to Computers and Engineering Problem Solving

Fall 2002

Problem Set 3

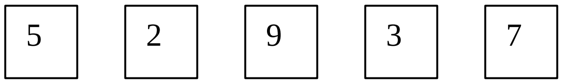
Due: Day 11

Problem 1. Finding the Median (15%)

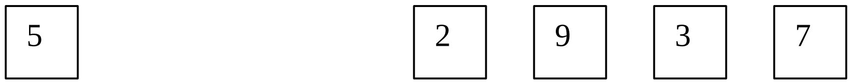
Write a method that takes in an integer array and returns an integer that is the median of the array. For example, if array A = { 5, 2, 10, 3, 7, 8, 6 }, the returned value should be 6. The median can be found by sorting the array first and then choosing the middle value. Therefore, if you have an array like {2, 3, 3, 3, 3, 3, 3, 3, 3, 4, 5, 6, 7}, the median should be 3. For an array with even number of elements, calculate the average of the two middle values. For example, array B = { 4, 6, 11, 9 }, the median should be (6+9)/2. Since this method should return an integer, we would accept value with either 7 or 8 in this case. In `main()`, write a test that verifies that your implementation is correct.

There are many sorting methods, you are welcome to use anything you know. Here we will introduce **insertion sort**. It is an efficient algorithm for sorting a small number of elements. Insertion sort works the way many people sort a hand of playing cards. We start with an empty left hand and the cards face down on the table. We then take the first card off the top and hold it in our left hand. Next we take the second card off the top and then compare with the first card and put it in place. Then the third card is taken and compared to the two cards we have already sorted and again placed in the right position.

Basically we take one card at a time from the table and insert it into the correct position in the left hand. The way to do this is by comparing it with each of the cards already in the hand, from right to left. At all times, the cards held in the left hand are sorted. If we are sorting an array { 5, 2, 9, 3, 7 } with increasing order,

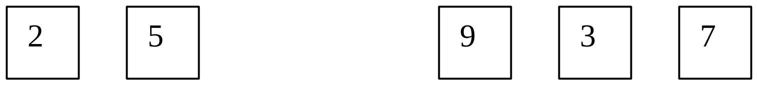


We take the first number out (which is 5) and by itself, it is sorted.

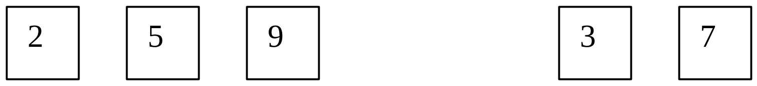


We look at the second number and notice that the left hand side is not sorted.

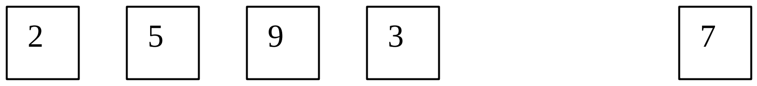
Therefore we switch the two numbers and make them sorted.



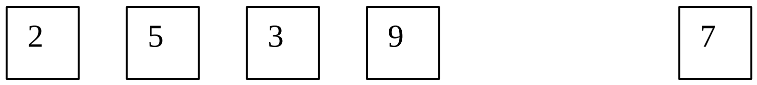
Now the left hand side is sorted, let us look at the next number 9, and the left hand side is still sorted.



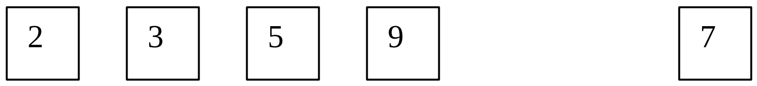
Next is 3,



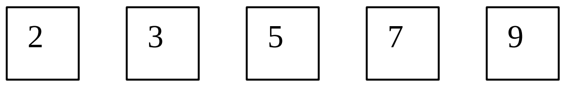
The left hand side is not sorted anymore, because 3 is not greater than 9. Therefore, we swap them.



The left hand side is still not sorted. We will continue to swap the number 3 with the number that comes before it until the left hand side is sorted.



Now we have one more number to go, we find that 7 is less than the number that comes before it(which is 9), so we will use the method that we used before to put the number 7 in the correct position such that the entire array is sorted. At the end, we would get



(Hint: In Java®, you would need two loops to achieve sorting, one within the other)

Problem 2. Median Filtering (85%)

Introduction

In this problem,nyou will learn a bit about how modern computers represent and manipulate images. More importantly, you will write a class that will perform a simple but important image restoration technique called median filtering. We will return to image display and computer graphics later in the course. In order to shield you from some of the messy details, this problem set will supply four of the five classes you will need to solve the problem. But in writing your class to fit into the structure provided by the prewritten code, you will practice another important skill -- learning to use a preexisting API (application programming interface).

Images and Color

Computers store images on secondary storage like hard disk in many formats. Often these formats perform image compression because images can be very large. Most modern computers, however, represent images in main memory in one of only a very limited number of formats. Java® has chosen one of these, arguably the simplest and most general, as the standard. In this format, an image is represented as a rectangular array of color points or pixels. Each pixel is represented by three small numbers, a red, green, and blue intensity, since most colors that we can perceive can be represented by a combination of these three additive primary colors. Java® also includes a fourth small number called the alpha component that represents the color's transparency, but we will ignore it in this problem set. Most humans can not distinguish more than 256 levels of any color so Java® uses 8 bits or a byte to represent each of the color and alpha components. This has the advantage that the whole pixel can be packed into a single 32-bit integer or int. (For those who are interested, the alpha goes into the high order byte, then the red and green components, and finally the blue component goes in the low order or rightmost position in the int.) Java® has a number of classes that can be used to represent and manipulate images. These classes allow you to access individual pixels or blocks of pixels and also to disassemble the color components.

The Problem

My baby sister’s birthday is coming up in two weeks. However, our beloved dog just passed away recently. She misses him a lot. There is an old picture of him, unfortunately our neighbor’s baby boy played around with it and bent it. There are a few scratches on the photo. I have scanned the picture and saved it as a jpeg file. I am hoping that you could implement a median filter function that will get rid of the defects in the photo. You will definitely make my sister very happy on her birthday.

Median Filter

In this problem, you will write a simple program that does median filtering. Median filters are useful when there are impulse spikes (noise; eg. dust, scratches) in the images. When the median filter is applied over a pixel A, it looks at the neighbor pixel values and its own pixel value, and assigns pixel A the median of those values.

There are some pre-written classes provided for this problem which you can find on the 1.00 class website. A simple description of these files is provided below:

`Pixel.java`, which contains the static methods to unpack and repack the color components from the **`int`** representation of a pixel.
`ImageComparator.java`, which displays the original and the resultant image side by side.
`FilterOp.java`, which is a glue class that provides some boilerplate needed for Java® image handling.

Problem2Main.java, which contains the **main()** method, loads the image and coordinates the rest of the program.
puppy.jpg, which is the picture you are going to filter.

In this problem, you are to write a class called **MedianFilter**. When combined with the four classes that we are supplying to you, it will form an application that will load and display an image named on the command line, perform median filtering on the red, green, blue components and then redisplay the filtered image side by side with the original.

This class should have one private integer data member **size**, which is the size of the filter. The size of the filter refers to the 'neighborhood' over which the median is to be found for any pixel except those near the edges of the image. Here we will only consider square filters with odd length (for example: 3x3, 5x5, etc.) You should initialize this value in the constructor.

Your should start your **MedianFilter** class with the following import statement:

```
import java.awt.image.*;
```

Then declare the data field and write its constructor. Your constructor should check if the filter size is valid.

Implement the following method in the **MedianFilter** class:
public void filter(BufferedImage source_image, BufferedImage result_image)

You can read about **BufferedImage** in the online documentation, but we will tell you everything that you need to know here. Here are some useful methods from the classes **BufferedImage** and **Pixel**.

BufferedImage:
int getWidth(): returns the width in pixels of the **BufferedImage** instance it is called on.
int getHeight(): returns the height.
int getRGB(int x, int y): returns the packed red, green, and blue components of pixel (x,y) in the **BufferedImage** it is called on, as an **int**.
void setRGB (int x, int y, int rgb) : sets a pixel in this BufferedImage to the specified RGB value.

Pixel:
static int getRed(int pix): takes an **int** argument as returned by **getRGB()** and returns its red component as an **int**.
static int getGreen(int pix), **static int Pixel.getBlue(int pix)**: similar to **getRed(int pix)** above.

`static public int createRGB(int r, int g, int b)`: takes three **`int`** arguments representing the red, green and blue component of a pixel and create the corresponding pixel.

The **`filter()`** method should go through the `source_image` pixel by pixel, and determine the new value for each pixel. For each pixel, the filter should perform median filtering on the red, green and blue components and use the results to create the new pixel value. You can assume that `source_image` and `result_image` are the same size. Pixel values have to be positive integers. The `result_image` is simply an empty image to contain the result of the median filter.

Lets suppose we have a 5x5 single color image, and a filter of size 3 (a 3x3 square filter).

2	7	3	2	2
5	3	7	5	6
7	4	9	6	4
1	3	5	5	3
9	2	8	4	6

So first we look at pixel at position (0, 0), the median among (2, 7, 5, 3) (values at position (0, 0), (0, 1), (1, 0), (1, 1)) is 4 and 4 should be its new value after using median filter.

2	7	3	2	2
5	3	7	5	6
7	4	9	6	4
1	3	5	5	3
9	2	8	4	6

The pixel at position (0, 1), the median among (2, 7, 3, 5, 3, 7) is 4.

2	7	3	2	2
5	3	7	5	6
7	4	9	6	4
1	3	5	5	3
9	2	8	4	6

The pixel at position (1, 1), the median among (2, 7, 3, 5, 3, 7, 7, 4, 9) is 5, and 5 will be the new value at position (1, 1).

If you have filtered this entire 5x5 image, you should get:

4	4	4	4	3
4	5	5	5	4
3	5	5	5	5
3	5	5	5	4
2	4	4	5	4

The median is always found using the pixel values from the original figure and not using the new pixel values. You should notice that for pixels that are at the four corners, you only look at 4 values and choose the median, whereas those on the sides(but not on the corners), you would have to look at 6 values and choose the median. For the rest(those in the middle), you would need to look at 9 values before choosing the median.

<u>How is the result image when you choose a filter size of 3? What if you use a filter size of 5? Comment on the advantage and the disadvantage of having a larger filter size.</u> (Include your answers to these questions at the end of your **`MedianFilter`** class.) Read the next few pages for instructions on how you should set command line arguments in Forte before you run/execute your code.

Extra Credit (25 %)

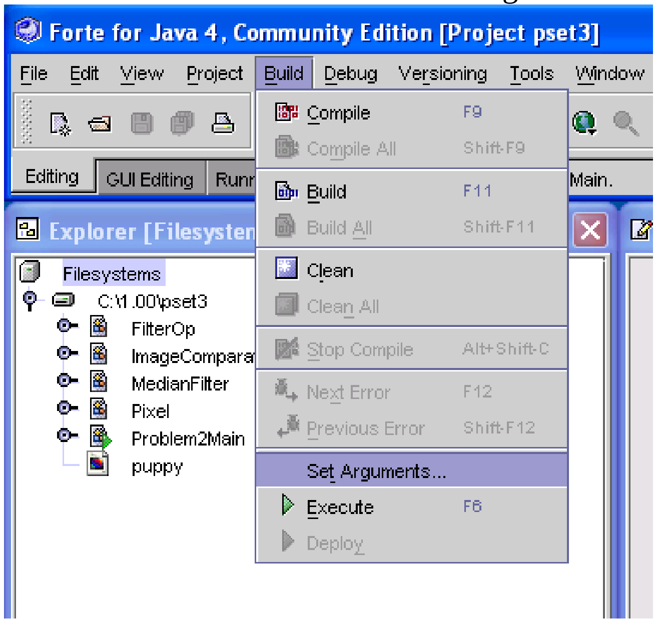
1 . Make changes in **`main()`** such that the user could enter the number of times the image will be filtered before outputing the final image.

2 . Explain(in a comment in the file) why there should not be a **`System.exit(0)`**, at the end of **`main()`**.

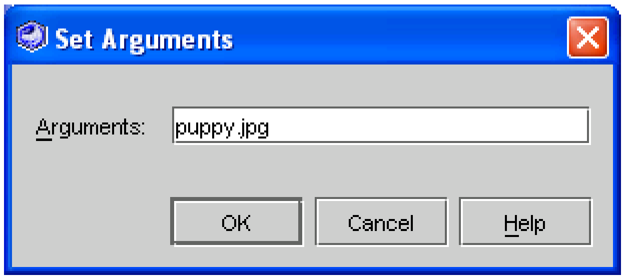
Compile and Run…

You should make sure all your files compile correctly before you do the following: (please also make sure you have mounted a file system (the folder where all your problem set 3 files are))

Go to "Build" and then choose "Set Arguments…"



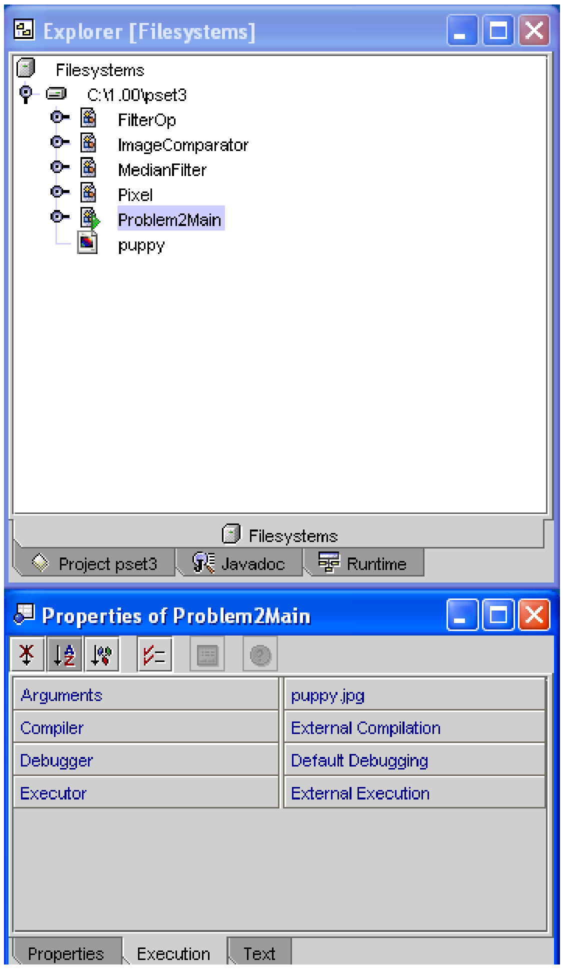
Set your argument as "puppy.jpg" as the following



Arguments: puppy.jpg

then click ok. Now we have set the argument we want to pass to the **`main()`** function, we are going to set the working directory to be the same as our Filesystems directory.

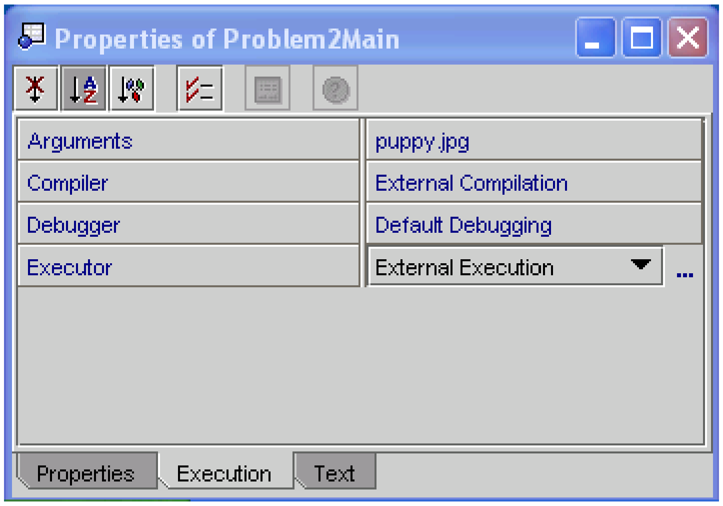
Click on “Problem2Main”, then you should see that below the “Explorer” window is the “Properties of Problem2Main” window. Click on the “Execution” Tab.



```
Filesystems
  C:\1.00\pset3
    FilterOp
    ImageComparator
    MedianFilter
    Pixel
    Problem2Main
    puppy
```

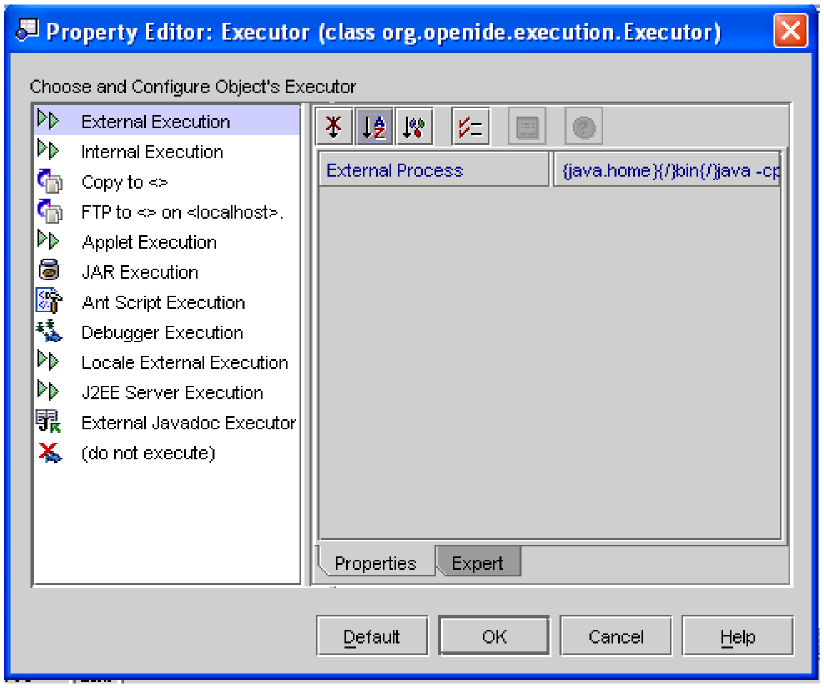
Arguments	puppy.jpg
Compiler	External Compilation
Debugger	Default Debugging
Executor	External Execution

Then click on “External Execution” and click again on “…”



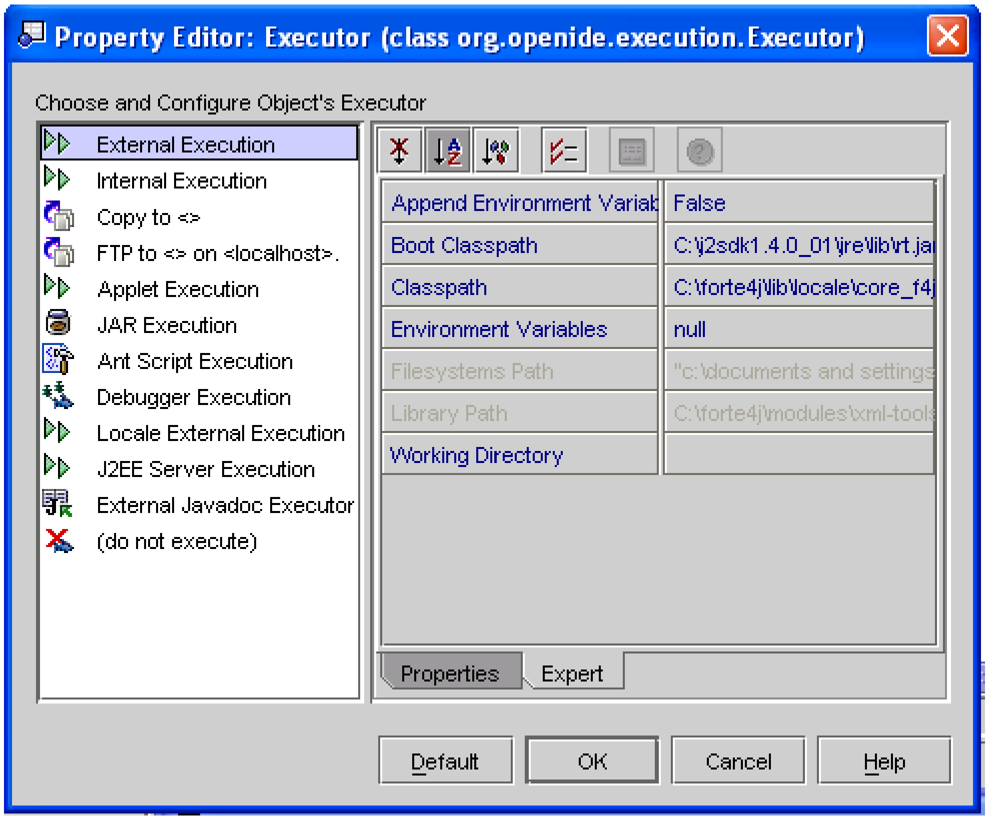
Arguments	puppy.jpg
Compiler	External Compilation
Debugger	Default Debugging
Executor	External Execution

Then a window which looks like the following should pop up.



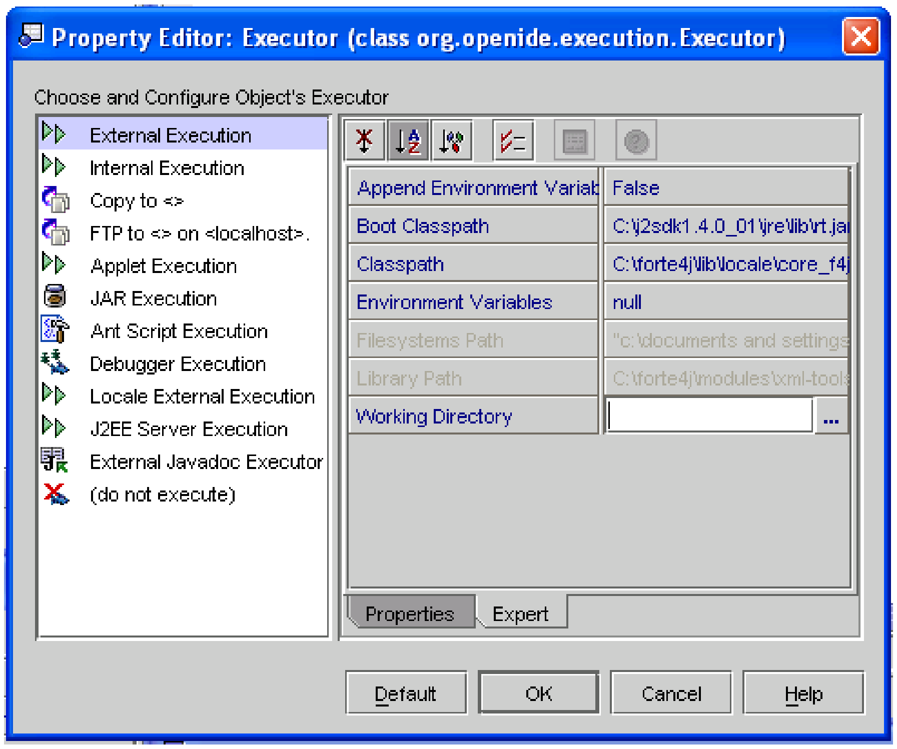
External Process	{java.home}{/}bin{/}java -cp

Click on the “Expert” tab, and you should see the following window.



Append Environment Variab	False
Boot Classpath	C:\j2sdk1.4.0_01\jre\lib\rt.ja
Classpath	C:\forte4j\lib\locale\core_f4j
Environment Variables	null
Filesystems Path	"c:\documents and settings
Library Path	C:\forte4j\modules\xml-tools
Working Directory	

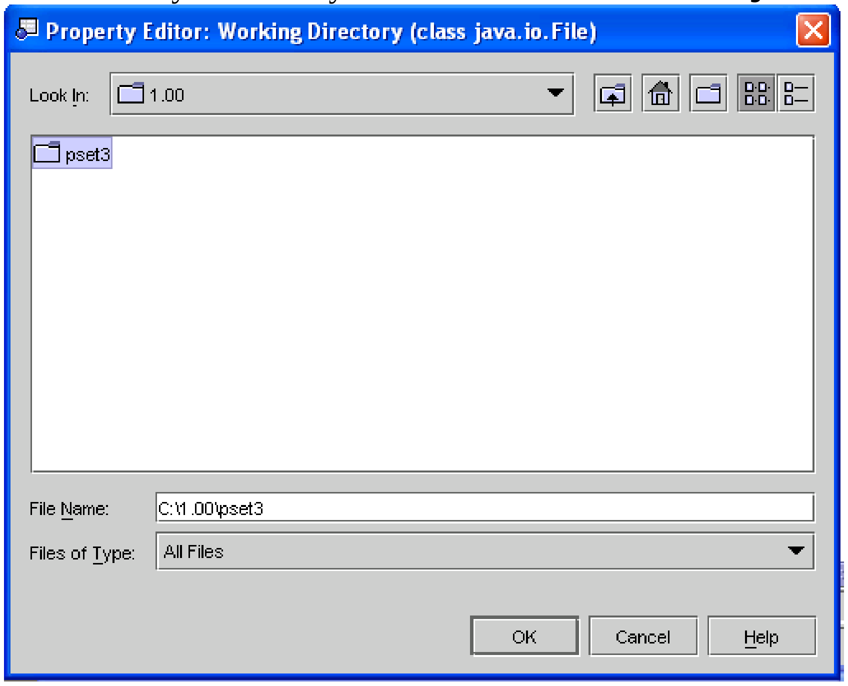
Then click on the cell next to “Working Directory” and click again on “…”



Append Environment Variab	False
Boot Classpath	C:\j2sdk1.4.0_01\jre\lib\rt.ja
Classpath	C:\forte4j\lib\locale\core_f4j
Environment Variables	null
Filesystems Path	"c:\documents and settings
Library Path	C:\forte4j\modules\xml-tools
Working Directory	...

Now please choose the same folder as your Filesystem. In this example, it is “C:\1.00\pset3”

Click ok after you are finished. Click ok again then all the pop up windows should not be closed. Now you are ready to run/execute **`Problem2Main.java`**.



File Name: C:\1.00\pset3
Files of Type: All Files

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.