Problem Solving

Definitions:

Problem – a general problem that you are asked to write a program (or an algorithm) for.

E.g.: Pb speed: Write a program that given speed (in miles per hour) and distance (in miles) computes required time (in hours and minutes) to reach a destination.

Instance of a problem – the same problem, but with specific data provided.

E.g.: Instance 1 (of pb speed): if speed is 60mph and distance is 120miles, what is the time?

Instance 2 (of pb speed): if speed is 60mph and distance is 100miles, what is the time?

Instance 3 (of pb speed): if speed is 60mph and distance is 35miles, what is the time?

READ IN THE DATA: Before starting problem solving and analyzing, it may help to familiarize yourself with the problem by writing a small program where you just read in the data. Advantages:

1. Have started on the problem
2. You detect/clarify any misunderstandings related to: data type (e.g. int, double, string), what format (on the same line, separate lines, strings with spaces in them, etc)
3. Later on you are ready to move on to the actual solving.
4. It may help you understand the problem better: it gives you a clear picture of how you represent in the program each piece of data discussed in the problem.

Steps:

1. Be able to “solve on paper” specific “instances” of that problem.
   a. You may need to use additional resources (like the web) to get missing information (e.g. conversion rate from km to miles, or dates used to horoscope signs). This info can (later) be hardcoded in your program.
2. Look at the data being used and name each piece of it
   a. Input - E.g. speed and distance
   b. Output - E.g. time
   c. Other intermediary data needed to produce the output - E.g. I may need to
      i. convert from km to miles, or
      ii. compute a sum (to be used later to compute an average) or
      iii. “collect” vowels seen so far (so I can only count them once).
3. Identify smaller component problems – E.g. “access” each letter of a string, produce numbers in decreasing order, computing a sum.
   a. If you identified a small problem that you do not know how to solve, look into that by itself.
      i. Try to find the closest thing to it that you know: Did you do or see anything at all similar to it
      ii. Identify as close as possible the “breaking point”: what you can do and what you cannot.
      iii. Get outside help. Use resources such as: Java API, web searches, past lectures, discuss with classmates/TA/instructor. If you did the first part above, when talking to someone you can explain what you know how to do and what makes “stuck”
      iv. Once you find a “possible answer” (e.g. a Java method, or a piece of Java code) UNDERSTAND it. Do not just use it without experimenting with it first to know how it behaves for special cases.
4. Identify the “solving process” that you used to solve the instances, and write it as a sequence of steps involving the type of data manipulations and “actions” allowed in a program:
   a. Collecting information (data) – either a complete overwrite or an update (e.g. sum = sum + current)
   b. Reading data from user
   c. Repetition of a sequence of actions
   d. Printing
   e. Decision (do one thing or another)
5. ***THINK ABOUT HOW THE DATA CHANGES throughout these steps*** - most of the time the gist of a program is manipulating data:
   a. Getting all the data needed as input: from user, or as method argument, or read from a file. Data can be hardcoded only if it can never change (e.g. transformation formulas from km to miles, number of days in a week), and even then, good practice uses a constant (use the “final” Java keyword)
   b. Storying data – you should not lose whatever you compute (use more variables). Most of the times, this refers to new pieces of data.
   c. Extracting data from other data – e.g. extract from a string char at a given position or a substring,
   d. Building new data – E.g. a sum, a concatenation of strings (e.g. first and last name to make full name, or repeated sentences to make a phrase.)
   e. Modifying data – update current bank balance according to transaction performed
6. Write (on paper) how the data you identified, changes for a specific instance of a problem. This has a double role:
   a. It verifies the process you just defined (as you should get the final result)
   b. You know exactly what you expect from your program and you can use this data to verify that at each step your program produces the data you want. Use a Debugger to execute the program step-by-step.
7. Write the program
   a. Use meaningful variable names
   b. Add comments especially for things that are not obvious or that are a bit special (e.g. prints a border of the pattern)
8. Test your program
   a. What if the user enters invalid data (e.g. negative speed) or data of the wrong type (string instead of int)
   b. What if the method is given an empty or a null string?
   c. Test border cases E.g., for the program with 5 chances to enter the correct password, what if the user guesses the correct password in the last allowed attempt (before being locked out)? The program should give access to the user, not lock them out.