Program Design Strategies

Design Strategies: Different strategies can be used to develop a computer program that implements a solution to a particular set of specifications.

Flowchart:

Pseudocode:

How much detail?

- possible to translate it into a programming language
- but it is free of language-dependent details

Program Design Strategies

Large software systems: Software engineering techniques are required to manage the process:

- requirements documents
- design documents
- design presentations and reviews
- software systems for rebuilding, performing tests, handling trouble reports.
- and many more...

Program Design Strategies

Smaller projects: Are usually maintained by a single programmer. They still require good clean design to reduce the effort in debugging and maintenance.

Object-Oriented design:

Design before implementing:

Pseudocode+Homeworks: For several homework assignments you will be asked to provide pseudocode for several methods.
Example: Olympic Scoring

(Semi)-Olympic Scoring:
- Input a list of integer scores,
- Drop the lowest score, and output the average of the remaining scores.
- Scores are given by the user as integers. The average is expressed as a double.
- Terminate when "quit" is entered.
- We assume at least two scores are given (otherwise the problem is not well defined). We do not do error checking, but this should be added for a complete program.

Example: Olympic Scoring

Attempt 1: (A really rough cut)

- Initialize variables
- Repeat
  - Input a score
  - Update variables
- until seeing "quit"
- Compute and output final average

Olympic Scoring: Attempt 2

Attempt 2: (A bit more detail) Will need at least the following variables:

- total: a running total of the scores seen so far.
- count: a count of the number of scores seen so far.
- min: the minimum score seen so far.
- score: the current score being processed.
- average: the final average

Question: Should total and count include the contribution of min?
Olympic Scoring: Attempt 2

Attempt 2: (continued)
- Initialize variables: total = 0; count = 0; min = ??;
- Repeat
  - Input score
  - Update total and count: total += score; count++
  - Update min: if (score < min) min = score
- until seeing "quit"
- Factor min out: total -= min; count--
- Compute the average: average = total / count;

Remaining issues:
  How to initialize min?

When "quit" is seen:

Olympic Scoring: Final Pseudocode

Attempt 3: We will initialize min to 11 (max score + 1). We will also add a check for "quit". We will create a boolean flag "isDone" and will set it to true when we see "quit".

- Initialize variables:
  total = 0; count = 0;
  min = 11;
  isDone = false;
- Repeat
  - Read inputString
  - If (inputString equals "quit") then set isDone = true
  - Otherwise convert inputString to numeric score and do the following:
    - Update total and count: total += score; count++
    - Update min: if (score < min) min = score
  - until isDone is true
- Factor min out: total -= min; count--
- Compute the average: average = total / count;

Sample Trace
It's a good idea to trace your pseudocode on a simple input to see that it will do what you expect. Input: {5, 7, 3, 2, 4, "quit"}

<table>
<thead>
<tr>
<th>Step</th>
<th>total</th>
<th>count</th>
<th>min</th>
<th>isDone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initially</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>false</td>
</tr>
<tr>
<td>Input 5:</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Input 7:</td>
<td>12</td>
<td>2</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Input 3:</td>
<td>15</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
Olympic Scoring: Almost correct

```java
import javax.swing.*;

public class OlympicScoring {

    public static void main( String[ ] args ) {
        int total = 0;   // sum of scores
        int count = 0;   // number of scores seen
        int min = 11;   // minimum so far
        int score;    // the current input value
        boolean isDone = false;  // are we done?

        do {
            // ...(the rest of the loop shown on the next slide)
        } while ( ! isDone );

        total -= min;   // remove minimum from total
        count--;    // ...and remove from the count
        double average =  total / count; // compute the final answer
        JOptionPane.showMessageDialog( null, "Average score: " + average );
        System.exit( 0 );   // terminate swing
    }
}
```

Olympic Scoring: Adding in the Loop

```java
do {     // repeat until sentinel seen
    String inputString = JOptionPane.showInputDialog( "Enter a score
        (Enter quit to terminate)" );
    if ( inputString.equals( "quit" ) ) {
        isDone = true;  // we are now done
    } else {
        score = Integer.parseInt( inputString );
        total += score;
        count++;
        if ( score < min )  // update minimum if needed
            min = score;
    }
} while ( ! isDone );
```
Olympic Scoring: Remaining Issues

Cryptic constants: In the initialization:

```java
int min = 11;
```

the constant 11 is hard to understand. We know how we got it (anything bigger than the maximum possible score of 10), but a reader of the code might not.

```java
final int MAX_VALID_SCORE = 10; // maximum valid score
int min = MAX_VALID_SCORE + 1;
```

The output is wrong: On the input test "3 4 5" we should drop the minimum 3 and return the average of 4 and 5, which is 4.5. But instead we get 4.0. Why? Total and count are int. Thus:

```java
double average = total / count;
```

We suffer from integer division truncation. Need a cast.

```java
double average = (double) total / (double) count;
```

General Suggestions for Implementation

Once you have designed a solution for a particular problem, proceed to implement the design. Here are some suggestions.

Incremental code development:

Make frequent backups: (in our environment AutoCVS takes care of that for us.)

Better variable names: Use refactoring to improve variables names. (Eclipse: Highlight a variable name, then right-click and select Refactor → Rename.)

Fix indentation: Stupid typing errors are often revealed by proper indentation (e.g. missing braces). (Eclipse: Highlight the entire procedure, right-click and select Source → Correct Indentation.)
More Suggestions for Implementation

Here are more suggestions.

**Test on "boundary cases":** Most errors occur in the limiting cases. Be sure to test these.

**Retest after each major change.**

**Don’t assume, verify:** Assumptions about how a construct or API method work can lead to future bugs.

**Debugger:** Later we will see a tool called a debugger, which will enable us to debug programs.

Testing and Debugging

**Good Testing is Critical:** A significant effort goes into this phase

**Start simple:** with basic test cases as you develop your code.

**Printing:** Use `System.out.println` in order to:

- determine the value associated with variables

```java
System.out.println( "total = " + total );
```

- trace the flow of execution

```java
public void foobar( ) {
    System.out.println( "Entering foobar" );
    // ... (the rest of foobar - omitted)
    System.out.println( "Exiting foobar" );
}
```