CS 241 — Introduction to Problem Solving and Programming

Fundamentals of Programming

The software development process and Input/Output

Jan 21, 2005

Software development

We've seen and practiced some basic tools. Let's apply them in a meaningful way.

The software development process:

- Specification
- Design
- Implementation
- Testing
- Maintenance

Along the way, documentation . . .

Specification

A program begins with a specification. What is the program supposed to do?

Example: a program to make change, as for a vending machine or cash register. Specification:

> Given an amount of money, display the number of quarters, dimes, nickels, and pennies needed to make that amount of change with the fewest number of coins.

Design

Once we have isolated the problem, we need an algorithm—a method for solving it.

> An algorithm is a set of instructions for solving a problem. To qualify as an algorithm, the instructions must be expressed so completely and so precisely that somebody could follow the instructions without having to fill in any details or make any decisions that are not fully specified in the instructions. (Savitch, 17)

Design

An algorithm should be

- Precise
- Easy to read by a human
- Programming language independent

Use pseudocode, an informal mixture of English and a programming language.

Design

An algorithm for our change-maker in plain English:

- Find the greatest number of quarters with value less than the amount.
- Record that number of quarters.
- Subtract the value of the quarters from the amount.
- Record the new amount.
- Find the greatest number of dimes with value less than the (new) amount.
- Record that number of dimes.
- Subtract the value of the dimes from the amount.
- Record the new amount.
- Find the greatest number of nickels with value less than the amount.
- Record the number of nickels.
- Subtract the value of the nickels from the amount.
- Record the new amount.
- Record the number of pennies as equal to the amount that is left.
- Display result

Design

The same algorithm in pseudocode:

- quarters = amount / 25
- quartersValue = quarters * 25
- amount = amount - quarterValue
- dimes = amount / 10
- dimesValue = dimes * 10
- amount = amount - dimesValue
- nickels = amount / 5
- nickelsValue = nickels * 5
- amount = amount - nickelsValue
- pennies = amount
- print quarters, dimes, nickels, and pennies

Now it's starting to look like Java.

Documentation and comments

We're ready to begin coding. Start by writing documentation, comments in code that the compiler ignores but explain the program to a human.

Comments in Java:

```
/*
     This is a block comment.
     Everything between the slash asterick pairs is ignored.
*/

// These are line comments
// Everything from the double slash to the end of the line is ignored
```

Documentation

Start each file with a block comment.

```
/**
 * MakeChange.java
 *
 * This program simulates a change-making machine. It computes
 * an arrangement of coins in US currency which use the minimum
 * number of coins whose value is a given amount.
 *
 * @author Thomas VanDrunen
 * Wheaton College, CS 241, Spring 2005
 * Assignment 27
 * Jan 21, 2005
 */
```

Documentation

Start with two asterisks, and line the left edge with asterisks.

```
/**
 * MakeChange.java
 *
 * This program simulates a change-making machine. It computes
 * an arrangement of coins in US currency which use the minimum
 * number of coins whose value is a given amount.
 *
 * @author Thomas VanDrunen
 * Wheaton College, CS 241, Spring 2005
 * Assignment 27
 * Jan 21, 2005
 */
```

Documentation

Begin with the name of the file.

```
/**
 * MakeChange.java
 *
 * This program simulates a change-making machine. It computes
 * an arrangement of coins in US currency which use the minimum
 * number of coins whose value is a given amount.
 *
 * @author Thomas VanDrunen
 * Wheaton College, CS 241, Spring 2005
 * Assignment 27
 * Jan 21, 2005
 */
```

Documentation

Then give a short description of the program.

```
/**
 * MakeChange.java
 *
 * This program simulates a change-making machine. It computes
 * an arrangement of coins in US currency which use the minimum
 * number of coins whose value is a given amount.
 *
 * @author Thomas VanDrunen
 * Wheaton College, CS 241, Spring 2005
 * Assignment 27
 * Jan 21, 2005
 */
```

Documentation

Put your name, preceded by "@author".

```
/**
 * MakeChange.java
 *
 * This program simulates a change-making machine. It computes
 * an arrangement of coins in US currency which use the minimum
 * number of coins whose value is a given amount.
 *
 * @author Thomas VanDrunen
 * Wheaton College, CS 241, Spring 2005
 * Assignment 27
 * Jan 21, 2005
 */
```

Documentation

Mention the college, class, semester, assignment, and date, like this.

```
/**
 * MakeChange.java
 *
 * This program simulates a change-making machine. It computes
 * an arrangement of coins in US currency which use the minimum
 * number of coins whose value is a given amount.
 *
 * @author Thomas VanDrunen
 * Wheaton College, CS 241, Spring 2005
 * Assignment 27
 * Jan 21, 2005
 */
```

Coding

Now we can start on the body of the program

```
public class MakeChange {
    public static void main(String[] args) {

        int amount;    // Holds the  amount of money to make change for

        amount = DocsIO.readint("Please enter a number--> ");

...
```

Coding

Document every variable declaration with an explanation in a line comment.

```
public class MakeChange {
    public static void main(String[] args) {

        int amount;    // Holds the  amount of money to make change for

        amount = DocsIO.readint("Please enter a number--> ");

...
```

Coding

Here's something new.

```
public class MakeChange {
    public static void main(String[] args) {

        int amount;    // Holds the  amount of money to make change for

        amount = DocsIO.readint("Please enter a number--> ");

...
```

This prompts the user for input. This displays the given string on the screen and pauses, waiting for the user to type at the keyboard and hit "enter." It returns an `int`.

Coding

```
        int quarters = amount / 25;            // The number of quarters
        int quartersValue = quarters * 25;  // The value of the quarters
        amount = amount - quatersValue;
        int dimes = amount / 10;             // The number of dimes
        int dimesValue = dimes * 10;      // The value of the dimes
        int amount = amount - dimesValue;
        int nickels = amount / 5;             // The number of nickels
        int nickelsValue = nickels * 5     // The value of the nickels
        int pennies = amount;       // The number (and value) of pennies
        System.out.println("Quarters: " + quarters);
        System.out.println("Dimes: " + dimes);
        System.out.println("Nickels: " + nickels);
        System.out.println("Pennies: " + pennies);
    }
}
```

Compiling

Let's try compiling.

```
ar1121: {400} javac MakeChange.java
MakeChange.java:23: cannot resolve symbol
symbol  : variable quatersValue
location: class MakeChange
        amount = amount - quatersValue;
                          ^
MakeChange.java:26: amount is already defined in main(java.lang.String[])
        int amount = amount - dimesValue;
            ^
MakeChange.java:28: ';' expected
        int nickelsValue = nickels * 5     // The value of the nickels
                                   ^
3 errors
```

What's wrong?

Compiling

```
MakeChange.java:23: cannot resolve symbol
symbol  : variable quatersValue
location: class MakeChange
        amount = amount - quatersValue;
                          ^
MakeChange.java:26: amount is already defined in main(java.lang.String[])
        int amount = amount - dimesValue;
            ^
MakeChange.java:28: ';' expected
        int nickelsValue = nickels * 5     // The value of the nickels
                                   ^
3 errors
```

It doesn't recognize the variable quatersValue because we have misspelled it. It should be quartersValue.

Compiling

```
MakeChange.java:23: cannot resolve symbol
symbol  : variable quatersValue
location: class MakeChange
        amount = amount - quatersValue;
                          ^
MakeChange.java:26: amount is already defined in main(java.lang.String[])
        int amount = amount - dimesValue;
            ^
MakeChange.java:28: ';' expected
        int nickelsValue = nickels * 5     // The value of the nickels
                                   ^
3 errors
```

This is a re-declaration of the variable amount. We should get rid of int, which makes it a declaration.

Compiling

```
MakeChange.java:23: cannot resolve symbol
symbol  : variable quatersValue
location: class MakeChange
        amount = amount - quatersValue;
                          ^
MakeChange.java:26: amount is already defined in main(java.lang.String[])
        int amount = amount - dimesValue;
            ^
MakeChange.java:28: ';' expected
        int nickelsValue = nickels * 5     // The value of the nickels
                                         ^
3 errors
```

We forgot a semi-colon.

Compiling

Compiler errors are not always so helpful. Compilers are not always so good at figuring out what's wrong.

Suppose we had used the -= shortcut:

```
ar1121: {17} javac MakeChange.java
MakeChange.java:26: ';' expected
        int amount -= dimesValue;
                   ^
1 error
```

The compiler gets hung up about it being a malformed declaration before noticing that the real problem is a re-declaration.

Coding

Corrected version:

```
int quarters = amount / 25;            // The number of quarters
int quartersValue = quarters * 25;  // The value of the quarters
amount -= quartersValue;
int dimes = amount / 10;             // The number of dimes
int dimesValue = dimes * 10;      // The value of the dimes
amount -= dimesValue;
int nickels = amount / 5;             // The number of nickels
int nickelsValue = nickels * 5;     // The value of the nickels
int pennies = amount;       // The number (and value) of pennies

System.out.println("Quarters: " + quarters);
System.out.println("Dimes: " + dimes);
System.out.println("Nickels: " + nickels);
System.out.println("Pennies: " + pennies);
```

Testing

Now we compile and run.

```
ar1121: 26 javac MakeChange.java
ar1121: 27 java MakeChange
Please enter a number--> 93
Quarters: 3
Dimes: 1
Nickels: 1
Pennies: 8
```

Testing

Now we compile and run.

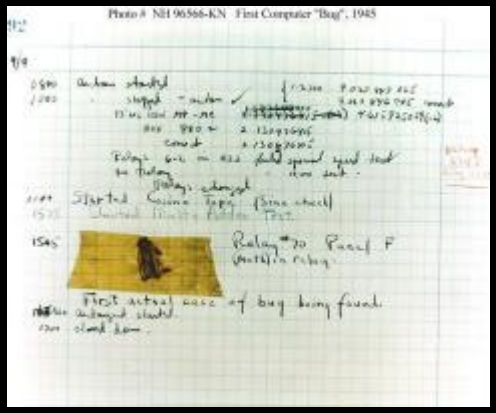
```
ar1121: 26 javac MakeChange.java
ar1121: 27 java MakeChange
Please enter a number--> 93
Quarters: 3
Dimes: 1
Nickels: 1
Pennies: 8
```

This adds up to 98, not 93. Something is wrong.

Testing and debugging

Our program has a bug. Looking for and fixing problems like this is called debugging.

The fun begins. . .



First computer bug, 1945.
http://www.uah.edu/colleges/liberal/womensstudies/hopper/computing.html

A lot of effort must go to training yourself to track down bugs efficiently.

Testing and debugging

First, try to reproduce the error.

```
ar1121: 48 java MakeChange
Please enter a number--> 93
Quarters: 3
Dimes: 1
Nickels: 1
Pennies: 8
ar1121: 49 java MakeChange
Please enter a number--> 15
Quarters: 0
Dimes: 1
Nickels: 1
Pennies: 5
ar1121: 50 java MakeChange
Please enter a number--> 23
Quarters: 0
Dimes: 2
Nickels: 0
Pennies: 3
```

Debugging

Note

- There are some inputs for which it does not fail.
- It only fails when a nickel is given out.
- When it fails, the problem is giving out five too many pennies.

Is `amount` correct after calculating dimes?

Debugging

Add some debugging output

```
amount -= dimesValue;
System.out.println("Amount before nickels: " + amount);
int nickels = amount / 5;            // The number of nickels
int nickelsValue = nickels * 5;     // The value of the nickels
System.out.println("Amount before pennies: " + amount);
int pennies = amount;       // The number (and value) of pennies
```

Debugging

Recompile and run. . .

```
Please enter a number--> 93
Amount before nickels: 8
Amount before pennies: 8
Quarters: 3
Dimes: 1
Nickels: 1
Pennies: 8
```

Amount doesn't change between computing nickels and pennies. Why?

Debugging

Because we don't change it.

```
amount -= dimesValue;
System.out.println("Amount before nickels: " + amount);
int nickels = amount / 5;            // The number of nickels
int nickelsValue = nickels * 5;     // The value of the nickels
System.out.println("Amount before pennies: " + amount);
int pennies = amount;       // The number (and value) of pennies
```

Debugging

Corrected version.

```
amount -= dimesValue;
System.out.println("Amount before nickels: " + amount);
int nickels = amount / 5;            // The number of nickels
int nickelsValue = nickels * 5;     // The value of the nickels
amount -= nickelsValue;
System.out.println("Amount before pennies: " + amount);
int pennies = amount;       // The number (and value) of pennies
```

Debugging

Now removing debugging output, recompiling, and running. . .

```
Please enter a number--> 93
Quarters: 3
Dimes: 1
Nickels: 1
Pennies: 3
ar1121: 66 java MakeChange
Please enter a number--> 15
Quarters: 0
Dimes: 1
Nickels: 1
Pennies: 0
ar1121: 67 java MakeChange
Please enter a number--> 23
Quarters: 0
Dimes: 2
Nickels: 0
Pennies: 3
```

Coding

Next, we can make it more efficient:

```
int quarters = amount / 25;            // The number of quarters
amount %= 25;
int dimes = amount / 10;             // The number of dimes
amount %= 10;
int nickels = amount / 5;             // The number of nickels
amount %= 5;
int pennies = amount;       // The number (and value) of pennies
```

This reduces

- the code size (10 lines to 7 lines).
- the number of variables (removing 3).
- the number of operations (modulus replaces multiplication and subtraction).

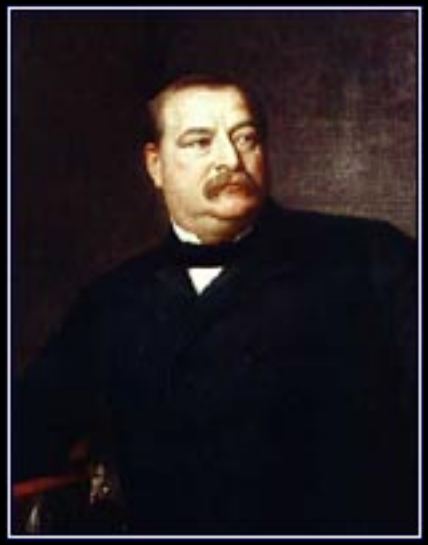
Not finished yet...

We have seen how programs are specified, designed, implemented, tested, and documented. . .

However, in the real world, specifications change, and software must be maintained.

Maintenance

Suppose the US government mints a new coin in honor of Grover Cleveland. It's called the cleve, and it is worth three cents. We need to upgrade our program so that it can handle the new coins.



www.whitehousehistory.org/

05/subs/05_a10.html

Maintenance

Declare a new variable, calculate the number of cleves, and don't forget also to change `amount`.

```
int quarters = amount / 25;            // The number of quarters
amount %= 25;
int dimes = amount / 10;              // The number of dimes
amount %= 10;
int nickels = amount / 5;              // The number of nickels
amount %= 5;
int cleves = amount / 3;              // The number of cleves
amount %= 3;
int pennies = amount;        // The number (and value) of pennies
```

Maintenance

Update the output.

```
System.out.println("Quarters: " + quarters);
System.out.println("Dimes: " + dimes);
System.out.println("Nickels: " + nickels);
System.out.println("Cleves: " + cleves);
System.out.println("Pennies: " + pennies);
```

Maintenance

Indicate the change in the documentation.

```
/**
 * MakeChange.java
 *
 * This program simulates a change-making machine. It computes
 * an arrangement of coins in US currency which use the minimum
 * number of coins whose value is a given amount.
 *
 * @author Thomas VanDrunen
 * Wheaton College, CS 241, Spring 2005
 * Assignment 27
 * Jan 21, 2005
 * Updated for the new cleve coin, Jan 32, 2025, Thomas VanDrunen
 */
```

Maintenance

Then compile and test.

```
Please enter a number--> 93
Quarters: 3
Dimes: 1
Nickels: 1
Cleves: 1
Pennies: 0
```

Summary

Be able to identify the following concepts:

- Algorithm
- Pseudocode
- Specification
- Design
- Implementation
- Testing
- Maintenance
- Documentation