Lecture 23

- Covers
  - Designing methods
  - Procedural abstraction, top-down design
  - Drivers and stubs
  - Testing
  - Formatting decimal output
- Reading: Savitch 5.3

Lecture outline

- Top-down decomposition (or, procedural abstraction)
- Testing strategies
- DecimalFormat class

Top-down Decomposition

- There are two basic techniques for developing complex methods:
  - Divide a complex task into a number of subtasks (divide the subtasks again if necessary)
  - Using methods to program the subtasks
- We may use the term "procedure abstraction" to refer to such an approach.
- A more commonly used term is "top-down decomposition"

Testing strategies

- Category testing: Test values in each category of input
- Boundary value testing: Test boundary values
- Unit testing: Test each method separately
- Integration testing: Test how the methods act together

Stubs

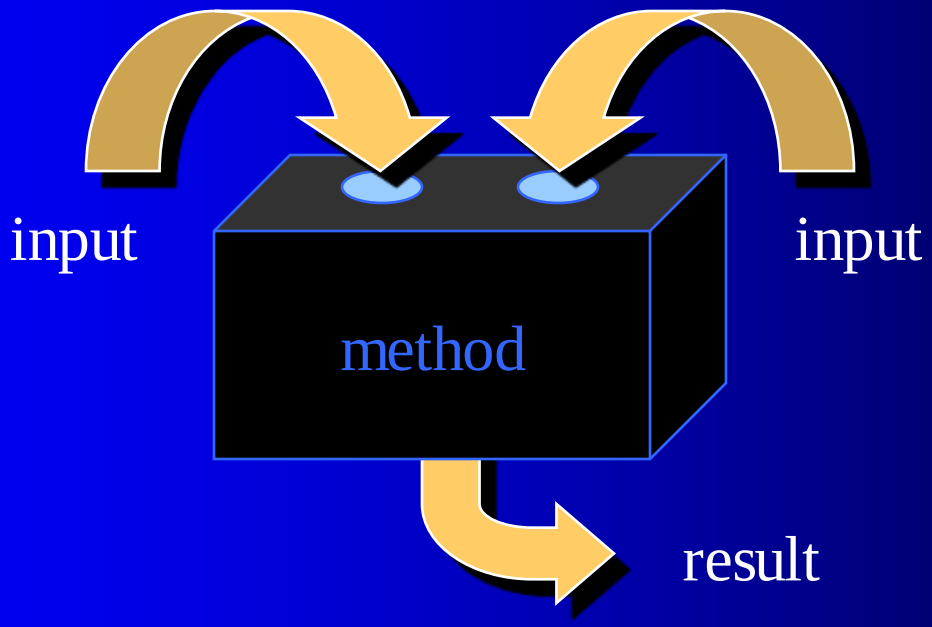
- In early phases of integration testing, we may use stubs for proper methods
- Stubs are methods that are not correctly coded
  - They have proper signatures. So they can be called in integration tests. But they may yield the wrong results as their body is incomplete.

Drivers

- Sometimes we write small programs in order to test particular methods or classes during the development process
- Such programs are referred to as driver programs (or drivers)

What is procedural abstraction?

- Use methods to progressively refine a problem
- Each method can then be treated as a “black box”



What is procedural abstraction?

- All the programmer needs to know to use the method is the method's header and a description of the processing of the method
- They do not need to know any of the details contained in this "black box"

Example

- Write a program that reads a value from the user and outputs the square root of the value entered (without using the sqrt method in the Math class)
- Use Heron's method for approximating the square root

Example

- Main steps

Prompt user to enter a number

Get input

Calculate Square root

Display result

Example

- Refinement of calculate square root method

METHOD calculateSquareRoot(in:num, out: sqrt)
Guess at sqrt
DO
Average guess and num/guess as better guess
Calculate square of guess
WHILE (square of guess is not close enough to num)
Return guess
ENDMETHOD

Example

```
public class SqrtProgram
{
    public static double calculateSquareRoot(double num)
    {
        double guess = Math.random( ) * num;
        double squareOfGuess;
        do
        {
           guess = (guess + num/guess) / 2;
           squareOfGuess = guess * guess;
        }
        while (Math.abs(squareOfGuess - num) > 0.0001);

        return guess;
    }
```

Example

```
    public static void main(String[ ] args)
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.print("Enter a number whose "
                         "square root you require: ");
        double num = keyboard.nextDouble( );
        double sqrtOfNum = calculateSquareRoot(num);
        System.out.print("The square root of the number is "
                          + sqrtOfNum);
    }
}
```

Testing strategies

- Test each category of input
- Test boundary values
- Test each method in the program separately

Testing categories of input

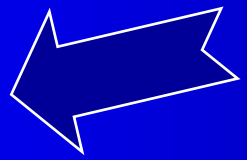
- Decide what kinds of values are to be dealt with and test each category

```
public static void main(String[ ] args)
{
    int value;
    Scanner keyboard = new Scanner(System.in);
    System.out.print("Input an integer value: ");
    value = keyboard.nextInt( );

    if (value > 0)                                  Positive numbers, e.g. 3
        System.out.println(value + " is a positive number");
    else                                            Negative numbers, e.g. -5
        System.out.println(value + " is a negative number");
}
```

Testing boundary values

- An input value is a boundary value if it is a value at which the program changes behaviour

```
public static void main(String[ ] args)
{
    Scanner keyboard = new Scanner(System.in);
    int value;
    System.out.print("Input an integer value: ");
    value = keyboard.nextInt( );

    if (value >= 0)
        System.out.println(value + " is a positive number");
    else
        System.out.println(value + " is a negative number");
}
```



Boundary value 0
(Test -1, 0, 1)

Testing and debugging methods

- Top down design translates one big problem into a series of smaller, more manageable methods
- Each method should be designed, coded and tested as a separate unit from the rest of the program
- How do we test a method outside the program for which it is intended?
  - Write a driver program

Testing and debugging methods

- By testing each method separately, locating the mistakes in the program is much easier
- A fully tested method can be used in a driver program for another method
- Sometimes to test a method, we must use another method that has not been written yet
- Use a simplified version of the missing or untested method called a stub

Case study

- Write a program that tells the user what coins are needed to give any amount of change from 1 cent to 99 cents

- For example if the amount is 85 cents, the output would be

85 Cents can be given as

1 fifty-cent(s), 1 twenty-cent(s), 1 ten-cent(s), and 1 five-cent(s)

Case study

- Main Steps

Get amount

Round to the nearest 5 cents

Calculate number of fifty-cents

Calculate number of twenty-cents

Calculate number of ten-cents

Calculate number of five-cents

Output the results to screen

Case study

- Refinement of Get amount

```
DO
    Prompt user for input
    Get value from user
    IF (1 ≤ input ≤ 99) THEN
        valid is true
    ELSE
        valid is false
        Output error message
    ENDIF
WHILE not valid
```

Case study

```
private static int getAmount( )
{
    int value;
    boolean valid;
    do
    {
        System.out.print("Enter change amount: ");
        value = keyboard.nextInt( );
        if ((value >= 1) && (value <= 99))
        {
            valid = true;
        }
        else
        {
            valid = false;
            System.out.println("Number must be between 1 and 99");
        }
    } while(valid == false);
    return value;
}
```

Case study

```
// Driver for the getAmount method
public static void main(String[ ] args)
{
    int changeAmount;
    String againString = “”;
    char again;

    do
    {
        changeAmount = getAmount( );
        System.out.println("The amount entered was: " + changeAmount);

        System.out.print("Again? ");
        againString = keyboard.nextLine( );
        again = againString.charAt(0);
    } while ((again == 'y') || (again == 'Y'));
}
```

Case study

Enter change amount: 44
The amount entered was: 44

Category test

Again? y
Enter change amount: 1
The amount entered was: 1

Lower boundary test

Again? y
Enter change amount: 0
Number must be between 1 and 99
Enter change amount: 2
The amount entered was: 2

Again? y
Enter change amount: 99
The amount entered was: 99

Upper boundary test

Again? y
Enter change amount: 100
Number must be between 1 and 99
Enter change amount: 98
The amount entered was: 98

Again? n

Case study

- Refinement of Round to nearest 5 cents

```
remainder = changeAmount % 5;

IF (remainder equals 1 or 2) THEN
    Subtract remainder from changeAmount
ELSE
    IF (remainder equals 3 or 4) THEN
        Add (5 - remainder) to changeAmount
    ENDIF
ENDIF
```

Case study

```
private static int round(int changeAmount)
{
    int remainder = changeAmount % 5;

    if ((remainder == 1) || (remainder == 2))
    {
        changeAmount = changeAmount - remainder;
    }
    else
    {
        if ((remainder == 3) || (remainder == 4))
        {
            changeAmount = changeAmount + (5 - remainder);
        }
    }
    return changeAmount;
}
```

Case study

```
// driver for the round method
public static void main(String[ ] args)
{
     int changeAmount;
     String againString = “”;
     char again;

     do
     {
          changeAmount = getAmount( );
          changeAmount = round(changeAmount);
          System.out.println("The rounded amount is: " + changeAmount);

          System.out.print("Again? ");
          againString = keyboard.nextLine( );
          again = againString.charAt(0);
     } while ((again == 'y') || (again == 'Y'));
}
```

Case study

```
Enter change amount: 5
The rounded amount is: 5

Again? y
Enter change amount: 6
The rounded amount is: 5

Again? y
Enter change amount: 7
The rounded amount is: 5

Again? y
Enter change amount: 8
The rounded amount is: 10

Again? y
Enter change amount: 9
The rounded amount is: 10

Again? y
Enter change amount: 10
The rounded amount is: 10

Again?
```

Case study

```
private static void display(int changeAmount, int fifties, int twenties,
                            int tens, int fives)
{
    System.out.println(changeAmount + " can be given as\n"
                       + fifties + " fifty-cent(s), "
                       + twenties + " twenty-cent(s), "
                       + tens + " ten-cent(s), and "
                       + fives + " five-cent(s)");
}
```

If we decide to write and test display before we have done the coin calculations, we need to create method stubs to use in the driver program

Case study

```
// Driver for the display method

public static void main(String[ ] args)
{
      int changeAmount = 0;
      int amountLeft = changeAmount;
      int fifties, twenties, tens, fives;

      fifties = computeCoin(50, amountLeft);
      twenties = computeCoin(20, amountLeft);
      tens = computeCoin(10, amountLeft);
      fives = computeCoin(5, amountLeft);

      display(changeAmount, fifties, twenties, tens, fives);
}
```

Case study

```
// This is a stub
// It is not correct but good enough for some tests

public static int computeCoin(int coinValue, int amountLeft)
{
    return 9;
}
```

0 can be given as
9 fifty-cent(s), 9 twenty-cent(s), 9 ten-cent(s), and 9 five-cent(s)

Case study

```
// filled in version of computeCoin

public static int computeCoin(int coinValue, int amountLeft)
{
      int numberOfCoins = amountLeft / coinValue;
      return numberOfCoins;
}
```

Case study

```
// final main method

public static void main(String[ ] args)
{
      int changeAmount = getAmount( );
      int amountLeft = round(changeAmount);
      int fifties, twenties, tens, fives;

      fifties = computeCoin(50, amountLeft);
      amountLeft -= fifties * 50;
      twenties = computeCoin(20, amountLeft);
      amountLeft -= twenties * 20;
      tens = computeCoin(10, amountLeft);
      amountLeft -= tens * 10;
      fives = computeCoin(5, amountLeft);

      display(changeAmount, fifties, twenties, tens, fives);
}
```

Case study

Enter change amount: 86
86 can be given as
1 fifty-cent(s), 1 twenty-cent(s), 1 ten-cent(s), and 1 five-cent(s)

Enter change amount: 48
48 can be given as
1 fifty-cent(s), 0 twenty-cent(s), 0 ten-cent(s), and 0 five-cent(s)

Enter change amount: 55
55 can be given as
1 fifty-cent(s), 0 twenty-cent(s), 0 ten-cent(s), and 1 five-cent(s)

- The DecimalFormat class

The DecimalFormat class

- The double output in the sqrt program may display with more decimal places than we want
- We can use the DecimalFormal class (previously seen) to format the output
- Of package java.text
- A DecimalFormat object can take a number and return a string representing that number in a specified format

The DecimalFormat class

- Selected methods

  public DecimalFormat( )

  public DecimalFormat(String pattern)

  Examples of patterns

  "###.##", "#,###.##", "AUS$#,###.##"

  public void setMaximumIntegerDigits(int num)

  public void setMinimumIntegerDigits(int num)

  public void setMaximumFractionDigits(int num)

  public void setMinimumFractionDigits(int num)

  public String format(double num)

Suggested usage

- Think of a format in terms of the desired
  - Maximum and minimum decimal digits
  - Maximum and minimum fraction digits
  - Grouping of digits (e.g. separate groups of 3 digits by commas)
  - Preceding and trailing strings (e.g. AUS$)
- Control the last two features by specifying the pattern
- Control the rest by set methods listed in the previous slide

Next lecture

- Overloading methods