Recap: Fence Post Problem

Examples:

- Print n numbers but read only n - 1 commas.
  ```
  5, 4, 3, 2, 1
  ```
- Ask n numbers from user but print only n - 1 error messages.

Problem: If we use a loop algorithm repeatedly placing post + wire, the last post will have an extra dangling wire.

```java
loop (length of fence-wire pair) {
    place a post.          // e.g., <number>
    place some wire.    // e.g., ,
}
```

Problem: If we use a loop algorithm repeatedly placing post + wire, the last post will have an extra dangling wire.

```java
Recap: Fencepost Loop Solutions

Example: `getInt()`
```java
public static int getInt(Scanner console, String prompt) {
    System.out.print(prompt);            // post
    while (!console.hasNextInt()) {     // check last post
        System.out.println("That is not an integer. " + "Please try again.");
        System.out.print(prompt);         // post
    }
    return console.nextInt();          // post
}
```

Example: `getGrade()`
```java
public static int getGrade(Scanner console, String prompt) {
    int grade = getInt(console, prompt);          // post
    while (!((grade >= -1) || 59 <= grade && grade <= 100)) {
        System.out.println("Range is invalid."); // wire
        grade = getInt(console, prompt);          // post
    }
    return grade;
```
Example: `getGrade()`

```java
public static int getGrade(Scanner console, String prompt) {
    int grade;
    while (true) {
        grade = getInt(console, prompt); // post
        if (grade == -1 || grade >= 59 && grade <= 100) {
            break;
        } System.out.println("Input grade not valid."); // wire
    return grade;
}
```

A More User-Friendly Version

```java
int grade;
boolean foundProblem; // loop flag
do {
    grade = getInt(console, prompt);
    foundProblem = false; // assume innocent
    if (grade != -1 && grade < 59) {
        System.out.println("Grade cannot < 59 unless -1.");
        foundProblem = true;
    } else if (grade > 100) {
        System.out.println("Grade cannot be higher than 100.");
        foundProblem = true;
    }
} while (foundProblem);
```

Programming Pattern

```java
boolean foundProblem; // loop flag
do {
    get user input;
    foundProblem = false; // assume innocent
    validate user input using a set of rules
    if (violating rule x) // guilty
        output that rule x is violated
        foundProblem = true;
    // check rule y ...
} while (foundProblem);
```

Comparison

```java
int grade;
boolean foundProblem; // loop flag
do {
    grade = getInt(console, prompt);
    foundProblem = false; // assume innocent
    if (grade != -1 && grade < 59) {
        System.out.println("Grade cannot < 59 unless -1.");
        foundProblem = true;
    } else if (grade > 100) {
        System.out.println("Grade cannot be higher than 100.");
        foundProblem = true;
    }
} while (foundProblem);
```

Comment: Which One to Use?

- It is often a personal style
- If the validation/evaluation of user input is a simple boolean expression and the error message is simple, the first one is good
- If the validation/input processing is complex (e.g., multiple steps), the second version often gives a good, extensible control structure.

Practice: A MathAdding Game

- Write a program that plays an adding game.
  - Ask user to solve random adding problems with n (2-5) numbers.
  - The user gets n-1 points for an addition with n numbers, 0 for incorrect.
  - A user has 3 lives (3 chances) to make mistakes.
- 4 + 10 + 3 + 10 = 27
- 8 + 2 = 10
- 9 + 5 = 14
- Wrong! The answer was 16
- 4 + 5 = 9
- 9 + 9 = 18
- Wrong! The answer was 13
- You earned 9 total points.
Design Questions

- Is the loop better controlled by for or while/do/while?
  ❍ Indeterminate logical condition (cumulate 3 errors) -> more likely a while or do/while loop

- How many times may the loop execute, at least once?
  ❍ at least once => we may try a do/while loop

MathAddingGame Structure

```java
// Asks the user to do adding problems and scores them.
import java.util.*;
public class MathAddingGame {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        // init. state
        int totalPoints = 0;
        int nChances = 3;
        do {
            // play a round:
            int roundPoints = playOneRound(console);
            totalPoints += roundPoints;
            if (roundPoints == 0) {
                nChances--;
            }
        } while (nChances > 0);
        System.out.println("You earned "+totalPoints+
                        " total points.");
    }
}
```

The user gets n-1 points for an addition with n numbers, 0 for incorrect.

MathAdding.java

```
mathAdding.java
```

MathAddingGame Program

```java
// Asks the user to do adding problems and scores them.
import java.util.*;
public class MathAddingGame {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        // play until user gets 3 wrong
        int totalPoints = 0;
        int nChances = 3;
        do {
            int roundPoints = playOneRound(console);
            totalPoints += roundPoints;
            if (roundPoints == 0) {
                nChances--;
            }
        } while (nChances > 0);
        System.out.println("You earned "+totalPoints+
                        " total points.");
    }
}
```

```
MathAdding.java
```

MathAddingGame Requirement

```java
// play a round:
// return the # of points earned;
// 0 means user made a mistake
static int playOneRound()
```

```
... // Builds one addition problem and presents it to the user.
// Returns # of points earned.
public static int playOneRound(Scanner console) {
    // print the operands being added, and sum them
    int operands = randInt(2, 5);
    int sum = 0;
    int n = randInt(1, 10);
    String question = "" + n; // post
    for (int i = 2; i <= operands; i++) {
        n = randInt(1, 10);
        question += " + " + n; // wire + post
        sum += n;
    }
    question += " = ";
    // read user's guess
    int guess = getInt(console, question);
    if (guess == sum) {
        return operands - 1;
    } else {
        return 0;
    }
}
```

```
MathAddingGame Requirement
```

```
playOneRound Structure

1 + 2 + 3 + 4 + 5 = 15
4 + 10 + 3 + 10 = 27
9 + 2 = 11

structure:
    pick n = # of operands to be added (2-5)
    build problem string
    output problem to user
    compute points to return
```

```
playOneRound Structure
```

```
MathAddingGame Requirement

```
```
MathAddingGame Program

// A class that asks the user to do adding problems and scores them.
import java.util.*;

public class MathAddingGame {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        // play until user gets 3 wrong
        int totalPoints = 0;
        int nChances = 3;
        while (nChances > 0) {
            int roundPoints = playOneRound(console);
            totalPoints += roundPoints;
            if (roundPoints == 0) {
                nChances--;
            }
        }
        System.out.println("You earned " + totalPoints + " total points.");
    }
}

Move the while condition up. Do we have the same result?

Summary: Program Flow Control

Program flow control is among the most challenging perspectives in terms of mastering computer programming:

- it may involve complex logical analysis
- it has substantial flexibility
  - multiple types of conditional statements (if/else, switch)
  - multiple types of loop statements (for, while, do/while)
  - break

Practice: Random Question

Write a program that simulates rolling of two 6-sided dice until their sum comes up as a "natural" (i.e., 7 or 11) in the casino game of craps.

2 + 4 = 6
3 + 5 = 8
5 + 5 = 10
1 + 1 = 2
4 + 3 = 7
You need 5 tries!

Design Questions

Is the loop better controlled by for or while/do/while?

- Indeterminate logical condition (sum == 7 or 11) -> more likely a while or do/while loop

How many times may the loop execute, at least once?

- at least once => we may try a do/while loop

NaturalCraps.java

Answer

// Rolls two dice until a sum of 7 or 11
import java.util.*;

public class NaturalCraps {
    public static void main(String[] args) {
        int tries = 0;
        int sum = 0; // force initial while to be true
        do {
            int roll1 = randInt(1,6); // one roll
            int roll2 = randInt(1,6);
            sum = roll1 + roll2;
            System.out.println(roll1 + " + " + roll2 + " = " + sum);
            tries++;
        } while (sum != 7 && sum != 11);
        System.out.println("You won after " + tries + " tries!");
    }
}

Answer 2: do->while

// Rolls two dice until a sum of 7
import java.util.*;

public class NaturalCraps {
    public static void main(String[] args) {
        int tries = 0;
        int sum = 0; // force initial while to be true
        do {
            int roll1 = randInt(1,6); // one roll
            int roll2 = randInt(1,6);
            sum = roll1 + roll2;
            System.out.println(roll1 + " + " + roll2 + " = " + sum);
            tries++;
        } while (sum != 7 && sum != 11);
        System.out.println("You won after " + tries + " tries!");
    }
}
Program Analysis

- A useful tool to understand flow control better is program analysis

Requirements

Program

Analyze program to check if program satisfies requirements

Foundation of Program Analysis: Logical Assertions

- Assertion: A statement that we focus on whether it is true, false, or sometimes true/sometimes false (unknown).

Examples:
- Yale was founded in 1776.
- The sky is purple.
- The capital of Connecticut is New Haven.
- Prof. Yang met a donkey this morning.
- int x;
  - x = x+10;
  - x divided by 2 equals 7. (depends on the value of x)

Logical Assertions on Program Variables

- One can make assertions on program variables at each point of a program
  - For example, right after a variable is initialized, its value is known, and we can make true/false statement:
    ```
    int x = 3;
    // is x > 0? True
    ```

- A common confusion: An assertion is not part of your program; it is a claim/property of your program

Difficulty of Making Assertions

- The value of a variable may become unknown after certain operations (hence leading to “unknown/sometimes” assertions)
  - reading from a Scanner
  - assigned a number from a random number
  - a parameter’s initial value to a method
    ```
    public static void mystery(int a, int b) {
    // is a == 10? UNKNOWN
    ```

Control Structure Establishes Assertions

- At certain execution point of a control structure (e.g., if, while, break), we may know something:

  ```
  public static void mystery(int a, int b) {
    if (a < 0) {
      // is a == 10? FALSE
    }
    ...
  }
  ```

  We know a < 0, which implies a != any positive number.

Assertions and Controls

- At the start of an if or loop’s body, the test must be true:
  ```
  while (y < 10) {
    // is y < 10? TRUE
    ...
  }
  ```

- In the else or after a loop w/o break, the test must be false:
  ```
  while (y < 10) {
    ...
    // is y < 10? FALSE
  }
  ```

- Note: Inside a loop’s body, the loop’s test may become false:
  ```
  while (y < 10) {
    y++;  // is y < 10? UNKNOWN
    // if y < 12  TRUE
  }
Program Analysis

Derive assertions to check if program satisfies requirements

More Rigorous Requirements Specification

Requirement on returning number:
- return number => it is non-negative
- reads in a non-negative number => return it

Requirement on error message:
- if print error message => user input negative
- if user input negative => print error message

Program Analysis: Returning Number

Precise requirement on returning number:
- reads in a non-negative number => return it
- return number => it is non-negative

Using Assertions to Understand Program (II)

Precise requirement on error message:
- if user input negative => print error message
- if print error message => user input negative

Requirements: Example

Program:
```java
public static double getNonNegativeNumber(Scanner console) {
    System.out.print("Type a nonnegative number: ");
    double number = console.nextDouble();
    while (number < 0.0) {
        System.out.print("Error: Negative; try again: ");
        number = console.nextDouble();
    }
    return number;
}
```

Informal requirements: read a non-negative number from user; if the user input is negative, output an error message and try again.
Using Assertions to Understand Program (II)

```java
public static double getNonNegativeNumber(Scanner console) {
    System.out.print("Type a nonnegative number: ");
    double number = console.nextDouble();
    while (number < 0.0) {
        System.out.print("Error: Negative; try again: ");
        number = console.nextDouble();
    }
    return number;
}
```

Precise requirement on error message:

- if user input negative => print error message
- if print error message => user input negative

Summary: Program Analysis

```java
public static double getNonNegativeDouble(Scanner console) {
    System.out.print("Type a nonnegative number: ");
    double number = console.nextDouble();
    while (number < 0.0) {
        // ASSERTION: number < 0
        System.out.print("Error: Negative; try again: ");
        number = console.nextDouble();
    }
    // ASSERTION: number >= 0.0
    return number;
}
```

Requirements

- print an error message only if user number is negative
- return number only if non-negative
- return the first non-negative number
- print error message for every negative user input number

Practice: Sentinel Sum

```java
int sum = 0;
System.out.print("Enter a number (-1 to quit): ");
int grade = console.nextInt();
while (true) {
    sum = sum + grade;
    System.out.print("Enter a number (-1 to quit): ");
    grade = console.nextInt();
}
```

Analyze this program.

Practice: Analyze if They Give the Same Result

```java
int sum = 0;
int grade = 0;
while (true) {
    System.out.print("Enter a number (-1 to quit): ");
    grade = console.nextInt();
    if (grade == -1) {
        // detect the last post
        break;
    }
    sum = sum + grade;
}
```

```java
int sum = 0;
int grade = 0;
do {
    System.out.print("Enter a number (-1 to quit): ");
    grade = console.nextInt();
    if (grade == -1) {
        // detect the last post
        break;
    }
    sum = sum + grade;
} while (true);
```

Practice: Analyze if They Give the Same Result

```java
// set initial value of month so that the while
// condition below is true initially to force a loop
int month = -1;
do {
    System.out.print("Enter a month (1 to 12): ");
    month = getInt(console, "Enter a month (1 to 12): ");
} while (month < 1 || month > 12);
```

Example: File Processing Loop

- Requirement:
  1. You are given a file containing a list of temperature values on each day of a week
  2. Write a program that prints out temperature fluctuation (the difference of temperature between two consecutive days)
Design Questions

- Is the loop better controlled by for or while/do/while?
  - A deterministic loop -> for loop

- Loop pattern?
  38.1 28.9 31.3 30.4 28.9 38.3 36.3
  -9.2 2.4 -0.9 -1.5 9.4 -2.0

File Input Answer

// Displays changes in temperature from data in an input file.
import java.io.*;  // for File
import java.util.*; // for Scanner
public class NHWeatherWeek {
    public static void main(String[] args)
        throws FileNotFoundException {
            Scanner input = new Scanner(new File(DATA_FILE));
            double prev = input.nextDouble(); // post
            for (int i = 0; i < 6; i++) {
                double next = input.nextDouble(); // next post
                double change = next - prev;       // wire
                System.out.printf("%.1f -> %.1f: %5.1f
", prev, next, change);
                prev = next;
            }
        }
    }

Reading an Entire File

- Suppose we want our program to work no matter how many numbers are in the file.
  - Currently, if the file has more numbers, they will not be read.
  - If the file has <= 7 numbers, what will happen?

- Modify the temperature program to process the entire file, regardless of how many numbers it contains.
  - Is the loop better controlled by for or while/do/while?
Reading an Entire File

// Displays changes in temperature from data in an input file.
import java.io.*; // for file
import java.util.*; // for Scanner
public class NHAll {
  public static void main(String[] args) throws FileNotFoundException {
    Scanner input = new Scanner(new File(DATA_FILE));
    if (input.hasNextDouble()) { // post
      double prev = input.nextDouble(); // post
      while (input.hasNextDouble()) { // post
        double next = input.nextDouble();
        double change = next - prev;
        System.out.printf("%.1f -> %.1f: %5.1f
", prev, next, change);
        prev = next;
      } // end of while
    } // end of if
  }
}

Is the program safe (i.e., not crash by reading a non double)?

Analysis: Original Program

// Displays changes in temperature from data in an input file.
import java.io.*; // for File
import java.util.*; // for Scanner
public class NHAll {
  public static void main(String[] args) throws FileNotFoundException {
    Scanner input = new Scanner(new File("nh-2013-01-14-annote.txt"));
    if (input.hasNextDouble()) { // post
      double prev = input.nextDouble(); // post
      while (input.hasNextDouble()) { // post
        double next = input.nextDouble();
        double change = next - prev;
        System.out.printf("%.1f -> %.1f: %5.1f
", prev, next, change);
        prev = next;
      } // end of while
    } // end of if
  }
}

Problem of Original Solution

It cannot make progress: the first non-number will cause input.hasNextDouble() to be false, leading the program to skip the rest of data points

Objective: making progress

Attempt 1

// Displays changes in temperature from data in an input file.
import java.io.*; // for file
import java.util.*; // for Scanner
public class NHAll {
  public static void main(String[] args) throws FileNotFoundException {
    Scanner input = new Scanner(new File("nh-2013-01-14-annote.txt"));
    if (input.hasNextDouble()) { // post
      double prev = input.nextDouble(); // post
      while (input.hasNextDouble()) { // post
        double next = input.nextDouble();
        double change = next - prev;
        System.out.printf("%.1f -> %.1f: %5.1f
", prev, next, change);
        prev = next;
      } // end of while
    } // end of if
  }
}

Removes defense to nextDouble() and can lead to crash
The condition before `nextDouble()` should be

- **safe**: true only if `hasNextDouble()` is true
- **making progress**: can skip non-Double tokens

Define `hasNextDoubleData()`:

```java
public static boolean hasNextDoubleData(Scanner input) {
    while ( !input.hasNextDouble() ) {
        input.next(); // making progress
    }
    return input.hasNextDouble(); // safety
}
```

Is there defense for `next()`?

Solution (another version)

```java
public static boolean hasNextDoubleData(Scanner input) {
    while (input.hasNext() && !input.hasNextDouble()) {
        input.next(); // making progress
    }
    return input.hasNextDouble(); // safety
}
```

```java
public static boolean hasNextDoubleData(Scanner input) {
    while (input.hasNext()) {
        if (input.hasNextDouble())
            return true;
        else
            input.next();
    }
    return false;
}
```