Exercise: MatchDB

- Design a program to query a match-making database
  - database (file) format:
    - line 1: number of candidates
    - each line represents a candidate: name age and then a sequence of words (tags) each describing a character
  - user commands
    - list: display each candidate
    - count <a list of tag words> // count # matching all tags
    - match1 <a list of tag words> // print first record that matches all tags
    - match <start> <end> <a list of tag words> // print matched record <start> to record <end>

match1

```java
public static void match1(String[] mTags)
{
    for (int i = 0; i < N; i++ ) {
        if ( checkMatch(mTags, i) ) {
            System.out.println("Find " + name[i]);
        }
    } // end of for
}
```

Problem: Loop does not stop after matching first.

Solution I: Using break

- break statement: Immediately exits a loop.
  - Can be used to write a loop whose test is in the middle.
  - Typical in a search problem
    - Find what you are looking for, i.e., an “evidence”
  - Exercise: fix match1 using break

Removing break

- Using break to jump out of a loop is considered a bad style by some programmers

```java
public static void match1(String[] mTags)
{
    for (int i = 0; i < N; i++ ) {
        if ( checkMatch(mTags, i) ) {
            System.out.println("Find " + name[i]);
            break;
        }
    } // end of for
}
```

Objective: design a single exit point.
Analysis

Question: logical conditions that the for loop should exit?

- \( i \geq N \) || matchTags(mTags, i)
- Condition to continue the loop (negate of the above):
  \( ! ( i \geq N \) || matchTags(mTags, i) \)

De Morgan’s Law

- Rules used to negate boolean tests.
  - Useful when you want the opposite of an existing test.

<table>
<thead>
<tr>
<th>Original Expression</th>
<th>Negate</th>
<th>Equivalent Negate</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a \land b )</td>
<td>( \neg(a \land b) )</td>
<td>( \neg a \lor \neg b )</td>
</tr>
<tr>
<td>( a \lor b )</td>
<td>( \neg(a \lor b) )</td>
<td>( \neg a \land \neg b )</td>
</tr>
</tbody>
</table>

Revised Version w/o break: II

```
public static void match1(String[] mTags){
    for (int i = 0; i < N; i++ ) {
        if ( checkMatch(mTags, i) ) {
            System.out.println("Find " + name[i]);
            break;
        }
    } // end of for
}
```

Revised Version w/o break: II

```
public static void match1(String[] mTags){
    for (int i = 0; i < N && !checkMatch(mTags, i); i++ ) {
        System.out.println("Find " + name[i]);
    } // end of for
}
```

Revised Version w/o break: II

```
public static void match1(String[] mTags){
    int i = 0;
    for (i = 0; i < N && !checkMatch(mTags, i); i++ ) {
        System.out.println("Find " + name[i]);
    } // end of for
    if (i < N)
        System.out.println("Find " + name[i]);
}
```

Revised Version w/o break: II

```
public static void match1(String[] mTags){
    int i;
    for (i = 0; i < N && !checkMatch(mTags, i); i++ ) {
        System.out.println("Find " + name[i]);
    } // end of for
    if (i < N)
        System.out.println("Find " + name[i]);
}
```

Revised Version w/o break: II

```
public static void match1(String[] mTags){
    int i = 0;
    while (i < N && !checkMatch(mTags, i) ) {
        i++;
    } // end of for
    if (i < N)
        System.out.println("Find " + name[i]);
}
```

Revised Version w/o break: II

```
public static void match1(String[] mTags){
    int i = 0;
    while (i < N && !checkMatch(mTags, i) ) {
        i++;
    } // end of for
    if (i < N)
        System.out.println("Find " + name[i]);
}
```
Practice

- Implement the match command:
  
  match <start> <end> <tags>

Roadmap: Program Flow Control

- Deterministic for loop
  (loop counter)

- cumulative scan loop

- Early exit loops

- Fencepost loops

Example Problem: User Input Protocol

- Two input styles
  
  - Header control protocol
    - User first specifies the number of data items
  
  - In-band control protocol
    - User finishes input by entering a sentinel value
      - e.g., 1 to signal the end of input grades, "quit" to finish the program
    - Why in-band sentinel: flexibility.
     - Implication: a data item just read can be either a real data item or the signaling sentinel

Solution

Scanner console = new Scanner(System.in);
int sum = 0;
String response;

do {
    System.out.print("Type a line (or nothing to exit): ");
    response = console.nextLine();
    sum += response.length();
} while (!response.equals(""));

System.out.println("You typed a total of 
  + sum 
  + " characters.");

Sentinel Values

- sentinel: A value that signals the end of user input.
  - sentinel loop: Repeats until a sentinel value is seen.

- Example: Write a program that prompts the user for text until the user types nothing, then outputs the total number of characters typed.
  - In this case, the empty string is the sentinel value.

  Type a line (or nothing to exit): hello
  Type a line (or nothing to exit): this is a line
  You typed a total of 19 characters.

Design question: for, while, or do loop?

Changing the Sentinel Value

- Changing the sentinel’s value to "quit"

  Scanner console = new Scanner(System.in);
  int sum = 0;
  String response;

  do {
      System.out.print("Type a line (or "quit" to exit): ");
      response = console.nextLine();
      sum += response.length();
    } while ( !response.equals("Quit"));

  System.out.println("You typed a total of 
    + sum + " characters.");

  Type a line (or "quit" to exit): hello
  Type a line (or "quit" to exit): this is a line
  You typed a total of 23 characters.

- This solution produces the wrong output. Why?
A “Simpler” Problem...

- Revisit the `countDown` method that prints from a given maximum (>=1) to 1, separated by commas.

For example, the call:

```
countDown(5)
```

should print:

```
5, 4, 3, 2, 1
```

Problem: Fence Post Analogy

- We print `n` numbers but need only `n-1` commas.

```
5, 4, 3, 2, 1
```

- If we use a loop algorithm that repeatedly places a post + wire, the last post will have an extra dangling wire.

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

Fencepost Loop Solution

- There are multiple solutions
- A typically one is to add a statement outside the loop to place the initial "post." Also called a "loop-and-a-half" solution.

```
place a post.
loop (length of fence - 1) {
    place a post.
    place some wire.  // e.g., ,
}
```

Fencepost Method Solution

```
public static void countDown(int max) {
    System.out.print(max);    // first post
    for (int i = max; i >= 2; i--) {
        System.out.print(i + " , ");  // wire + post
    }
    System.out.println();    // to end the line
}
```

```java
5, 4, 3, 2, 1
```

Previous “Solution”

```
public static void countDown(int max) {
    System.out.print(max + " , ");  // to end the line of output
}
```

```java
Output from countDown(5): 5, 4, 3, 2, 1
```

Problem: Fence Post Analogy

- The sentinel read is also a fencepost problem: Must read `N` lines, but sum only the lengths of the first `N-1`.

```
Scanner console = new Scanner(System.in);
int sum = 0;
String response;
do {
    System.out.print("Type a line (or "quit" to exit): ");
    response = console.nextLine();
    sum += response.length();
} while ( !response.equals("quit");
System.out.println("You typed a total of "+ sum + " characters.");
```

```
Type a line (or "quit" to exit):
hello
Type a line (or "quit" to exit):
this is a line
Type a line (or "quit" to exit):
quit
You typed a total of 23 characters.
```

Fencepost Method Solution

```
public static void countDown(int max) {
    for (int i = max; i >= 2; i--) {
        System.out.print(i + " , ");  // post + wire
    }
    System.out.println(1);    // last post
}
```
Fencepost Sentinel Loop

```java
public static final String SENTINEL = "quit";
public static final String PROMPT = 
"Type a line (or " + SENTINEL + ") to exit):
int sum = 0;
Scanner console = new Scanner(System.in);
// pull one prompt/read ("post") out of the loop
String response =(getString(PROMPT, console);
while (!response.equals(SENTINEL)) {
    sum += response.length();
    // wire
    response = getString(PROMPT, console);  // post
}
System.out.println("You typed a total of 
" + sum + " characters before quit.");
}
public static String getString(String prompt, Scanner console)
{
    System.out.print(prompt);
    return console.nextLine();
}
```

Fencepost Sentinel Loop: Grade

```java
public static final int SENTINEL = -1;
public static final String PROMPT = "Type a grade (" + SENTINEL + ") to exit"):
Scanner console = new Scanner(System.in);
int sum = 0; int count = 0;
// pull one prompt/read ("post") out of the loop
int grade = getInt(PROMPT, console);
while (grade != SENTINEL) {
    sum += grade; count++;
    // wire
    grade = getInt(PROMPT, console); // post
} if (count > 0)
{
    System.out.println("Avg: 
" + 1.0 * sum / count);
}
public static String getInt(String prompt, Scanner console)
{
    System.out.print(prompt);
    return console.nextInt();
}
```

Summary: Fencepost Loops

```
read a number
update sum
```

Alternative Design: Motivation

```
place a post
loop (length of fence - 1) {
    place a wire.
    place a post.
}
```

Alternative Design

```
Scanner console = new Scanner(System.in);
int sum = 0; int grade = 0;
while (grade != -1) {
    // post
    System.out.print("Enter a number (-1 to quit): ");
    grade = console.nextInt();
    if (grade != -1) {
        // detect the last post
        sum = sum + grade;    // wire
        grade = console.nextInt();
    }
}
System.out.println("The total was "+ sum);
```
Comparison

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

```
Duplicated code
```

```
Duplicated code
```

```
Duplicate execution
```

```
Alternative Design: Analysis
```
```
So that we can exit the loop
```

```
So that we don't have an infinite loop
```

Q: Can we remove the duplication, say the first test?

Sentinel Loop with break

```java
Scanner console = new Scanner(System.in);
int sum = 0;
while (true) {
    System.out.print("Enter a number (-1 to quit): ");
    int number = console.nextInt();
    if (number == -1) {
        break;
    }
    sum = sum + number;
    // number != -1 here
}
System.out.println("The total was "+ sum);
```

Comments

- Choice between "sentinel loop with break" vs "non-break" is typically a personal style
  - The "non-break" version may be preferred, because
    - the high-level structure ("topical sentence") indicates that the loop is controlled by a sentinel
    - the high-level structure ("topical sentence") of the "break" version indicates that the loop is an infinite loop.

Refining GradeAnalyzer

- Problem statement: users are known to give wrong input. A rule of robust programming is to remember Murphy's Law
  - Murphy's Law: "If anything can go wrong, it will."

- What may be wrong when a user gives an input for a grade?
  - Not a valid integer
  - Not in the right range

Step 1: getInt

- Write a method getInt to read an integer. If the user does not give an integer, output an error message that we are reading an integer, and ask again.
**getInt()**

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

**getGrade()**

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

**Step 2: Application Logic Validation**

- Even when the user gives an integer, it may not be a valid grade input, i.e. -1 or between 59 and 100 for "the new grading system" that Yale is considering.
- Write a method `getGrade` to read a valid integer for a grade. If the user does not give a valid integer, output an error message indicating invalid range, and repeat.

**getGrade() using Break**

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