Steve Cooper
CS 106A

Handout #12A
April 12, 2013

More on Repetition

Indefinite interation (e.g. the **`while`** loop) tends to be used when we do not know how many times to iterate through a loop. There are two special cases of indefinite iteration, involving user input. In both cases the user is asked to input data, and the loop needs to be entered at least once. The first case is prompting the user to enter data as long as the user enters data that is not yet valid. The second case is prompting the user to enter data until the user enters some special sentinel value. This handout explores various solutions to these problems.

Case 1: Prompting the user to enter some data until a valid input is entered. This situation is best illustrated by means of an example. Suppose we wish the use to enter in the month of the year. Obviously, we only wish to have the user enter in a value between 1 and 12. Using a simple **`while`** loop, the code would look something like the following:

```
int month;
month = readInt("Enter month: ");
while ((month > 12) || (month < 1)) {
    month = readInt("Enter month: ");
}
```

While this code works, it is somewhat inconvenient in that the code prompting the user to enter the month is required twice: once to determine if it is necessary to enter the **`while`** loop the first time, and again to determine if the **`while`** loop should be re-entered. It would be easier to only include the prompt once. An alternate solution is the following:

```
int month;
do {
    month = readInt("Enter month: ");
} while ((month > 12) || (month < 1));
```

This code, using a **`do while`** loop, only requires a single prompt. The reason is that the **`do while`** loop is always entered at least once, with the test occuring at the end of the loop rather than at the beginning.

Both code snippets accomplish the same result, and both are considered to be "correct".

Case 2: Prompting the user to enter some data until a a sentinel value is entered. This is the so-called "loop-and-a-half problem described in the text. It is best explained by means of an example. The example presented is Soloway's famous "rainfall problem", where a sequence of monthly rainfall amounts is entered followed by a -999 to indicate end of input. The program must compute the average monthly rainfall, paying particular attention to not process the sentinel value. Using a **`while`** loop, the task can be solved as follows:

```
final int SENTINEL = -999;
int months = 0;
int rainfall;
int total = 0;
rainfall = readInt("Enter monthly rainfall, or " +
                   SENTINEL + ": ");
while (rainfall != SENTINEL) {
    months ++;
    total += rainfall;
    rainfall = readInt("Enter monthly rainfall, or " +
                       SENTINEL + ": ");

}

if (months > 0) {
   println("The average rainfall is " + (total / months));
}
```

Much as in the first case, it is not ideal to need to prompt the user to enter the monthly rainfall twice (once to get into the loop the first time, and once to determine whether to remain in the loop). Much as the **`do while`** loop was useful in the first case, it also helps in this case as well. The following code accomplishes the same goal:

```
final int SENTINEL = -999;
int months = 0;
int rainfall;
int total = 0;
do {
    rainfall = readInt("Enter monthly rainfall, or " +
                       SENTINEL + ": ");
    if (rainfall != SENTINEL) {
       months ++;
       total += rainfall;
    }
}  while (rainfall != SENTINEL);

if (months > 0) {
   println("The average rainfall is " + (total / months));
}
```

Here, there is no need to prompt the user to enter the monthly rainfall twice. However, after getting the monthly value, it is necessary to check to make sure that it is not the sentinel before adding it to the total rainfall.

There is also a third solution to this problem, using a **`break`** instruction to exit a **`while`** loop. This is the solution presented in the text to address the loop and a half problem. The following code accomplishes this goal:

```
final int SENTINEL = -999;
int months = 0;
int rainfall;
int total = 0;
while (true) {
    rainfall = readInt("Enter monthly rainfall, or " +
                       SENTINEL + ": ");
    if (rainfall == SENTINEL)
          break;

    months ++;
    total += rainfall;
}

if (months > 0) {
   println("The average rainfall is " + (total / months));
}
```

Since all of these solutions work, any of them may be used to solve this problem. However, many students tend to over-rely on the second and/or third solution. In *almost all cases*, it is the regular **`while`** loop that should be used. These latter two solutions are typically only used to solve the loop-and-a-half problem and the input validation problem, as has been illustrated above. In general, the solution involving using the **`break`** instruction is often viewed as "worse". The reason is that this is a dangerous habit to get into – breaking out of **`while`** loops is extremely error-prone, especially when **`while`** loops are nested inside of one another!