Control Statements
Loops

October 8, 2003

Example

```java
System.out.println("Enter a number between 1 and 5");
int num = Integer.parseInt(br.readLine());
if (num == 1)
    System.out.println("Do a headstand.");
else if (num == 2)
    System.out.println("Fat your head and rub your tummy.");
else if (num == 3)
    System.out.println("Do 3 jumping jacks.");
else if (num == 4)
    System.out.println("Recite a limerick.");
else if (num == 5)
    System.out.println("Count backwards from 100 by 7's.");
else
    System.out.println("You cannot follow directions, butthead!");
```

switch statement

- Sometimes using if statements can be cumbersome

switch statement

- Replacement for if statements when the boolean expression involves testing an integer (int) or a character (char)
  - also: byte, short
- Examples of integer tests:
  - num (from previous example)
  - 25 % 3
  - a / 3 (if a is an integer!)
Example

```java
System.out.println("Enter a number between 1 and 5");
int num = Integer.parseInt(br.readLine());
switch (num) {
    case 1:  System.out.println("Do a headstand.");
             break;
    case 2:  System.out.println("Pat your head and rub your tummy.");
             break;
    case 3:  System.out.println("Do 3 jumping jacks.");
             break;
    case 4:  System.out.println("Recite a limerick.");
             break;
    case 5:  System.out.println("Count backwards from 100 by 7's.");
             break;
    default:  System.out.println("You cannot follow directions, butthead!");
}
```

Format of a switch statement

```java
switch {<expression>} {
    case <constant 1>:  <statements>
                          break;
    case <constant 2>:  <statements>
                          break;
    ...
    case <constant n>:  <statements>
                          break;
    default:  <statements>
}
```

Notes

- The `<constant i>` values must be constants
  - integers/chars/bytes or defined constants of these types
- Cases can be in any order
  - typically, default goes last
- break is used to make sure we only execute one of the statement blocks
- default clause is optional
  - handles unexpected input or program behavior
  - good programming practice

Another example

```java
System.out.println("Enter a number between 1 and 7");
int num = Integer.parseInt(br.readLine());
switch (num) {
    case 1:
    case 2:
    case 3:
    case 5:
    case 7:  System.out.println("Prime number.");
             break;
    case 4:  System.out.println("Factors are 1, 2, and 4.");
             break;
    case 6:  System.out.println("Factors are 1, 2, 3, and 6.");
             break;
    default:  System.out.println("Invalid number entered.");
}
```
Example using `char` values

```java
System.out.print("Enter C to continue, R to repeat, or Q to quit:");
char choice = br.read(); // reads in one character from stdin
switch (choice) {
    case 'C': System.out.println("We will continue the game.");
        ... break;
    case 'R': System.out.println("I will repeat the rules.");
        ... break;
    case 'Q': System.out.println("Thanks for playing!");
        break;
    default: System.out.println("Invalid option entered.");
}
```

Control statement gotchas

- Not using `breaks` in your `switch` statements
- Using values that are not integers, chars, or bytes when using `switch`
- Making your `if` statements too complex
  - simplify your logic!
- Using `&&` when you mean to use `||` and vice versa
- Using `=` (assignment) when you mean to use `==` (equality)
- Using `==` (equality) to compare objects
  - Strings: `equals()` or `equalsIgnoreCase()` should be used instead

Repetition statements

- Control statements allow us to execute certain blocks of code when certain conditions are met
- `Repetition statements` allow us to execute certain blocks of code more than once
  - until some condition is met
  - a set number of times

Uses of repetition

- MiniYahtzee: allow the user to keep playing until he/she decides to quit
- Keep prompting the user for input until the user enters valid input
- Mathematical calculations
  - average
  - sum
- And many more!
Repetition statements

- **while**
- **do-while**
- **for**

### while statement

- “While some condition is true, do the following...”
- **pre-test** loop: test the condition first, then execute the loop

#### Example:

```java
int sum = 0, count = 1, maxSum = 500;
while (sum < maxSum) {
    sum = sum + count;
    count = count + 2;
}
System.out.println("Sum = "+sum);
```

#### Another example: valid user input

```java
System.out.print("Enter an interest rate: ");
rate = Double.parseDouble(br.readLine());
while (rate < 0 || rate > 100) {
    System.out.println("You entered an invalid rate. Please enter a rate between 0 and 100:");
    rate = Double.parseDouble(br.readLine());
}
...```

#### Another example: averaging numbers

```java
System.out.print("Enter a positive number (or any negative number to quit): ");
int num = Integer.parseInt(br.readLine());
int sum = 0, count = 0;
while (num >= 0) {
    count++;
    sum = sum + num;
    System.out.print("Enter a positive number (negative number to quit): ");
    num = Integer.parseInt(br.readLine());
}
double average = ((double)sum)/count;
System.out.println("The average of the numbers is "+average);
```
Format of a `while` statement

```plaintext
while (<condition>) {
    <statement>
}
```

- `<condition>` is a boolean statement
  - changes each time through the loop
- As long as `<condition>` is true, the loop will execute
- `<statement>`: *loop body*

Types of loops

- *count-controlled*: the while loop executes a certain number of times
  - example:
    ```plaintext
    count = 0;
    while (count < 100) {
        count++;
    }
    ```
- *sentinel-controlled*: the while loop executes until a variable becomes a certain value
  - the interest rate and averaging examples are this type
  - the variable of interest is the *sentinel*
    - interest rate: rate between 0 and 100
    - averaging: negative number entered by the user

Gotchas

- Infinite loop
- Overflow
- Using real numbers
- Off-by-one
- Other logic mistakes

Infinite loop

- Probably the most common error
- Occurs when `<condition>` is never met
  - program has no stopping point
  - loops forever
- Example:
  ```plaintext
  int sum = 0;
  while (sum < 100) {
      count = 0;
      sum = sum + count;
  }
  ```
Other examples of infinite loops

```java
int prod = 0;
while (prod < 100) {
    prod = prod * 3;
}

int count = 0;
while (count != 25) {
    count = count + 3;
}
```

Overflow

- Usually occurs when you have an infinite loop and are using `double` or `float` variables
- The program will continue, but the variable will be set to infinity
- Integers do not overflow
  - instead, they “wrap around” and become negative.

Using real numbers

- Be careful when you are using real numbers in `<condition>`!
- Example:
  ```java
double num = 100.0;
while (num != 0.0) {
    num = num - 1.0000000000000001;
}
```

What's wrong with the example?

- Computers store real numbers as an *approximate value*
  - real numbers are rounded at some point
- Numbers may not be what we think they are
  - e.g., “1” may actually be stored as “1.000000000000001” or “0.99999999999999”
- We should avoid testing equality with real numbers
  - in the previous example, use `while (num >= 0)`
Off-by-one errors

Q: How would you write a loop that repeats 10 times?

```java
count = 0;
while (count < 10) {
    count++;
}
```

count = 0;
while (count <= 10) {
    count++;
}

What do the previous examples show?

- The first loop executes 10 times
- The second loop executes 11 times
- Be careful when determining `<condition>`!

Logic errors

- We saw some examples previously:
  - resetting `sum` to 0 inside the loop
  - initializing `prod` to 0
  - executing our loop one too many times
- Solution: test, test, test your code!