Building Java Programs

Chapter 5: Program Logic and Indefinite Loops

Chapter outline
- Indefinite loops
  - The while loop
  - Sentinel loops
  - Generating random numbers with Random objects
- Boolean logic
  - Boolean expressions and variables
  - Logical operators
- Testing for valid user input
- Indefinite loop variations
  - The do/while loop
  - Logical assertions

Definite loops
- Definite loop: One that executes a known number of times.
  - The for loops we have seen so far are definite loops.

Examples:
- Print 'twas 10 times.
- Find all the prime numbers up to an integer.
- Print each odd number between 5 and 127.

Indefinite loops
- Indefinite loop: One where it is not obvious in advance how many times it will execute.
  - The while loops in this chapter are indefinite loops.

Examples:
- Prompt the user until they type a non-negative number.
- Print random numbers until a prime number is printed.
- Continue looping while the user has not typed "n" to quit.

The while loop statement
- While loop: Executes a group of statements as long as a condition is true.
  - Well suited to writing indefinite loops

The while loop general syntax:
while (condition) {
  <statement(s)>
}

Example:
```java
int number = 1;
while (number <= 100) {
  System.out.println(" + ");
  number += 2;
}
```

Output:
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99

while loops
reading: 5.1
**Example while loop**

- Finds and prints a number's first factor other than 1:
  ```java
  Scanner console = new Scanner(System.in);
  System.out.println("Type a number: ");
  int number = console.nextInt();
  int factor = 2;
  while (number % factor != 0) {
    factor++;
  }
  System.out.println("First factor: ");
  ```

- Example log of execution:
  ```
  Type a number: 8
  First factor: 2
  ```

**for/while loop example**

- What while loop is essentially equivalent to the following for loop?
  ```java
  for (int i = 1; i <= 10; i++) {
    System.out.println(i + " squared = " + i * i);
  }
  ```

- ANSWER:
  ```java
  int i = 1;
  while (i <= 10) {
    System.out.println(i + " squared = " + i * i);
    i++;
  }
  ```

**While loop question**

- Write a piece of Java code that uses a while loop to repeatedly prompt the user to type a number until the user types a non-negative number, then square it.

- Example log of execution:
  ```
  Type a non-negative integer: -4
  Invalid number, try again: -3
  Invalid number, try again: -2
  Valid number, try again: 8
  8 squared is 64
  ```

**While loop answer**

- Solution:
  ```java
  System.out.println("Type a non-negative integer: ");
  int number = console.nextInt();
  int square = 0;
  while (number < 0) {
    System.out.println("Invalid number, try again: ");
    number = console.nextInt();
  }
  System.out.println("Square of "+ number + " is "+ square);
  ```

- Notice that `number` has to be declared outside the loop in order to remain in scope.
**While loop question**

- Write a method named `digitSum` that accepts an integer as a parameter and returns the sum of the digits of that number.
  - `digitSum(12345)` returns 15
  - `digitSum(987654)` returns 30
- You may assume that the number is non-negative.
- Hint: Use the `%` operator to extract the last digit of a number. If we do this repeatedly, when should we stop?

**While loop answer**

- The following code implements the method:
  ```java
  public static int digitSum(int n) {
      int sum = 0;
      while (n > 0) {
          sum += n % 10; // add last digit to sum
          n = n / 10; // remove last digit
      }
      return sum;
  }
  ```

**Sentinel loops**

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

**Example:** Write a program that repeatedly prompts the user for numbers to add until the user types 0, then outputs their sum. (In this case, 0 is our sentinel value.)

**Example log of execution**

- Enter a number [-1 to quit]: 5
- Enter a number [-1 to quit]: 3
- Enter a number [-1 to quit]: -1
- Enter a number [-1 to quit]: 2
- Enter a number [-1 to quit]: 1

  The total was 11

**Flawed sentinel solution**

- What’s wrong with this solution?

  ```java
  Scanner console = new Scanner(System.in);
  int sum = 0;
  int inputNumber = 1; // dummy value, anything but 0
  while (inputNumber != 0) {
      System.out.print("Enter a number [-1 to quit]: ");
      inputNumber = console.nextInt();
      sum += inputNumber;
  }
  System.out.println("The total was "+ sum);
  ```

**A different sentinel value**

- Modify your program to use a sentinel value of -1.

  **Example log of execution**

  - Enter a number [-1 to quit]: 5
  - Enter a number [-1 to quit]: 3
  - Enter a number [-1 to quit]: -1
  - Enter a number [-1 to quit]: 2
  - Enter a number [-1 to quit]: 1

  The total was 11
Changing the sentinel value

- To see the problem, change the sentinel's value to -1:

  ```java
  Scanner console = new Scanner(System.in);
  int sum = 0;
  int inputNumber = 1; // dummy value, anything but -1
  while (inputNumber != -1) {
    System.out.print("Enter a number [-1 to quit]: ");
    inputNumber = console.nextInt();
    sum += inputNumber;
  }
  System.out.println("The total was "+ sum);
  ```

- Now the solution produces the wrong output. Why?
  The total was 266

The problem with our code

- Our code uses a pattern like this:

  ```java
  sum = 0;
  while (inputNumber != -1) {
    prompt for input, read input.
    sum += inputNumber;
  }
  System.out.println("The total was "+ sum);
  ```

- On the last pass through the loop, the sentinel value -1 is added to the sum:

  ```java
  sum += inputNumber; // never moved to top of loop
  System.out.println("The total was "+ sum);
  ```

- This is a fencepost problem.
  - We want to read N numbers (N is not known ahead of time),
  - but only sum the first N - 1 of them.

A fencepost solution

- We need the code to use a pattern like this:

  ```java
  sum = 0;
  prompt for input, read input.
  while (inputNumber is not the sentinel) {
    add input to the sum,
    prompt for input, read input.
  }
  ```

- Sentinel loops often utilize a fencepost-style "loop-and-a-half" solution by pulling some code out of the loop.

Correct code

- This solution produces the correct output:

  ```java
  Scanner console = new Scanner(System.in);
  int sum = 0;
  System.out.print("Enter a number [-1 to quit]: ");
  int inputNumber = console.nextInt();
  while (inputNumber != -1) {
    sum += inputNumber; // moved to top of loop
    System.out.print("Enter a number [-1 to quit]: ");
    inputNumber = console.nextInt();
  }
  System.out.println("The total was "+ sum);
  ```

Constant with sentinel

- A better solution creates a constant for the sentinel:

  ```java
  public static final int SENTINEL = -1;
  ```

- This solution uses the constant:

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

Generating random numbers

reading: 5.1
The Random class

- Random objects generate pseudo-random numbers.
- Class Random is found in the java.util package.
- Import java.util.*;

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>randomInt()</td>
<td>return a random integer</td>
</tr>
<tr>
<td>randomDouble()</td>
<td>return a random real number in the range [0, max)</td>
</tr>
</tbody>
</table>

Example:

```java
Random rand = new Random();
int randomNumber = rand.nextInt(10);
// randomNumber has a random value between 0 and 9
```

Generating random numbers

- Common usage: to get a random number from 1 to N
  - Example: A random integer between 1 and 20, inclusive:
    ```java
    int n = rand.nextInt(20) + 1;
    ```
  - To get a number in arbitrary range [min, max]:
    ```java
    nextInt(sizeOfRange) + min
    ```
    where `sizeOfRange` equals `max - min` + 1
  - Example: A random integer between 5 and 10 inclusive:
    ```java
    int n = rand.nextInt(5) + 5;
    ```

Random questions

- Given the following declaration, how would you get:
  ```java
  Random rand = new Random();
  ```
  - A random number between 0 and 100 inclusive?
  ```java
  int randomNum = rand.nextInt(100); // randomNum has a random value between 0 and 100
  ```
  - A random number between 1 and 100 inclusive?
  ```java
  int randomNum = rand.nextInt(100) + 1;
  ```
  - A random number between 4 and 17 inclusive?
  ```java
  int randomNum = rand.nextInt(14) + 4;
  ```

Random answers

- Given the following declaration, how would you get:
  ```java
  Random rand = new Random();
  ```
  - A random number between 0 and 100 inclusive?
    ```java
    int randomNum = rand.nextInt(100); // randomNum has a random value between 0 and 100
    ```
  - A random number between 1 and 100 inclusive?
    ```java
    int randomNum = rand.nextInt(100) + 1;
    ```
  - A random number between 4 and 17 inclusive?
    ```java
    int randomNum = rand.nextInt(14) + 4;
    ```

Random question

- Write a program that simulates rolling of two six-sided dice until their combined result comes up as 7.

  ```java
  // Roll two dice until sum of 7 is reached
  import java.util.*;

  public class Roll {
    public static void main(String[] args) {
      Random rand = new Random();
      int sum = 0;
      int tries = 0;
      while (sum < 7) {
        int side1 = rand.nextInt(6) + 1;
        int side2 = rand.nextInt(6) + 1;
        sum = side1 + side2;
        System.out.print(side1 + " + " + side2 + " = " + sum + " in try: ");
        tries ++;
      }
      System.out.println("You won after " + tries + " tries.");
    }
  }
  ```

Random answer

```java
// Roll two dice until sum of 7 is reached
import java.util.*;

public class Roll {
    public static void main(String[] args) {
        Random rand = new Random();
        int sum = 0;
        int tries = 0;
        while (sum < 7) {
            int side1 = rand.nextInt(6) + 1;
            int side2 = rand.nextInt(6) + 1;
            sum = side1 + side2;
            System.out.print(side1 + " + " + side2 + " = " + sum + " in try: ");
            tries ++;
        }
        System.out.println("You won after " + tries + " tries.");
    }
}
```
Random/while question

Write a multiplication tutor program. Example log of execution:

This program helps you practice multiplication by asking you random multiplication questions with numbers ranging from 1 to 12 and counting how many you solve correctly.

1 * 6 = 6
Correct!
2 * 5 = 20
Correct!
3 * 4 = 12
Correct!
4 * 3 = 24
Correct!
5 * 8 = 32
Correct!
6 * 9 = 16
Incorrect! The answer was 54
You solved 6 correctly.

Random/while answer

// Make the same as a multiplication tutor program and store them. run.
package java3;
public class Multiplication {
    public static void main(String[] args) {
        System.out.println("Multiplication Tutor Program");
        int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
        int[] answers = {6, 10, 12, 24, 40, 54, 21, 32, 36, 45, 72, 112};
        int correct = 0;
        for (int i = 0; i < numbers.length; i++) {
            int rand = (int) (Math.random() * counts.length);
            System.out.print(numbers[i] + " * " + answers[i] + " = " + answers[i] + "? ");
            if (input.nextInt() == answers[i] + 1) {
                System.out.println("Correct!");
                correct++;
            } else {
                System.out.println("Incorrect! The answer was " + answers[i] + "; you solved " + correct + " correctly.");
            }
        }
    }
}

Random/while answer 2

// Make the same as a multiplication tutor program and store them. run.
package java3;
public class Multiplication {
    public static void main(String[] args) {
        System.out.println("Multiplication Tutor Program");
        int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
        int[] answers = {6, 10, 12, 24, 40, 54, 21, 32, 36, 45, 72, 112};
        int correct = 0;
        for (int i = 0; i < numbers.length; i++) {
            int rand = (int) (Math.random() * counts.length);
            System.out.print(numbers[i] + " * " + answers[i] + " = " + answers[i] + "? ");
            if (input.nextInt() == answers[i] + 1) {
                System.out.println("Correct!");
                correct++;
            } else {
                System.out.println("Incorrect! The answer was " + answers[i] + "; you solved " + correct + " correctly.");
            }
        }
    }

Random text and others

Random can be used in text processing.

- Code to pick a random lowercase letter:
  char letter = (char) (\'a\' + rand.nextInt(26));

- Code to pick a random character from a string (in this case, a random vowel):
  char vowels = "aeiou";
  char vow = vowels.charAt(rand.nextInt(vowels.length());

- Another example: code to pick a random letter representing a base in a DNA strand (A, C, G, or T):
  char base = bases.charAt(rand.nextInt(bases.length()));

Other random values

- Random can be used with double:
  double randDouble = rand.nextDouble() * 2.5 + 1.0;

- Random can be used to pick between arbitrary choices
  double randChoice = rand.nextDouble();
  if (r < 0.5) {
    g.setColor(Color.RED);
  } else if (r < 1.0) {
    g.setColor(Color.GREEN);
  } else {
    g.setColor(Color.BLUE);

Boolean logic

reading: 5.2
**Type boolean**

- boolean: A primitive type to represent logical values.
- A boolean expression produces either true or false.
- The `<condition>` in if statements, for loops are boolean.

Examples:
- boolean isPrime = isPrime < 21
- boolean isLarge = isPrime > 500.01
- if (isPrime)
  - System.out.println("Can't purchase alcohol!");
  - }
- You can create boolean variables, pass boolean parameters, return boolean values from methods, ...

**Logical operators && || !**

- Boolean expressions can use logical operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
<td>x &amp;&amp; y</td>
<td>true</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>or</td>
</tr>
<tr>
<td>!</td>
<td>not</td>
<td>! x</td>
<td>false</td>
</tr>
</tbody>
</table>

- Truth tables of each operator used with boolean values p and q:

| p   | q   | p && q | p || q |
|-----|-----|--------|--------|
| true| true|  true  |  true  |
| true| false| false  |  true  |
| false| true| false  |  true  |
| false| false|  false |  false |

**Methods that return boolean**

- There are methods in Java that return boolean values.
- A call to one of these methods can be used as a `<condition>` in an if, for, loop, while loop, or if statement.

Examples:
- Scanner scanner = new Scanner(System.in);
  - System.out.print("Tell your name: ");
  - String line = scanner.nextLine();
  - if (line.startsWith("Dr."))
    - System.out.print("айте меня ");
    - } else if (line.endsWith("!");
    - System.out.print("And I am Ted "Thomson" Logan!");
    - }

**String boolean methods**

- The following string methods return boolean values:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>equalsIgnoreCase()</td>
<td>whether two strings contain exactly the same characters</td>
</tr>
<tr>
<td>equals()</td>
<td>whether two strings contain the same characters, ignoring uppercase/lower case differences</td>
</tr>
<tr>
<td>startsWith()</td>
<td>whether one string contains the other's characters at its start</td>
</tr>
<tr>
<td>endsWith()</td>
<td>whether one string contains the other's characters at its end</td>
</tr>
</tbody>
</table>

**Writing boolean methods**

- Methods can return a boolean result.
  - public boolean isWholeNumber(double x) {...
  - if (x / 2 == 0)
  - return true;
  - else
  - return false;
  - }

- Calls to such methods can be used as conditions:
  - if (isWholeNumber(7.0))
  - ...
**Writing boolean methods 2**

- Another example that returns a boolean result:
  ```java
  public static boolean containsVowel(char ch) {
      if ("aeiouAEIOU".contains(ch)) return true;  
      return false;
  }
  ```

- Example call to this method:
  ```java
  String name = "a.e. cunning";
  char firstVowel = name.charAt(0);  
  if (containsVowel(firstVowel)) System.out.println("You forgot to capitalize your name!");
  ```

**Boolean practice problems**

- Write a method named isVowel that returns whether a particular character is a vowel (a, e, i, o, u). Count only lowercase vowels.
  ```java
  public boolean isVowel(char ch) {
      return ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u';
  }
  ```

- Write a method named allDigitsOdd that returns whether every digit of a positive integer is odd.
  ```java
  public boolean allDigitsOdd(int n) {
      while (n > 0) {
          int digit = n % 10;
          if (digit % 2 == 0) return false;
          n /= 10;
      }
      return true;
  }
  ```

- Write a method named countVowels that returns the number of lowercase vowels in a String.
  ```java
  public int countVowels(String text) {
      int count = 0;
      for (int i = 0; i < text.length(); i++) {
          char c = text.charAt(i);
          if (isVowel(c) && Character.isLowerCase(c)) count++;
      }
      return count;
  }
  ```

**Boolean practice solutions**

- **isVowel**
  ```java
  public static boolean isVowel(char ch) {
      return ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u';
  }
  ```

- **allDigitsOdd**
  ```java
  public static boolean allDigitsOdd(int n) {
      while (n > 0) {
          int digit = n % 10;
          if (digit % 2 == 0) return false;
          n /= 10;
      }
      return true;
  }
  ```

- **countVowels**
  ```java
  public static int countVowels(String text) {
      int count = 0;
      for (int i = 0; i < text.length(); i++) {
          char c = text.charAt(i);
          if (isVowel(c) && Character.isLowerCase(c)) count++;
      }
      return count;
  }
  ```

**Boolean question**

- Modify your previous multiplication tutor program to use a static method that returns a boolean value.
  ```java
  public static boolean isCorrect(int num1, int num2) {
      int result = num1 * num2;
      if (result == 144) return true;
      return false;
  }
  ```

- Test the program with different inputs and check if it returns the correct boolean results.
Boolean practice problem

Write a program that determines if a number is prime. If the number is prime, output
1, and if it is not prime, output 0.

Sample input:
5

Sample output:
1

Boolean practice solution

As part of your solution, write two methods:

- isPrime(int n)
- getPrimeFactors(int n)

Use these methods to solve the problem.

Boolean answer

The program should output 1 if the number is prime, and 0 otherwise. The
method isPrime should return true if the number is prime, and false otherwise.

Boolean answer 2

The program should output the prime factors of a given number.

Boolean practice solution

```java
public class Main {  
    public static void main(String[] args) {  
        // your code here
    }
}
```

Boolean practice solution 2

```java
public class Main {  
    public static void main(String[] args) {  
        // your code here
    }
}
```

Indefinite loop variations

*reading: 5.4*

The do/while loop

* do/while loop: Executes statements repeatedly while a condition is true, testing it at the end of each repetition.
* Example:

```java
// roll until we get a number other than 3
int dice;  
while (dice != 3) {  
    dice = new Random().nextInt(6) + 1;  
}
```

* The do/while loop, general syntax:

```java
do {  
    <statement(s)>;  
} while (<condition>);  
```

* Example:

```java
// roll until we get a number other than 3
int dice;  
while (dice != 3) {  
    dice = new Random().nextInt(6) + 1;  
}
```

* The do/while question

* Modify the previous dice program to use a do/while loop.

```java
// Example log of execution:
2 + 4 = 6
3 + 5 = 8
5 + 6 = 11
1 + 1 = 2
4 + 3 = 7
You won after 5 trials!
```
do/while solution

```java
// Rolls two dice until a sum of 7 is reached.
import java.util.*;
public class Roll {
    public static void main(String[] args) {
        Random rand = new Random();
        int sum = 0;
        while (true) {
            int n1 = rand.nextInt(6) + 1;
            int n2 = rand.nextInt(6) + 1;
            sum = n1 + n2;
            System.out.println("The sum after " + n1 + " + " + n2 + " = " + sum);
            if (sum == 7) {
                System.out.println("The sum is 7.");
                break;
            }
        }
    }
}
```

"Forever" loop with break

```java
// "Forever" loop with break.

```java

Sentinel loop with break

• A working sentinel loop solution using break:

```java
Scanner console = new Scanner(System.in);
int sum = 0;
while (true) {
    System.out.print("Enter a number [-1 to quit]: ");
    int inputNumber = console.nextInt();
    if (inputNumber == -1) { // don't add -1 to sum
        break;
    }
    sum += inputNumber; // inputNumber != -1 here
}
System.out.println("The total was "+ sum);
```

User errors

reading: 5.3

Testing for valid user input

• A Scanner object has methods that can be used to "look ahead" to test whether the upcoming input token is of a given type.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasNext()</td>
<td>Whether the next token can be read as a String</td>
</tr>
<tr>
<td>hasNextInt()</td>
<td>Whether the next token can be read as an int</td>
</tr>
<tr>
<td>hasNextDouble()</td>
<td>Whether the next token can be read as a double</td>
</tr>
</tbody>
</table>

• Each method waits for the user to type input and press Enter, then reports a true or false answer based on what was typed.

Scanner condition example

• The has methods are useful for testing whether the user typed the kind of token we wanted.

```java
Scanner console = new Scanner(System.in);
if (console.hasNextInt())) {
    int age = console.nextInt(); // will not throw an exception
    System.out.println("The age is + age + " + years");
}
if (console.hasNextDouble()) {
    System.out.println("Please enter a whole number for your age.");
}
if (console.hasNext()) {
    System.out.println("You did not type a number.");
}
```
Logical assertions

* assertion: A statement that is either true or false.

Examples:

- Java was created in 1995.
- The sky is purple.
- 23 is a prime number.
- 10 is greater than 20.
- x divided by 2 equals 7. (depends on the value of x)

Assertions in code

* We can make assertions about our code and ask whether they are true at various points in the code.

Valid answers are ALWAYS, NEVER, or SOMETIMES.

```java
System.out.println("Type a nonnegative number: ");
double number = console.nextDouble();
// Point A: is number < 0.0 bare? (SOMETIMES)
while (number < 0.0) { // Point B: is number < 0.0 bare? (ALWAYS)
    System.out.println("Negative; try again: ");
    number = console.nextDouble();
    // Point C: is number < 0.0 bare? (SOMETIMES)
}
// Point D: is number < 0.0 bare? (NEVER)
```

Assertion example 1

```java
public static void mystery() throws ConsoleException
    int x = 5;
    // Point A
    while (x > 0) { // Point B
        System.out.println("x = ");
        x--; // Point C
        // Point D
    }
    System.out.println("x = 0");
```

Assertion example 2

```java
public static void mystery() throws ConsoleException
    int x = 5;
    // Point A
    while (x > 0) { // Point B
        System.out.println("x = ");
        x--; // Point C
        // Point D
    }
    System.out.println("x = 0");
```

Assertion example 3

```java
public static void mystery() throws ConsoleException
    int x = 5;
    // Point A
    while (x > 0) { // Point B
        System.out.println("x = ");
        x--; // Point C
        // Point D
    }
    System.out.println("x = 0");
```