Module 5 - Loops and Strings

- Module 5 - Loops and Strings
- General Notes
  - Domains
- Loops and Strings
  - Example 1
  - Example 2 - Replace Double Spaces
- Nested Loops
  - Example 1
  - Example 2
- Developing Programs Incrementally
- Variable Name Scope
  - Scope of Names
  - For Loop Index
    - Common Error
- Enumerations

General Notes

Domains

- A **top-level domain (TLD)** name is the last part of an Internet domain name like .com in example.com.
- A **core generic top-level domain (core gTLD)** is a TLD that is either .com, .net, .org, or .info. A second-level domain is a single name that precedes a TLD as in apple in apple.com

Loops and Strings

A programmer commonly iterates through a string, examining each character. The following example counts the number of letters in a string, not counting digits, symbols, etc:

Example 1

```java
import java.util.Scanner;

public class CountLetters {
  public static void main(String[] args) {
    Scanner scnr = new Scanner(System.in);
    String inputWord;
    int numLetters;
    int i;

    System.out.print("Enter a word: ");
    inputWord = scnr.next();

    numLetters = 0;
    for (i = 0; i < inputWord.length(); ++i) {
      if (Character.isLetter(inputWord.charAt(i))) {
        numLetters += 1;
      }
    }

    System.out.println("Number of letters: " + numLetters);
  }
}
```

Example 2 - Replace Double Spaces

```java
import java.util.Scanner;

public class SingleSpaces {
  public static void main(String[] args) {
    Scanner scnr = new Scanner(System.in);
    String userText;

    System.out.print("Enter sentence: ");
    userText = scnr.nextLine();

    while (userText.indexOf("  ") != -1) {
      // At least one double-space still exists
      // Replace all double-spaces by one space
      userText = userText.replace("  ", " ");
    }

    System.out.print("New sentence: " + userText);
  }
}
```

Nested Loops

A **nested loop** is a loop that appears in the body of another loop. The nested loops are commonly referred to as the **inner loop** and **outer loop**.

Example 1

```java
import java.util.Scanner;

public class DomainNamePrinter {

  public static void main(String[] args) {
    Scanner scnr = new Scanner(System.in);
    char letter1;
    char letter2;

    System.out.println("Two-letter domain names:");

    letter1 = 'a';
    while (letter1 <= 'z') {
      letter2 = 'a';
      while (letter2 <= 'z') {
        System.out.println("" + letter1 + "" + letter2 + ".com");
        ++letter2;
      }
      ++letter1;
    }
  }
}
```

Example 2

```
import java.util.Scanner;

public class IntHistogram {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      int numAsterisk;  // Number of asterisks to print
      int i;            // Loop counter

      numAsterisk = 0;

      while (numAsterisk >= 0) {
         System.out.print("Enter an integer (negative to quit): ");
         numAsterisk = scnr.nextInt();

         if (numAsterisk >= 0) {
            System.out.println("Depicted graphically:");
            for (i = 1; i <= numAsterisk; ++i) {
               System.out.print("*");
            }
            System.out.println("\n");
         }
      }

      System.out.println("Goodbye.");
   }
}
```

Developing Programs Incrementally

Experienced programmers develop programs **incrementally**, meaning they create a simple program version, and then grow the program little-by-little into successively more-complete versions.

- What many new programmers do, but shouldn't, is write the entire program, compile it, and run it—hoping it works.
  - Debugging such a program can be difficult because there may be many distinct bugs.

Variable Name Scope

Scope of Names

- A declared name is only valid within a region of code known as the name's **scope**.
- A **block** is a brace-enclosed `{...}` sequence of statements, such as found with an `if-else`, `for` loop, or `while` loop. A variable name's scope extends from the declaration to the closing brace `}`.

For Loop Index

for loop	Equivalent while loop
for (int i = 0; i < 5; ++i) {	{
x = x + i;	int i = 0;
}	while (i < 5) {
	x = x + i;
x = x + i; // ERROR	++i;
	}
	}
	x = x + i; // ERROR

- The approach of declaring a for loop's index variable in the for loop's initialization statement makes clear that the variable's sole purpose is to serve as that loop's index.

Common Error

A common error is to declare a variable inside a loop whose value should persist across iterations.

Enumerations

- Useful for when variables only need to store a small set of named values.
- An **enumeration type (enum)** declares a name for a new type and possible values for that type.

```java
// Declaration
public enum identifier {enumerator1, enumerator2, ...}

// Example 1
import java.util.Scanner;

/* Manual controller for traffic light */
public class TrafficLightControl {
  // enum type declaration occurs outside the main method
  public enum LightState {RED, GREEN, YELLOW, DONE}

  public static void main(String[] args) {
    Scanner scnr = new Scanner(System.in);
    LightState lightVal;
    String userCmd;

    lightVal = LightState.RED;
    userCmd = "-";

    System.out.println("User commands: n (next), r (red), q (quit).\n");

    while (lightVal != LightState.DONE) {

      if (lightVal == LightState.GREEN) {
        System.out.print("Green light  ");
        userCmd = scnr.next();
        if (userCmd.equals("n")) { // Next
          lightVal = LightState.YELLOW;
        }
      }
      else if (lightVal == LightState.YELLOW) {
        System.out.print("Yellow light  ");
        userCmd = scnr.next();
        if (userCmd.equals("n")) { // Next
          lightVal = LightState.RED;
        }
      }
      else if (lightVal == LightState.RED) {
        System.out.print("Red light  ");
        userCmd = scnr.next();
        if (userCmd.equals("n")) { // Next
          lightVal = LightState.GREEN;
          }
        }

        if (userCmd.equals("r")) { // Force immediate red
          lightVal = LightState.RED;
        }
        else if (userCmd.equals("q")) { // Quit
          lightVal = LightState.DONE;
        }
      }

      System.out.println("Quit program.");
    }
  }
```

- The example illustrates the idea of a **state machine** that is sometimes used in programs, especially programs that interact with physical objects, wherein the program moves among particular situations ("states") depending on input;
  - See What is: State machine.
- Enumerations are safer than strings, generating compiler errors for invalid values where a String would not.