Logic & program control part 3:

Compound selection structures
Multi-way selection

• Many algorithms involve several possible pathways to a solution
• A simple if/else structure provides two alternate paths; if more than two paths are possible, a multi-way branching structure, such as a nested if/else or switch statement
Nested if/else structures

- An if/else structure can contain another if/else structure
- When this happens, the structures are said to be nested
- This occurs when one condition test depends on the result of another such test
Nested if/else syntax

if (expression 1)
{
    statement(s); // perform these if expression 1 is true
}
else if (expression 2)
{
    statement(s); // perform these if expression 1 is false, expression 2 is true
}
...
else if (expression n)
{
    statement(s); // perform if nth expression true, all previous false
}
else
{
    statements(s); // perform if all expressions false
}
Nested if/else Statements

- Each expression is evaluated in sequence, until some expression is found that is true.
- Only the specific statement following that particular true expression is executed.
- If no expression is true, the statement following the final else is executed.
- Actually, the final else and final statement are optional. If omitted, and no expression is true, then no statement is executed.
Example

• Suppose you are assigned the task of writing a simple math quiz program:
  – The user selects from a menu of options describing different types of problems
  – When an option has been chosen, the program displays a problem (using randomly-generated numbers) and prompts the user for an answer
  – The program displays a message informing the user if s/he was right or wrong, and provides the correct answer if the user didn’t

• A fragment of this program appears on the next 2 slides
Simple calculator – variable declaration & menu processing

int op1, op2;  // operands for problem; generated randomly
int choice;    // user’s choice of operation to perform

// part of program not shown …

// ************** display problem & generate correct answer **************
if (choice == 1)
{
    System.out.println(op1 + " + " + op2 + " = ");
    answer = op1 + op2;
}
else if (choice == 2)
{
    System.out.println(op1 + " - " + op2 + " = ");
    answer = op1 - op2;
}
else if (choice == 3)
{
    System.out.println(op1 + " * " + op2 + " = ");
    answer = op1 * op2;
}
else if (choice == 4)
{
    System.out.println(op1 + " / " + op2 + " = ");
    answer = op1 / op2;
}
else  // user chose something that wasn't on the menu
{
    System.out.println
    ("That was not a valid option. You must be punished.");
}
Example

- Check the number of credit hours earned by a student, and print his/her class status as a freshman, sophomore, junior, or senior, as follows:
  - if more than 90 credit hours earned, print senior
  - if between 60 and 89 credit hours, print junior
  - if between 30 and 59 credit hours, print sophomore
  - otherwise, print freshman
Java code

if ( creditsEarned >= 90 )
    System.out.print( “SENIOR STATUS\n”);
else if (creditsEarned <= 89 && creditsEarned >= 60 )
    System.out.print(“JUNIOR STATUS\n”);
else if ( creditsEarned >= 30 && creditsEarned <= 59 )
    System.out.print(“SOPHOMORE STATUS\n”);
else if (creditsEarned < 30)
    System.out.print(“FRESHMAN STATUS\n”);
Equivalent logic

- The logic in a selection structure can often be written in several different ways.
- For example, on the previous slide we had logic that printed a student’s class status based on credit hours earned.
- The next slide illustrates the same logic, but written more efficiently.
- In general, strive for the simplest expressions, with the least amount of evaluation required.
- This not only makes code more efficient, but also easier to test and debug.
Java code

if ( creditsEarned >= 90 )
    System.out.print("SENIOR STATUS\n");
else if ( creditsEarned >= 60 )
    System.out.print("JUNIOR STATUS\n");
else if ( creditsEarned >= 30 )
    System.out.print("SOPHOMORE STATUS\n");
else
    System.out.print("FRESHMAN STATUS\n");
Which is preferable?

Random rg = new Random();
int x = rg.nextInt();
if (x > 0)
    System.out.printf
    ("%d is positive\n", x);
else if (x < 0)
    System.out.printf
    ("%d is negative\n", x);
else
    System.out.println
    ("the value is 0");
// A

Random rg = new Random();
int x = rg.nextInt();
if (!((x > 0) || (x < 0)))
    System.out.println
    ("the value is 0");
else if (x != 0) {
    if (x > 0)
        System.out.printf
        ("%d is positive\n", x);
    else
        System.out.printf
        ("%d is negative\n", x);
} // B
Example problem

• You are asked to write a program that provides information about air quality
• Your city classifies a pollution index
  – less than 35 as “Pleasant”,
  – 35 through 60 as “Unpleasant”,
  – above 60 as “Health Hazard.”
• What is the condition? How should it be tested?
Notes on nested if/else structures

• The number of “ifs” and “eloses” need not be equal, but each else must be preceded by an if
• By default, an else clause is associated with the closest preceding if that doesn’t already have an else
Example - what is the output?

```java
float average;
average = 100.0;
if ( average >= 60.0 )
    if ( average < 70.0 )
        System.out.print
            ("Marginal PASS\n"); // A
else
    System.out.print("FAIL\n"); // B
```
Write a program to control your life using selection structures:

Every Monday thru Friday you go to class. When it is raining you take an umbrella. But on the weekend, what you do depends on the weather.

If it is raining you read in bed. Otherwise, you have fun outdoors.
// One variation

import java.util.*;
public class HowToLive {
    public static void main (String [] args) {
        int day;
        boolean raining;
        String tf;
        Scanner kb = new Scanner (System.in);
        System.out.print("Enter day (use 1 for Sunday: ");
        day = kb.nextInt();
        System.out.print("If it’s raining, enter T; otherwise, enter F ");
        if (tf.equalsIgnoreCase("T"))
            raining = true;
        else
            raining = false;
    }
}
if ( ( day == 1) || (day == 7) ) { // Sat or Sun
    if (raining )
        System.out.println("Read in bed");
    else
        System.out.println("Have fun outdoors");
} // end if day
else {
    System.out.println("Go to class ");
    if (raining)
        System.out.println("Take an umbrella");
} // end else
} // end main
} // end class
import java.util.*;
public class HowToLive2 {
    public static void main (String [] args) {
        int day;
        boolean raining=false;
        String input;
        Scanner kb = new Scanner (System.in);
        System.out.print("Is it raining? (1=yes, 0=no): ");
        input = kb.next();
        if(input.equals("1"))
            raining=true;
        System.out.print("What day is it? ");
        input = kb.next();
Another variation continued

```java
if (raining)
{
    if (input.equals("Sunday") || input.equals("Saturday"))
        System.out.println("Read in bed");
    else
        System.out.println("Take an umbrella to class");
} // ends outer if
else
{
    if(!input.equals("Sunday") && !input.equals("Saturday"))
        System.out.println("Go to class");
    else
        System.out.println("Go outside & play");
} // ends outer else
} // ends main
} // ends class HowToLive2
```
Which version is better?

A: version 1
B: version 2
Logical equivalence

• We can see that nested if statements can be equivalent to anded expressions:
  
  ```
  if ((a > b) && (b > c)) could be written as:
  if (a > b) {
    if (b > c) …
  }
  ```

• In the next example, we’ll use logical or to determine if a value lies outside a given range
Example using if/else if

```java
if (value < 1)
    System.out.println("Value out of range");
else if (value > 100)
    System.out.println("Value out of range");
```
Example using logical or

if ((value < 0) || (value > 100))
    System.out.println("Value out of range");
Same example using && and !

```java
if (!((value >= 0) && (value <= 100)))
    System.out.println("Value out of range");
```
Which if clauses use the same logic?

if(!((a > b) || (a > c)))  A. First & second
if (!((a > b) && !((a > c)))  B. First & third
if ( ((a <= b) && (a <=c))) C. Second & third
D. All are equivalent
E. None are equivalent
An alternative: the switch/case structure

• For problems like the calculator menu and how to spend your day examples, an alternative kind of selection control structure is available

• In a switch/case structure, the variable to be tested must be testable to discrete values; in other words, tested for equality
Switch/case syntax

switch(expression)
{
    case value1:
        statement(s);
        break;
    case value2:
        statement(s);
        break;
    ...
    case valueN:
        statement(s);
        break;
    default:
        statement(s);
}

Notes:
• “expression” is usually, but doesn’t have to be, a single variable
• Each case label contains a possible value for the expression
• The values must be integral values; e.g. ints, longs, etc. - may be literal values or named constants
Simple calculator – if/else vs. switch

```java
if (choice == 1)
{
    answer = op1 + op2;
}
else if (choice == 2)
{
    answer = op1 - op2;
}
...
else
{
    System.out.println("not a valid option");
}
```

```java
switch (choice)
{
    case 1:
        answer = op1 + op2;
        break;
    case 2:
        answer = op1 - op2;
        break;
    ...
    default:
        System.out.println("not a valid option");
}
```
How switch works

- The expression is evaluated; if a matching value is found next to one of the case labels, all statements following that case are executed until either:
  - a break statement is encountered or
  - the end bracket of the switch/case structure is encountered
- If no matching case is found, the statements following the default label are performed
The importance of the break statement

- The break statement at the end of each case terminates the switch statement at that point
- Case labels are just labels; once any valid case is found, all statements from then on will execute sequentially in the absence of a break – subsequent labels are ignored
- With break statements, only the statement(s) associated with the valid case will be performed
Syntax notes on switch structure

• Begin/end brackets surround the whole structure from the end of the switch() clause until the end of the last case
• No brackets surround the statements within individual cases; brackets would only be used there if a case contained another control structure which required its own set of brackets
• Default case is optional; works like final else in if/else structure
Switch with no brakes (breaks)
int light; // 1 means red, 2 means green, 3 means yellow
...
switch (light)
{
    case 1:
        System.out.println("STOP!");
    case 2:
        System.out.println("GO!");
    case 3:
        System.out.println("Go real fast!");
}

*What happens when light is red?*
Testing Selection Control Structures

- To test a program with branches, use enough data sets so that every branch is executed at least once
- This is called minimum complete coverage
How to Test a Program

• Design and implement a test plan

• A test plan is a document that specifies the test cases to try, the reason for each, and the expected output

• Implement the test plan by verifying that the program outputs the predicted results