Logic & program control part 2:

Simple selection structures
Summary of logical expressions in Java

• “boolean expression” means an expression whose value is true or false
• An expression is any valid combination of operators and operands
• Use of parentheses is encouraged; otherwise, use precedence chart to determine order
Logical expressions & program control

• Relational expressions can be used to control the flow of logic in a program
• Depending on the truth value of an expression, a program can be made to perform one task or another (but not both)
• A control structure that fits this description is called a selection structure
Selection & Java

• In Java, a simple selection structure is created using an **if statement**, which has the following syntax:
  
  if (relational expression)
  {
      statement(s);
  }

• If only one statement follows the if clause, the curly brackets are unnecessary
Example

double otpay = 0.0;  // overtime pay – time and 1/2

if (hours > 40)
{
    otpay = hours - 40;
    otpay = otpay * wage * 1.5;
}

pay = otpay + hours * wage;
Potential error condition

```c
float average; // average price
float total;   // sum of prices
int howMany;   // entered
              // number of prices entered

average = total / howMany;
```
Improved Version

```c
float average,    // average price
total;           // sum of prices entered
int   howMany;   // number of prices entered
...
if ( howMany > 0 )
{
    average = total / howMany;
}
```
After which 2 lines should brackets be added?

// 0
if (code == ‘T’) // 1 – do 2-7 if this is true
    System.out.print (“Enter income:”); // 2
income = kb.nextDouble(); // 3
System.out.print (“Enter tax rate:”); // 4
taxRate = kb.nextDouble(); // 5
taxDue = income * taxRate; // 6
System.out.printf(“You owe $%.2f”, taxDue); // 7
The if-else control structure

- In the previous examples, a set of statements either executed, or didn’t, based on the truth value of an expression
- In many situations, two options make more sense – one set of statements executes if the condition is true, while the other executes if it is false
- This slightly more complicated version is called an if/else structure
If/else in Java

Basic syntax is:

if (relational expression)
{
    statement(s);
}
else
{
    statement(s);
}
Example

Read an int value from the keyboard
If value is less than 0, print its square
If value is greater than 0, print its square root
Program logic in Java

String s;
double value;
Scanner kb = new Scanner(System.in);
System.out.print(“Enter a number: ”);
s=kb.nextLine();
value = Double.parseDouble(s);
if (value < 0)
    System.out.printf(“Value squared is %.1f”,
                        Math.pow(value, 2.0));
else
    System.out.printf(“Square root of value is %.1f ”,
                        Math.sqrt(value));
Use of blocks recommended

```java
if ( Expression )
{

}
else
{

}

“if clause”

“else clause”
```
What output? and Why?

```java
boolean code;
code = false;

if (!code)
    System.out.println("Yesterday");  // A
else
    System.out.println("Tomorrow");  // B
```
int carDoors, driverAge;
double premium, monthlyPayment;
.

if ((carDoors == 4) && (driverAge > 24))
{
    System.out.println("LOW RISK");
    premium = 650.00;
}
else
{
    System.out.println("HIGH RISK");
    premium = 1200.00;
}

monthlyPayment = premium / 12.0 + 5.00;
What happens if you omit braces?

```java
if ( (carDoors == 4) && (driverAge > 24) )
    System.out.println("LOW RISK");
    premium = 650.00 ;
else
    System.out.println("HIGH RISK");
    premium = 1200.00 ;

monthlyPayment = premium / 12.0 + 5.00 ;
```

**COMPILE ERROR OCCURS.** The “if clause” is the single statement following the if.
Adding braces:

```java
if ( (carDoors == 4) && (driverAge > 24) )
{
    System.out.println(“LOW RISK”); 
    premium = 650.00 ;
}
else
    System.out.println(“HIGH RISK”); 
    premium = 1200.00 ; 

monthlyPayment = premium / 12.0 + 5.00 ;
```

**PROGRAM COMPILES.** Is it correct? (T = yes, F = no)
Which is the correct translation of the following?

If taxCode is 1, increase price by adding taxRate times price to it.

A. if (taxCode = 1) {price += price * taxRate;}
B. If (taxCode == 1) price += price * taxRate;
C. if (taxCode ==1) price += price * taxRate;
D. All are correct
Short-Circuit Benefits

- one boolean expression can be placed first to “guard” a potentially unsafe operation in a second boolean expression

- Time is saved in evaluation of complex expressions using operators || and &&
Use Precedence Chart

```java
int number;
float x;

if (number != 0 && x < 1 / number)
```

/ 
 has highest priority
< 
 next priority
!= 
 next priority
&& 
 next priority

What happens if number has value 0?
Our Example Revisited

```java
int number;
float x;

if(((number != 0) && (x < 1 / number))

is evaluated first and has value false

Because operator is &&, the entire expression will have value false. Due to short-circuiting the right side is not evaluated in Java.
```
Logical expressions & floating-point numbers

• In general, it isn’t a good idea to compare two floating-point numbers for equality
• This is because floats and doubles always represent approximations of values, and, although 2 equals 2, 2.0000000000003 does not equal 2.00000000000019
• A better method for comparing floating-point numbers involves deciding how close is close enough to equal
Comparing floating-point numbers for equality

- The expression below assumes numbers are close enough to equal if they are within 1/100,000,000 of one another:
  
```java
if(Math.abs(a-b) < 1.0e-10)
    System.out.printf("%f = %f (or close enough)\n", a, b);
else
    System.out.printf("%f and %f are not equal\n", a, b);
```
Comparing Objects

- When comparing primitive-type variables, constants, and values, the relational operators are adequate.
- When comparing objects, we can use the relational operators, but they don’t mean the same thing they mean with the primitive types.
Comparing Objects

- Recall that, when we declare an object, the identifier in the declaration doesn’t contain an object until we initialize it by calling the object’s constructor (using the new operator)
- When we invoke the constructor, a new object is created, and its memory address is associated with the identifier from the declaration
Comparing objects: method equals

• Because the relational operator `==` compares only the addresses of objects, many objects have a member method to compare object contents for equality
• The equals method performs a comparison that depends on its definition within the class
• For example, for String objects, the equals method performs a letter-by-letter comparison between two Strings, evaluating true if the Strings’ contents are identical
Example: comparing Strings

String s1, s2;
s1 = new String ("a string");
s2 = new String ("a string");
• The expression s1 == s2 evaluates false
• The expressions s1.equals(s2) and s2.equals(s1) evaluate true
More String comparison methods

- The equals method returns true if the calling object and its argument are identical in both spelling and case
- A second method, equalsIgnoreCase, can be used to compare Strings for spelling only; for example:
  String s1 = new String (“hello”);
  String s2 = new String (“HELLO”);
  - s1.equals(s2) returns false
  - s1.equalsIgnoreCase(s2) returns true
More String comparison methods

• The String class includes two comparison methods besides equals and equalsIgnoreCase:
  – compareTo is similar to equals; it is case-sensitive
  – compareToIgnoreCase, as the name implies, ignores case
String compare methods

- Both compare methods work as follows:
  - if the calling object is less than the argument, the method returns a negative number
  - if the calling object is greater than the argument, the method returns a positive number (greater than 0)
  - if the Strings are equal, the method returns 0
- In this context, “less than” and “greater than” refer to alphabetical order – so, for example, “abc” is less than “bcd” because “a” comes before (is less than) “b”
String compare methods

• If the case-sensitive compare method is used, then if two Strings have the same spelling but one contains capital letters, the one with the capital letters will evaluate as less than the one with equivalent lowercase letters
• So, for example, “Hello” is less than “hello”
Exception to the rules

• One important point about Strings – they can sometimes act like primitive objects

• If a String is instantiated without the new operator, as in the example below:
  
  String s1 = “no news is good news”;
  String s2 = “no news is good news”;
  – then the expression s1 == s2 evaluates true
  – this is because, if the same String literal is assigned without “new” to 2 different objects, both objects refer to the same memory location
  – however, if s1 then gets assigned a different String literal, the expression s1 == s2 will be false, because now s2 refers to the original address, but s1 now refers to a new address