ICS 111
Conditional Statements

- Review: Sequence, Repetition, Conditional
  - and Parallel Execution, Program Structure
- Conditional Statements
  - Conditional Expressions
  - Switch Statements
- Syntax and Semantics

Review: Sequence, Repetition, Conditional

- The past few lectures were spent talking of some basic elements of programming:
  - print statements
  - variable assignments
  - string operations
  - arithmetic operations
- These can be combined into larger programs by sequence, repetition, and conditionals
  - and parallel execution

Review: Parallel Execution, Program Structure

- Parallel Execution is important in the real world to help our programs run fast and be distributed across systems
  - but we still won't look at it in ICS 111
- Program Structure is built-in to everything we do as programmers. We will see more of it throughout the course, including in this lecture.

Review: Sequence

- You've already seen and used sequences
- sequences are easy and natural, just write statements one after another
- building blocks to be sequenced include:
  - print statements
  - variable assignments
  - string and arithmetic operations
  - loops, conditionals, and other sequences

Review: Repetition

- We've talked about repetition (loops), but haven't yet seen it in Java programs
- we will talk about it from September 16th

Review: Conditional

- We have seen some examples of conditional
- Inside the “if” or “else” parts we have seen print statements and variable assignments
- Anything could be inside the if:
  - any basic statement
  - another conditional
  - a sequence of statements
  - a loop

Code Blocks: motivation

```
if (b)
  x = x + 1;
```

- this is a complete if statement
- the body of the if (and the body of the else, if present) is a single statement
- loops are similar in that the body of a loop can be a single statement
- usually, we want more than a single statement
- we can group a sequence of statements into a block
  - bracketed by { curly braces }

Code Blocks in conditionals

```
if (b) {
  x = x + 1;
}
```

- This is so common, that using braces has become a standard, like camelCase
- This also helps to prevent errors: when you add a new statement, the curly brace reminds you whether you are in or out of the if or else part
- In ICS 111, you should always use braces for your if statements
  - and for the loops too!

Code Blocks in conditionals

```
if (x > 0) {
  x = x + 1;
  System.out.println (“new x: “ + x);
} else {
  System.out.println (“x < 0: “ + x);
}
```

Programming Languages: Semantics and Syntax

- We write programs to specify a computation
  - what we're telling a program to do is the program's **semantics**
  - for example, the semantics of x + 1 is that the value computed is one more than the value stored in x
- The rules for how we can write programs are the programming language **syntax**
- Compilers are very good at detecting syntax errors
  - semantic errors are when the programmer tells the program to do something other than what the programmer is trying to do
  - most semantic errors cannot be detected by the compiler

Syntax of Conditionals in Java

- keyword `if`, followed by:
- a boolean expression in parentheses, the **condition**
- the **if** statement(s) (also known as **then** statement(s))
  - **then** is never written, and is not a Java keyword
- zero or more **else-if** statements, each beginning with the keywords `else if`, then a boolean expression in parentheses, then the else-if statement(s)
- possibly an `else` keyword followed by its **else** statement(s)

Semantics of Conditionals in Java

- The boolean expression is evaluated
- if the boolean expression evaluates to `true`, the if statement (then statement) is executed
- otherwise, the boolean expressions for any subsequent else-if parts are evaluated in sequence. If the expression is `true`, the corresponding statement is executed, and evaluation of the entire conditional ends
- if a final else statement is reached, it is executed

Example of a conditional

```
if (price >= 10) {
  buy immediately
} else if (price >= 20) {
  think about it
} else if (price >= 30)
  complain
} else {
  buy two
}
```

- note the parentheses around the booleans
- what is wrong with this example?
  - look for one syntax error and one semantic error
- inspired by section 3.3 in the book

Syntax Error

```
if (price >= 10) {
  buy immediately
} else if (price >= 20) {
  think about it
} else if (price >= 30)
  complain
} else {
  buy two
}
```

- the second else-if is missing a brace
- this might be ok if complain is a single Java statement, but the closing brace is present
- the compiler will let you know that there is an error

Semantic Error

- We fix the syntax error by adding that brace, and the program now compiles, but it is still wrong. What is wrong?

```
if (price >= 10) {
  buy immediately
} else if (price >= 20) {
  think about it
} else if (price >= 30) {
  complain
} else {
  buy two
}
```

Semantic Error

- The semantic error is that there is no way to enter the two else-if statements. If the price is 10, 20, 30, or more, we satisfy the condition for the if, and ignore the rest

```
if (price >= 10) {  // 20 > 10, 30 > 10
  buy immediately
} ...
```

- It is usually a semantic error when there is no possible way for the condition of an else-if to evaluate to true
- Sometimes the compiler will detect this

Semantic Error, Fix 1: reorder the conditions

- By testing first for the highest price range, all the else-if statements become relevant

```
if (price >= 30) {
  complain
} else if (price >= 20) {
  think about it
} else if (price >= 10) {
  buy immediately
} else {
  buy two
}
```

- We could also first test for the lowest price

```
  if (price < 10) ... else if (price < 20) ... else if (price < 30) ... else ...
```

Semantic Error, Fix 2: have more detailed conditions

- We can also test for the price being in a range:

```
if ((price >= 10) && (price < 20)) {
  buy immediately
} else if ((price >= 20) && (price < 30)) {
  think about it
} else if (price >= 30) {
  complain
} else {
  buy two
}
```

Conditional Expressions

- Java also has conditional expressions:

```
final int X = ((x < 0) ? -x : x);
```

- the condition comes first, then the `?`
- next is the if/then expression, then `:`
- finally, the else expression
  - unlike in conditional statements, the else expression is required
- ?: is the conditional operator -- a two-part operator
- conditional statements are very common in code, conditional expressions are much less common

Another Example

```
if (taxStatus.equals (“single”)) {
  if (income <= 2400) {
    tax = income * 0.014;
  } else {
    tax = 34 + (income – 2400) * 0.032;
  }
} else {
  if (income < 4800) {
    tax = income * 0.014;
  } else {
    tax = 67 + (income – 4800) * 0.032;
  }
}
```

- inspired by section 3.4 in the book and the first two Hawaii tax rates
- some problems naturally work well with nested if statements!

Switch Statement

- When testing a single value for equality against one of a number of values of strings or basic types, you can use a switch statement

```
long powerOfTwo = Math.round(Math.pow(2, x));
switch (powerOfTwo) {
  case 1: exp = 0; break;
  case 2: exp = 1; break;
  case 4: exp = 2; break;
  case 8: exp = 3; break;
  default: exp = -1; break
}
```

- break is required at the end of every case
  - otherwise execution continues with the next case!
- if none of the cases match, the default statement (if any) is executed
- only one set of braces! The entire switch statement is considered a single block
- switch statements are not common in most code
  - but are found from time to time

Summary

- if statements provide conditionals in Java
  - if statements rely on the boolean value of the condition to decide what statements to execute
  - else if and else allow us to consider multiple cases
  - conditional expressions and switch statements can be used in specific cases
- Syntax tells us how to write programs
  - all programming languages have a formally-defined syntax
- Semantics tells us the meaning of syntactically correct programs
  - some programming languages have a formally-defined semantics