Building Java Programs

Chapter 4
Lecture 4-1: `if` and `if/else` Statements

Cumulative Sum Exercise

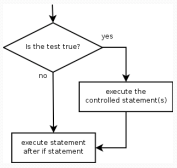
- Write a program that computes the average of any given number of doubles.
- Example output:

```
How many numbers would you like to average? 4
Enter number 1: 3
Enter number 2: 2
Enter number 3: 6.6
Enter number 4: 77
The average is: 22.15
```


The `if` statement

Executes a block of statements only if a test is true

```
if (test) {
   statement;
   ...
   statement;
}
```



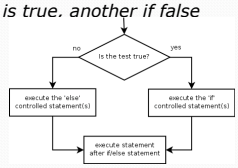
- Example:

```
double gpa = console.nextDouble();
if (gpa >= 2.0) {
    System.out.println("Application accepted.");
}
```


The `if/else` statement

Executes one block if a test is true, another if false

```
if (test) {
   statement(s);
} else {
   statement(s);
}
```



- Example:

```
double gpa = console.nextDouble();
if (gpa >= 2.0) {
    System.out.println("Welcome to Mars University!");
} else {
    System.out.println("Application denied.");
}
```


Relational expressions

- A **test** in an `if` is the same as in a `for` loop.

```
for (int i = 1; i <= 10; i++) { ...
if (i <= 10) { ...
```

  - These are `boolean` expressions, seen in Ch. 5.
- Tests use *relational operators*:

Operator	Meaning	Example	Value
==	equals	1 + 1 == 2	true
!=	does not equal	3.2 != 2.5	true
<	less than	10 < 5	false
>	greater than	10 > 5	true
<=	less than or equal to	126 <= 100	false
>=	greater than or equal to	5.0 >= 5.0	true

Logical operators: &&, ||, !

- Conditions can be combined using *logical operators*:

Operator	Description	Example	Result
&&	and	(2 == 3) && (-1 < 5)	false
||	or	(2 == 3) || (-1 < 5)	true
!	not	!(2 == 3)	true

- Relational operators cannot be "chained" as in algebra.

```
2 <= x <= 10          (assume that x is 15)
true   <= 10
  error!
```

  - Instead, combine multiple tests with `&&` or `||`

```
2 <= x && x <= 10     (assume that x is 15)
true   && false
  false
```

hi

Loops with if/else

- `if/else` statements can be used with loops or methods:

```
int evenSum = 0;
int oddSum = 0;

for (int i = 1; i <= 10; i++) {
    if (i % 2 == 0) {
        evenSum = evenSum + i;
    } else {
        oddSum = oddSum + i;
    }
}

System.out.println("Even sum: " + evenSum);
System.out.println("Odd sum: " + oddSum);
```


Nested `if/else`

reading: 4.1, 4.5

Sequential `if` bug

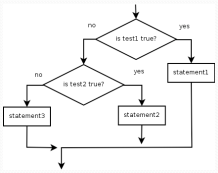
- What's wrong with the following code?

```
Scanner console = new Scanner(System.in);
System.out.print("What percentage did you earn? ");
int percent = console.nextInt();
if (percent >= 90) {
    System.out.println("You got an A!");
}
if (percent >= 80) {
    System.out.println("You got a B!");
}
if (percent >= 70) {
    System.out.println("You got a C!");
}
if (percent >= 60) {
    System.out.println("You got a D!");
}
else {
    System.out.println("You got an F!");
}
...
```

Nested `if/else`

Chooses between outcomes using many tests

```
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else {
    statement(s);
}
```



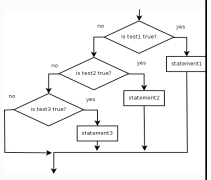
- Example:

```
if (number > 0) {
    System.out.println("Positive");
} else if (number < 0) {
    System.out.println("Negative");
} else {
    System.out.println("Zero");
}
```


Nested `if/else/if`

- If it ends with `else`, one code path **must** be taken.
- If it ends with `else if`, the program might not execute any path.

```
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else if (test) {
    statement(s);
}
```



- Example:

```
if (place == 1) {
    System.out.println("You win the gold medal!");
} else if (place == 2) {
    System.out.println("You win a silver medal!");
} else if (place == 3) {
    System.out.println("You earned a bronze medal.");
}
```

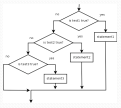

Structures

- Exactly 1 path: (mutually exclusive)

```
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else {
    statement(s);
}
```

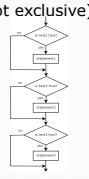
- 0 or 1 path:

```
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else if (test) {
    statement(s);
}
```



- 0, 1, or many paths: (independent tests, not exclusive)

```
if (test) {
    statement(s);
}
if (test) {
    statement(s);
}
if (test) {
    statement(s);
}
```



bye

Which nested `if/else`?

- **(1) if/if/if (2) nested if/else (3) nested if/else/if**
  - Reading the user's GPA and printing whether the student is on the dean's list (3.8 to 4.0) or honor roll (3.5 to 3.8).
    - **(3)** nested if / else if
  - Printing whether a number is even or odd.
    - **(N/A)** simple if / else
  - Printing whether a user is lower-class, middle-class, or upper-class based on their income.
    - **(2)** nested if / else if / else
  - Reading a number from the user and printing whether it is divisible by 2, 3, and/or 5.
    - **(1)** sequential if / if / if
  - Printing a grade of A, B, C, D, or F based on a percentage.
    - **(2)** nested if / else if / else if / else if / else

Factoring `if/else` code

- **factoring**: extracting common/redundant code
  - Factoring `if/else` code can reduce the size of `if/else` statements or eliminate the need for `if/else` altogether.
- Example:

```
if (a == 1) {
    x = 3;
} else if (a == 2) {
    x = 6;
    y++;
} else {  // a == 3
    x = 9;
}
```

```
x = 3 * a;
if (a == 2) {
    y++;
}
```


Code in need of factoring

```
if (money < 500) {
    System.out.println("You have, $" + money + " left.");
    System.out.print("Caution!  Bet carefully.");
    System.out.print("How much do you want to bet? ");
    bet = console.nextInt();
} else if (money < 1000) {
    System.out.println("You have, $" + money + " left.");
    System.out.print("Consider betting moderately.");
    System.out.print("How much do you want to bet? ");
    bet = console.nextInt();
} else {
    System.out.println("You have, $" + money + " left.");
    System.out.print("You may bet liberally.");
    System.out.print("How much do you want to bet? ");
    bet = console.nextInt();
}
```


Code after factoring

```
System.out.println("You have, $" + money + " left.");

if (money < 500) {
    System.out.print("Caution!  Bet carefully.");
} else if (money < 1000) {
    System.out.print("Consider betting moderately.");
} else {
    System.out.print("You may bet liberally.");
}

System.out.print("How much do you want to bet? ");
bet = console.nextInt();
```

- If the start of each branch is the same, move it *before* the if/else.
- If the end of each branch is the same, move it *after* the if/else.
- If similar but code exists in each branch, look for patterns.

The "dangling if" problem

- What can be improved about the following code?

```
if (x < 0) {
    System.out.println("x is negative");
} else if (x >= 0) {
    System.out.println("x is non-negative");
}
```

- The second if test is unnecessary and can be removed:

```
if (x < 0) {
    System.out.println("x is negative");
} else {
    System.out.println("x is non-negative");
}
```

  - This is also relevant in methods that use `if` with `return`...

`if/else` with `return`

- Methods can return different values using `if/else`:

```
// Returns the largest of the three given integers.
public static int max3(int a, int b, int c) {
    if (a >= b && a >= c) {
        return a;
    } else if (b >= c && b >= a) {
        return b;
    } else {
        return c;
    }
}
```

- Whichever path the code enters, it will return the appropriate value.
- Returning a value causes a method to immediately exit.
- All code paths must reach a return statement.
  - All paths must also return a value of the same type.

All paths must return

```
public static int max3(int a, int b, int c) {
    if (a >= b && a >= c) {
        return a;
    } else if (b >= c && b >= a) {
        return b;
    }
    // Error: not all paths return a value
}
```

- The following also does not compile:

```
public static int max3(int a, int b, int c) {
    if (a >= b && a >= c) {
        return a;
    } else if (b >= c && b >= a) {
        return b;
    } else if (c >= a && c >= b) {
        return c;
    }
}
```

- The compiler thinks `if/else/if` code might skip all paths.

if/else, return question

- Write a method `countFactors` that returns the number of factors of an integer.
  - `countFactors(24)` returns 8 because 1, 2, 3, 4, 6, 8, 12, and 24 are factors of 24.
- Write a program that prompts the user for a maximum integer and prints all prime numbers up to that max.

```
Maximum number? 52
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
15 primes (28.84%)
```


if/else, return answer 1

```
// Prompts for a maximum number and prints each prime up to that maximum.
import java.util.*;

public class Primes {
    public static void main(String[] args) {
        // read max from user
        Scanner console = new Scanner(System.in);
        System.out.print("Maximum number? ");
        int max = console.nextInt();
        printPrimes(max);
    }

    // Prints all prime numbers up to the given maximum.
    public static void printPrimes(int max) {
        int primes = 0;
        for (int i = 2; i <= max; i++) {
            if (countFactors(i) == 2) {       // i is prime
                System.out.print(i + " ");
                primes++;
            }
        }
        System.out.println();
        double percent = 100.0 * primes / max;
        System.out.printf("%d primes (%.2f%%)\n", primes, percent);
    }
```


if/else, return answer 2

```
...

// Returns how many factors the given number has.
public static int countFactors(int number) {
    int count = 0;
    for (int i = 1; i <= number; i++) {
        if (number % i == 0) {
            count++;  // i is a factor of number
        }
    }
    return count;
}
}
```