Nested if statements

- if-else statements can reside within other if-else statements
  
Example (pseudocode)

Get interest rate from the user.
If interest rate is less than zero or greater than 100,
  print an error message.
Else,
  if interest rate is greater than 1,
    divide interest rate by 100.0;
  calculate the new balance.
Print out the account's balance.

Interest rate (in Java code)

```java
String rateStr = JOptionPane.showInputDialog(null, "Enter interest rate: ");
double rate = Double.parseDouble(rateStr);
if (rate > 100 || rate < 0) {
    System.out.println("Invalid interest rate entered.");
}
else {
    if (rate > 1) {
        rate = rate/100.0;
    }
    ...}
```
Comparing objects

- When we compare two numbers, we use the equality (==) operator
  - if (a == b)
  - equality for primitive types means “the values are equivalent”
- Equality for objects means “at the same location in memory”
  - cannot use the == operator

Example: comparing Strings

- The String class has an `equals` method and an `equalsIgnoreCase` method
- Example:
  ```java
  String s1 = "this is a string";
  String s2 = "This is a STRING";
  if (s1.equals(s2)) {
    System.out.println("match");
  }
  else if (s1.equalsIgnoreCase(s2)) {
    System.out.println("match, ignoring case");
  }
  else {
    System.out.println("no match");
  }
  ```

Comparing two objects

- If we want to compare two objects (Canvases, Accounts, etc), the class has to contain an `equals` method
  - this method defines what constitutes equality for this class
- Example: Compare 2 Canvas objects
  - “2 Canvas objects are equal if they have the same title and the same dimensions.”

switch statement

- Sometimes using `if` statements can be cumbersome
  - conditions involve integer or character values, for example
Example

```java
String numStr = JOptionPane.showInputDialog(null, "Enter your choice (1-5): ");
int num = Integer.parseInt(numStr);
if (num == 1) {
    System.out.println("Do a headstand.");
} else if (num == 2) {
    System.out.println("Pat your head and rub your tummy.");
} else if (num == 3) {
    System.out.println("Do 3 jumping jacks.");
} else if (num == 4) {
    System.out.println("Recite a limerick.");
} else if (num == 5) {
    System.out.println("Count backwards from 100 by 7's.");
} else {
    System.out.println("You cannot follow directions, butthead!");
}
```

**switch statement**

- Replacement for if statements when the boolean expression involves testing an integer (int) or a character (char)
  - also: byte, short
- Examples of integer tests:
  - num (from previous example)
  - 25 % 3
  - a / 3 (if a is an integer!)

Example (revised)

```java
String numStr = JOptionPane.showInputDialog(null, "Enter your choice (1-5): ");
int num = Integer.parseInt(numStr);
switch (num) {
    case 1:  System.out.println("Do a headstand.");
    break;
    case 2:  System.out.println("Pat your head and rub your tummy.");
    break;
    case 3:  System.out.println("Do 3 jumping jacks.");
    break;
    case 4:  System.out.println("Recite a limerick.");
    break;
    case 5:  System.out.println("Count backwards from 100 by 7's.");
    break;
    default:  System.out.println("You cannot follow directions, butthead!");
}
```

Format of a switch statement

```java
switch {<expression>} {
    case <constant 1>:  <statements>
    break;
    case <constant 2>:  <statements>
    break;
    ...
    case <constant n>:  <statements>
    break;
    default:  <statements>
}
```
Notes

- The constant values must be constants
  - integers/chars/bytes or defined constants of these types
- Cases can be in any order
  - typically, default goes last
- break is used to make sure we only execute one of the statement blocks
- default clause is optional
  - handles unexpected input or program behavior
  - good programming practice

Another example

```java
String choice = JOptionPane.showInputDialog(null, "Enter a number between 1 and 7");
int num = Integer.parseInt(choice);
switch (num) {
    case 1:
    case 2:
    case 3:
    case 5:
        System.out.println("Prime number.");
        break;
    case 4:
        System.out.println("Factors are 1, 2, and 4.");
        break;
    case 6:
        System.out.println("Factors are 1, 2, 3, and 6.");
        break;
    default:
        System.out.println("Invalid number entered.");
}
```

Control statement gotchas

- Not using breaks in your switch statements
- Using values that are not integers, chars, or bytes when using switch
- Making your if statements too complex
  - simplify your logic!
- Using && when you mean to use || and vice versa
- Using = (assignment) when you mean to use == (equality)
- Using == (equality) to compare objects
  - Strings: equals() or equalsIgnoreCase() should be used instead

Repetition statements

- Control statements allow us to execute certain blocks of code when certain conditions are met
- Repetition statements allow us to execute certain blocks of code more than once
  - until some condition is met
  - a set number of times
Uses of repetition

- Game: allow the user to keep playing until he/she decides to quit
- Keep prompting the user for input until the user enters valid input
- Mathematical calculations
  - average
  - sum
- And many more!

Repetition statements

- **while**
- **do-while**
- **for**

**while statement**

- “While some condition is true, do the following...”
- **pre-test loop**: test the condition first, then execute the loop
- Example:

```java
int sum = 0, count = 1, maxSum = 500;
while (sum < maxSum) {
    sum = sum + count;
    count = count + 2;
}
System.out.println("Sum = "+sum);
```

Another example: valid user input

```java
String rateStr = JOptionPane.showInputDialog(null,"Enter an interest rate: ");
rate = Double.parseDouble(rateStr);
while (rate < 0 || rate > 100) {
    rateStr = JOptionPane.showInputDialog(null,"You entered an invalid rate. Please enter a rate between 0 and 100:");
    rate = Double.parseDouble(rateStr);
}
...```
Another example

```java
String choice = JOptionPane.showInputDialog(null,"Enter a positive number (or any negative number to quit): ");
int num = Integer.parseInt(choice);
int sum = 0, count = 0;
while (num >= 0) {
    count++;
    sum = sum + num;
    choice = JOptionPane.showInputDialog(null,"Enter a positive number (or any negative number to quit): ");
    num = Integer.parseInt(choice);
}
double average = ((double)sum)/count;
System.out.println("The average of the numbers is "+average);
```

Format of a while statement

```java
while (<condition>) {
    <statement>
}
```

- `<condition>` is a boolean statement
- `<condition>` changes each time through the loop
- As long as `<condition>` is true, the loop will execute
- `<statement>`: loop body

Types of loops

- **count-controlled**: the while loop executes a certain number of times
  - example:
    ```java
    count = 0;
    while (count < 100) { count++; }
    ```
- **sentinel-controlled**: the while loop executes until a variable becomes a certain value
  - the interest rate and averaging examples are this type
  - the variable of interest is the **sentinel**
    - interest rate: rate between 0 and 100
    - averaging: negative number entered by the user

Gotchas

- Infinite loop
- Overflow
- Using real numbers
- Off-by-one
- Other logic mistakes
Infinite loop

- Probably the most common error
- Occurs when <condition> is never met
  - program has no stopping point
  - loops forever
- Example:
  ```java
  int sum = 0;
  while (sum < 100) {
    count = 0;
    sum = sum + count;
  }
  ```

Other examples of infinite loops

```java
int prod = 0;
while (prod < 100) {
  prod = prod * 3;
}

int count = 0;
while (count != 25) {
  count = count + 3;
}
```