Flow of Control: Selection

Reading: Appendix A

- Forming conditions.
- if statements
- Comparing floating-point numbers.
- Comparing Objects.
- The switch statement.
- The Conditional Operator.

Flow of Control:
- Programs can control the order in which their instructions are executed.
  - A major feature of software and why it is useful!
- Four types of flow:
  - Sequential:
    - Execute instructions in the order listed in the code. (This is mostly what you have been doing.)
  - Method calls:
    - Transfer flow control to the code inside the method; i.e.
      ```java
      String zap = "Once upon a time in a galaxy far, far away...";
      String word = zap.substring(22, 28);
      System.out.println("word contains: " + word);
      ```
    - Control returns back to the point of the call. Some calls also return a value.
  - Selection: (now!)
    - Which set of instructions are executed depends on the data.
  - Looping: (next)
    - Repeat a set of instructions, changing some of the data each time through the set of instructions.
Comparison Operators:

Reading: Appendix A

- We need ways to compare values.
  - Will then use these techniques to make choices about which instructions to execute.
- All of these techniques have one goal: to compute a boolean value (true or false).
- Equality, or the lack thereof:
  - There are two “equality” operators:
    - == compares two values and returns true if one is equal to the other.
    - != compares two values and returns true if one is not equal to the other.
- Example:

```java
boolean answer;
Scanner inputScan = new Scanner(System.in);
int aNum, bNum;
System.out.print("Enter two integers: ");
aNum = inputScan.nextInt();
bNum = inputScan.nextInt();

answer = (aNum == bNum);
System.out.println("aNum == bNum returns " + answer);
answer = (aNum != bNum);
System.out.println("aNum != bNum returns " + answer);
```

Comparison Operators (continued):

Equality Operators (continued):

- Do not confuse these two:
  - == for comparing two values.
  - = for assigning a value to a variable.
- Examples:

```java
int aNum = 42, bNum = 96;
boolean answer;
answer = (aNum = bNum);
```

Gives the error: “Incompatible types”.

Why?

```java
int zap;
zap = (aNum = bNum);
zap = aNum = bNum;
System.out.println("zap is " + zap);
```

Compiles without error.

Runs without error.

What is printed?
Comparison Operators (continued):

**Relational Operators**

- Result is boolean
  - `true` or `false`

<table>
<thead>
<tr>
<th>Relational Operators</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;</code></td>
<td>binary two operands</td>
<td>is less than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>binary two operands</td>
<td>is less than or equal to</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>binary two operands</td>
<td>is greater than</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>binary two operands</td>
<td>is greater than or equal to</td>
</tr>
</tbody>
</table>

Example:

```java
import java.util.Scanner;
public class Relational
{
    public static void main(String[] args)
    {
        boolean answer;
        Scanner inputScan = new Scanner( System.in );
        int aNum, bNum;

        System.out.print("Enter two integers: ");
        aNum = inputScan.nextInt();
        bNum = inputScan.nextInt();

        System.out.println("Result of (aNum < bNum) is " + (aNum < bNum));
        System.out.println("Result of (aNum <= bNum) is " + (aNum <= bNum));
        System.out.println("Result of (aNum > bNum) is " + (aNum > bNum));
        System.out.println("Result of (aNum >= bNum) is " + (aNum >= bNum));
    } // end of method main
} // end of class Relational
```
**Comparison Operators (continued):**

**Relational and Equality Operators (continued):**
- The two-character operators (==, !=, <=, >=):
  - **No whitespace** between the two characters.
  - The order of the two characters does matter (except for ==, of course :-):  
    
    ```java
    boolean result = (aNum => bNum);
    ```
  - Will not compile, gives the error: “illegal start of expression”

**Logical Operators:**
- The ! operator:
  - Performs a NOT operation.
  - Has only **one** operand.
  - Returns **false** if the operand is **true**. Returns **true** if the operand is **false**.
  - Example:
    ```java
    float income;
    income = inputScan.nextFloat();

    boolean rich, poor;
    rich = (income > 1e6);
    poor = !(income > 1e6);
    poor = !rich;
    ```
  - Both of these give the same result.
Comparison Operators (continued):

Logical Operators (continued):

- The `&&` operator:
  - Performs an AND operation.
  - Has two operands.
  - Returns `true` if both operands are `true`; otherwise, `false` is returned.
  - Example:
    ```java
    int age;
    age = inputScan.nextInt();
    boolean teenAge;
    teenAge = (age >= 13) && (age <= 19);
    System.out.println("teenAge is "+ teenAge);
    teenAge = (age > 12) && (age < 20);
    ```

- The `||` operator:
  - Performs an OR operation.
  - Has two operands.
  - Returns `false` if both operands are `false`; otherwise, `true` is returned.
  - Example:
    ```java
    int myAge;
    myAge = inputScan.nextInt();
    int sisterAge = 34, brotherAge = 39;
    boolean notYoungest;
    notYoungest = (myAge > brotherAge) || (myAge > sisterAge);
    System.out.println("notYoungest is "+ notYoungest);
    ```
Comparison Operators (continued):
Logical Operators (continued):

• Summary in the form of a *truth table*:

|   |   | !a | a && b | a || b |
|---|---|----|-------|-------|
| true | true | false | true | true |
| true | false | false | false | true |
| false | true | true | false | true |
| false | false | true | false | false |

Comparison Operators (continued):
Logical Operators (continued):

• Suppose we want to know if a value falls within a range.
  • Examples: The teenager age range of slide #9, or the temperature range for liquid water.
  • It is tempting to write this as:
    ```java
    float waterTemp;
    waterTemp = inputScan.nextFloat();
    boolean liquidWater;
    liquidWater = 0.0F <= waterTemp <= 100.0F;
    ```
  • The last line produces the error: *"operator <= cannot be applied to boolean, float"
  • What is the right way to make this range comparison?
    ```java
    liquidWater = (0.0F <= waterTemp) && (waterTemp <= 100.0F);
    ```

• How would we calculate the non-liquid case?
  ```java
  boolean iceOrSteam;
  iceOrSteam = !( (0.0F <= waterTemp) && (waterTemp <= 100.0F) );
  iceOrSteam = !liquidWater;
  ```
Comparison Operators (continued):

Logical Operators (continued):

- DeMorgan’s Laws:
  1. NOT(A AND B) = (NOT A) OR (NOT B)
  2. NOT(A OR B) = (NOT A) AND (NOT B)

- Consider the teenage example from slide #9:
  
  ```
  notTeenAge = !(age >= 13) || !(age <= 19);
  ```

- Ask the question: is the age not a teenager?
  
  ```
  notTeenAge = !( (age >= 13) && (age <= 19) );
  ```

- Apply the first of DeMorgan’s Laws:
  
  ```
  notTeenAge = !((age >= 13) || (age <= 19));
  ```

Comparison Operators (continued):

Logical Operators (continued):

- From the previous slide:
  
  ```
  notTeenAge = !(age >= 13) || !(age <= 19);
  ```

- This can be simplified:
  
  - Consider: !(age >= 13)
  
  - How can this be written without the NOT?
    
    ```
    age < 13
    ```
  
  - Do the same for !(age <= 19)
    
    ```
    age > 19
    ```

- The final result:
  
  ```
  notTeenAge = (age < 13) || (age > 19);
  ```
Comparison Operators (continued):

Logical Operators (continued):

• One more look at this. The following statements are all equivalent:

  teenAge = (age >= 13) && (age <= 19);
  teenAge = !( (age < 13) || (age > 19) );
  teenAge = !(age < 13) && !(age > 19);

if Statements:
Simple if:

Reading: Appendix A

• Used when the program should perform an operation for one set of data, but not for all other data.

• Syntax:

  if ( condition-goes-here ) {
      // true block
      // code to execute when condition is true
  }
  // code here that executes after the if statement
if Statements (continued):

Simple if (continued):

- Example:
  ```java
  float battingAverage, salary = 500000.0F;
  battingAverage = inputScan.nextFloat();
  if (battingAverage >= 0.300F) {
    System.out.println("Great hitter, pay him more!");
    salary = salary + 1e6F;
  }
  System.out.println("Salary is "+ salary);
  ```
**if Statements** (continued):

**Simple if** (continued):

- Be careful:
  - Do not put a semi-colon after the condition.
  - What happens if the code is?

```java
float battingAverage, salary = 500000.0F;
battingAverage = inputScan.nextFloat();
if (battingAverage >= 0.300F);    // The mistake
{
    System.out.println("Great hitter, pay him more!");
    salary = salary + 1e6F;
}
System.out.println("Salary is " + salary);
```

**if Statements** (continued):

**Simple if** (continued):

- When there is only one statement inside the if, the { }'s can be left off.
- Example:

```java
float battingAverage, salary = 500000.0F;
battingAverage = inputScan.nextFloat();
if (battingAverage < 0.100F )
    System.out.println("Send him down to the minor leagues.");
System.out.println("Salary is " + salary);
System.out.println("done");
```
if Statements (continued):

Simple if (continued):

- Indentation of the code inside an if statement is very important for us humans.
- But, the Java compiler ignores the indentation.
- What happens here?

```java
float battingAverage, salary = 500000.0F;
battingAverage = inputScan.nextFloat();
if (battingAverage >= 0.300F)
    System.out.println("Great hitter, pay him more!");
salary = salary + 1e6F;
System.out.println("Salary is " + salary);
```

When does this line get executed?

if Statements (continued):

if/else:

Reading: Section 5.3

- Allows separate actions for both true and false results.
- The else clause contains the code for false.
- Example:

```java
import java.util.Scanner;
public class WaterSport {
    public static void main(String[] args) {
        Scanner inputScan = new Scanner(System.in);
        int waterTemp;
        System.out.print("Enter the water temperature: ");
        waterTemp = inputScan.nextInt();
        if (waterTemp <= 0) {
            System.out.println("Ice skating time!");
        } else {
            System.out.println("Go for a swim!");
        }
        System.out.println("Have a good time!");
    }
}
```

![Diagram](image-url)

waterTemp = inputScan.nextInt();

true

waterTemp <= 0

false

Print Ice Skating msg

Print Swimming msg

Print msg;

Rest of program

Have a good time!
if Statements (continued):

if/else (continued):

• Indent the statement(s) inside the else clause.
  • Use a tab, or at least 3 spaces.
• When more than one statement is in the else clause, you must use { }’s.
• Example:

```java
if ( waterTemp <= 0 )
    System.out.println("Ice skating time!");
else {
    System.out.println("Go for a swim!");
    System.out.println("Might need a wet suit...");
}
System.out.println("Have a good time!");
```

The { can be on the next line.
Align it with the e in else.

if Statements (continued):

Block Scope:

Reading: Appendix A

• The scope of a variable is the region of code within a program where the variable can be referenced (or used).
• Scope is determined by the block of code containing the variable declaration.
• Code blocks:
  • The main method is a code block.
  • Code in the true clause of an if statement is a block.
  • Code in the false clause of an if statement is a block.
  • Code inside { }’s is a block.
**if Statements (continued):**

**Block Scope (continued):**

- Scope example:

```java
public static void main(String[] args) {
    Scanner inputScan = new Scanner(System.in);
    int waterTemp;
    System.out.print("Enter the water temperature: ");
    waterTemp = inputScan.nextInt();
    if ( waterTemp <= 0 )
        System.out.println("Ice skating time!");
    else {
        System.out.println("Go for a swim!");
        System.out.println("Might need a wet suit...");
    }
    System.out.println("Have a good time!");
} // end of method main
```

**if Statements (continued):**

**Block Scope (continued):**

- Scope example (continued)

```java
public static void main(String[] args) {
    Scanner inputScan = new Scanner(System.in);
    int waterTemp;
    System.out.print("Enter the water temperature: ");
    waterTemp = inputScan.nextInt();
    if ( waterTemp <= 0 )
        System.out.println("Ice skating time!");
    else {
        System.out.println("Go for a swim!");
        System.out.println("Might need a wet suit...");
    }
    System.out.println("Have a good time!");
} // end of method main
```
if Statements (continued):

Block Scope (continued):

• This code works. There are two distinct variables, both named area. Each exists in a different scope.

```java
import java.util.Scanner;
public class Area
{
    public static void main(String[] args)
    {
        Scanner inputScan = new Scanner(System.in);
        int width, height;
        System.out.print("Enter the width and height: ");
        width = inputScan.nextInt();
        height = inputScan.nextInt();
        if (width == height)
        {
            int area = width * width;
            System.out.println("Area of square = " + area);
        }
        else
        {
            int area = width * height;
            System.out.println("Area of rectangle = " + area);
        }
    } // end of method main
} // end of class Area
```

if Statements (continued):

Block Scope (continued):

• This code does not work.
• There are (still) two distinct variables, both named area. Each exists in a different scope.
• The reference to area after the if is invalid, since it lies outside the scope of both area variables.

```java
import java.util.Scanner;
public class Area
{
    public static void main(String[] args)
    {
        Scanner inputScan = new Scanner(System.in);
        int width, height;
        System.out.print("Enter the width and height: ");
        width = inputScan.nextInt();
        height = inputScan.nextInt();
        if (width == height)
        {
            int area = width * width;
            System.out.println("Area of square = " + area);
        }
        else
        {
            int area = width * height;
            System.out.println("Area of rectangle = " + area);
        }
        int inchesArea;
        inchesArea = area * 144; // 144 sq inches in 1 sq foot
        System.out.println("Area in square inches = " + inchesArea);
    } // end of method main
}
```
if Statements (continued):

if/else:

Reading: Section 5.4

• Used when data falls into multiple mutually exclusive categories.
• Want program to do different things for each category.
• Water example:
  • Still need Ice Skating.
  • Different swimming temperatures:
    • Cold — wet suit needed.
    • Medium — normal swimming.
    • Hot — hot tub, need warning.

if Statements (continued):

if/else if (continued):

Scanner inputScan = new Scanner( System.in );
int waterTemp;
System.out.print("Enter the water temperature: ");
waterTemp = inputScan.nextInt();

if ( waterTemp <= 0 )
    System.out.println("Ice skating time!");
else if ( waterTemp <= 18 ) {
    System.out.println("Go for a swim!");
    System.out.println("Bring a wet suit!");
}
else if ( waterTemp <= 37 ) {
    System.out.println("Go for a swim!");
}
else {
    System.out.println("Hot tub time!");
    System.out.println("Don't stay in too long!");
}

System.out.println("Have a good time!");
if Statements (continued):

Nested if Statements:

Reading: Section 5.5

- if statements can be written as part of the true or false block of another if statement.
- Typically, nest if statements when more information is required beyond the results of the first if condition.

```java
waterTemp = inputScan.nextInt();
if (waterTemp <= 0)
    true
else
    false
    if (waterTemp <= -10)
        Print Ice Skating msg
    else if (waterTemp <= 18) {
        System.out.println("Ice skating time!");
        System.out.println("Bring a wet suit!");
        System.out.println("Go for a swim!");
    }
    else if (waterTemp <= 37) {
        System.out.println("Go for a swim!");
    }
    else {
        System.out.println("Hot tub time!");
        System.out.println("Don't stay in too long!");
    }
    System.out.println("Have a good time!");
```

if Statements (continued):

Nested if Statements (continued):

- This code does not work.

```java
System.out.print("Enter the water temperature: ");
waterTemp = inputScan.nextInt();
if (waterTemp <= 0)
    if (waterTemp <= -10 )
        System.out.println("Ice skating time!");
else if (waterTemp <= 18 ) {
    System.out.println("Ice skating time!");
    System.out.println("Bring a wet suit!");
    System.out.println("Go for a swim!");
} else if (waterTemp <= 37 ) {
    System.out.println("Go for a swim!");
} else {
    System.out.println("Hot tub time!");
    System.out.println("Don't stay in too long!");
} System.out.println("Have a good time!");
```
if Statements (continued):

Nested if Statements (continued):

• When looking at an else, how does the compiler know what if it belongs to?
  - Recall: The compiler ignores indentation!
  - Each else is associated with the most recent if that does not already have an else.

```java
System.out.print("Enter the water temperature: ");
waterTemp = inputScan.nextInt();
if (waterTemp <= 0)
    if (waterTemp <= -10)
        System.out.println("Ice skating time!");
    else
        if (waterTemp <= 18){
            System.out.println("Go for a swim!");
            System.out.println("Bring a wet suit!");
        }
    else if (waterTemp <= 37) {
        System.out.println("Go for a swim!");
    }
else {
    System.out.println("Hot tub time!");
}
```

if Statements (continued):

Nested if Statements (continued):

• Can fix this by adding an else clause for the ice skating if statement.
  - This works.
  - But, can be awkward when we did not have anything to put in the else.

```java
System.out.print("Enter the water temperature: ");
waterTemp = inputScan.nextInt();
if (waterTemp <= 0)
    if (waterTemp <= -10)
        System.out.println("Ice skating time!");
    else
        System.out.println("...something goes here..."i);    
    else if (waterTemp <= 18) {
        System.out.println("Go for a swim!");
        System.out.println("Bring a wet suit!");
    }
    else if (waterTemp <= 37) {
        System.out.println("Go for a swim!");
    }
```
if Statements (continued):

Nested if Statements (continued):

- Use { }’s to surround the if statement that does not have an else.
- When { }’s are present, the if statement has to fit entirely inside the { }’s.
- This tells the compiler that the if statement does not have an else clause.

```java
System.out.print("Enter the water temperature: ");
waterTemp = inputScan.nextInt();
if ( waterTemp <= 0 ) {
    if ( waterTemp <= -10 )
        System.out.println("Ice skating time!");
}
else if ( waterTemp <= 18 ) {
    System.out.println("Go for a swim!");
    System.out.println("Bring a wet suit!");
}
else if ( waterTemp <= 37 ) {
    System.out.println("Go for a swim!");
}
else {
```

if Statements (continued):

Testing Techniques:

Reading: Appendix A

- Execution Path Testing:
  - Develop a test plan that includes:
    - Running the program multiple times with data values that cause all true blocks to be executed,
    - AND all false blocks to be executed.
  - Check results against the program specifications.

- Black Box Testing:
  - Treat the program like a black box:
    - Assume you do not know how the code is written.
    - Develop test data on program specifications.
if Statements (continued):

Testing Techniques (continued):

- Consider the flowchart.
- What values of \((x\text{ray}, z\text{ap})\) are needed to test each path?

```
x\text{ray} = 
y\text{oke} =
```

```
\begin{verbatim}
if x\text{ray} > 7 then print "medium" else
if x\text{ray} < -7 then print "heavy" else
if z\text{ap} > 10 then print "now" else
if z\text{ap} < 20 then print "light" else
rest of program
\end{verbatim}
```

Comparing Floats and Doubles:

Reading: Appendix A

- Can compare floats and doubles using \(<\), \(\leq\), \(>\), \(\geq\), \(==\), and \(!=\).
- But, equality is a problem. Consider

```
double z\text{ap}, w\text{obble};
w\text{obble} = 1.1;
z\text{ap} = 0.1;
z\text{ap} = z\text{ap} + 0.1; // repeat this line 10 times
```

- The two should now both be \(1.1\), but \(z\text{ap}\) is not!

```
w\text{obble} = 1.1
z\text{ap} = 1.0999999999999999
```

- An equality, or inequality, test will yield an unexpected result!
  - A less-than test or a greater-than test will also lead to an incorrect result.
if Statements (continued):

Comparing Floats and Doubles (continued):

• Establish a “close enough” criteria.
  • How close together do the values need to be to be considered “equal”?
  • Example: choose 0.001 as being “close enough”.
  • Find the difference between zap and wobble. Is this difference less than 0.001?
    
    ```java
    double zap, wobble;
    wobble = 1.1;
    zap = 0.1;
    zap = zap + 0.1;    // repeat this line 10 times
    if ( Math.abs(zap - wobble) < 0.001 )
        System.out.println("zap and wobble are equal (close enough)");
    else
        System.out.println("zap and wobble are not equal");
    ```

if Statements (continued):

switch:

• An if/else if statement can (sometimes) be replaced by a switch statement.

• Requirements:
  • Must be comparing the value of a char, byte, short, or int.
  • Note: cannot be a long, float, double, String, or anything else!

    ```java
    switch ( /* char, byte, short, or int expression goes here */ ) { 
        case constant1:
            // statement(s);
            break; // optional
        case constant2:
            // statement(s);
            break; // optional
        ...
        default: // optional (but generally a very good idea!)
            statements(s);
    } // switch ends here
    ```
**if Statements** (continued):

**switch** (continued):

- The *expression* is evaluated, then its value is compared to the *case* constants in order.
- When a match is found, the statements under that *case* constant are executed in sequence until:
  - a *break* statement is reached, OR
  - the end of the *switch* block is reached.
- Example:
  - A program that reads a year from the keyboard.
  - Determines if the year is:
    - A Presidential election year.
    - A House of Representatives year.
    - A year with no federal election.

```java
Scanner inputScan = new Scanner(System.in);
short year;
System.out.print("Enter the year: ");
year = inputScan.nextShort();

switch (year % 4) {
    case 0: // if ( year % 4 == 0 )
        System.out.println("Elect a President");
        System.out.println("Elect members of the US House");
        break;

    case 1: // if ( year % 4 == 1 )
        System.out.println("No federal election");
        break;

    case 2:
        System.out.println("Elect members of the US House");
        break;

    case 3:
        System.out.println("No federal election");
        break;
}
```
**if Statements** (continued):

**switch** (continued):

- The non-election years can be combined:
  ```java
  case 1:
  case 3:
      System.out.println("No federal election");
  break;
  ```
  - It is not a requirement that the `case`'s be listed in order.
  - The `case`'s are checked in the order in which they are listed.
- When can `break` statements be left out? What happens if they are?
  - How can the two election years be combined?

**if Statements** (continued):

**switch** (continued):

- A simplistic class standing example:
  - Students “advance” from freshman to sophomore, etc. every 30 credit hours.
    - Freshmen can only take freshman classes.
    - Sophomores can take sophomore and freshman classes.
    - Etc.
  - Once they reach 120 credit hours, they “advance” to graduate status.
    - From 120 to 134 credit hours, they can take 500-level courses.
    - From 135 to 150 credit hours, they can take 500- and 600-level courses.
  - Number of credit hours beyond 150 (and below 0) are not allowed.
if Statements (continued):

switch (continued) — ClassStanding example continued.

- Can use the “fall through” feature of switch.
- What goes on each case expression?

```java
switch (creditHours / 30) {
    case 3:
        System.out.println("Can take Senior courses");
    case 2:
        System.out.println("Can take Junior courses");
    case 1:
        System.out.println("Can take Sophomore courses");
    case 0:
        System.out.println("Can take Freshman courses");
        break;
}
```

if Statements (continued):

switch (continued) — ClassStanding example continued.

- Can use integer division to break the credit hours into the 30-unit categories.
- Can use the “fall through” feature of switch.

```java
switch (creditHours / 30) {
    case 3:
        System.out.println("Can take Senior courses");
    case 2:
        System.out.println("Can take Junior courses");
    case 1:
        System.out.println("Can take Sophomore courses");
    case 0:
        System.out.println("Can take Freshman courses");
        break;
}
```
if Statements (continued):

switch (continued) — ClassStanding example continued.

• Graduate students are in the range 120 to 149.
  • This group can be found by integer division by 30.
  • But, there are two sub-categories.

```
switch (creditHours / 30) {
    case 4:
        if ( creditHours >= 135 ) {
            System.out.println("Can take 600-level courses");
            System.out.println("Can take 500-level courses");
        } else
            System.out.println("Can take 500-level courses");
            break;
    case 3:
        System.out.println("Can take Senior courses");
    case 2:
        System.out.println("Can take Junior courses");
    case 1:
        System.out.println("Can take Sophomore courses");
    case 0:
        System.out.println("Can take Freshman courses");
            break;
    default:
        System.out.println("The credit hours must be in the range 0 to 149");
}
```

if Statements (continued):

switch (continued) — ClassStanding example continued.

• How to handle creditHours that are negative or too large?

```
switch (creditHours / 30) {
    case 4:
        if ( creditHours >= 135 ) {
            System.out.println("Can take 600-level courses");
            System.out.println("Can take 500-level courses");
        } else
            System.out.println("Can take 500-level courses");
            break;
    case 3:
        System.out.println("Can take Senior courses");
    case 2:
        System.out.println("Can take Junior courses");
    case 1:
        System.out.println("Can take Sophomore courses");
    case 0:
        System.out.println("Can take Freshman courses");
            break;
    default:
        System.out.println("The credit hours must be in the range 0 to 149");
}
```
**if Statements** (continued):

**switch** (continued):

- Where should the **default** clause go inside the **switch** statement?
  - The **default** clause can go anywhere.
  - If placed somewhere other than the bottom, a **break** is (almost) always required.
  - Strongly suggest: put it in the same place every time.
    - Most common locations are at the **top** or at the **bottom**.
    - Not in the middle!
  - Is a **default** clause required?
    - No.
  - Should the **default** clause be present anyway?
    - Yes!
    - Having a **default** clause even when I was **absolutely certain** the **default** clause could never be executed has saved me countless hours of trouble-shooting!

**if Statements** (continued):

**Comparing Objects**:

Reading: Appendix A

- Consider the following code fragment. What is printed?

  ```java
  String zebra, xray;
  zebra = new String("Hello");
  xray = new String("Hello");
  if ( zebra == xray )
    System.out.println("zebra == xray is true");
  else
    System.out.println("zebra == xray is false");
  ```

- Why?
if Statements (continued):

Comparing Objects (continued):

- *zebra* is a reference to a String. *xray* is a reference to a (different!!) String.
- The contents of the two String's are the same, but the value of *zebra* and the value of *xray* are different.
- The value of *zebra* is the reference to its String; the value of *xray* is the reference to its String.
- The equality (==, !=) and relational (<, <=, >, >=) operators can only compare values, not contents.

```java
String zebra, xray;
zebra = new String("Hello");
xray = new String("Hello");
if ( zebra == xray )
    System.out.println("zebra == xray is true");
else
    System.out.println("zebra == xray is false");
```

if Statements (continued):

Comparing Objects (continued):

- Consider the following code fragment (first part is the same as the previous two slides). What is printed?

```java
String zebra, xray;
zebra = new String("Hello");
xray = new String("Hello");
if ( zebra == xray )
    System.out.println("zebra == xray is true");
else
    System.out.println("zebra == xray is false");
String oscar;
oscar = xray;
if ( oscar == xray )
    System.out.println("oscar == xray is true");
else
    System.out.println("oscar == xray is false");
```
if Statements (continued):

Comparing Objects (continued):

• What is different?

```java
String zebra, xray;
zebra = new String("Hello");
xray = new String("Hello");

if ( zebra == xray )
    System.out.println("zebra == xray is true");
else
    System.out.println("zebra == xray is false");

String oscar;
oscar = xray;
if ( oscar == xray )
    System.out.println("oscar == xray is true");
else
    System.out.println("oscar == xray is false");

oscar = new String("Tomorrow");
```

if Statements (continued):

Comparing Objects (continued):

• What is different?

```java
int alpha, bravo;
alpa = 42;
bravo = 42;
if ( alpha == bravo )
```

```java
zebra
String "Hello"
```

```java
xray
String "Hello"
```

```java
oscar
```

```java
String "Hello"
```
if Statements (continued):

Comparing Strings:

- To compare the contents of two String's, we need to be able to “look inside” each String.
- The String class (and most classes) has an equals() method that does this.
- Syntax: `someString.equals( someOtherString )`

```java
String zebra, xray;
zebra = new String("Hello");
xray = new String("Hello");

if ( zebra.equals(xray) )
    System.out.println("the contents of zebra and xray are the same");
else
    System.out.println("the contents of zebra and xray are NOT the same");

// This will also work:
if ( xray.equals(zebra) )
    System.out.println("the contents of zebra and xray are the same");
else
    System.out.println("the contents of zebra and xray are NOT the same");
```

if Statements (continued):

Comparing Strings (continued):

- Example:

```java
String zebra, xray;
zebra = new String("Hello");
xray = new String("Hello");

if ( zebra.equals(xray) )
    System.out.println("the contents of zebra and xray are the same");
else
    System.out.println("the contents of zebra and xray are NOT the same");
```

- Another example:

```java
String zebra, xray;
zebra = new String("Hello");

if ( zebra.equals(zebra) )
    System.out.println("the contents of zebra and zebra are the same");
else
    System.out.println("the contents of zebra and zebra are NOT the same");
```
if Statements (continued):

Comparing Strings (continued):
• The String class provides additional methods to compare the contents of two String’s.
  • The equalsIgnoreCase() method will “match” strings when the only differences are in upper vs. lower case.
  • Example:
    ```java
    String zebra, xray;
    zebra = new String("HeLLO");
    xray = new String("Hello");

    if ( zebra.equalsIgnoreCase(xray) )
        System.out.println("the contents of zebra and xray are the same");
    else
        System.out.println("the contents of zebra and xray are NOT the same");

    if ( xray.equals(zebra) )
        System.out.println("the contents of zebra and xray are the same");
    else
        System.out.println("the contents of zebra and xray are NOT the same");
    ```

• The compareTo() method determines the alphabetical order of the two strings.
  • Example:
  ```java
  int answer = oneString.compareTo(anotherString);
  ```
  • If the String object is less than the String argument, a negative integer is returned.
  • oneString is the object, and anotherString is the argument.
  • If the String object is greater than the String argument, a positive integer is returned.
  • Zero is returned if the contents of the two String’s are the same.
  • A character with a lower Unicode value is considered less than a character with a higher Unicode value.
  • Examples: ’A’ is less than ’a’. ’A’ is less than ’a’. ’Y’ is less than ’X’. ’Z’ is less than ’a’
  • Follows the normal rules for sorting text in English.
if Statements (continued):

Comparing Strings (continued):

- The `String` class provides additional methods to compare the contents of two `String`'s.

  Example:

  ```java
  String xray, yoke, zebra;
  xray = new String("Hello");
yoke = new String("Good-bye");
zebra = new String("baseball playoffs");
  int result;
  System.out.println("result = " + yoke.compareTo(zebra));
  System.out.println("result = " + yoke.compareTo(xray));

  if ( yoke.compareTo(zebra) < 0 )
     System.out.println("yoke comes before zebra");

  if ( yoke.compareTo(xray) < 0 )
     System.out.println("yoke comes before xray");
  
  Output:
  result = -27
  result = -1
  yoke comes before zebra
  yoke comes before xray
  ```

if Statements (continued):

Comparing Objects (continued):

- Other classes have an `equals()` method. Consider the `DecimalFormat` class:

  ```java
  DecimalFormat xray, yoke, zebra;
  xray = new DecimalFormat("$ #,###.00");
yoke = new DecimalFormat("$ #,###.00");
zebra = new DecimalFormat("###.#");

  if ( yoke.equals(xray) )
     System.out.println("the contents of yoke and xray are the same");
  else
     System.out.println("the contents of yoke and xray are NOT the same");

  if ( yoke.equals(zebra) )
     System.out.println("the contents of yoke and zebra are the same");
  else
     System.out.println("the contents of yoke and zebra are NOT the same");
  ```
if Statements (continued):

Conditional Operator (continued):

- The conditional operator contributes one of two values to an expression based on the value of the condition.
- Syntax:

  ```java
  condition ? trueExp : falseExp
  kilo = (hotel < golf) ? 42 : -378;
  ```

- Has the same meaning as:

  ```java
  if ( hotel < golf )
  kilo = 42;
  else
  kilo = -378;
  ```

- The conditional operator `?` is a tertiary operator in that it requires 3 operands: `condition, trueExp, falseExp`.
  - It is the only tertiary operator in Java.
- As a review, name:
  - Binary operators in Java:
  - Unary operators in Java:

if Statements (continued):

Conditional Operator (continued):

- Want to print a message that uses the correct singular or plural form:
  - Without the conditional operator, we can write:

    ```java
    if ( numBoxes == 1 )
    System.out.println("We need 1 box.");
    else
    System.out.println("We need " + numBoxes + " boxes.");
    ```

  - With the conditional operator, we can write:

    ```java
    int numBoxes = 7;
    System.out.println("We need " + numBoxes + ((numBoxes == 1) ? " box." : " boxes.");
    numBoxes = 1;
    System.out.println("We need " + numBoxes + ((numBoxes == 1) ? " box." : " boxes.");
    ```