Announcement

- Office hour is **permanently changed**
  - Wednesday, 12:30PM – 2:30PM
  - I’ll try to be at office before 3:30PM
Review: If-Else Statement

• If statement
  
  - \textit{if} (\textit{boolean expression})
    
    \{ \textit{statements}; \}
    
    \textit{other statements};

• If-else statement
  
  - \textit{if} (\textit{boolean expression})
    
    \{ \textit{statements 1}; \}
    
    \textit{else} \{ \textit{statement 2}; \}
Review: If-Else Statement

• Pay attention to the brackets {} 
  – You can discard them if there is only one statement in them 
  – if (inputInt > 0) 
    System.out.println(“Positive”);
  else 
    System.out.println(“Negative or zero”);
Review: If-Else Statement

- Pay attention to **the brackets {}**
  - You can discard them if there is only one statement in them
  - Discarding it may cause problems
  - **if** (`inputInt > 0`)
    ```java
    System.out.println("Positive");
    ```
  - **else**
    ```java
    System.out.println("Negative or zero");
    System.out.println("What’s happening?");
    // will always be executed
    ```
Review: If-Else Statement

- Pay attention to the brackets `{}`
  - You can discard them if there is only one statement in them
  - Discarding it may cause problems
  - \texttt{if (inputInt > 0)}
    \begin{verbatim}
    System.out.println("Positive");
    System.out.println("What’s happening now?");
    // Will cause a syntax error
    \end{verbatim}
  - \texttt{else}
    \begin{verbatim}
    System.out.println("Negative or zero");
    \end{verbatim}
Review: If-Else Statement

• Pay attention to the brackets `{}`
  – You can discard them if there is only one statement in them
  – Discarding it may cause problems
  – As a good habit, don’t discard them, even if you have only one statement in it
    • The only exception: multibranch if-else
Review: If-Else Statement

• Never put a semicolon after `if` or `else`
  
  – `if (inputInt > 0);`
  
  System.out.println("What’s happening now?");

• Compiler will interpret it as
  
  – `if (inputInt > 0)`
    
    `{ ; }`
  
    System.out.println("What’s happening now?");
Review: If-Else Statement

• Never put a semicolon after `if` or `else`
  
  ```java
  if (inputInt > 0)
      System.out.println("Positive");
  else;
      System.out.println("What's happening?");
  ```

• Compiler will interpret it as
  
  ```java
  if (inputInt > 0)
      { System.out.println("Positive");}
  else { ;}
  System.out.println("What's happening?");
  ```
Review: Boolean Expression

• Assignment vs. equal to
  – \( \text{if } ( n1 = n2 ) \)
    • Error!!!! It’s an assignment statement!
  – \( \text{if } ( n1 == n2 ) \)
    • Correct. It’s a boolean expression now.

• Use `equals()` to compare Strings
  – One_String.equals(Other_String)
  – One_String.equalsIgnoreCase(Other_String)
Nested If-Else Statement

• Without brackets, every *else* will automatically match the nearest *if*

```java
if (time < 7)
    if (!fridge.isEmpty())
        grab something from the fridge;
else
    go to school;
```

• Is this piece of code correct?
Nested If-Else Statement

- Without brackets, every `else` will automatically match the nearest `if`

```java
if (time < 7){
    if (!fridge.isEmpty())
        grab something from the fridge;
    else
        go to school;
} // Otherwise?
```

- Use brackets to avoid such errors
Multibranch If-Else Statement

• Example
  – Write a program that takes as input your year in college (as an integer) and outputs your year as freshman, sophomore, junior, senior, or super senior
Multibranch If-Else Statement

- Flow chart:

1. Prompt user for year
2. Which year?
3. freshman
4. sophomore
5. junior
6. senior
7. super senior
Multibranch If-Else Statement

• We can write a program like this

```java
if (year == 1)
    System.out.println("freshman");
else {
    if (year == 2)
        System.out.println("sophomore");
    else {
        if (year == 3)
            System.out.println("junior");
        else {
            if (year == 4)
                System.out.println("senior");
            else {
                if (year == 5)
                    System.out.println("super senior");
                else
                    System.out.println("huh?");
            }
        }
    }
}
```
Multibranch If-Else Statement

- Because the previous version is too ugly, we use the **multibranch** statement instead
  - It is not a new syntax rule. We only ignore the brackets so that the logical structure is clear.

```java
if (year == 1)
    System.out.println("freshman");
else if (year == 2)
    System.out.println("sophomore");
else if (year == 3)
    System.out.println("junior");
else if (year == 4)
    System.out.println("senior");
else if (year == 5)
    System.out.println("super senior");
else
    System.out.println("huh?");
```
Multibranch If-Else Statement

- Though all the branches look equal, there is a precedence order among them
  - Only the **first** satisfied branch will be executed
Multibranch If-Else Statement

• You can think the program flow as a highway. When you are driving on it, you always check the first exit – and then exit if possible.
  – If the exists are not listed properly, you will be lost
Multibranch If-Else Statement

• What’s wrong with this piece of code?

```python
if (time < 7)
    grab something from the fridge;
else if (time < 6)
    cook hams and scramble eggs;
else
    go to school;
```
Multibranch If-Else Statement

• What’s wrong with this piece of code?

```java
if (time < 7)
    grab something from the fridge;
else if (time < 6)
    cook hams and scramble eggs;
else
    go to school;
```

Will this branch get executed?
Switch Statement

```java
if (year == 1)
    System.out.println("freshman");
else if (year == 2)
    System.out.println("sophomore");
else if (year == 3)
    System.out.println("junior");
else if (year == 4)
    System.out.println("senior");
else if (year == 5)
    System.out.println("super senior");
else
    System.out.println("huh?");
```

```java
switch (year) {
    case 1:
        System.out.println("freshman");
        break;
    case 2:
        System.out.println("sophomore");
        break;
    case 3:
        System.out.println("junior");
        break;
    case 4:
        System.out.println("senior");
        break;
    case 5:
        System.out.println("super senior");
        break;
    default:
        System.out.println("unknown");
        break;
}
```
**Switch Statement**

- Syntax rules:
  - If `controlling expression == case_label_n`, then execute `statements_n`;
  - The `break` means jumping out of the statement.
  - Without the `break`, next statement will also be executed.

```java
switch (controlling expression /variable)
{
    case case_label_1:
        statements_1;
        break;
    case case_label_2:
        statements_2;
        break;
    default:
        statements;
        break;
}
```
Switch Statement

- Without a break (after `case 'A'` and `case 'C'`), the program will continue to the next case and execute that statement.
- Pay attention to colons and semicolons.

```java
switch (eggGrade) {
    case 'A':
    case 'a':
        System.out.println("Grade A");
        break;
    case 'C':
    case 'c':
        System.out.println("Grade C");
        break;
    default:
        System.out.println("We only buy grade A and grade C.");
        break;
}
```
Switch Statement

- The *default* case is optional
  - It means “everything else”
- The case labels must be of the same type as the controlling expression
- The controlling expression can only be *int*, *short*, *byte* or *char*
  - Why not *float* or *double*?
  - Why not *String*?
    - Hint: Think about “==”
Switch Statement

• The controlling expression can only be \textit{int}, \textit{short}, \textit{byte} or \textit{char}
  
  – Why not \textit{float} or \textit{double}?
    
    • \textit{float} and \textit{double} are only approximate values
    • They are inaccurate for "=="

  – Why not \textit{String}?
    
    • You can not use "==" to compare Strings
Multibranch vs. Switch

• Switch statement is more straightforward if you only use “==” to check a single expression/variable
  – Using proper `break` can have shorter code
• Multibranch if-else statement is more powerful
  – You can use it to check the range of a variable
  – You can use it to check the value of `float/double/String`
  – You can check more than one variable
Loop Statement

• Loop statements are designed to repeat instructions
  – Think about the requirement: Print number 1 to 10
    • It’s easy
      – System.out.println("1");
      – System.out.println("2");
      – ......
  – Think about the requirement: Print number 1 to 100
    • We can still do this
  – Let the user input a value n, then print 1 to n
    • We are in trouble......
Loop Statement

What is the pseudo code to fulfill the requirement?

- Count to 1, if 1<=n, write it down, otherwise stop
- Count to 2, if 2<=n, write it down, otherwise stop
- Count to 3, if 3<=n, write it down, otherwise stop
- ......
- Count to i, if i<=n, write it down, otherwise stop
- Count to i+1, if i+1<=n, write it down, otherwise stop
- ......

- While a counter<=n, write it down, increase the counter. Otherwise stop
While Statement

• Flow of while statement
  – Start from expression evaluation
  – As long as it’s true, repeat instructions in brackets

```java
while (count <= number) {
    System.out.println(count);
    count++;
}
```
While Statement

• You have to do some initialization before the statement

• The loop body typically contains an action that ultimately causes the controlling boolean expression to become false.

```java
number = keyboard.nextInt();
count = 1;
while (count <= number) {
    System.out.println(count);
    count++;
}
```
While Statement

• Usually there is a counter variable in the statement
  – You can use it in different ways
• Requirement: print the odd numbers from 1 to 10000

```java
int count = 1;
while (count < 10000) {
    System.out.println(count);
    count += 2;
}
```

```java
int count = 1;
while (count * 2 - 1 < 10000) {
    System.out.println(count * 2 - 1);
    count++;
}
```
Do-While Statement

• Another pseudo code to fulfill the requirement?
  • Write down 1 and count to 2, continue if 2<=n, otherwise stop
  • Write down 2 and count to 3, continue if 3<=n, otherwise stop
  • Write down 3 and count to 4, continue if 4<=n, otherwise stop
  • ……

• See the difference?
  • Count to 1, if 1<=n, write it down, otherwise stop
  • Count to 2, if 2<=n, write it down, otherwise stop
  • Count to 3, if 3<=n, write it down, otherwise stop
  • ……
Do-While Statement

• The main difference: do-while statement will at least execute the body statements once
  – If start from \( \text{count} = 1 \) and \( \text{number} = 0 \)
  – While statement will output nothing
  – Do-While statement will output 1, then stop

```
while (count <= number) {
    System.out.println(count);
    count++;
}
```
```
do {
    System.out.print(count);
    count++;
} while (count <= number);
```
Do-While Statement

• Flow of do-while statement
  – Start from body statements
  – Repeat instructions in brackets as long as the expression is true

```java
do {
    System.out.print(count);
    count++;
} while (count <= number);
```
Do-While Statement

• Don’t forget the semicolon after the whole statement
• Not recommended, unless you really need the body statement to be executed once

```java
do {
   System.out.print(count);
   count++;
} while (count <= number);
```
While and Do-While Statements

• These two pieces of code perform identically

```c
initialization;
do {
   body_statments
} while (boolean_expression);
```

```c
initialization;
body_statements;
while (boolean_expression) {
   body_statements;
}
```
Infinite Loops

• Q: How did the programmer die in the shower?
  A: He read the shampoo bottle instructions:  
    \textit{Lather, Rinse, Repeat}.

• If the controlling boolean expression never 
  becomes false, a \texttt{while} loop or a \texttt{do-while} loop will 
  continue to repeat without ending.
Infinite Loops

• Always make sure that your loop will end
  – Never forget to change the counter

```java
while (count <= number) {
    System.out.println(count);
}
```

```java
while (count <= number); {
    System.out.println(count);
}
```

```java
while (count <= number) {
    ;
}
System.out.println(count);
```
Infinite Loops

• Always make sure that your loop will end
  – Never forget to change the counter
  – Use comparison instead of “==” or “!=” in the control expression
  – Know whether your counter is increasing or decreasing

```java
while (count != number) {
    System.out.println(count);
    count+=2;
}
while (count < number) {
    System.out.println(count);
    count--;
}
```
Infinite Loops

- If you wrote an infinite loop and executed it
- Use the **terminate** button of eclipse
  - If it is red, the program is **running**
Infinite Loops

• Infinite loop is not a syntax error. It’s a logical error
• eclipse will not help you in this case
• Write pseudo code, think, and rethink before coding