

# COMP-202

---

## Unit 3: Conditional Programming

### **CONTENTS:**

The `if` and `if-else` Statements

Commenting

Library methods

Boolean Expressions

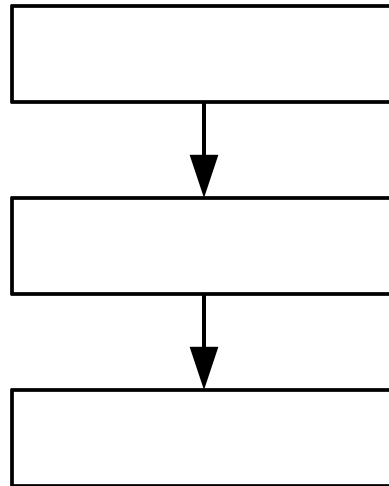


# Control Flow

---

- The default order of statement execution through a method is linear: one statement after the other, in the order they are written (from the top of the page down towards the bottom)
- Some programming statements modify that order, allowing us to:
  - decide whether or not to execute some statements, or
  - perform some statements over and over repetitively
- The order of statement execution is called *control flow* or *flow of control*

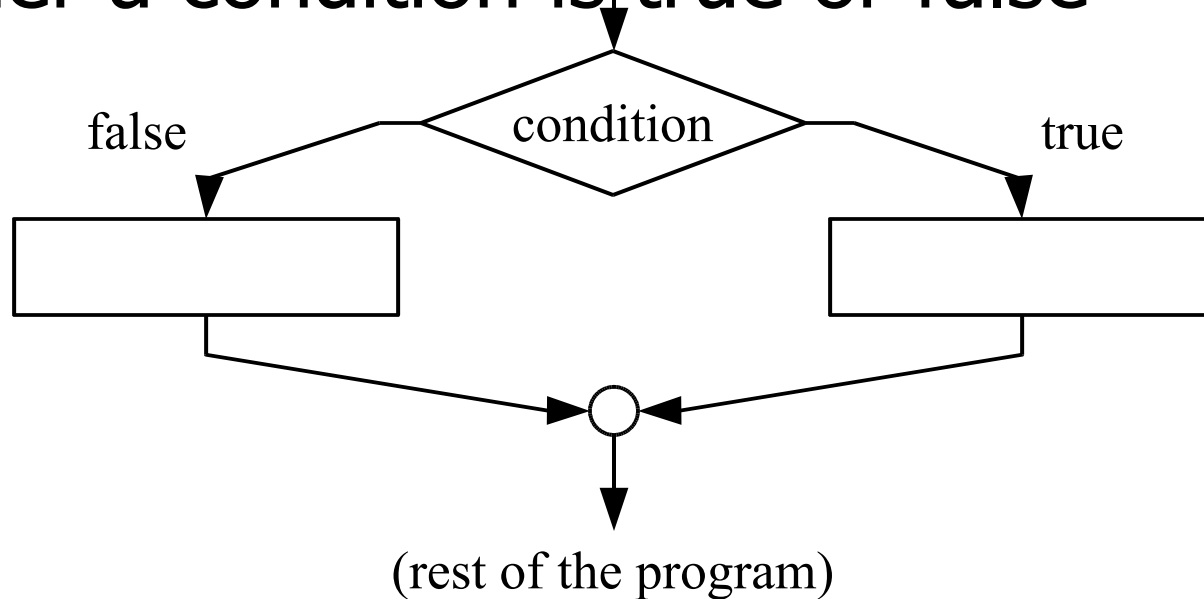
# Control Flow: Sequence



- In the **sequence** structure, statements are executed in the order they appear in the code

# Control Flow: Conditional

- In the **conditional** structure, one of two courses of action is taken depending on whether a condition is true or false





# if statements in Java

---

```
if ( condition){  
    //do something only if condition is true  
}  
//code goes here no matter what
```

condition can refer to any *boolean expression*

*This means anything that evaluates to be true or false*



# if statements in Java : Example

---

```
if ( x > 0 ){  
    System.out.println("positive");  
}
```



# if statements in Java : Example

---

```
if ( calculateComp202Grade() > 85 ){  
    System.out.println("Cool!");  
}
```

Here the condition is still a boolean expression. It means, "is the return value of the method calculateComp202Grade() greater than 85"



# if statements in Java : Example

---

```
if ( calculateComp202Grade() > 85 ){  
    System.out.println("Cool!");  
}
```

The code inside the { } of the if statement is only executed if the condition is true.

The code afterwards is executed no matter what.





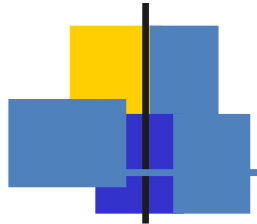
# if statements in Java : Example

---

```
int x = 4;
```

```
if ( Math.pow(x+1,2) > 16) {  
    System.out.println("OK");  
}
```

# if / else



Sometimes, you will have some code that you want to be executed if the condition is true and other when it is **false**

Option 1: 2 if statements using a !

Any time you have a boolean expression, you can get the opposite value of it by writing ! before it.


# if / else



So, for example,  $!(x > 0)$  is the same thing as  $x \leq 0$

Using this, I could do the following:

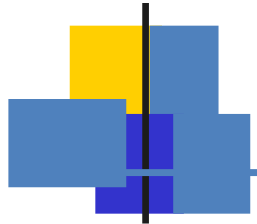
# if / else



---

```
if (x > 0) {  
    System.out.println("It is positive");  
}  
if ( ! (x > 0) )  
    System.out.println("It is negative");  
}
```

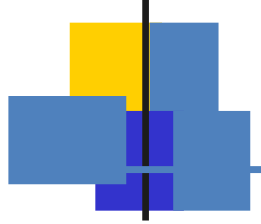
# if / else



This code is a bit cumbersome:

- 1) There is no obvious link between the 2 statements. Someone looking at the code wouldn't know
- 2) Since there is no link, if we had a more complicated statement there, it could end up that BOTH statements happened.

# if / else



```
if ( condition) {  
    // happens if condition is true  
}  
else {  
    // happens if condition was false  
}  
//happens no matter what
```



# An example of the difference:

---

```
int x = 3;
if (x > 0 ) {
    System.out.println("Positive.Now resetting"
        + "it's value.");
    x = 0;
}
```

```
if ( ! (x > 0) ) {
    System.out.println("Not positive");
}
```

# An example of the difference:

---

```
int x = 3;
if (x > 0 ) {
    System.out.println("Positive.Now resetting"
        + "it's value.");
    x = 0;
}
else {
    System.out.println("Not positive");
}
```





# An example of the difference:

---

In the first case, both if statements get executed because the value of the condition is true at both points.

However, with the if/else, only the first if is executed. The else is not, because the condition was evaluated at the beginning. If it isn't true at first, then the else never is entered.

# A fair game!



```
String coin = getARandomFlip();  
if (coin.equals("heads") ){  
    System.out.println("I Win!");  
}  
else {  
    System.out.println("You lose!");  
}
```

# if /else if /else



Sometimes, you will have a more complicated thing where there are more than 2 options.

Option one: Nested if statements:

```
if (option1) { }  
else {  
    if (option 2) { }  
    else {  
        if (option 3) { } } } }
```

# if /else if /else



A better way to do this is using “else if”

---

```
if (condition1) {  
}  
else if (condition2) {  
}  
else if (condition 3) {  
}  
else {  
}
```



# if /else if /else

---

Much like else, the “else if” is only entered if the prior conditions were false.

So there is a big difference if we change the order of the if / else if statements.

# if /else if /else



```
if ( x > 0 ){
    System.out.println("positive");
}
else if ( x > -1) {
    System.out.println("<-1 but <0");
}
else {
    System.out.println("zero");
}
```

# if statements in Java : Example

---

```
if ( x > 0 ){
    System.out.println("positive");
}
else if ( x > -1) {
    System.out.println(">-1 but <=0");
}
else if (x > -2) {
    System.out.println(">-2 but <= -1");
}
else {
    System.out.println("zero");
}
```

# if statements in Java : Example



---

```
if ( x > 0 ){
    System.out.println("positive");
}
else if ( x > 1) {
    System.out.println("This statement is unreachable!");
}
else {
    System.out.println("zero");
}
```

Why is the highlighted section impossible to get to?





# Comments : Single line

---

A comment is something only used by the programmer.

If you write

```
//
```

anywhere in the code, the rest of the line is ignored by the compiler.



# Comments : Multi-line

---

A comment is something only used by the programmer.

If you write

```
/*
```

anywhere in the code, then the rest of the code is ignored by the compiler until it sees

```
*/
```



# Comments : Purposes

---

Comments are generally used to help the programmer understand what he's doing.

This is useful both for when other people read your code or if you go back to your code at a later point.



# Good Comments

---

A good comment will make it clear what a complicated piece of code will do.

A bad comment will either mislead the user or provide unnecessary information (i.e. over commenting)



# Comments

---

```
//  
// Dear maintainer:  
//  
// Once you are done trying to 'optimize' this program,  
// and have realized what a terrible mistake that was,  
// please increment the following counter as a warning  
// to the next guy:  
//  
// total_hours_wasted_here = 39  
//
```



# Comments

---

```
/* <summary>
```

```
Class used to work around Richard being a @#*$ idiot
```

```
</summary>
```

```
<remarks>
```

```
The point of this is to work around his poor design so that paging  
will
```

```
work on a mobile control. The main problem is the  
BindCompany() method,  
which he hoped would be able to do everything. I hope he dies.
```

```
</remarks> */
```



# Comments

---

// sometimes I believe compiler ignores all my comments



# Comments

---

// drunk, fix later





# Comments

---

/\*

- \* You may think you know what the following code does.
- \* But you dont. Trust me.
- \* Fiddle with it, and youll spend many a sleepless
- \* night cursing the moment you thought youd be clever
- \* enough to "optimize" the code below.
- \* Now close this file and go play with something else.
- \*/



# Comments

---

```
// if i ever see this again i'm going to start bringing guns to  
// work
```



# Java Libraries

---

The Java Sdk comes with many *libraries* which contain classes and methods for you to use. These are not technically part of the language, but they are *types* of classes that others have written to extend Java and come with Java.

Some of these include:

- a Math library
- String library
- Graphics library
- Networking library



# Java Libraries

---

You can find a list of libraries in Java 6

<http://download.oracle.com/javase/6/docs/api/index.html?overview-summary>.

To use a library, first find the name of the library you wish to use. For example, the first library at the link has the name

`java.applet`

Then, write at the top of your program

```
import name;
```

# Example: Reading from Keyboard



One example is to read information from the keyboard that a user enters. There are methods that do this defined in the class

<http://download.oracle.com/javase/6/docs/api/java/util/Scanner.html>

To use the Scanner class in your program, write at the top

```
import java.util.Scanner;
```

(you can also write `import java.util.*;` if you want)



# Example: Reading from Keyboard

---

To use the Scanner class, after writing the import statement above, you simply declare a variable of type Scanner.

Scanner can work to read both files and from the keyboard, so when you create a Scanner class variable, you tell it where to scan.

Scanner has defined in it methods that will do things such as search for the next integer.



# RTFM

---

One of the best ways to learn about various methods that exist in classes such as this, is by reading the documentation.

Speaking of this, some of you may be wondering what the title stands for....To find that out, RTFM

# Example: Reading from Keyboard



---

```
import java.util.Scanner;
```

```
public class ReadInput {  
    public static void main(String[] args) {  
        Scanner s = new Scanner(System.in);  
        System.out.println("Enter a number");  
        int number = s.nextInt();  
        System.out.println("You entered the number " + number);  
    }  
}
```

**Exercise to try: What will happen if the user enters letters instead of a number? What can we do about this?**





# Exercise: Computing the Pythagorean theorem

---

Write a program that does the following:

- 1) Reads two doubles as input from the user
- 2) Calls a method that computes the hypotenuse of a right triangle with those 2 entered numbers as sides. (The function should return a double)
- 3) Print the result

**Hint: The square root method can be found in a class `java.lang.Math`**



# Exercise: Absolute value

---

Write a program that does the following:

- 1) Reads an integer from the user
- 2) Calls a method that computes the absolute value of the integer.
- 3) Calls the absolute value method defined in `java.lang.Math` on the integer
- 4) Print the result of each method and make sure they are the same



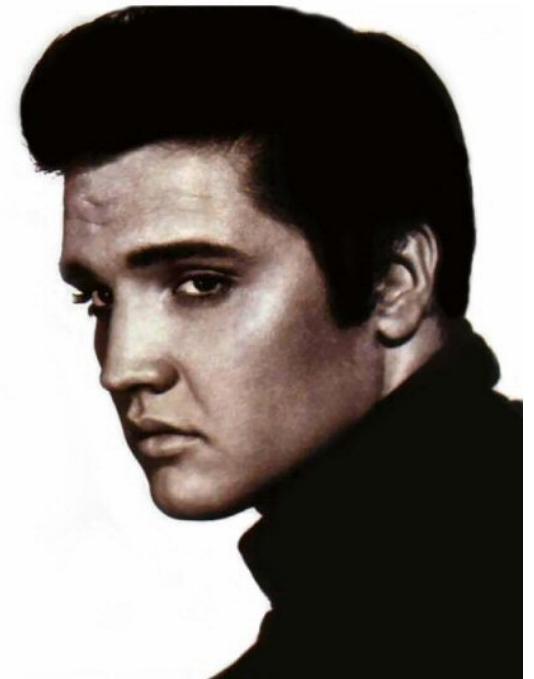
---

# **Part 1: Boolean Expressions --**

**(a.k.a. the things you put in if statements and other things as conditions)**

# Boolean Expressions: True or False

Expression	Value
2010 is an odd number	F
Elvis Presley died in 1977.	T
Elvis Presley was born in the United States and had red hair.	F





# Boolean Expressions

- Instead of evaluating to a numeric value, *boolean expressions* evaluate to either `true` or `false`

```
myNumber > 0 // can be either true or false
```

- You can assign the result of a boolean expression to a variable of type `boolean`:

```
boolean positive;  
positive = (myNumber > 0);
```



# Boolean Expressions in Java

- A *boolean expression* is a combination of operators and operands, and it evaluates to a **boolean** value
- A boolean expression can be:
  - The comparison of two values using a comparison operator
  - A variable which has type `boolean`
  - A call to a method that returns a type `boolean`
  - `true` or `false` (Java's boolean literals)
  - The negation of another boolean expression using the `!` operator
  - The combination of two or more other boolean expressions using the `&&` or `||` operators

# Boolean Expressions with Comparison Operators

Expression	Meaning	Value
<code>5 == 2</code>	5 is equal to 2	F
<code>!(10 &gt;= 5)</code>	10 is greater or equal to 5	T
<code>-2 &lt; 4</code>	-2 strictly less than 4	T

# Boolean Expressions with Logical Operators

Expression	Meaning	Value
$x \neq 5 \ \&\& \ y > 2$	x is not equal to 5 <b>and</b> y is greater than 2.	Depends on x and y
$!(10 == 5)$	The <b>negation</b> of "10 is equal to 5"	T
$1 < 0 \    \ 1 > 0$	1 is less than 0 <b>or</b> 1 is greater than 0	T





# More Boolean Expressions

Expression	Meaning	Value
true	Something that is always true.	T
<b>!</b> false	The <b>negation</b> of something that is always false.	T



# Comparison Operators (1)

The result of a comparison is always `true` or `false`

Used to compare numeric or character values

`==` : equal to

`!=` : not equal to

`<` : less than

`>` : greater than

`<=` : less than or equal to

`>=` : greater than or equal to



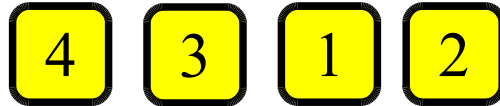
## Comparison Operators (2)

- Equality (`==`) and inequality (`!=`) operators apply to values that have any type
- The other comparison operators (`<`, `<=`, `>`, `>=`) only apply to values which have a numeric type (`byte`, `short`, `int`, `long`, `float`, `double`) or that have type `char`
- If the operands of a comparison operator have different types, the operand whose type has lower precision gets promoted to the other operand's type

# Comparison Operator Precedence

- Comparison operators have
  - **lower** precedence than **arithmetic** operators
  - **higher** precedence than the **assignment** operator

`boolean b = a > c * d + e;`



1. The product of `c` and `d` is evaluated first
2. Then, the value of `c * d` is added to `e`
3. Then, the value of `c * d + e` is compared to the value of `a`
4. Finally, the result of the comparison is stored in variable `b`



# Character Comparisons (1)

---

- In Java, each character (like '?' or 'm') is associated with a number.
- The following expression evaluates to **true** because the number assigned to the character '+' by the Unicode character set is lower than the number assigned to the character 'J' by the same character set:

```
boolean lessThan = '+' < 'J';
```

- Do not hesitate to use this property of characters in your programs.

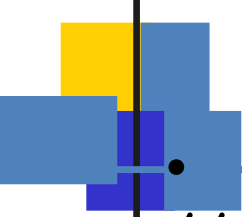


# Floating Point Comparisons (1)

---

- You should rarely use the equality operator (`==`) when comparing two floating point values (`float` or `double`)
- In many situations, you might consider two floating point numbers to be "close enough" even if they aren't exactly equal
  - Better approach: check if their difference is less than a certain threshold

# Floating Point Comparisons: Example



---

```
// Assuming f1 >= f2  
difference = f1 - f2;  
boolean essentiallyEqual = difference < 0.00001;
```

# Logical Operators



- Boolean expressions can also use the following *logical operators*:

! Logical NOT

|| Logical OR

&& Logical AND

- All three operators take operands of type `boolean` and produce results of type `boolean`



# Truth Tables

- The possible values of boolean expressions can be shown using *truth tables*
- A truth table contains all possible combinations of values for the terms in the expression
- The value of the expression for each combination is also shown
- Below is the truth table for boolean expression  $\neg a$

<b>a</b>	<b><math>\neg a</math></b>
true	false
false	true



## Exercise : More Complex

What would be the truth table for

$(a \ \&\& \ b) \ || \ ! \ (c \ \&\& \ b)$

That is to ask, what values of a,b, and c will make it so the above is true and what values will make it so it is false

# Logical Operator Precedence

( 1 )

• Like arithmetic operators, logical operators have precedence rules among themselves

- 1) !
- 2) &&
- 3) ||

a || b && !c

3	2	1
---	---	---

1. First, the negation of  $c$  is evaluated
2. Then,  $b$  is "AND-ed" with the value of  $!c$
3. Finally,  $a$  is "OR-ed" with the value of  $b \ \&\& \ !c$

# Logical Operator Precedence

(?)

- Logical operators have
  - **lower** precedence than **comparison** operators
  - **higher** precedence than the **assignment** operator

```
boolean b = a && c < d;
```

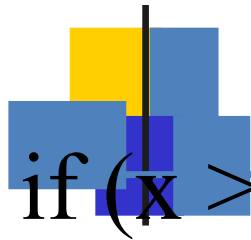
3 2 1



## `if` Statement trap

If you don't put `{` after the `if` statement, Java compiler will assume you only want 1 line inside your `if` statement.

# `if` Statement trap



```
if (x > 0)
```

---

```
System.out.println("this is part of the if");  
System.out.println("This is not");
```



# `if` Statement trap

---

Another thing to be careful of is not to put a `;` after the `if` statement. Always put a `{` instead.

What would happen if you just put a `;` by mistake?



# `if` Statement Example 2

---

The statements between braces `{ }` form a **block**.

Whenever you want more than one statement to be executed as part of an “if” statement, use braces create a block.



## Exercise:

What will be displayed if this code executes?



---

```
int x = 0, y = 0;  
if (y > 0 && y < 5 || !(x != 2))  
System.out.println("yes");  
System.out.println("congratulations!");
```



# How The Java Compiler Evaluates `&&` / `||`

---

- If left operand of a `&&` expression evaluates to *false*, the remaining operands are not evaluated
  - In `p1 && p2`, if `p1` is false, `p2` is never looked at.
- If left operand of a `||` expression evaluates to *true*, the remaining operands are not evaluated
  - In `p1 || p2`, if `p1` is true, `p2` is never looked at.
  - This is called “short-circuit” evaluation.



# Aside: How The Java Compiler Evaluates `&&` / `/ /`

---

This is useful in the following case. Suppose you have an int variable `x` and you aren't sure if `x` is equal to zero or not.

```
if ( x != 0 && 1 / x < 5 )
```



# Example: Bad Style

---

- - `boolean tall = height > 6.0;`
  - `if(tall == true) x = 5;`

# Example: Bad Style Leads to a Common Error

---

- 
- `boolean tall = height > 6.0;`
- `if(tall = true) x = 5;`

# Example: Bad Style Leads to a Common Error

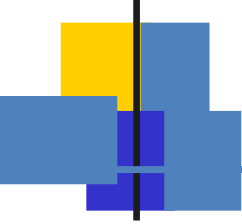
---

- - `boolean tall = height > 6.0;`
  - `if(tall = true) x = 5;`

This sets tall to true, so “x = 5” always executes, regardless of the value of height.

This is a logical error: **compiler will not detect it!**

# Eliminates Possibility of Common Error

- 
- 
- - `boolean tall = height > 6.0;`
  - `if(tall) x = 5;`



# Nested if statements

---

- Sometimes, you will put if statements inside of each other. This gets a bit confusing when you have else statements. For example, which else does something belong to?





# Nested `if` Statements (2)

- One can write nested `if-else` statements like this:

```
if ( condition1 )
    if ( condition2 )
        statement1;
    else
        statement2;
else
    if ( condition3 )
        statement3;
    else
        statement4;
```

Each  $e \in s e$  paired with most recent  
uninterrupted  $i f$  in same block



---

- 
- $\text{if}(x \neq 0) \text{ width} = x + 5;$
- $x = 2;$
- $\text{else } \{$
- $\text{width} = x + 10;$
- $x = x + 1;$
- $\}$

Each  $e \ell s e$  paired with most recent  
**uninterrupted**  $i f$  in same block

---

- 
- $\text{if}(x \neq 0) \text{ width} = x + 5;$
- $x = 2;$
- $\text{else } \{$
- $\text{width} = x + 10;$
- $x = x + 1;$
- $\}$

**Compile-time error:**  
 $e \ell s e$  isn't associated  
with any  $i f$  statement.

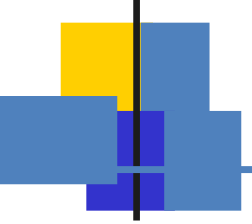
$x = 2;$  “interrupts” the  
if on the first line.

Each  $e \in s e$  paired with most recent  
uninterrupted  $i f$  in same block

---

```
if(width > 0) {  
    if(x != 0) width = x + 5;  
    x = 2;  
}  
else {  
    width = x + 10;  
    x = x + 1;  
}
```

# recent uninterrupted *if* in **same block**



```
if(width > 0) {  
    if(x != 0) width = x + 5;  
    x = 2;  
}  
else {  
    width = x + 10;  
    x = x + 1;  
}
```

Each *else* paired with most recent  
uninterrupted *if* in **same block**

```
if(width > 0) {  
    if(x != 0) width = x + 5;  
    X = 2;  
}  
else {  
    width = x + 10;  
    x = x + 1;  
}
```

This *if* and this  
*else* are in  
different blocks,  
so they are not  
paired together.

# Nested `if` Statements:

## Exercise

- Complete the `main()` method of the `MinOfThree` class by adding code which determines which of the three numbers entered by the user is the smallest number, and displays that number
- Can you write this code both with and without using block statements?



# MinOfThree.java

---

```
import java.util.Scanner;

public class MinOfThree {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        int num1, num2, num3, min;

        System.out.print("Enter a number: ");
        num1 = keyboard.nextInt();
        System.out.print("Enter another number: ");
        num2 = keyboard.nextInt();
        System.out.print("Enter a third number: ");
        num3 = keyboard.nextInt();

        // Add your code here
    }
}
```





---

# **Advanced Conditional Statements (not examinable)**


- 1) The switch statement
- 2) The conditional operator

# The “switch” statement: example

---

```
int x,y;  
switch(x+y){  
    case 5: System.out.print(“A”);  
    case 8: System.out.print(“B”);  
            System.out.print(“F”);  
    case 1: System.out.print(“C”);  
    default: System.out.print(“D”);  
}
```

# The "switch" statement



```
int x,y;  
switch(x+y){  
    case 5: System.out.print("A");  
    case 8: System.out.print("B");  
    case 1: System.out.print("C");  
    default: System.out.print("D");  
}
```

Case values must be **literals** or **constants**; all of same type as "test expression".

# What will this display?

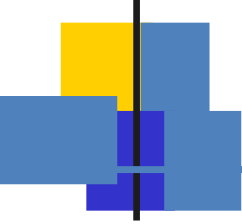


---

```
int section = 4;
```

```
switch(section) {  
  case 1:  
    System.out.println("A");  
  default:  
    System.out.println("B");  
  case 2:  
  case 3:  
    System.out.println("C");  
    System.out.println("4");  
}
```

# "switch" statement with "break"

- 
- 
- `int x,y;`
  - `switch(x+y){`
  - `case 5: System.out.print("A");`
  - `break;`
  - `case 8: System.out.print("B");`
  - `System.out.print("F");`
  - `case 1: System.out.print("C");`
  - `break;`
  - `default: System.out.print("D");`
  - `}`

# What will this display?



```
int section;
```

---

```
System.out.print("Enter your COMP-202 section: ");  
section = keyboard.nextInt();
```

```
switch(section) {  
    case 1:  
        System.out.println("Your section number is not prime.");  
        break;  
    case 2:  
    case 3:  
        System.out.println("Your section number is prime.");  
        break;  
    default:  
        System.out.println("There must be lots of students!");  
}
```

# What will this display?



```
int section;
```

---

```
System.out.print("Enter your COMP-202 section: ");  
section = keyboard.nextInt();
```

```
switch(section) {  
    default:  
        System.out.println("There must be lots of students!");  
    case 1:  
        System.out.println("Your section number is not prime.");  
        break;  
    case 2:  
    case 3:  
        System.out.println("Your section number is prime.");  
        break;  
}
```



# Summary of `switch` Statements

- The expression of a `switch` statement must evaluate to a value of type `char`, `byte`, `short` or `int`; it **cannot** be a floating point value, a `long`, a `boolean`, or any reference type, including `String`
- Note that the implicit boolean expression in a `switch` statement is equality
  - The `switch` statement tries to match the expression with a value (it is never `<`, `<=`, `>`, nor `>=`)
- You **cannot** perform relational checks with a `switch` statement
- The value of each `case` must be a constant (either a literal or a `final` variable)
  - It cannot be a plain (that is, non-`final`) variable



# The Conditional Operator



---

*condition* ? *expression1* : *expression2*

- 
- 
- If *condition* evaluates to `true`, then *expression1* is evaluated; if it evaluates to `false`, then *expression2* is evaluated

# Conditional Operator Examples

(1)



---

```
larger = (num1 > num2) ? num1 : num2;
```

- 
- If `num1` is greater than `num2`, then `num1` is assigned to `larger`; otherwise, `num2` is assigned to `larger`



# Conditional operator vs. if-else

---

- The conditional operator is like an if-else statement, except that instead of executing one of two possible **branches**, it evaluates to one of two possible **values**.  
 $larger = (num1 > num2) ? num1 : num2;$

...is the same as:

```
if (num1 > num2)
    larger = num1;
else
    larger = num2;
```



# Conditional Operator Examples

---

```
System.out.println ("Your change is " +  
    count + " dime" +  
    ((count == 1) ? "" : "s"));
```

- 
- If `count` evaluates to 1, then "dime" is printed
- If `count` evaluates to any value other than 1, then an "s" is added at the end of "dime"



- Exercise:
- Use the conditional operator to
- express “absolute value”.
- 
- Absolute value examples:
- The absolute value of 6 is 6.
- The absolute value of -1 is 1.



# Constants (1)

---

A constant is like a variable except that it holds one value for its entire existence

The compiler will issue an error if you try to assign a value to a constant more than once in the program



# Advantages of Constants

---

- - Constants can make programs easier to understand
  - Constants facilitate changes to the code
  - Constants prevent inadvertent errors