Revised: 11/23/2025

Logic and Branching

Michael C. Hackett

Associate Professor, Computer Science

COMMUNITY COLLEGE OF PHILADELPHIA

Lecture Topics

- Boolean Logic and Expressions
  - Relational Operators and Expressions
  - Logical Operators and Expressions
  - Operator Precedence
- Branching
  - If Statements
  - If-Else Structures
  - If-Else-If Structures
- Methods of String Comparison
- Switch Structures
- Conditional Operator

Colors/Fonts

- Local Variable Names – Brown
- Primitive data types – Fuchsia
- Literals – Blue
- Keywords – Orange
- Object names – Green
- Operators/Punctuation – Black
- Field Names – Lt Blue
- Method Names – Purple
- Parameter Names – Gold
- Comments – Gray
- Package Names – Pink

Source Code – **Consolas**
Output – Courier New

Boolean expression is false

Boolean expression is true

Relational Operators

- ***Relational operators*** perform a comparison that determines how two values relate to each other.
  - Each operator returns True or False

==	Equality Operator
!=	Inequality Operator
>	Greater Than Operator
<	Less Than Operator
>=	Greater Than or Equal To Operator
<=	Less Than or Equal To Operator

Relational Expressions

- A ***relational expression*** is an expression using a relational operator.
  - 1 == 5 (false)
  - 7 != 3 (true)
  - 16 > 5 (true)
  - 56 < 22 (false)
  - 10 >= 10 (true)
  - 9 <= 5 (false)
- A relational expression is a type of ***Boolean expression***.
  - A Boolean expression is one that evaluates to true or false.

Equality Operator ==

- Returns **true** if the operands are the same value.
- Returns **false** if the operands are different values.

```
int i = 8;
int j = 10;
boolean result1 = i == j;
```

false

```
int k = 10;
int m = 10;
boolean result2 = k == m;
```

true

Inequality Operator !=

- Returns **true** if the operands are different values.
- Returns **false** if the operands are the same value.

```
int i = 8;
int j = 10;
boolean result1 = i != j;
```

true

```
int k = 10;
int m = 10;
boolean result2 = k != m;
```

false

Greater Than Operator >

- Returns **true** if the first operand is larger than the second operand.
- Returns **false** if the first operand is equal to or smaller than the second operand.

```
int i = 8;
int j = 10;
boolean result1 = i > j;

int k = 10;
int m = 10;
boolean result2 = k > m;
```

false

false

Less Than Operator <

- Returns **true** if the first operand is smaller than the second operand.
- Returns **false** if the first operand is equal to or larger than the second operand.

```
int i = 8;
int j = 10;
boolean result1 = i < j;

int k = 10;
int m = 10;
boolean result2 = k < m;
```

true

false

Greater Than or Equal To Operator >=

- Returns **true** if the first operand is equal to or larger than the second operand.
- Returns **false** if the first operand is smaller than the second operand.

```
int i = 8;
int j = 10;
boolean result1 = i >= j;
```

false

```
int k = 10;
int m = 10;
boolean result2 = k >= m;
```

true

Less Than or Equal To Operator <=

- Returns **true** if the first operand is equal to or smaller than the second operand.
- Returns **false** if the first operand is larger than the second operand.

```
int i = 8;
int j = 10;
boolean result1 = i <= j;

int k = 10;
int m = 10;
boolean result2 = k <= m;
```

true

true

Logical Operators

- A ***logical operator*** connects two or more Boolean expressions or values into one **true** or **false** result.
  - Or, in the case of the logical not operator, reverse the logic of a Boolean expression or value.

&& **||** **!**

- A ***logical expression*** is an expression using a logical operator.

AND

- Evaluates to true if and only if both boolean values are true.
- AND Truth Table:

B_1	B_2	B_1 && B_2
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

"Both must be true"

OR

- Evaluates to true if at least one of the boolean values is true.
- OR Truth Table:

B_1	B_2	B_1 || B_2
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

"At least one must be true"

NOT

- Inverts/Negates a boolean value.
- NOT Truth Table:

B_1	$!B_1$
FALSE	TRUE
TRUE	FALSE

And Operator

```
boolean b1 = false;
boolean b2 = false;

boolean result = b1 && b2;
```

false

B_1	B_2	B_1 && B_2
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

And Operator

```
boolean b1 = false;
boolean b2 = true;

                    false
boolean result = b1 && b2;
```

B_1	B_2	B_1 && B_2
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

And Operator

```
boolean b1 = true;
boolean b2 = false;

                    false
boolean result = b1 && b2;
```

B_1	B_2	B_1 && B_2
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

And Operator

```
boolean b1 = true;
boolean b2 = true;

                    true
boolean result = b1 && b2;
```

B_1	B_2	B_1 && B_2
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

Or Operator

```
boolean b1 = false;
boolean b2 = false;

                    false
boolean result = b1 || b2;
```

B_1	B_2	B_1 || B_2
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

Or Operator

```
boolean b1 = false;
boolean b2 = true;

                    true
boolean result = b1 || b2;
```

B_1	B_2	B_1 || B_2
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

Or Operator

```
boolean b1 = true;
boolean b2 = false;

                    true
boolean result = b1 || b2;
```

B_1	B_2	B_1 || B_2
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

Or Operator

```
boolean b1 = true;
boolean b2 = true;

                    true
boolean result = b1 || b2;
```

B_1	B_2	B_1 || B_2
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

Not Operator

```
boolean b1 = true;

                    false
boolean result = !b1;
```

B_1	not B_1
FALSE	TRUE
TRUE	FALSE

Not Operator

```
boolean b1 = false;

                    true
boolean result = !b1;
```

B_1	$!B_1$
FALSE	TRUE
TRUE	FALSE

Logical Operator Precedence

1. **!** **Not** Operator
2. **&&** **And** Operator
3. **||** **Or** Operator

Operator Precedence

1.	**!, -**	Not Operator, Unary Negation (-5)
2.	***, /, %**	Multiplication, Division, Modulus
3.	**+, -**	Addition, Subtraction
4.	**<, >, <=, >=**	Less than (or equal), Greater than (or equal)
5.	**==, !=**	Equal to, Not equal to
6.	**&&**	And Operator
7.	**||**	Or Operator
8.	**=, +=, -=, *=, /=, %=**	Assignment and Combined Assignment

Operator Precedence

```
int i = 4;
int j = 8;
boolean b1 = false;

boolean result = !b1 && i + j >= 9;
```

- What is the value of the result variable?

Operator Precedence

```
int i = 4;
int j = 8;
boolean b1 = false;

boolean result = !b1 && i + j >= 9;
```

```
true && i + j >= 9;
true && 12 >= 9;
true && true
true
```

Operator Precedence

```
int i = 4;
int j = 5;
boolean b1 = false;

boolean result = b1 || i + j == 9;
```

- What is the value of the result variable?

Operator Precedence

```
boolean b1 = false;
boolean b2 = false;
boolean b3 = false;

boolean result = !b1 || b2 && b3;
```

- What is the value of the result variable?

Operator Precedence

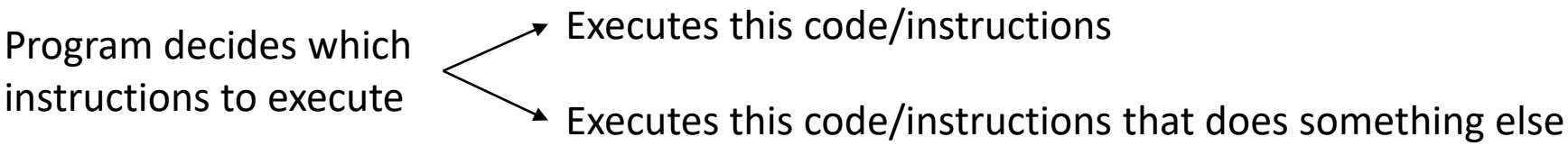
```
boolean b1 = false;
boolean b2 = false;
boolean b3 = false;

boolean result = !(b1 || b2) && b3;
```

- What is the value of the result variable?

Branching

- ***Branching***, in computer science, is when a computer program or algorithm departs from executing its current set of instructions to begin executing different instructions.

- In programming terms, branching normally refers to when a program or algorithm *decides* which set of instructions to execute.



If Statements

- An ***if statement*** tests a Boolean expression and will only execute its instructions if the expression evaluates to true.
  - The code will be "skipped" if the Boolean expression evaluates to false.
- The syntax for an if statement in Java is shown below.

```
if(Boolean Expression) {
    //code that will be
    //executed if the Boolean Expression
    //evaluates to true
}
```

- The Boolean expression as part of an if statement forms a ***conditional expression***.

Example If Statement (<)

```
int i = 8;
int j = 10;

if(i < j) {
   //ANY CODE HERE WILL EXECUTE
   //BECAUSE i (8) IS LESS THAN
   //j (10)
}
```

Example If Statement (<)

```
int i = 10;
int j = 8;

if(i < j) {
   //ANY CODE HERE WILL NOT EXECUTE
   //BECAUSE i (10) IS NOT LESS THAN
   //j (8)
}
```

Example If Statement (>)

```
int i = 10;
int j = 8;

if(i > j) {
   //ANY CODE HERE WILL EXECUTE
   //BECAUSE i (10) IS GREATER THAN
   //j (8)
}
```

Example If Statement (<=)

```
int i = 8;
int j = 10;

if(i <= j) {
   //ANY CODE HERE WILL EXECUTE
   //BECAUSE i (8) IS LESS THAN
   //OR EQUAL TO j (10)
}
```

Example If Statement (<=)

```
int i = 10;
int j = 10;

if(i <= j) {
   //ANY CODE HERE WILL EXECUTE
   //BECAUSE i (10) IS LESS THAN
   //OR EQUAL TO j (10)
}
```

Example If Statement (>=)

```
int i = 10;
int j = 8;

if(i >= j) {
   //ANY CODE HERE WILL EXECUTE
   //BECAUSE i (10) IS GREATER THAN
   //OR EQUAL TO j (8)
}
```

Example If Statement (==) - Numbers

```
int i = 8;
int j = 8;

if(i == j) {
   //ANY CODE HERE WILL EXECUTE
   //BECAUSE i (8) IS EQUAL TO j (8)
}
```

Example If Statement (==) - chars

```
char c = 'A';
char d = 'A';

if(c == d) {
   //ANY CODE HERE WILL EXECUTE
   //BECAUSE c ('A') IS EQUAL TO d ('A')
}
```

Example If Statement (==) - chars

```
char c = 'A';
char d = 'Z';

if(c == d) {
   //ANY CODE HERE WILL NOT EXECUTE
   //BECAUSE c ('A') IS NOT EQUAL TO d ('Z')
}
```

Example If Statement (==) - booleans

```
boolean b1 = true;
boolean b2 = true;

if(b1 == b2) {
   //ANY CODE HERE WILL EXECUTE
   //BECAUSE b1 (true) IS EQUAL TO b2 (true)
}
```

Example If Statement (==) - booleans

```
boolean b1 = true;
boolean b2 = false;

if(b1 == b2) {
   //ANY CODE HERE WILL NOT EXECUTE
   //BECAUSE b1 (true) IS NOT EQUAL TO b2 (false)
}
```

Example If Statement (!=) - Numbers

```
int i = 8;
int j = 10;

if(i != j) {
   //ANY CODE HERE WILL EXECUTE
   //BECAUSE i (8) IS NOT EQUAL TO j (10)
}
```

Example If Statement (!=) - chars

```
char c = 'A';
char d = 'Z';

if(c != d) {
   //ANY CODE HERE WILL EXECUTE
   //BECAUSE c ('A') IS NOT EQUAL TO d ('Z')
}
```

Example If Statement (!=) - booleans

```
boolean b1 = true;
boolean b2 = false;

if(b1 != b2) {
   //ANY CODE HERE WILL EXECUTE
   //BECAUSE b1 (true) IS NOT EQUAL TO b2 (false)
}
```

Example If Statement

```
int i = 8;
int j = 8;

if(i + j == 20) {
   //WILL ANY CODE HERE EXECUTE?
}
```

Example If Statement

```
int i = 8;
int j = 8;

if(i + 2 >= j - 1) {
   //WILL ANY CODE HERE EXECUTE?
}
```

Booleans and If Statements

```
boolean b1 = true;

if(b1 == true) {
   //ANY CODE HERE WILL EXECUTE
   //BECAUSE b1 IS TRUE
}
```

Booleans and If Statements

```
boolean b1 = true;

if(b1) {
   //ANY CODE HERE WILL EXECUTE
   //BECAUSE b1 IS TRUE
}
```

Booleans and If Statements

```
boolean b1 = false;

if(b1) {
   //ANY CODE HERE WILL NOT EXECUTE
   //BECAUSE b1 IS FALSE
}
```

Booleans and If Statements

```
boolean b1 = false;

if(b1 != true) {
   //ANY CODE HERE WILL EXECUTE
   //BECAUSE b1 (false) IS NOT EQUAL TO TRUE
}
```

Booleans and If Statements - Negation

- ! – Negation Operator (“Not” Operator)
  - Inverts Boolean values.

```
boolean b1 = false;

if(!b1) {
   //ANY CODE HERE WILL EXECUTE
   //BECAUSE b1 (false) IS TRUE WHEN NEGATED
}
```

If-Else Structures

- An ***else clause*** is a set of instructions that will only execute when its associated if statement's Boolean expression evaluates to false.

```
if(Boolean Expression 1) {
    //code that will be
    //executed if the Boolean Expression
    //evaluates to true
}
else {
    //code that will be executed if
    //Boolean Expression 1 is false
}
```

If-Else Structures

```
int i = 10;

if(i < 9) {
   //ANY CODE HERE WILL NOT EXECUTE
}
else {
   //ANY CODE HERE WILL EXECUTE INSTEAD
}
```

If-Else Structures

```
char c = 'A';

if(c == 'B') {
   //ANY CODE HERE WILL NOT EXECUTE
}
else {
   //ANY CODE HERE WILL EXECUTE INSTEAD
}
```

If-Else Structures

```
boolean b = false;

if(b) {
   //ANY CODE HERE WILL NOT EXECUTE
}
else {
   //ANY CODE HERE WILL EXECUTE INSTEAD
}
```

If-Else-If Structures

- An ***else-if clause*** is an additional if statement that allows testing alternative Boolean expressions.

```
if(Boolean Expression 1) {
    //code that will be
    //executed if the Boolean Expression
    //evaluates to true
}
else if(Boolean Expression 2) {
    //code that will be executed if Boolean Expression 1 is false
    //and this Boolean Expression 2 evaluates to True
}
```

If-Else-If Structures

- An ***else-if clause*** is an additional if statement that allows testing alternative Boolean expressions.
  - Allows multiple conditions to be tested before executing code.
- **Executes the first condition that evaluates to true.**

```
if(condition1){
  //CODE TO EXECUTE IF THE CONDITION IS TRUE
}
else if(condition2){
  //IF THE FIRST CONDITION WAS FALSE
  //ANY CODE HERE WILL BE EXECUTED IF THIS
  //CONDITION (condition2) IS TRUE
}
else {
  //IF ALL CONDITIONS ABOVE WERE FALSE
  //ANY CODE HERE WILL BE EXECUTED
}
```

If-Else-If Structures

```
char c = 'A';

if(c == 'B') {
   //ANY CODE HERE WILL NOT EXECUTE
}
else if(c == 'A') {
   //ANY CODE HERE WILL EXECUTE
}
else {
   //ANY CODE HERE WILL NOT EXECUTE
}
```

If-Else-If Structures

```
int i = 15;

if(i > 15) {
  //ANY CODE HERE WILL NOT EXECUTE
}
else if(i < 15) {
  //ANY CODE HERE WILL NOT EXECUTE
}
else if(i == 15) {
  //ANY CODE HERE WILL EXECUTE
}
else {
  //IS THIS “ELSE” EVEN NECESSARY?
}
```

If, If-Else, and If-Else-If Rules

- If Statements
  - **Must** always be first.
  - May be followed by any number of else ifs.
  - May be followed by one else.
- Else If
  - Optional.
  - **Must** follow an if statement or else if.
  - No limit to the number of else ifs.
  - May be followed by one else.
- Else
  - Optional.
  - **Must** follow an if statement or else if.
  - Only one else.
  - **Always** last.

String Comparison

- Strings should be compared using its equals method.

- Strings can be compared using == or !=, but it’s not always a good idea to do so.

String Comparison

```
String catString = "Cats";
String dogString = "Dogs";

if(catString.equals(dogString)) {
   System.out.println("The Strings are equal");
}
else {
   System.out.println("The Strings are not equal");
}
```

String Comparison - Negation

```
String catString = "Cats";
String dogString = "Dogs";

if(!catString.equals(dogString)) {
   System.out.println("The Strings are not equal");
}
else {
   System.out.println("The Strings are equal");
}
```

String Comparison – Ignore Case

```
String catUpper = "CATS";
String catLower = "cats";

if(catUpper.equalsIgnoreCase(catLower)) {
   System.out.println("The Strings are equal");
}
else {
   System.out.println("The Strings are not equal");
}
```

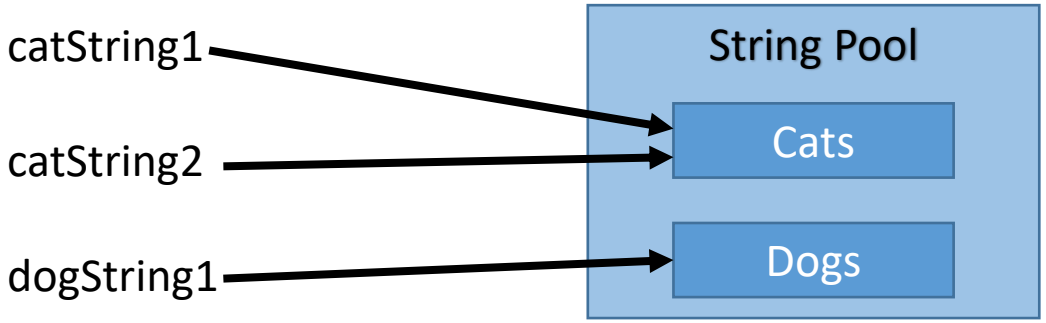
String Comparison using ==

```
String catString1 = "Cats";
String catString2 = "Cats";

if(catString1 == catString2) {
   System.out.println("The Strings are equal");
}
```

- The above code will work. That is, the condition in the if statement will evaluate to true, but not for the reasons you may think.
- To understand why it is a bad idea to compare the equality of two Strings using == or !=, we must first have a better understanding of how String literals work.

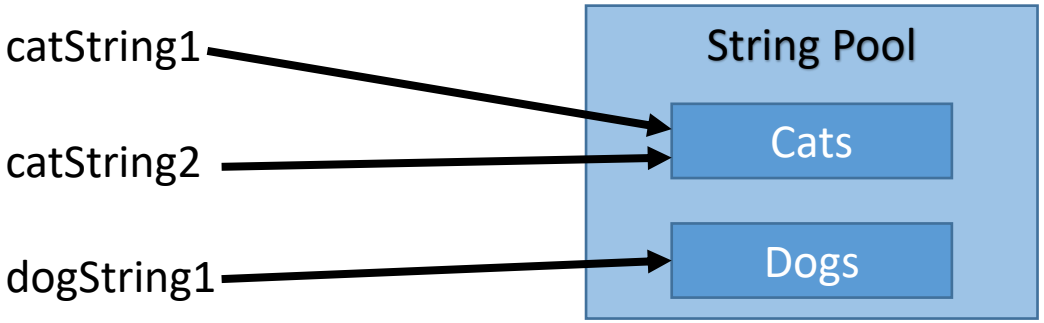
String Comparison using ==

```
String dogString1 = "Dogs";
String catString1 = "Cats";
String catString2 = "Cats";
```



- When you assign a String a literal, that literal gets added to a pool.
- When you assign another String the same literal, it references (or points to) the same value in the pool.

String Comparison using ==




```
if(catString1 == catString2) {
   System.out.println("The Strings are equal");
}
```

- When we compare catString1 and catString2 using the equality operator, it is testing if the **reference** is equal, **NOT the value**.

String Comparison using ==

```
String catString1 = "cats";
String catString2 = "CATS";

if(catString1 == catString2.toLowerCase()) {
   System.out.println("The Strings are equal");
}
```

- The condition in the above if statement actually returns false. Why?
- Because the toLowerCase method returns a new String (not a literal from the pool), the reference is different.
- When it comes to objects (remember, Strings are objects), comparing with == only compares **reference**, **not value**.

String Comparison – Length

```
String catString = "Cat";

if(catString.length() == 3) {
   //ANY CODE HERE WILL EXECUTE
}
```

startsWith(String)

- The startsWith method checks to see if the String begins with the String you provide it.
- Takes one parameter, a String, that will be tested against the beginning of the String's value.
- Returns true (the String did begin with that sequence) or false (the String did not begin with that sequence)

```
String hello = "Hello World!";

if(hello.startsWith("H")) {
   //ANY CODE HERE WILL EXECUTE
}
```

startsWith(String) (Another Example)

- The last example just used a single character. Here we use a whole word.

```
String hello = "Hello World!";

if(hello.startsWith("Goodbye")) {
   //ANY CODE HERE WILL NOT EXECUTE
}
```

startsWith(String) (Another Example)

```
String hello = "Hello World!";

if(hello.startsWith("Hello W")) {
   //ANY CODE HERE WILL EXECUTE
}
```

String.startsWith(String) – Case Sensitive!!

- The startsWith method is case sensitive.

```
String hello = "Hello World!";

if(hello.startsWith("h")) {
   //ANY CODE HERE WILL NOT EXECUTE
}
```

String.endsWith(String)

- Similar the startsWith method, the endsWith method tests if the String *ends* with a particular character sequence.

```
String hello = "Hello World!";

if(hello.endsWith("!")) {
   //ANY CODE HERE WILL EXECUTE
}
```

endsWith(String) (Another Example)

```
String hello = "Hello World!";

if(hello.endsWith("World")) {
   //ANY CODE HERE WILL NOT EXECUTE
}
```

String methods

Method	Return Type	Description	Possible Exceptions
equals(String)	boolean	Returns true if the strings are equal, false if not equal; Case sensitive.	None
equalsIgnoreCase(String)	boolean	Returns true if the strings are equal, false if not equal; Case insensitive.	None
length()	int	Returns the length of the String (number of characters; includes symbols and whitespace)	None
startsWith(String)	boolean	Returns true if the String begins with the supplied String, false if it does not.	None
endsWith(String)	boolean	Returns true if the String ends with the supplied String, false if it does not.	None

Switch Structures

- Works with byte, int, char, short, and Strings.

```
switch(value) {
    case valueThatMatches: Code to execute
}
```

Switch Structures

```
int myNumber = 2;

switch(myNumber) {
  case 0:  //ANY CODE HERE EXECUTES IF myNumber
           //WAS EQUAL TO 0
  case 1:  //ANY CODE HERE EXECUTES IF myNumber
           //WAS EQUAL TO 1
  default: //ANY CODE HERE IS EXECUTED IF NO
           //OTHER CASES WERE PREVIOUSLY MATCHED
}
```

Switch Structures

```
int myNumber = 4;
switch(myNumber) {
  case 5:  System.out.println("The number ");
           System.out.println("is five.");
  case 6:  System.out.println("The number ");
           System.out.println("is six.");
  default: System.out.println("This is");
           System.out.println("the default.");
}
```

```
This is
the default.
```

The value 4 doesn't match any of the cases, so the default case is executed.

Switch Structures

```
int myNumber = 5;
switch(myNumber) {
  case 5:  System.out.println("The number ");
           System.out.println("is five.");
  case 6:  System.out.println("The number ");
           System.out.println("is six.");
  default: System.out.println("This is");
           System.out.println("the default.");
}
```

```
The number
is five.
The number
is six.
This is
the default.
```

The value 5 matched a case, so that case's code is executed.
But it doesn't stop there… it keeps going!

Switch Structures

```
int myNumber = 6;
switch(myNumber) {
  case 5:  System.out.println("The number ");
           System.out.println("is five.");
  case 6:  System.out.println("The number ");
           System.out.println("is six.");
  default: System.out.println("This is");
           System.out.println("the default.");
}
```

```
The number
is six.
This is
the default.
```

Break Statement

```
break;
```

- When encountered, the switch will immediately stop where it is.

Switch Structures – break

```
int myNumber = 6;
switch(myNumber) {
  case 5:  System.out.println("The number ");
           System.out.println("is five.");
  case 6:  System.out.println("The number ");
           System.out.println("is six.");
           break;
  default: System.out.println("This is");
           System.out.println("the default.");
}
```

```
The number
is six.
```

A break statement will stop the switch's execution, and prevent it from continuing on.

Switch Structures – break

```
int myNumber = 5;
switch(myNumber) {
  case 5:  System.out.println("The number ");
           System.out.println("is five.");
           break;
  case 6:  System.out.println("The number ");
           System.out.println("is six.");
           break;
  default: System.out.println("This is");
           System.out.println("the default.");
}
```

```
The number
is five.
```

Switch Structures – break

```
String moneyString = "USD";
switch(moneyString) {
  case "MXN": System.out.print("Peso");
              break;
  case "EUR": System.out.print("Euro");
              break;
  case "USD": System.out.print("US Dollar");
              break;
  default:    System.out.print("Unknown Currency");
}
```

```
US Dollar
```

Switch Structures

- A default case is not required.
- No limit to the number of cases.
- An If-Else-If structure allows for only one execution path
  - The first condition that is true; subsequent conditions could also be true but the code won't be executed.
- A switch allows for multiple execution paths.
  - Without a break statement, the switch keeps falling through to the next case.

Conditional (Ternary) Operator (?:)

- Shorthand If-Else statement
  - Also called an *inline if statement*
- Uses three operands.

condition ? then : else

Conditional Operator (?:)

```
int fiveItems = 5;
int tenItems = 10;
boolean wantsTenItems = false;
int choice = 0;

choice = wantsTenItems ? tenItems : fiveItems;

System.out.println(choice); //Prints 5
```

Conditional Operator (?:)

```
choice = wantsTenItems ? tenItems : fiveItems;
```

…is equivalent to…

```
if(wantsTenItems) {
    choice = tenItems;
}
else {
    choice = fiveItems;
}
```

Conditional Operator (?:)

```
int fiveItems = 5;
int tenItems = 10;
boolean wantsTenItems = true;
int choice = 0;

choice = wantsTenItems ? tenItems : fiveItems;

System.out.println(choice); //Prints 10
```

Conditional Operator (?:)

```
String stateName = "NEW JERSEY";
String stateAbbr = "NJ";
boolean fullStateName = true;
String choice = "";

choice = fullStateName ? stateName.toLowerCase() : stateAbbr.toLowerCase();

System.out.println(choice); //Prints new jersey
```

Conditional Operator (?:)

```
String stateName = "NEW JERSEY";
String stateAbbr = "NJ";
boolean fullStateName = false;
String choice = "";

choice = fullStateName ? stateName.toLowerCase() : stateAbbr.toLowerCase();

System.out.println(choice); //Prints nj
```

Conditional Operator (?:)

```
String stateName = "NEW JERSEY";
String stateAbbr = "NJ";
boolean fullStateName = false;
String choice = "";

System.out.println(fullStateName ? stateName.toLowerCase() : stateAbbr.toLowerCase()); //Prints nj
```