CS 200 - Programming I: Branches

Marc Renault

Department of Computer Sciences
University of Wisconsin – Madison

Fall 2019

TopHat Sec 3 (1:20 PM) Join Code: 682357
TopHat Sec 4 (3:30 PM) Join Code: 296444

Boolean Statements

Booleans

Primitive

- `boolean b;`
- Values: `true` or `false`
- Wrapper class: `Boolean`

Boolean Expressions

An expression that evaluates to a `boolean`. I.e. `true` or `false`.

Boolean Expressions

An expression that evaluates to a `boolean`. I.e. `true` or `false`.

Comparison Operators

- Binary (*`expr1 oper expr2`*):
  - `==` : Equal to
  - `!=` : Not equal to
  - `>` : Greater than
  - `<` : Less than
  - `>=` : Greater than or equal
  - `<=` : Less than or equal

Boolean Expressions

An expression that evaluates to a `boolean`. I.e. `true` or `false`.

Logical Operators

- Unary (*`oper expr`*):
  - `!` : Logical NOT
- Binary (*`expr1 oper expr2`*):
  - `&&` : Logical AND
  - `||` : Logical OR
- Don't confuse with the *bitwise* operators: `&`, `|`, and `^`.

Operator Precedence

Level	Operator	Description	Associativity
higher	(<expression>)	grouping with parentheses	left to right
	[ ] ( ) .	array index, method call, member access (dot operator)	left to right
	++ –	post-increment, post-decrement	left to right
	++ – + - !	pre-increment, unary plus/minus, logical negation	right to left
	(type) new	casting and creating object	right to left
	* / %	multiplication, division, modulus	left to right
	+ - +	addition, subtraction, concatenation	left to right
	< <= > >= instanceof	relational and Java's `instanceof` operator	left to right
	== !=	equality	left to right
	&&	conditional AND (short-circuits)	left to right
	||	conditional OR (short-circuits)	left to right
	? :	ternary conditional	right to left
lower	= += -= *= /= %=	assignment, arithmetic (compound) assignment	right to left

Truth Tables

a	b	$a \wedge b$	$a \vee b$	$a \oplus b$	$\neg a$
0	0				
0	1				
1	0				
1	1				

Logic Rules

- $\wedge$ AND (`&&`, `&`) – True only if both operands are true.
- $\vee$ OR (`||`, `|`) – True if either operand is true.
- $\oplus$ XOR (`^`) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- $\neg$ NOT (`!`) – Flips the truth value.

Truth Tables

a	b	$a \wedge b$	$a \vee b$	$a \oplus b$	$\neg a$
0	0	0			
0	1				
1	0				
1	1				

Logic Rules

- $\wedge$ AND (`&&`, `&`) – True only if both operands are true.
- $\vee$ OR (`||`, `|`) – True if either operand is true.
- $\oplus$ XOR (`^`) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- $\neg$ NOT (`!`) – Flips the truth value.

Truth Tables

a	b	$a \wedge b$	$a \vee b$	$a \oplus b$	$\neg a$
0	0	0	0		
0	1				
1	0				
1	1				

Logic Rules

- $\wedge$ AND (`&&`, `&`) – True only if both operands are true.
- $\vee$ OR (`||`, `|`) – True if either operand is true.
- $\oplus$ XOR (`^`) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- $\neg$ NOT (`!`) – Flips the truth value.

Truth Tables

a	b	$a \wedge b$	$a \vee b$	$a \oplus b$	$\neg a$
0	0	0	0	0	
0	1				
1	0				
1	1				

Logic Rules

- $\wedge$ AND (`&&`, `&`) – True only if both operands are true.
- $\vee$ OR (`||`, `|`) – True if either operand is true.
- $\oplus$ XOR (`^`) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- $\neg$ NOT (`!`) – Flips the truth value.

Truth Tables

a	b	$a \wedge b$	$a \vee b$	$a \oplus b$	$\neg a$
0	0	0	0	0	1
0	1				
1	0				
1	1				

Logic Rules

- $\wedge$ AND (`&&`, `&`) – True only if both operands are true.
- $\vee$ OR (`||`, `|`) – True if either operand is true.
- $\oplus$ XOR (`^`) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- $\neg$ NOT (`!`) – Flips the truth value.

Truth Tables

a	b	$a \wedge b$	$a \vee b$	$a \oplus b$	$\neg a$
0	0	0	0	0	1
0	1	0			
1	0				
1	1				

Logic Rules

- $\wedge$ AND (`&&`, `&`) – True only if both operands are true.
- $\vee$ OR (`||`, `|`) – True if either operand is true.
- $\oplus$ XOR (`^`) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- $\neg$ NOT (`!`) – Flips the truth value.

Truth Tables

a	b	$a \wedge b$	$a \vee b$	$a \oplus b$	$\neg a$
0	0	0	0	0	1
0	1	0	1		
1	0				
1	1				

Logic Rules

- $\wedge$ AND (`&&`, `&`) – True only if both operands are true.
- $\vee$ OR (`||`, `|`) – True if either operand is true.
- $\oplus$ XOR (`^`) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- $\neg$ NOT (`!`) – Flips the truth value.

Truth Tables

a	b	$a \wedge b$	$a \vee b$	$a \oplus b$	$\neg a$
0	0	0	0	0	1
0	1	0	1	1	
1	0				
1	1				

Logic Rules

- $\wedge$ AND (`&&`, `&`) – True only if both operands are true.
- $\vee$ OR (`||`, `|`) – True if either operand is true.
- $\oplus$ XOR (`^`) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- $\neg$ NOT (`!`) – Flips the truth value.

Truth Tables

a	b	$a \wedge b$	$a \vee b$	$a \oplus b$	$\neg a$
0	0	0	0	0	1
0	1	0	1	1	1
1	0				
1	1				

Logic Rules

- $\wedge$ AND (`&&`, `&`) – True only if both operands are true.
- $\vee$ OR (`||`, `|`) – True if either operand is true.
- $\oplus$ XOR (`^`) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- $\neg$ NOT (`!`) – Flips the truth value.

Truth Tables

a	b	$a \wedge b$	$a \vee b$	$a \oplus b$	$\neg a$
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0			
1	1				

Logic Rules

- $\wedge$ AND (`&&`, `&`) – True only if both operands are true.
- $\vee$ OR (`||`, `|`) – True if either operand is true.
- $\oplus$ XOR (`^`) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- $\neg$ NOT (`!`) – Flips the truth value.

Truth Tables

a	b	$a \wedge b$	$a \vee b$	$a \oplus b$	$\neg a$
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1		
1	1				

Logic Rules

- $\wedge$ AND (`&&`, `&`) – True only if both operands are true.
- $\vee$ OR (`||`, `|`) – True if either operand is true.
- $\oplus$ XOR (`^`) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- $\neg$ NOT (`!`) – Flips the truth value.

Truth Tables

a	b	$a \wedge b$	$a \vee b$	$a \oplus b$	$\neg a$
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	
1	1				

Logic Rules

- $\wedge$ AND (`&&`, `&`) – True only if both operands are true.
- $\vee$ OR (`||`, `|`) – True if either operand is true.
- $\oplus$ XOR (`^`) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- $\neg$ NOT (`!`) – Flips the truth value.

Truth Tables

a	b	$a \wedge b$	$a \vee b$	$a \oplus b$	$\neg a$
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1				

Logic Rules

- $\wedge$ AND (`&&`, `&`) – True only if both operands are true.
- $\vee$ OR (`||`, `|`) – True if either operand is true.
- $\oplus$ XOR (`^`) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- $\neg$ NOT (`!`) – Flips the truth value.

Truth Tables

a	b	$a \wedge b$	$a \vee b$	$a \oplus b$	$\neg a$
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1			

Logic Rules

- $\wedge$ AND (`&&`, `&`) – True only if both operands are true.
- $\vee$ OR (`||`, `|`) – True if either operand is true.
- $\oplus$ XOR (`^`) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- $\neg$ NOT (`!`) – Flips the truth value.

Truth Tables

a	b	$a \wedge b$	$a \vee b$	$a \oplus b$	$\neg a$
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1		

Logic Rules

- $\wedge$ AND (`&&`, `&`) – True only if both operands are true.
- $\vee$ OR (`||`, `|`) – True if either operand is true.
- $\oplus$ XOR (`^`) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- $\neg$ NOT (`!`) – Flips the truth value.

Truth Tables

a	b	$a \wedge b$	$a \vee b$	$a \oplus b$	$\neg a$
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	

Logic Rules

- $\wedge$ AND (`&&`, `&`) – True only if both operands are true.
- $\vee$ OR (`||`, `|`) – True if either operand is true.
- $\oplus$ XOR (`^`) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- $\neg$ NOT (`!`) – Flips the truth value.

Truth Tables

a	b	$a \wedge b$	$a \vee b$	$a \oplus b$	$\neg a$
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Logic Rules

- $\wedge$ AND (`&&`, `&`) – True only if both operands are true.
- $\vee$ OR (`||`, `|`) – True if either operand is true.
- $\oplus$ XOR (`^`) – *Exclusive OR*; True if one (but not both) operand(s) is true.
- $\neg$ NOT (`!`) – Flips the truth value.

TopHat Question 1

If `A` is 1 and `B` is 1, what is `!A || B`?

a. 1

b. 0

Bitwise Operations

Performs the logical operations on each of the corresponding bits of the operands.

Bitwise Operators

- Binary (*expr1 oper expr2*):
  - `&` – Bitwise AND
  - `|` – Bitwise OR
  - `^` – Bitwise XOR

Bitwise Operations

Performs the logical operations on each of the corresponding bits of the operands.
E.g. `7 & 5 = 5`
Consider the bits:

```
  111
& 101
-----
  101
```

Bitwise Operators

- Binary (*expr1 oper expr2*):
  - `&` – Bitwise AND
  - `|` – Bitwise OR
  - `^` – Bitwise XOR

TopHat Question 2

What is 7 | 5 ?

a. 12
b. 7
c. 5
d. 3

Comparing References

Reference Comparison

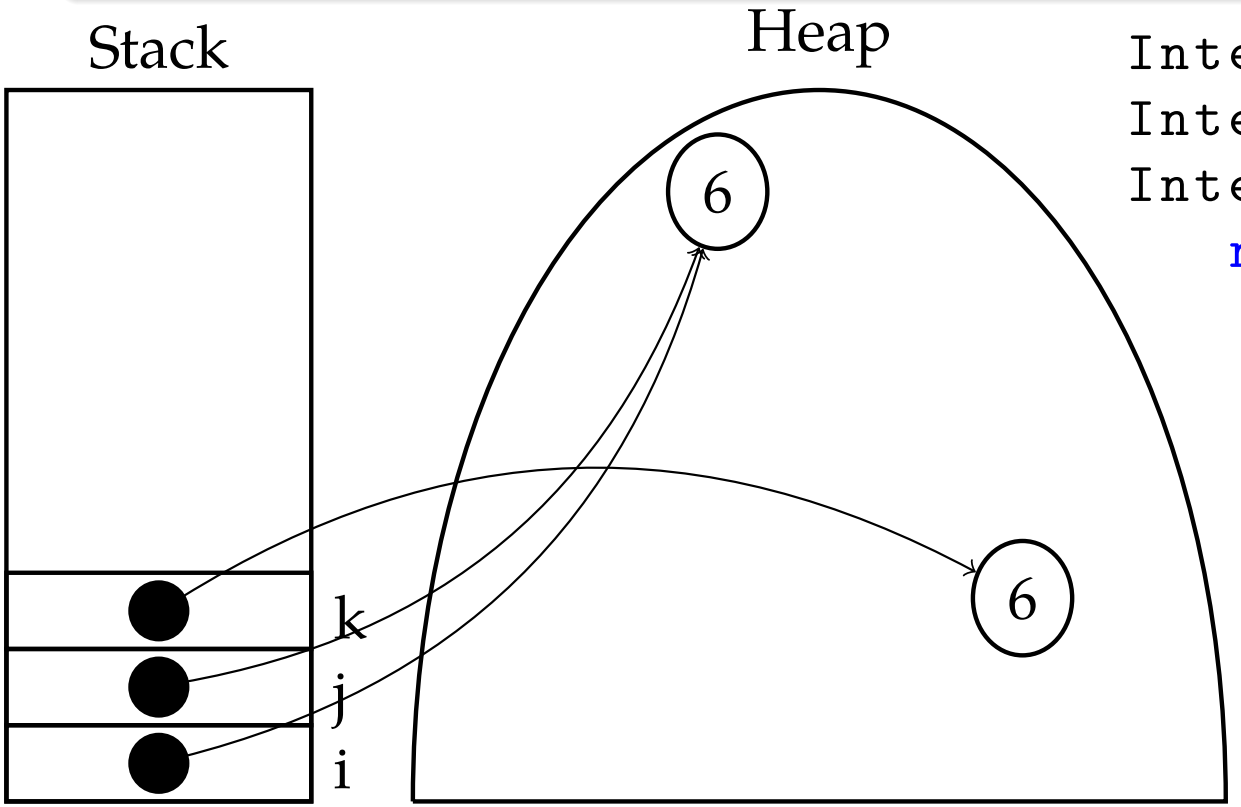
Equality Operator (==)

- Generally, the equality operator compares the values of two variables.
- Recall: value of a reference type is the referral information.

Reference Comparison

Equality Operator (==)

- Generally, the equality operator compares the values of two variables.
- Recall: value of a reference type is the referral information.

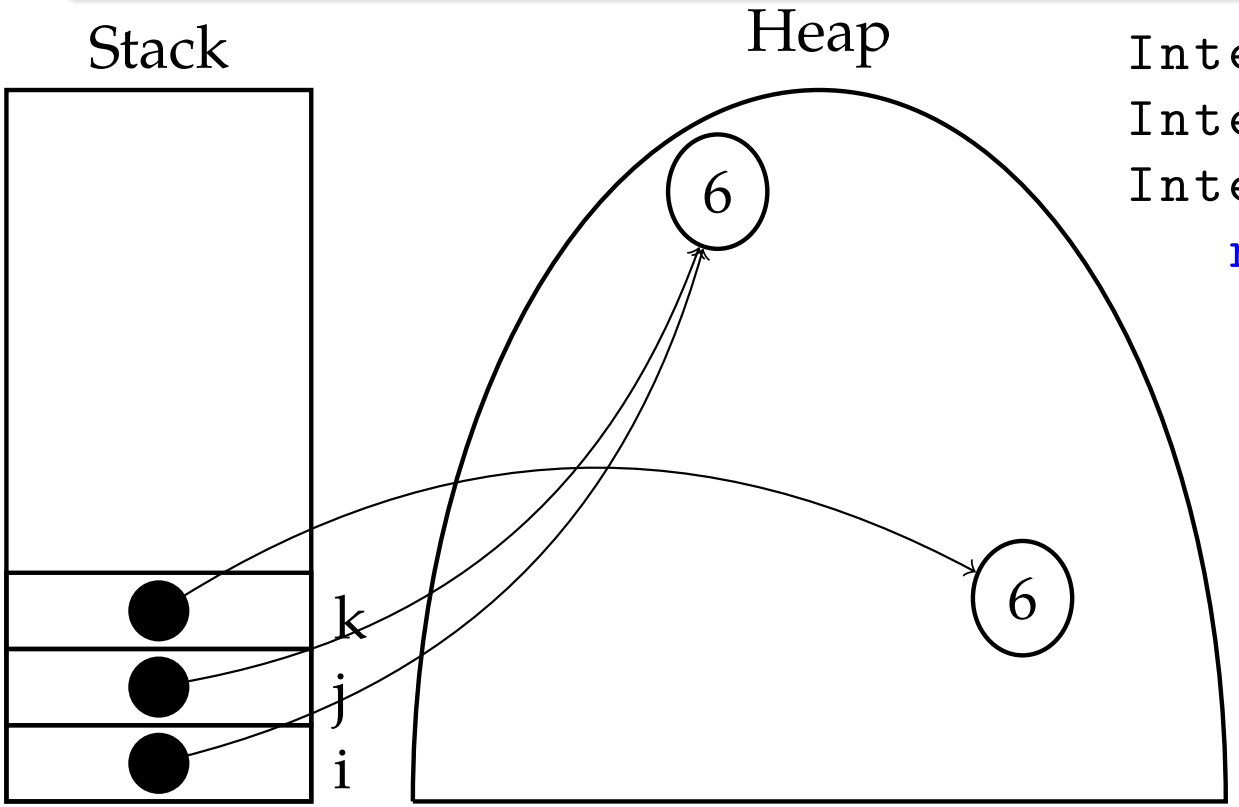



```
Integer i = 6;
Integer j = 6;
Integer k =
    new Integer(6);
```

Reference Comparison

Equality Operator (==) [Resp: Inequality Operator (!=)]

- Generally, the equality operator compares the values of two variables.
- Recall: value of a reference type is the referral information.




```
Integer i = 6;
Integer j = 6;
Integer k =
    new Integer(6);
```

TopHat Question 3

What is the output?

```
Integer i = 5;
Integer j = 5;
Integer k = new Integer(5);

System.out.print(i == j);
System.out.print(j == k);
```

Comparing Values of Primitive Wrappers

```
Integer i = 6;
Integer j = 5;
Integer k = new Integer(6);
int m = 5;
```

Instance Methods for Comparison

- `equals(`*`otherObj`*`)` – True if values of the objects are equal. False otherwise.
  E.g. `i.equals(k) // true`

Comparing Values of Primitive Wrappers

```
Integer i = 6;
Integer j = 5;
Integer k = new Integer(6);
int m = 5;
```

Instance Methods for Comparison

- `equals(`*`otherObj`*`)` – True if values of the objects are equal. False otherwise.
  E.g. `i.equals(k) // true`
- `compareTo(`*`otherObj`*`)` – 0 if equal, negative if less than *`otherObj`*, and positive if greater than *`otherObj`*.
  E.g. `i.compareTo(j) // +ve`

Comparing Values of Primitive Wrappers

```
Integer i = 6;
Integer j = 5;
Integer k = new Integer(6);
int m = 5;
```

Instance Methods for Comparison

- `equals(`*`otherObj`*`)` – True if values of the objects are equal. False otherwise.
  E.g. `i.equals(k) // true`
- `compareTo(`*`otherObj`*`)` – 0 if equal, negative if less than *`otherObj`*, and positive if greater than *`otherObj`*.
  E.g. `i.compareTo(j) // +ve`

Other Comparison Operators

- (In)Equality exception: `j == m // true`
- Other operators work as expected: `<`, `<=`, `>`, `>=`.

TopHat Question 4

What is the output?

```
Scanner sc = new Scanner("foo");
String s = "foo";
String t = sc.next();
System.out.print(s == t);
```

String Comparison

Only Instance Methods

- `equals(`*`otherStr`*`)`
- `compareTo(`*`otherStr`*`)`

String Comparison

Only Instance Methods

- `equals(`*`otherStr`*`)`
- `compareTo(`*`otherStr`*`)`

Lexicographical Comparison

Starting from index 0 and compare each pair of characters at index x according to the following rules:

1. The character with the smaller Unicode value is considered smaller lexicograpically.
2. No character is smaller than an existing character. (E.g. `"foo"` is smaller than `"foobar"`)

String Comparison

Only Instance Methods

- `equals(`*`otherStr`*`)`
- `compareTo(`*`otherStr`*`)`
- `equalsIgnoreCase(`*`otherStr`*`)`
- `compareToIgnoreCase(`*`otherStr`*`)`

Lexicographical Comparison

Starting from index 0 and compare each pair of characters at index x according to the following rules:

1. The character with the smaller Unicode value is considered smaller lexicograpically.
2. No character is smaller than an existing character. (E.g. `"foo"` is smaller than `"foobar"`)

TopHat Question 5

Arrange these words in lexicographical order (smallest to largest):

a. Javascript
b. C
c. Java
d. C++
e. C#

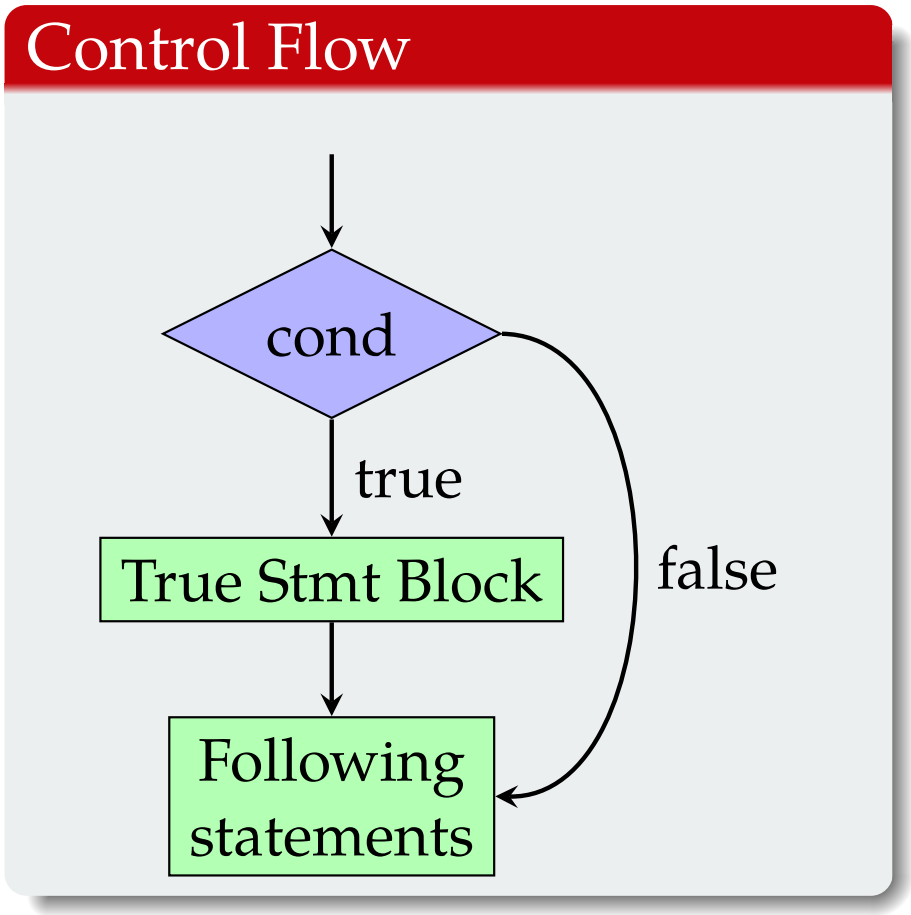
Conditional Statements

If Statement

```
if (cond) {
    trueStatement;
        .
        .
        .
    lastTrueStatement;
}
```

If Statement

```
if (cond) {
    trueStatement;
        .
        .
        .
    lastTrueStatement;
}
```

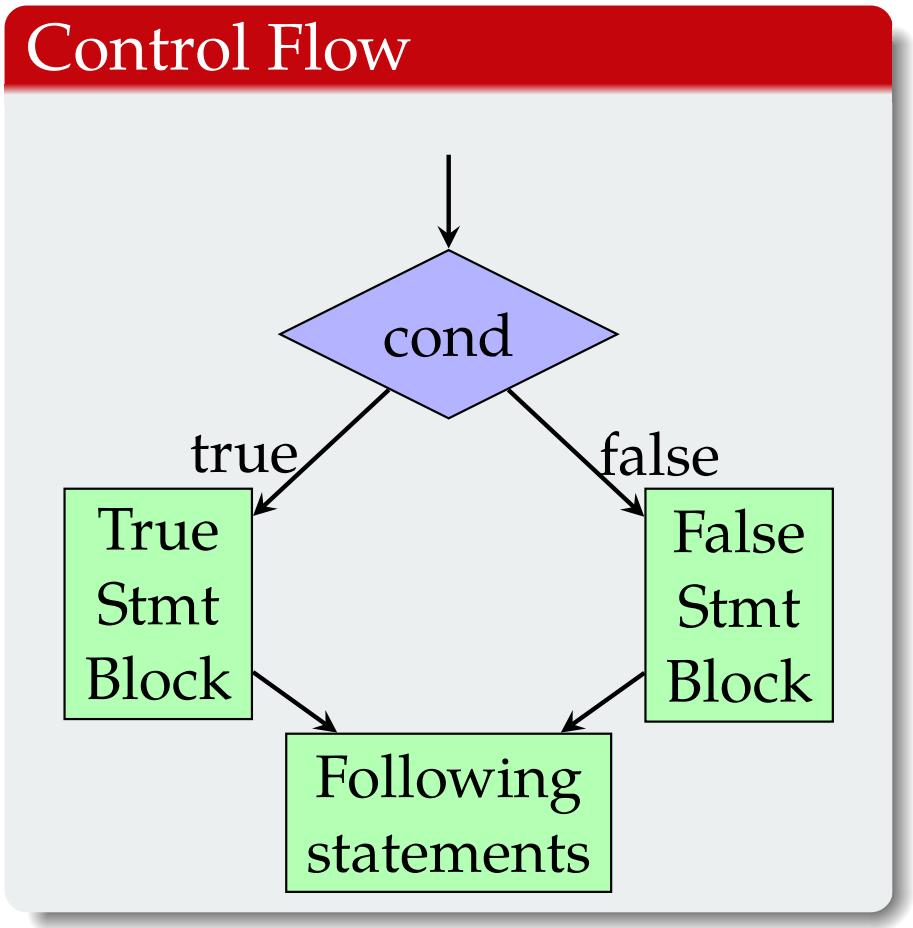


If-Else Statement

```
if (cond) {
    trueStatement;
        .
        .
        .
    lastTrueStatement;
}
else {
    falseStatement;
        .
        .
        .
    lastFalseStatement;
}
```

If-Else Statement

```
if (cond) {
    trueStatement;
        .
        .
        .
    lastTrueStatement;
}
else {
    falseStatement;
        .
        .
        .
    lastFalseStatement;
}
```



TopHat Question 6

What is printed out in be following code block:

```
boolean toBe = false;
if(toBe || ! toBe) {
    System.out.print("Tautology");
}
else {
    System.out.print("Contradiction");
}
```

TopHat Question 7

What is the boolean expression to complete the following code?

Assume that `i` is an `int` variable. Replace `??????` in the code with the appropriate boolean expression.

```
if( ?????? ) {
    System.out.println(i + " is divisible by 3");
}
else {
    System.out.println(i + " is not divisible by 3");
}
```

TopHat Question 8

What is the output?

```
int i = 3;
if ((i = 5) > 4) {
    System.out.print("a");
} else {
    System.out.print("b");
}
```

If-Else If Statement

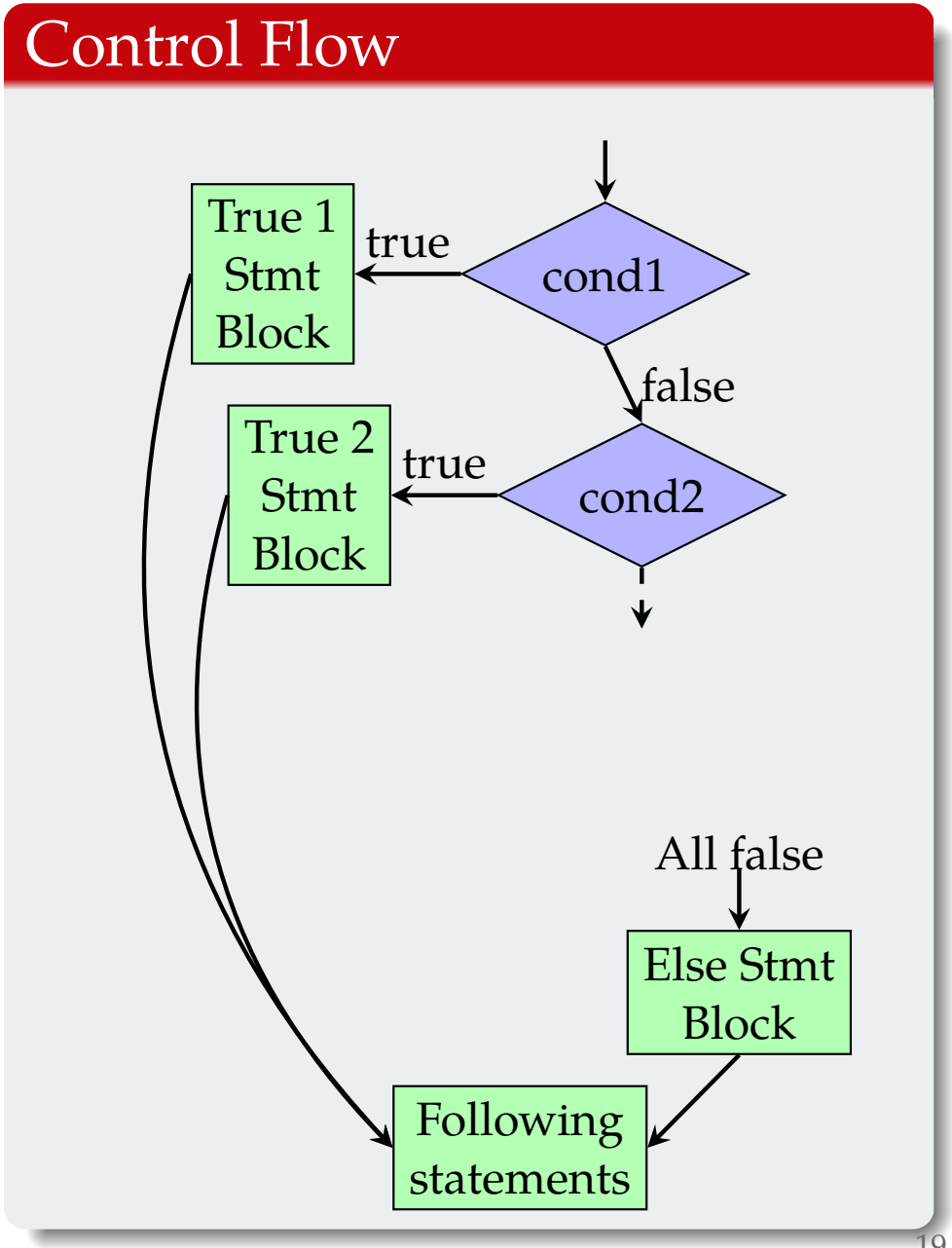
Ending Else

```
if (cond1) {
    trueStatements1;
}
else if (cond2) {
    trueStatements2;
}
.
.
.

else {
    allFalseStatements;
}
```

If-Else If Statement

Ending Else

```
if (cond1) {
    trueStatements1;
}
else if (cond2) {
    trueStatements2;
}
.
.
.

else {
    allFalseStatements;
}
```



If-Else If Statement

No Ending Else

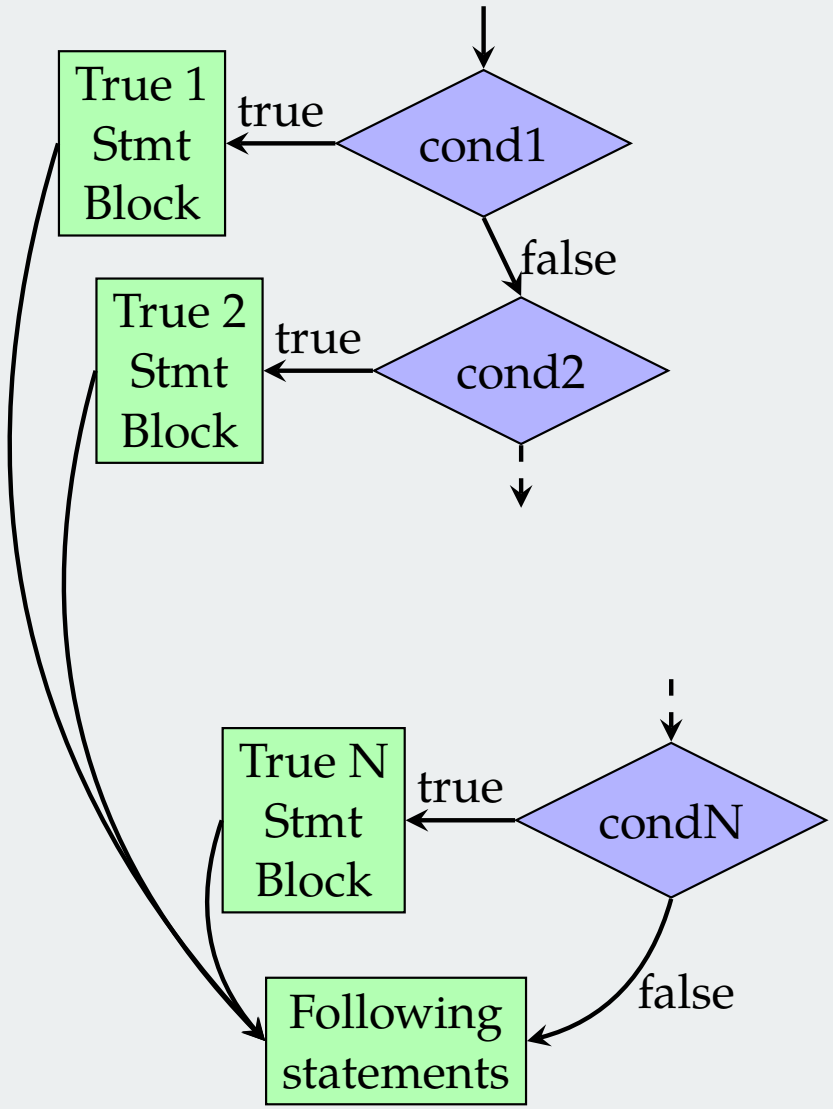
```
if (cond1) {
    trueStatements1;
}
else if (cond2) {
    trueStatements2;
}
.
.
.

else if (condN) {
    trueStatementsN;
}
```

If-Else If Statement

No Ending Else

```
if (cond1) {
    trueStatements1;
}
else if (cond2) {
    trueStatements2;
}
.
.
.

else if (condN) {
    trueStatementsN;
}
```

Control Flow



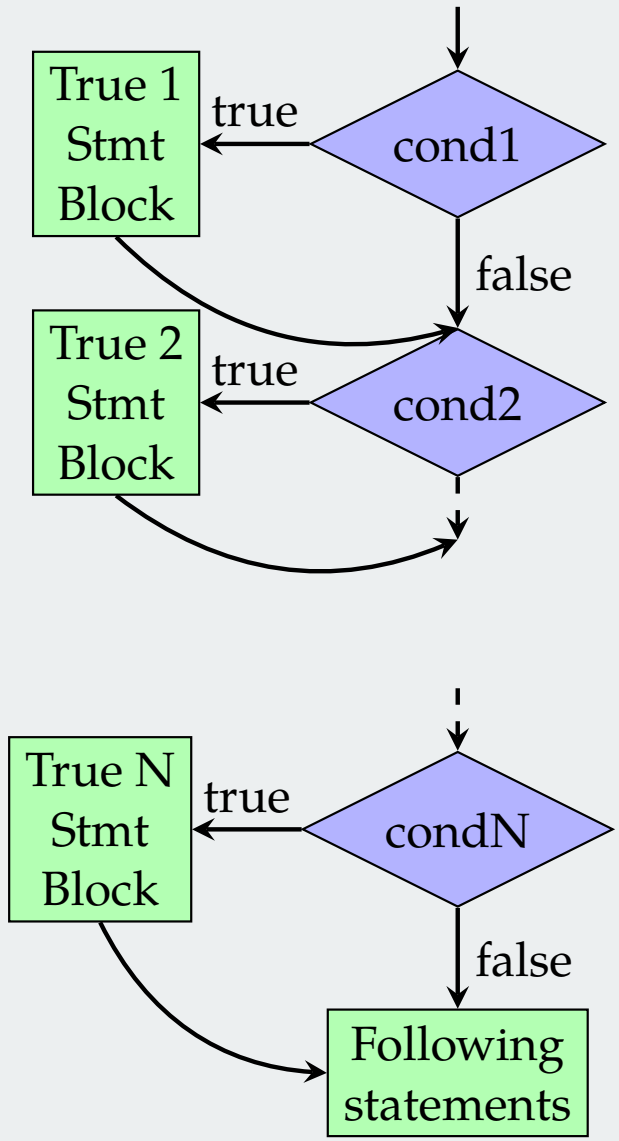
Multiple Ifs

```
if (cond1) {
    trueStatements1;
}
if (cond2) {
    trueStatements2;
}
.
.
.

if (condN) {
    trueStatementsN;
}
```

Multiple Ifs

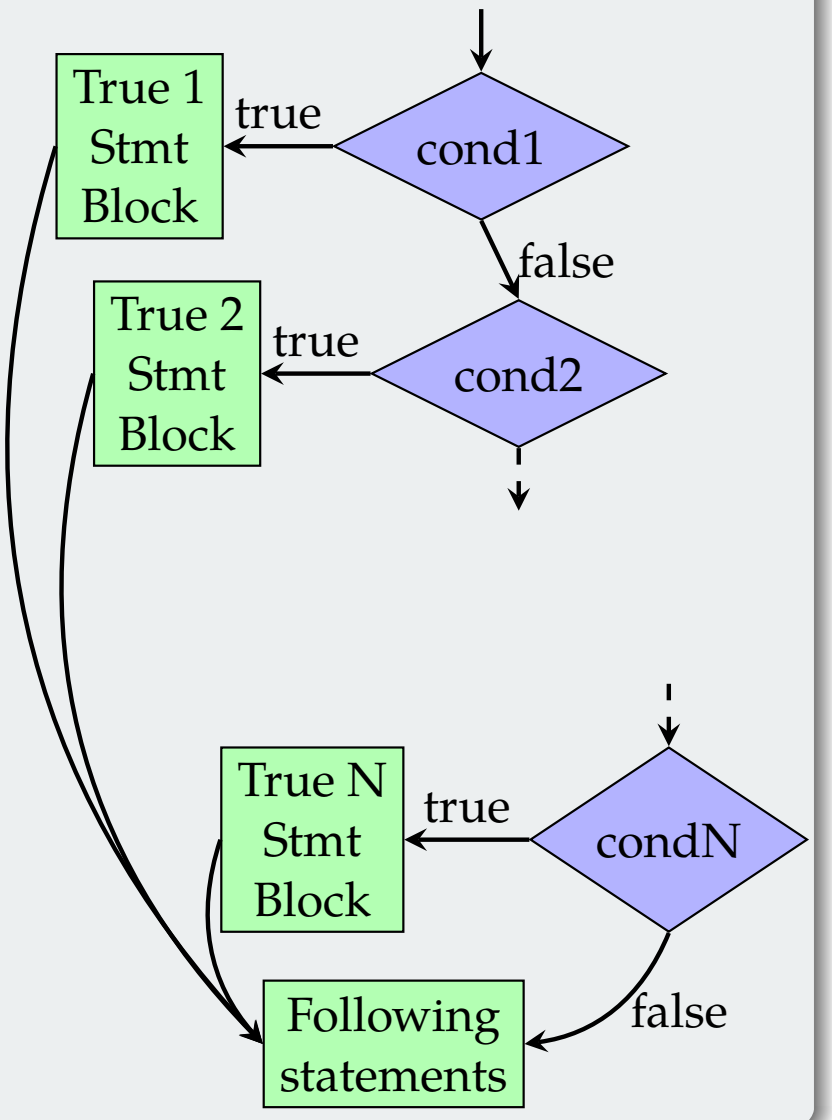
```
if (cond1) {
    trueStatements1;
}
if (cond2) {
    trueStatements2;
}
.
.
.

if (condN) {
    trueStatementsN;
}
```

Control Flow

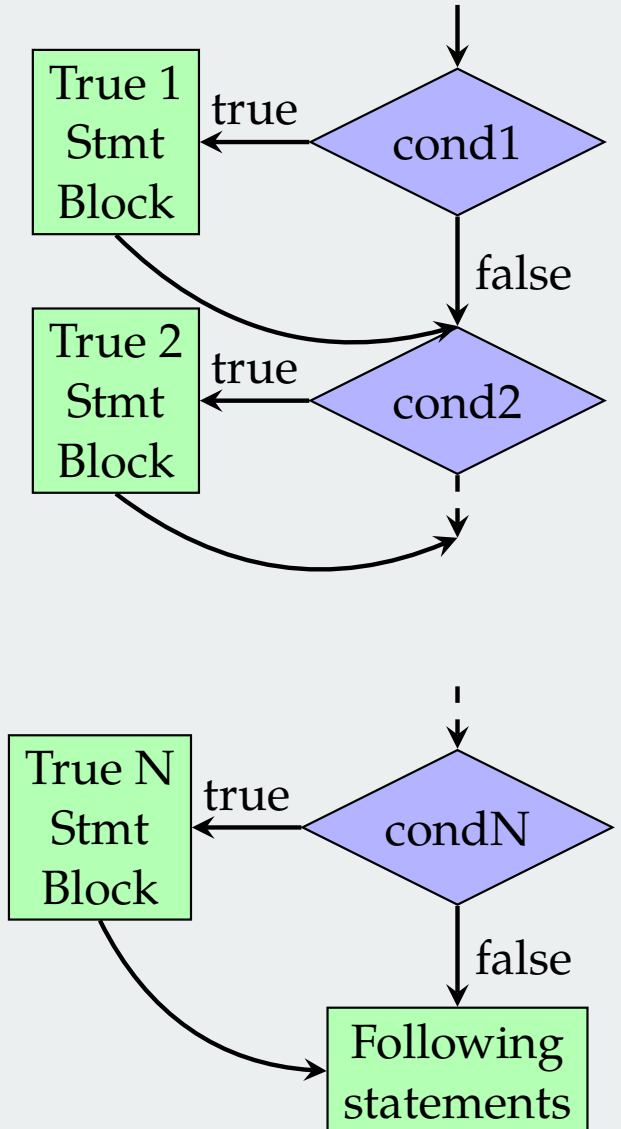


If-Else If vs Multiple Ifs

Control Flow – If-Else If



Control Flow – Multiple Ifs



TopHat Question 9

What does the following code print?

```
int v1 = 3, v2 = 6;
if (v1 + 2 < v2) System.out.print(" tic ");
else if (v1 + 4 < v2) System.out.print(" tac ");
else if (v1 + 4 > v2) System.out.print(" toe ");
```

TopHat Question 10

What does the following code print?

```
int v1 = 3, v2 = 6;
if (v1 + 2 < v2) System.out.print(" tic ");
if (v1 + 4 < v2) System.out.print(" tac ");
if (v1 + 4 > v2) System.out.print(" toe ");
```

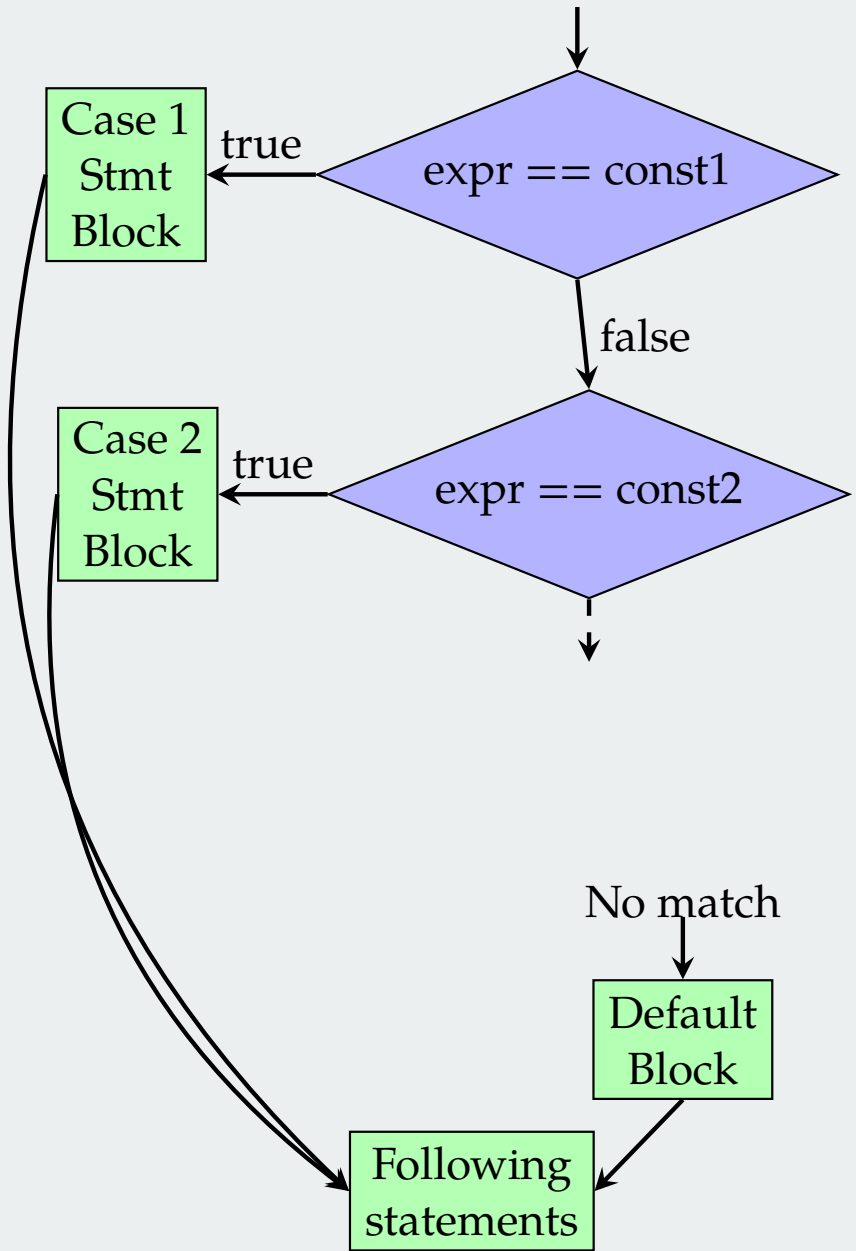
Switch Statements

```
switch (expr) {
    case const1:
        case1Statements;
        break;

    case const2:
        case2Statements;
        break;
    .
    .
    .
    default:
        defStatements;
        break;
}
```

Switch Statements

```
switch (expr) {
    case const1:
        case1Statements;
        break;

    case const2:
        case2Statements;
        break;
    .
    .
    .
    default:
        defStatements;
        break;
}
```

Control Flow

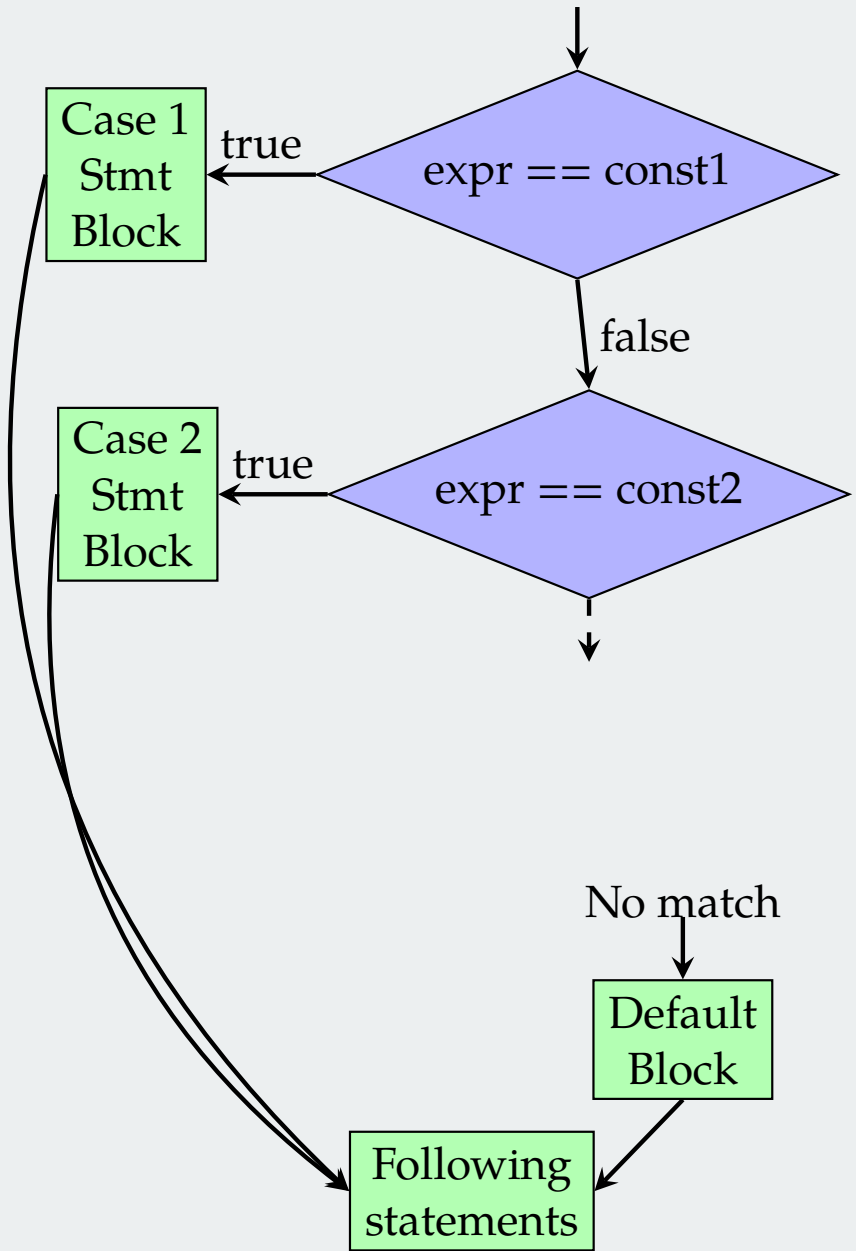


Switch Statements

Control flow depends on `break;`

```
switch (expr) {
    case const1:
        case1Statements;
        break;

    case const2:
        case2Statements;
        break;
    .
    .
    .
    default:
        defStatements;
        break;
}
```

Control Flow

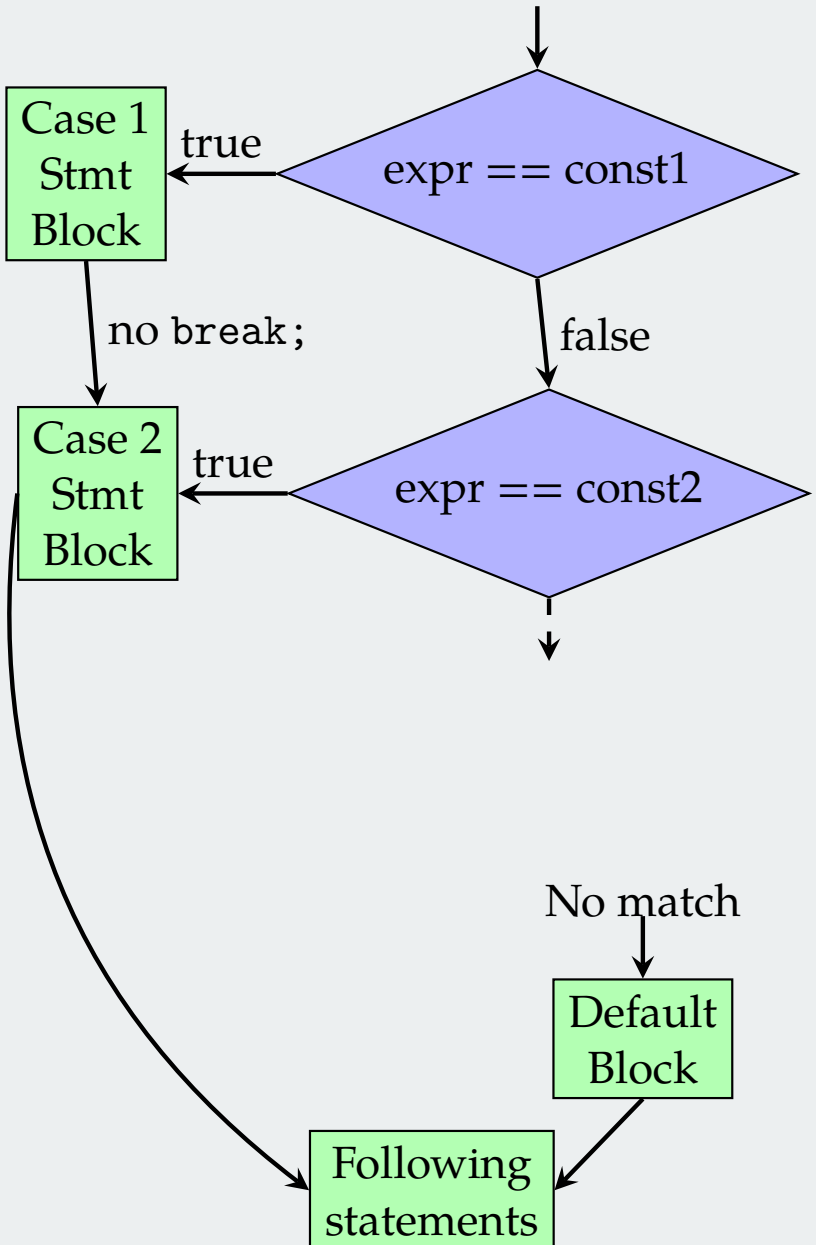


Switch Statements

Control flow depends on `break;`

```
switch (expr) {
    case const1:
        case1Statements;

    case const2:
        case2Statements;
        break;
    .
    .
    .
    default:
        defStatements;
        break;
}
```

Control Flow



Switch Statements Keys

Rules

- Finds the *first* match going from the top to the bottom in order.
- *Fall through* – A case without a `break` will fall through to the next case's statements.

Switch Statements Keys

Rules

- Finds the *first* match going from the top to the bottom in order.
- *Fall through* – A case without a `break` will fall through to the next case's statements.

Good Practices

- Only use fall though when the cases execute the same code.
- Generally, it is best to always have a `default` case at the end.

If-Else if and Switch Statements

```
if (i == 1) {
    System.out.println("one");
}
else if (i == 2) {
    System.out.println("two");
}
else {
    System.out.println("Not 1 or 2");
}
```

If-Else if and Switch Statements

```
if (i == 1) {
    System.out.println("one");
}
else if (i == 2) {
    System.out.println("two");
}
else {
    System.out.println("Not 1 or 2");
}
```

```
switch (i) {
   case 1:
      System.out.println("one");
      break;
   case 2:
      System.out.println("two");
      break;
   default:
      System.out.println("Not 1 or 2");
      break;
}
```

TopHat Question 11

What does the following code print?

```
int month = 8;
switch (month) {
    case 1:  System.out.print("January");
             break;
    case 2:  System.out.print("February");
             break;
    case 3:  System.out.print("March");
             break;
    case 4:  System.out.print("April");
             break;
    case 5:  System.out.print("May");
             break;
    case 6:  System.out.print("June");
             break;
    case 7:  System.out.print("July");
             break;
    case 8:  System.out.print("August");
             break;
    case 9:  System.out.print("September");
             break;
    case 10: System.out.print("October");
             break;
    case 11: System.out.print("November");
             break;
    case 12: System.out.print("December");
             break;
    default: System.out.print("Invalid month");
             break;
}
```

TopHat Question 12

What does the following code print?

```
int month = 8;
switch (month) {
    case 1:  System.out.print("January");
    case 2:  System.out.print("February");
    case 3:  System.out.print("March");
    case 4:  System.out.print("April");
    case 5:  System.out.print("May");
    case 6:  System.out.print("June");
    case 7:  System.out.print("July");
    case 8:  System.out.print("August");
    case 9:  System.out.print("September");
    case 10: System.out.print("October");
    case 11: System.out.print("November");
    case 12: System.out.print("December");
             break;
    default: System.out.print("Invalid month");
             break;
```

Ordinal Abbreviation Exercise

Write a method that takes an integer as input and returns a string with the integer converted to an abbreviated ordinal. I.e. 1 becomes 1st, 2 becomes 2nd, 3 becomes, 3rd, etc...

Using Eclipse

Eclipse

Eclipse

- Integrated development environment (IDE) that we will use in the course.
- In 2017, 40.5% use Eclipse (48% use IntelliJ) [a]
- In 2016, 48.2% use Eclipse (43.6% use IntelliJ) [b]

[a] Source: `http://www.baeldung.com/java-in-2017`
[b] Source: `http://www.baeldung.com/java-ides-2016`

Eclipse

Eclipse

- Integrated development environment (IDE) that we will use in the course.
- In 2017, 40.5% use Eclipse (48% use IntelliJ) [a]
- In 2016, 48.2% use Eclipse (43.6% use IntelliJ) [b]

[a] Source: `http://www.baeldung.com/java-in-2017`
[b] Source: `http://www.baeldung.com/java-ides-2016`

Installing and Using

- Installation: `https://cs200-www.cs.wisc.edu/wp/install-eclipse/`
- Tutorials: `https://cs200-www.cs.wisc.edu/wp/resources/eclipse-tutorial/`

Style Guide

`https://cs200-www.cs.wisc.edu/wp/style-guide/`
Examples:

- `http://pages.cs.wisc.edu/~cs200/resources/Frame.java`
- `http://pages.cs.wisc.edu/~cs200/resources/Circle.java`

Key Elements

- File Comment Header
- Class Comment Header
- Method Comment Header
- Good names with proper formatting
- Proper use of whitespace

Further Reading

COMP SCI 200: Programming I
zyBooks.com, 2015.
zyBook code:
WISCCOMPSCI200Fall2019

- Chapter 4. Branches

Appendix

References

Image Sources I

https://brand.wisc.edu/web/logos/

http://www.zybooks.com/