SELECTIONS

- Statements that let you choose actions with alternative courses
- Selection statements use conditions that are Boolean expressions

-
  ```
  if (radius < 0) {
  System.out.println("Incorrect input");
  }
  else {
  area = radius * radius * 3.14159;
  System.out.println("Area is " + area);
  }
  ```

boolean Data Type

- declares a variable with the value either true or false
- The equality testing operator is two equal signs (==)

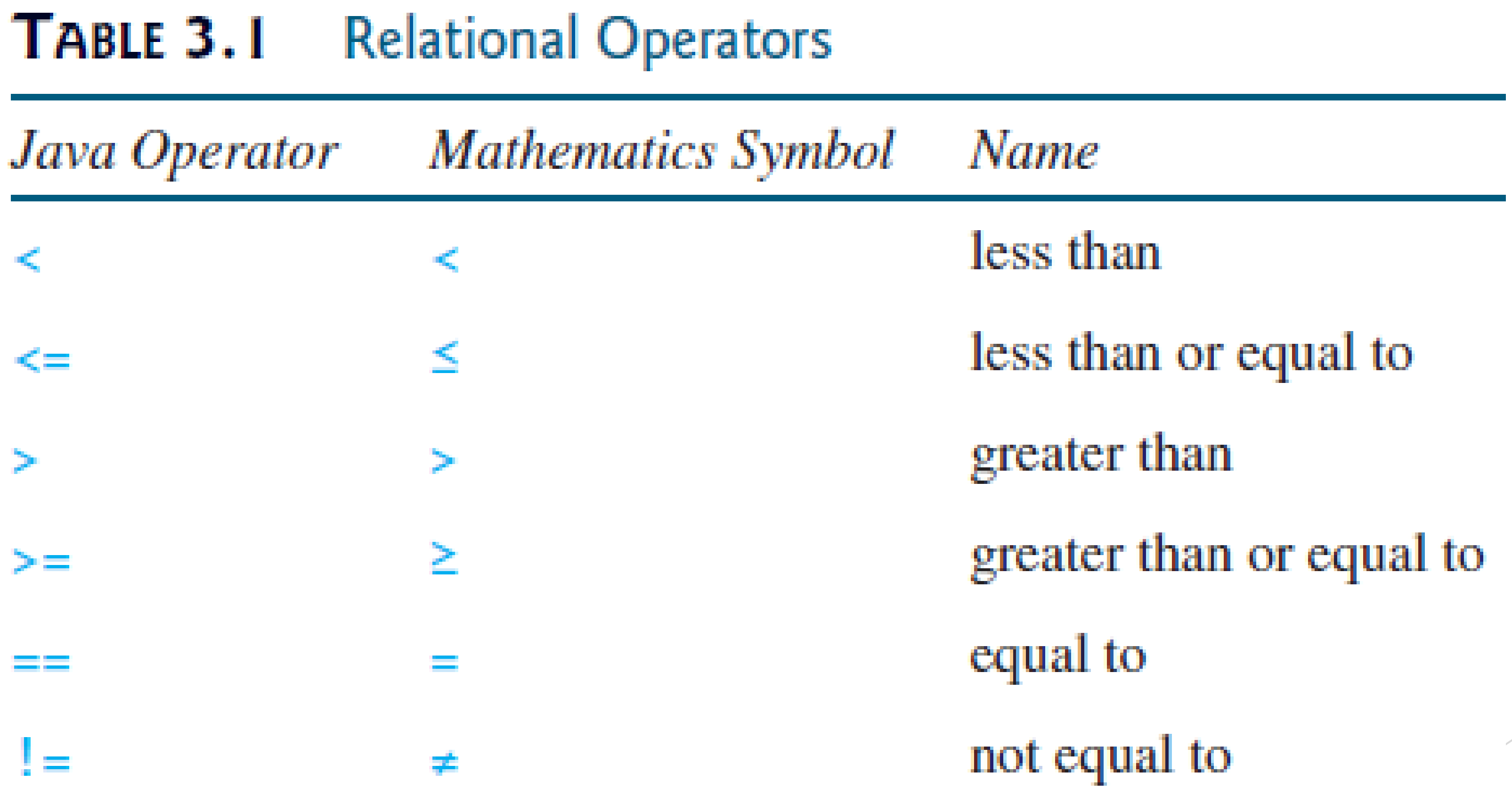


TABLE 3.1 Relational Operators

Java Operator	*Mathematics Symbol*	*Name*
<	$<$	less than
<=	$\leq$	less than or equal to
>	$>$	greater than
>=	$\geq$	greater than or equal to
==	$=$	equal to
!=	$\neq$	not equal to

- **double** radius = **1**;

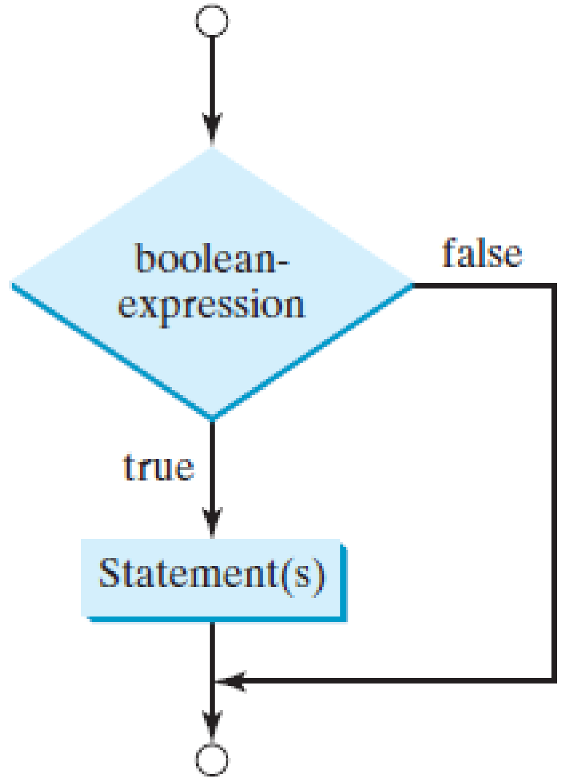
  System.out.println(radius > **0**);
- A variable that holds a Boolean value is known as a *Boolean variable*
- hold one of the two values

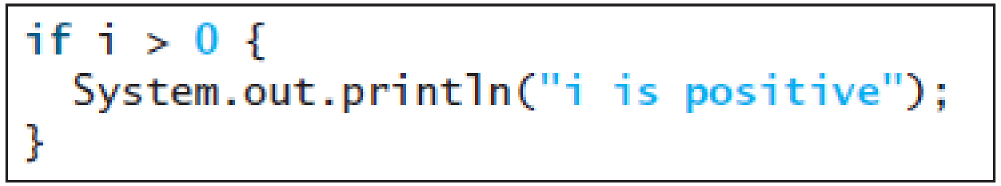
if Statements

- *It is a construct that enables a program to specify alternative paths of execution*
- one-way **if** statement executes an action if and only if the condition is **true**
- **if** (boolean-expression) {

  statement(s);

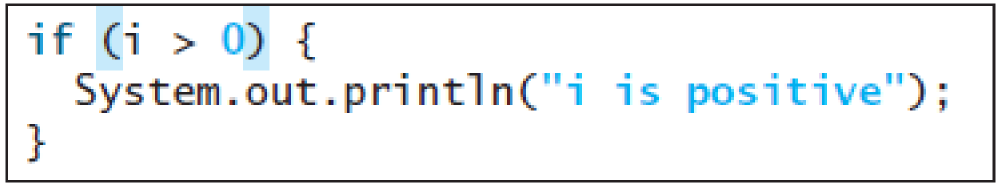
  }





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

(a) Wrong



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

(b) Correct

The block braces can be omitted if they enclose a single statement

Two-Way **if-else** Statements

- *An* **if-else** *statement decides the execution path based on whether the condition is true or false*
- **if** (boolean-expression) {

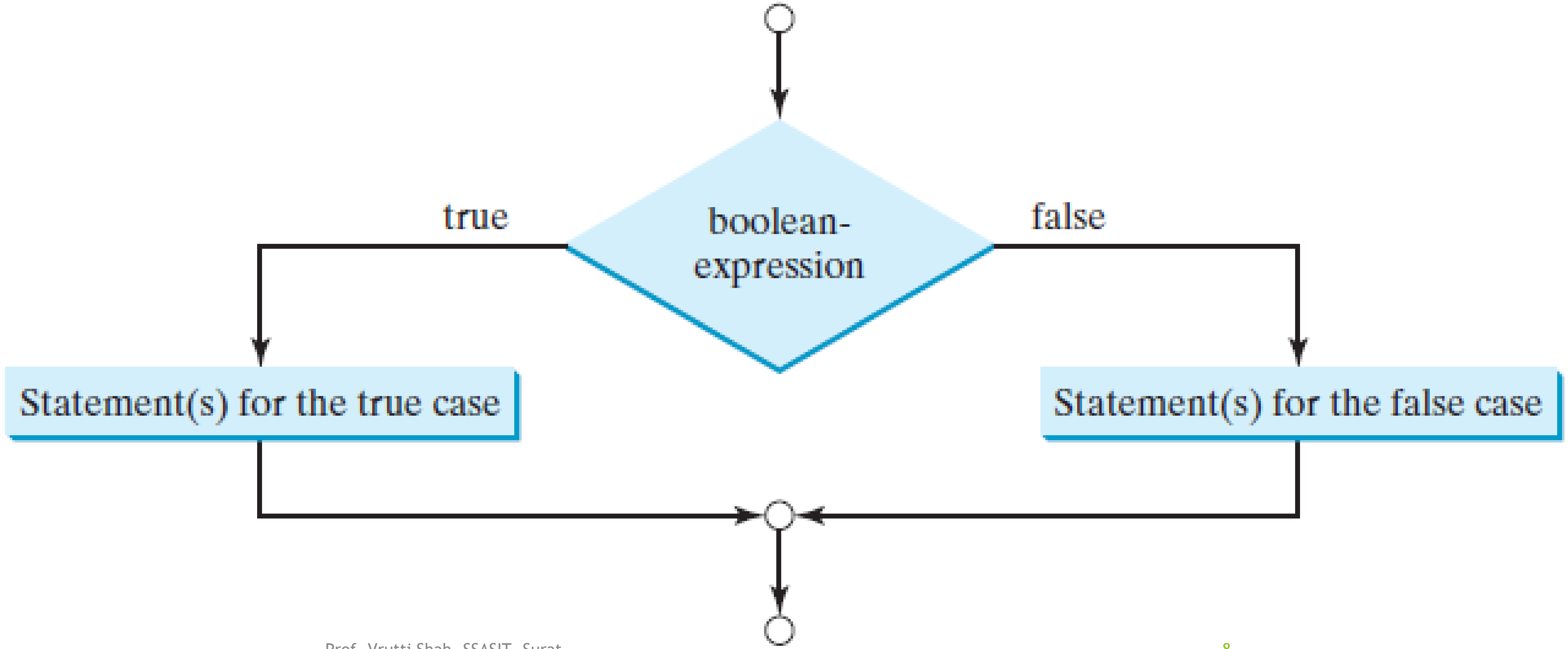
  statement(s)-for-the-true-case;

  }

  else {

  statement(s)-for-the-false-case;

  }


true
boolean-expression
false
Statement(s) for the true case
Statement(s) for the false case

Example

```
if (radius >= 0) {
  area = radius * radius * PI;
  System.out.println("The area for the circle of radius " +
  radius + " is " + area);
  }
  else {
  System.out.println("Negative input");
  }
```

Example

- **if** (number % **2** == **0**)

  System.out.println(number + " **is even.**");

  else

  System.out.println(number + " **is odd.**");

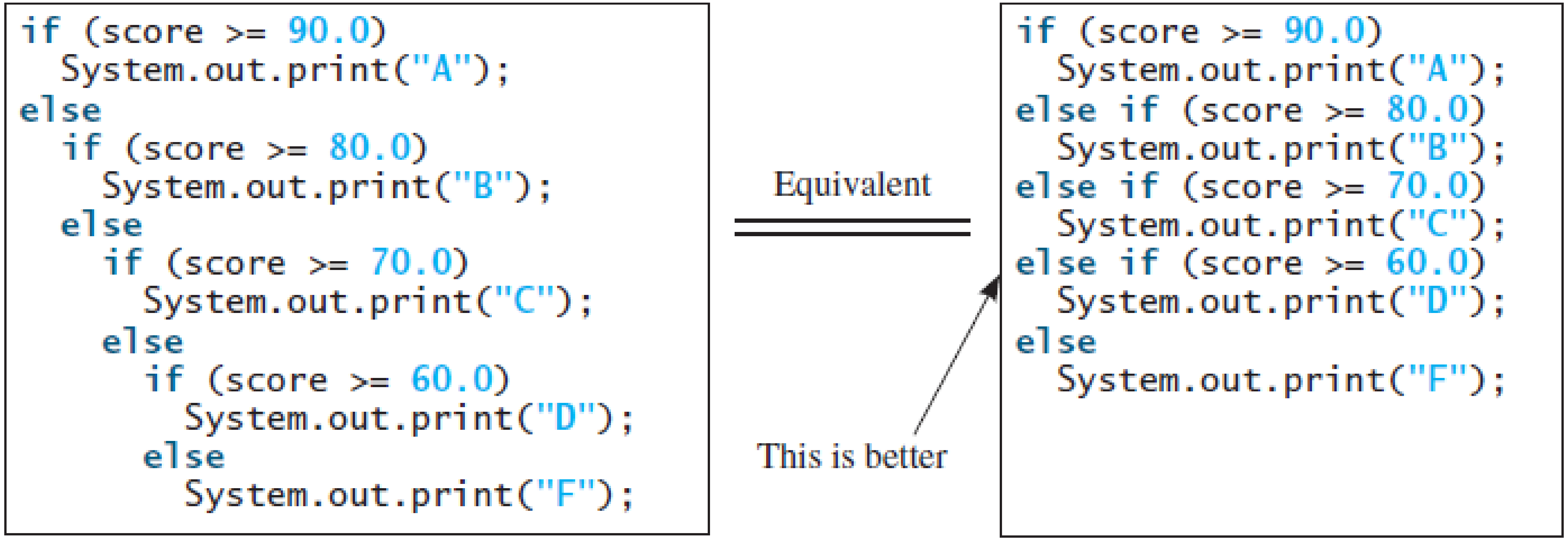
- Write an **if** statement that increases **pay** by 3% if **score** is greater than **90**, otherwise increases **pay** by 1%.

Nested **if** and Multi-Way **if-else** Statements

- *An* **if** *statement can be inside another* **if** *statement to form a nested* **if** *statement*
- **if** (i > k) {

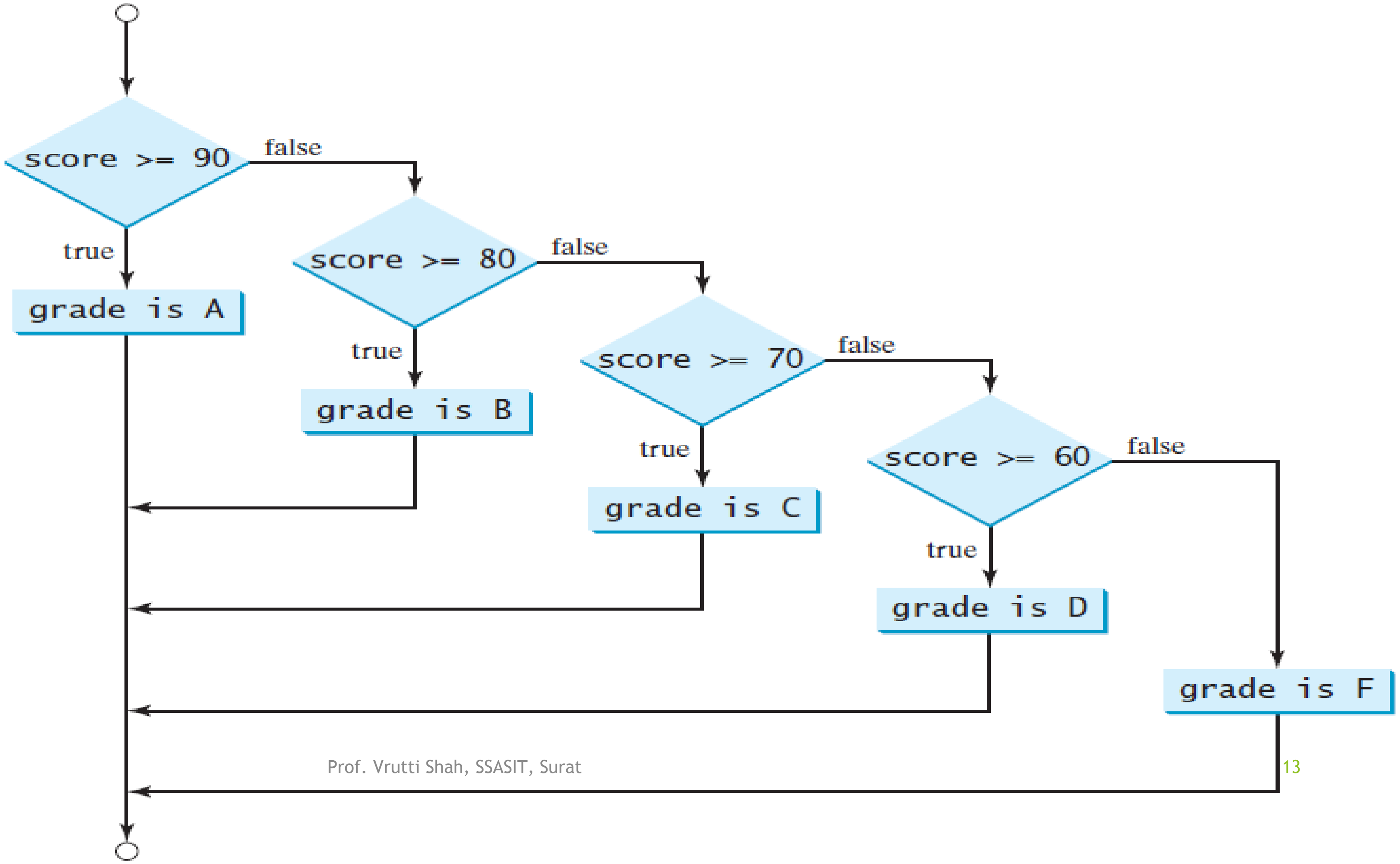
  ```
  if (j > k)
  System.out.println("i and j are greater than k");
  }
  else
  System.out.println("i is less than or equal to k");
  ```
- The nested **if** statement can be used to implement multiple alternatives

Example

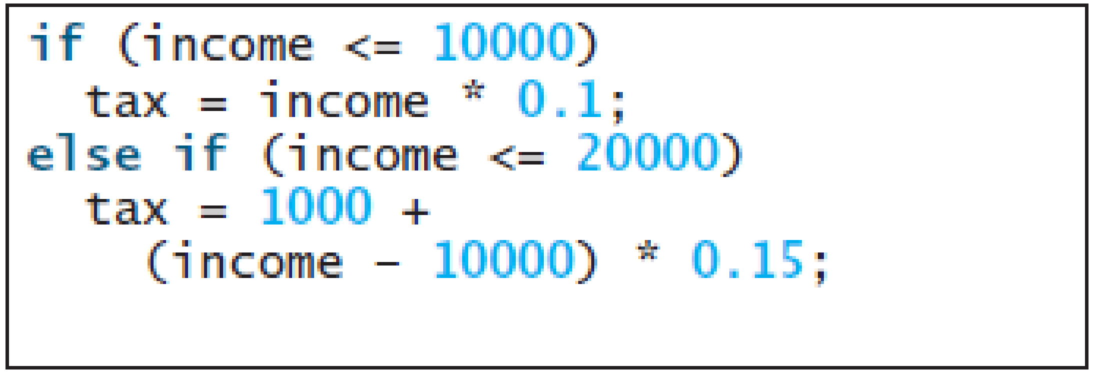

Equivalent
This is better

```
if (score >= 90.0)
  System.out.print("A");
else
  if (score >= 80.0)
    System.out.print("B");
  else
    if (score >= 70.0)
      System.out.print("C");
    else
      if (score >= 60.0)
        System.out.print("D");
      else
        System.out.print("F");
```

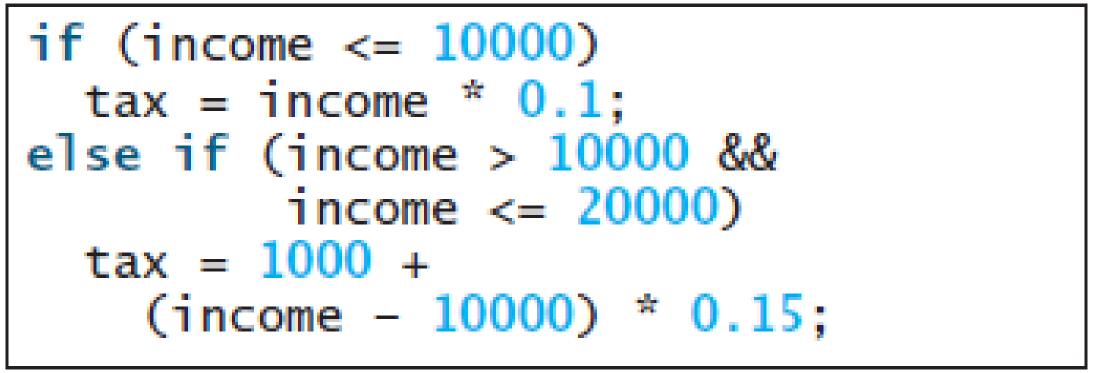
```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```


score >= 90
false
true
grade is A
score >= 80
false
true
grade is B
score >= 70
false
true
grade is C
score >= 60
false
true
grade is D
grade is F

- Following two statements are equivalent or not?



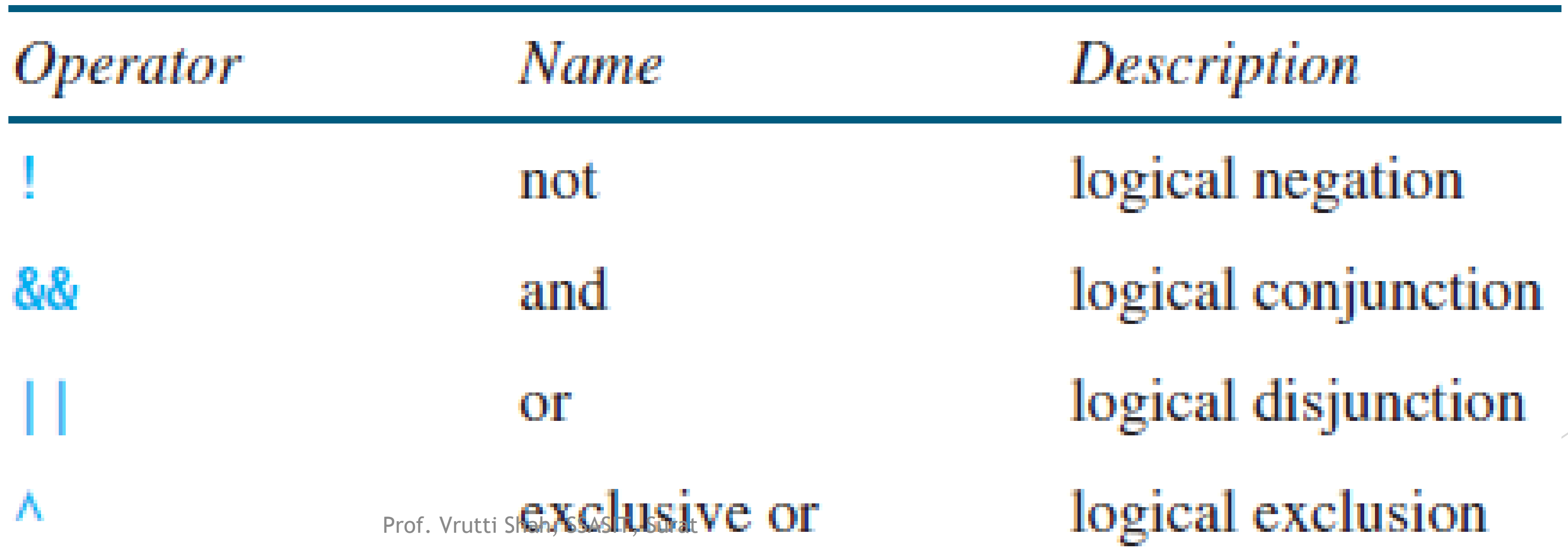
```
if (income <= 10000)
  tax = income * 0.1;
else if (income <= 20000)
  tax = 1000 +
    (income - 10000) * 0.15;
```



```
if (income <= 10000)
  tax = income * 0.1;
else if (income > 10000 &&
         income <= 20000)
  tax = 1000 +
    (income - 10000) * 0.15;
```

Logical Operators (*Boolean operators*)

- *logical operators* `!`, `&&`, `||`, *and* `^` *can be used to create a compound Boolean expression*
- use logical operators to combine the conditions to form a compound Boolean expression

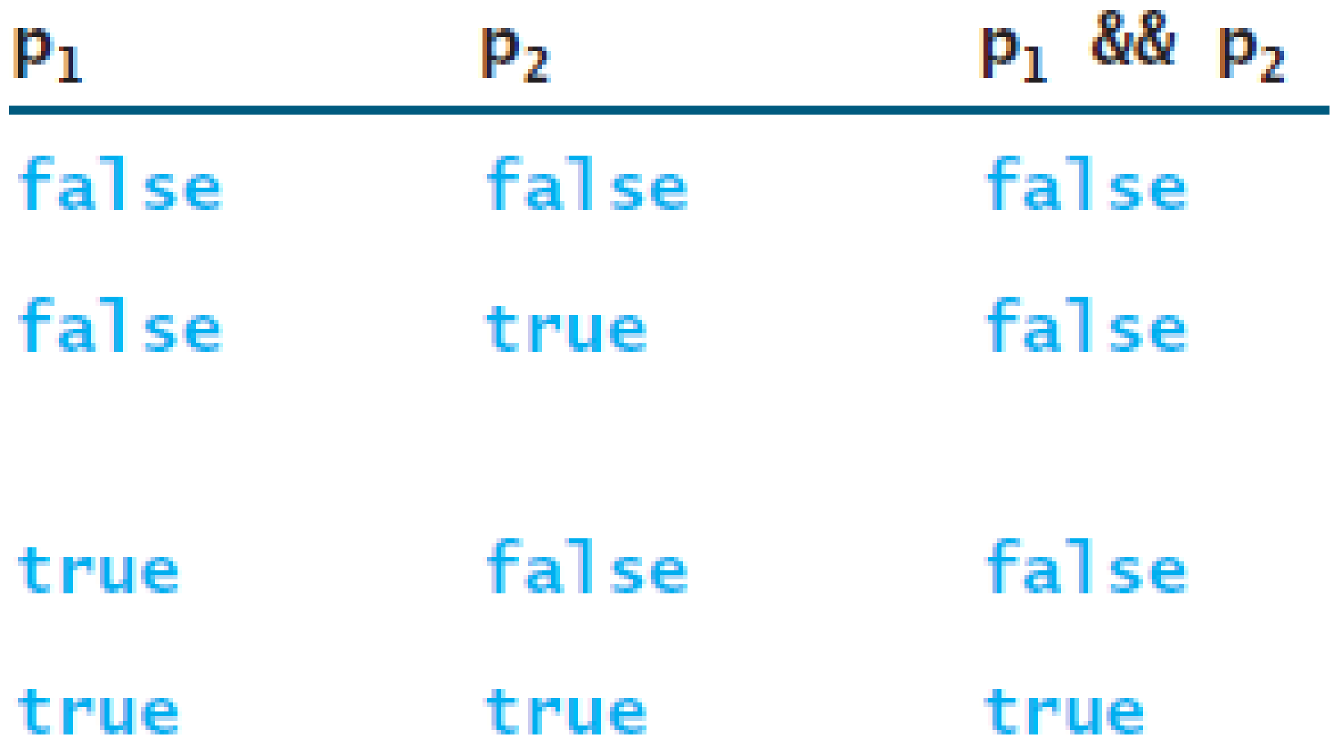


Operator	*Name*	*Description*
!	not	logical negation
&&	and	logical conjunction
||	or	logical disjunction
^	exclusive or	logical exclusion

Truth Table for Operator !

p	!p
true	false
false	true

Truth Table for Operator &&



p_1	p_2	p_1 && p_2
false	false	false
false	true	false
true	false	false
true	true	true

Truth Table for Operator ||

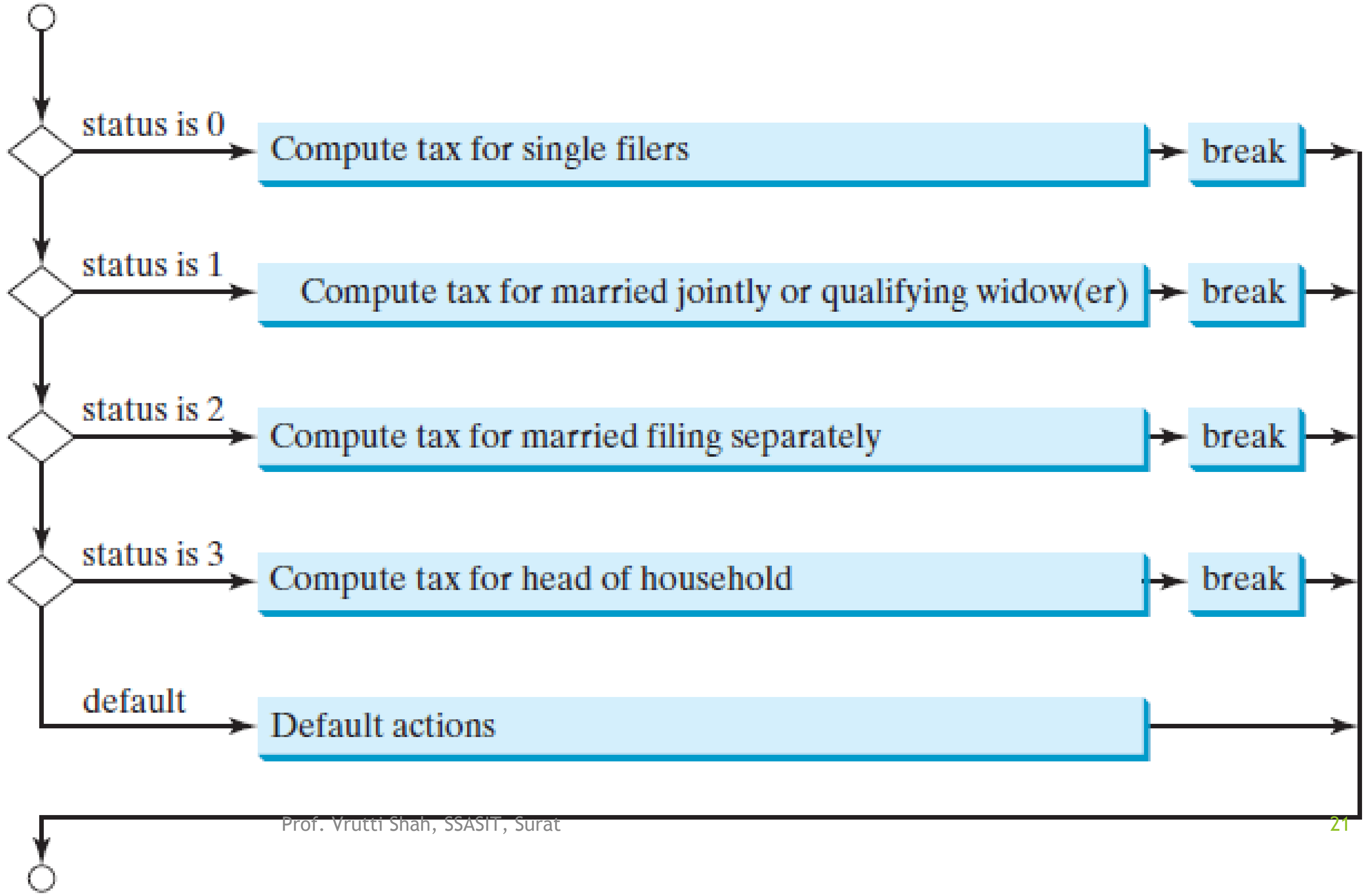
p_1	p_2	p_1 || p_2
false	false	false
false	true	true
true	false	true
true	true	true

Truth Table for Operator ^

p_1	p_2	p_1 ^ p_2
false	false	false
false	true	true
true	false	true
true	true	false

switch Statements

- **switch** *statement executes statements based on the value of a variable or an expression*
- Java provides a **switch** statement to simplify coding for multiple conditions.


status is 0
Compute tax for single filers
break
status is 1
Compute tax for married jointly or qualifying widow(er)
break
status is 2
Compute tax for married filing separately
break
status is 3
Compute tax for head of household
break
default
Default actions

- **switch** (**switch**-expression) {

```
case value1: statement(s)1;
break;
case value2: statement(s)2;
break;
…
case valueN: statement(s)N;
break;
default: statement(s)-for-default;
}
```

switch Statements (Contd…)

- It must yield a value of **char**, **byte**, **short**, **int**, or **String** type
- must always be enclosed in parentheses
- **value1**, . . ., and **valueN** must have the same data type as the value of the **switch-expression**
- **value1**, . . ., and **valueN** are constant expressions
- they cannot contain variables, such as **1 + x**
- When the value in a **case** statement matches the value of the **switch-expression**, the statements *starting from this case* are executed until either a **break** statement or the end of the **switch** statement is reached
- The **default** case is optional

switch Statements (Contd…)

```
switch (day) {
 case 1:
 case 2:
 case 3:
 case 4:
 case 5: System.out.println("Weekday"); break;
 case 0:
 case 6: System.out.println("Weekend");
 }
```

Conditional Expressions

- *A conditional expression evaluates an expression based on a condition*
- boolean-expression ? expression1 : expression2;
- **if** (x > **0**)

  y = **1**;

  else

  y = **-1**;
- y = (x > **0**) ? **1** : **-1**;
- max = (num1 > num2) ? num1 : num2;
- System.out.println((num % **2** == **0**) ? **"num is even"** : **"num is odd"**);

Operator Precedence and Associativity

- *It determine the order in which operators are evaluated.*
- **3** + **4** * **4** > **5** * (**4** + **3**) – **1** && (**4** - **3** > **5**)
- expression within parentheses is evaluated first
- without parentheses, the operators are applied according to the precedence and associativity rule.

Precedence	*Operator*
	`var++` and `var--` (Postfix)
	`+`, `-` (Unary plus and minus), `++var` and `--var` (Prefix)
	(type) (Casting)
	`!` (Not)
	`*`, `/`, `%` (Multiplication, division, and remainder)
	`+`, `-` (Binary addition and subtraction)
	`<`, `<=`, `>`, `>=` (Relational)
	`==`, `!=` (Equality)
	`^` (Exclusive OR)
	`&&` (AND)
	`||` (OR)
↓	`=`, `+=`, `-=`, `*=`, `/=`, `%=` (Assignment operator)

- All binary operators except assignment operators are *left associative*

$$\texttt{a - b + c - d} \quad \overset{\text{is equivalent to}}{\Longleftrightarrow} \quad \texttt{((a - b) + c) - d}$$

- Assignment operators are *right associative*

$$\texttt{a = b += c = 5} \quad \overset{\text{is equivalent to}}{\Longleftrightarrow} \quad \texttt{a = (b += (c = 5))}$$

- Write a program that prompts the user to enter a three-digit integer and determines whether it is a palindrome number.
- Write a program that reads three edges for a triangle and computes the perimeter if the input is valid. Otherwise, display that the input is invalid. The input is valid if the sum of every pair of two edges is greater than the remaining edge.