

6. Selection Statements

10 Sep 2015

Objectives

- Implement selection control in a program using if statements.
- Implement selection control in a program using switch statements.
- Write boolean expressions using relational and boolean operators.
- Evaluate given boolean expressions correctly.

Objectives

- Nest an if statement inside another if statement correctly.
- Describe how objects are compared.
- Define and use enumerated constants.

Selection Statements

- Any practical computer program contains many statements that make decisions. The statement that alters the control flow is called a 'control statement'. In this chapter we describe some important control statements, called 'selection statements'.

The if Statement

- There are two versions of the if statement
 1. if-then-else
 2. if-then

if–then–else

- The if–then–else is used when we want to choose one of two possible actions. The if–then–else is like a fork in the road. Under the control of the boolean test, one or the other will be taken, but not both.

if-then-else

- Ex. Suppose we wish to enter a student's test score and print out the message You did not pass if the score is less than 70 and You did pass if the score is 70 or higher.

```
int testScore = scanner.nextInt();  
  
if (testScore < 70)  
    System.out.println("You did not pass");  
  
else  
    System.out.println("You did pass");
```

This statement is executed if **testScore** is less than 70.

This statement is executed if **testScore** is 70 or higher.

if-then-else

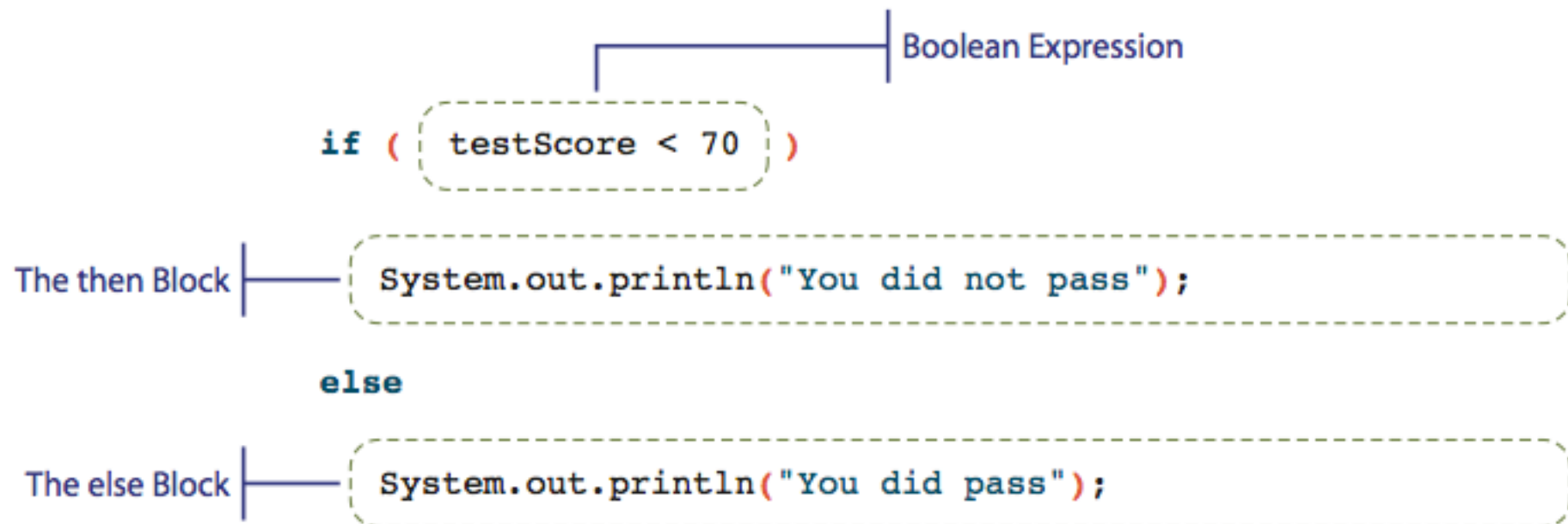
- The if-then-else statement follows this general format:

```
if ( <boolean expression> )  
    <then block>  
  
else  
    <else block>
```


if-then-else

- If the <boolean expression> evaluates to true, then the statements in the <then block> are executed. Otherwise, the statements in the <else block> are executed.

if-then-else



if-then-else

- The <boolean expression> is a conditional expression that is evaluated to either 'true' or 'false'. For example, the following three expressions are all conditional:

```
testScore < 80  
testScore * 2 > 350  
30 < w / (h * h)
```

if-then-else

- The six 'relational operators' we can use in conditional expressions are:

```
<      // less than
<=     // less than or equal to
==     // equal to
!=     // not equal to
>      // greater than
>=     // greater than or equal to
```

```
a * a <= c    //true if a * a is less than or equal to c
x + y != z    //true if x + y is not equal to z
a == b        //true if a is equal to b
```

if-then-else

- One very common error in writing programs is to mix up the assignment and equality operators. We frequently make the mistake of writing

```
if (x = 5) ...
```

when we actually wanted to say

```
if (x == 5) ...
```

if-then-else

- We can reverse the relational operator and switch the then and else blocks to derive the equivalent code, for example, from this code

```
if (testScore < 70)
    System.out.println("You did not pass");
else
    System.out.println("You did pass");
```

if-then-else

We can reverse to this code

```
if (testScore >= 70)
    System.out.println("You did pass");

else
    System.out.println("You did not pass");
```

if-then-else

- The if statement is called a selection or branching statement because it selects (or branches to) one of the alternative blocks for execution.

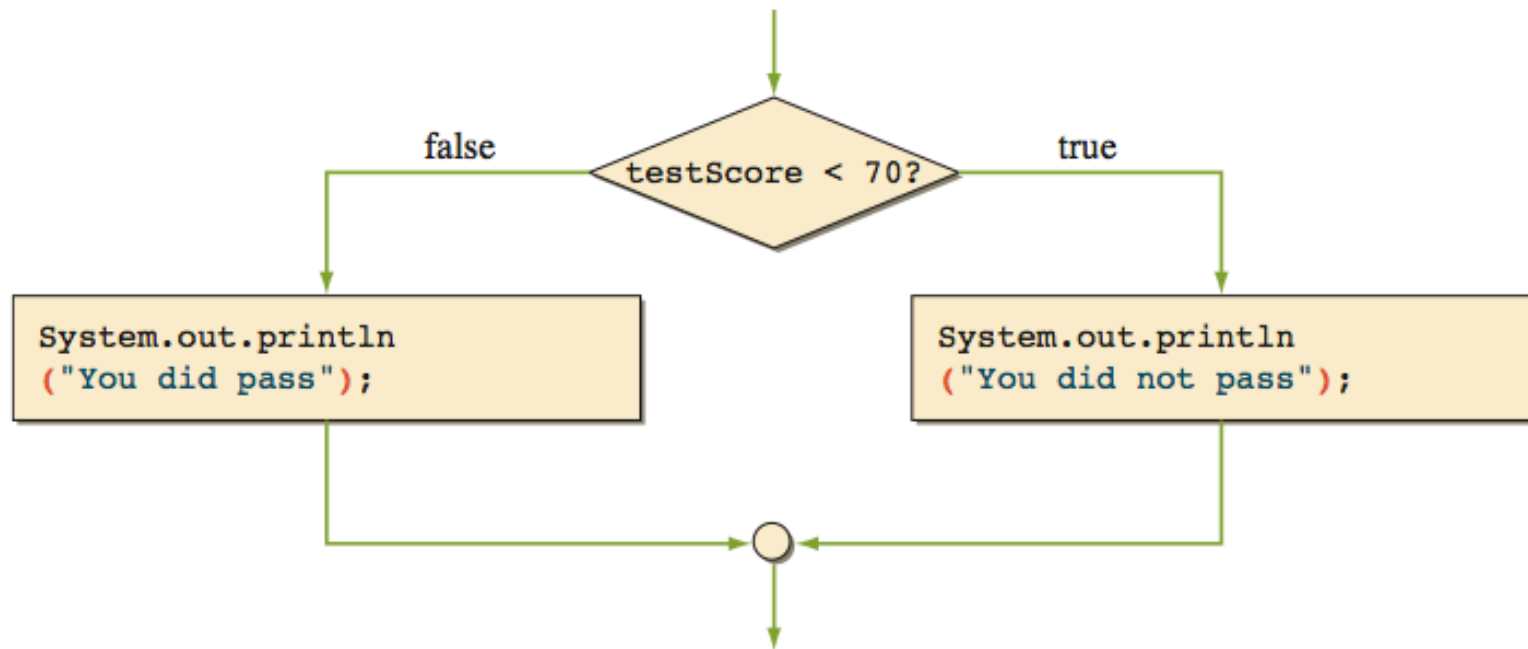
```
System.out.println("You did not pass");
```

or

```
System.out.println("You did pass");
```


if-then-else

- We can illustrate a branching path of execution with the diagram shown below.



if-then-else

- In the preceding if statement, both blocks contain only one statement. The then or else block can contain more than one statement. The general format for both the <then block> and the <else block> is either a

`<single statement>`

or a

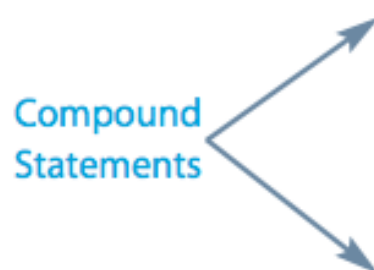
`<compound statement>`

if-then-else

- If multiple statements are needed in the <then block> or the <else block>, they must be surrounded by braces { and }.

Compound Statements

```
if (testScore < 70)
{
    System.out.println("You did not pass");
    System.out.println("Try harder next time");
}
else
{
    System.out.println("You did pass");
    System.out.println("Keep up the good work");
}
```



if-then-else

- It is valid for a single statement to be surrounded by braces { and } too.

```
if (testScore < 70)
{
    System.out.println("You did not pass");
}
else
{
    System.out.println("You did pass");
}
```

if-then

- The if-then statement is the most basic of all the control flow statements. Your program will execute a section of code only if a test evaluates to 'true'.
- The if-then statement follows this general format:

```
if ( <boolean expression> )  
    <then block>
```

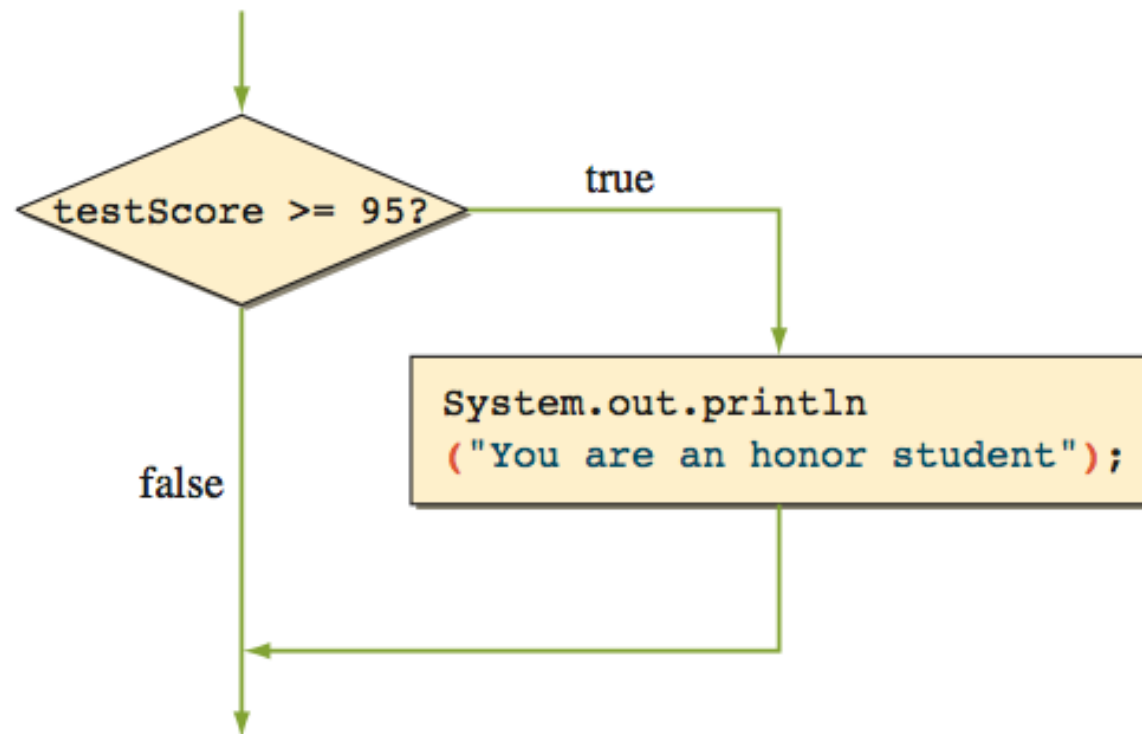
if-then

- Suppose we want to print out the message
You are an honor student if the test score is
95 or above and print out nothing otherwise.

```
if (testScore >= 95) {  
    System.out.println("You are an honor student");  
}
```

if-then

- Figure below shows the diagram that illustrates the control flow for this if-then statement.



if-then

- Ex. Ch5Circle class. We will include a test in this class so the methods such as getArea() and getCircumference() return the constant INVALID_DIMENSION when the dimension of the radius is invalid.

```
class Ch5Circle {  
    public static final int INVALID_DIMENSION = -1;  
    private double radius;  
    public Ch5Circle(double r) {  
        setRadius(r);  
    }  
}
```


if-then

```
public void setRadius(double r) {  
    if (r > 0) {  
        radius = r;  
    } else {  
        radius = INVALID_DIMENSION;  
    }  
}  
  
private boolean isRadiusValid( ) {  
    return radius != INVALID_DIMENSION;  
}
```

if-then

```
public double getArea( ) {  
    double result = INVALID_DIMENSION;  
    if (isRadiusValid()) {  
        result = Math.PI * radius * radius;  
    }  
    return result;  
}  
  
public double getCircumference( ) {  
    double result = INVALID_DIMENSION;  
    if (isRadiusValid()) {  
        result = 2.0 * Math.PI * radius;  
    }  
    return result;  
}
```

if-then

```
class Ch5Sample1 {  
    public static void main(String[] args) {  
        double    radius, circumference, area;  
  
        Ch5Circle circle;  
  
        Scanner scanner = new Scanner(System.in);  
  
        System.out.print("Enter radius: ");  
        radius = scanner.nextDouble();  
  
        circle = new Ch5Circle(radius);  
  
        circumference = circle.getCircumference();  
  
        area          = circle.getArea();  
  
        System.out.println("Input radius:  " + radius);  
        System.out.println("Circumference: " + circumference);  
        System.out.println("Area:         " + area);  
    }  
}
```

if-then

- Output (If we input invalid value):

```
Enter radius: -10  
Input radius: -10.0  
Circumference: -1.0  
Area: -1.0
```

Nested if Statements

- The then and else blocks of an if statement can contain any statement including another if statement. An if statement that contains another if statement in either its then or else block is called a nested if statement.

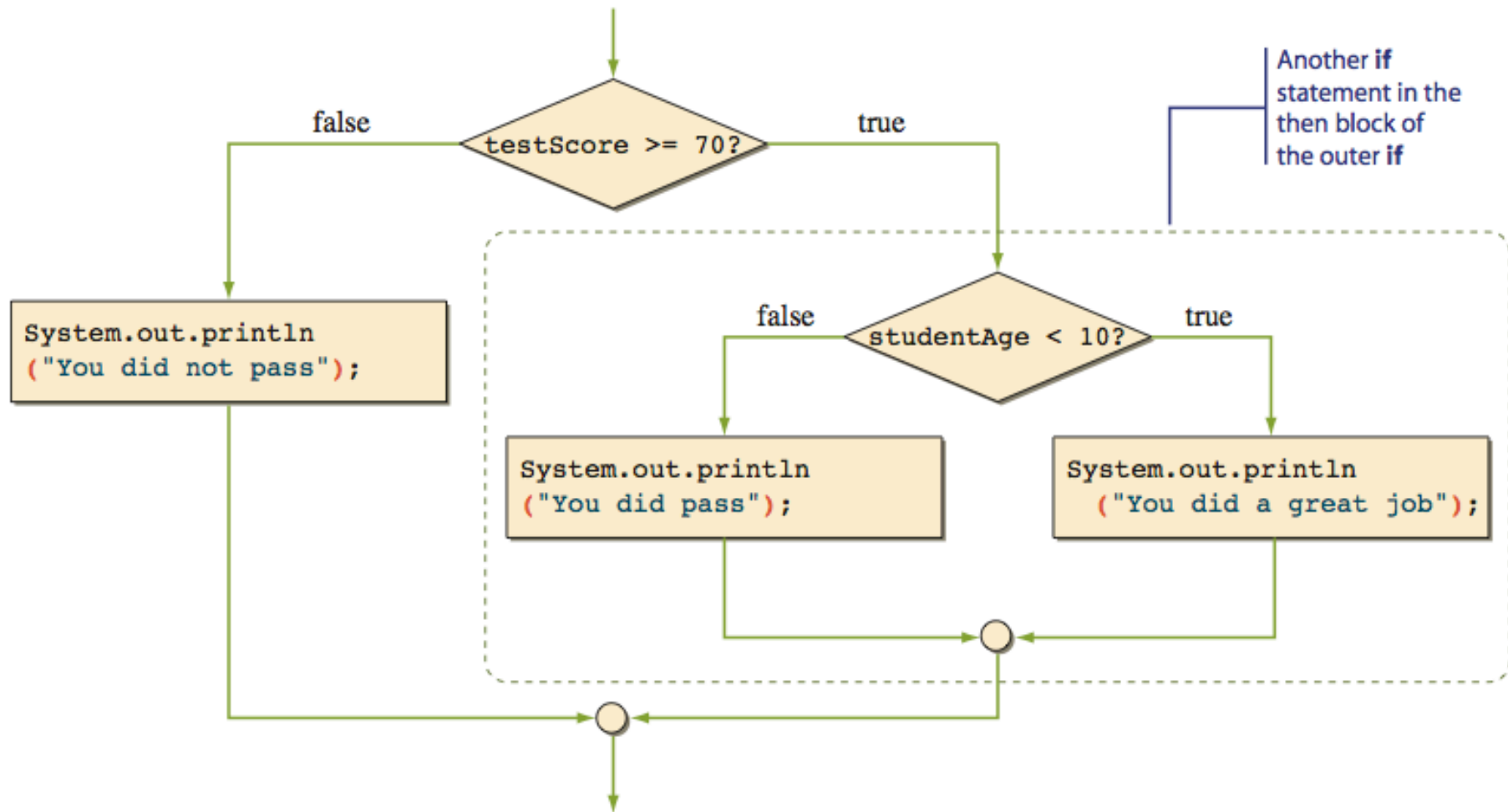
Nested if Statements

- Ex. If the test score is lower than 70, then we print You did not pass, as before. If the test score is 70 or higher, then we will check the student's age. If the age is less than 10, we will print You did a great job.

Nested if Statements

```
if (testScore >= 70) {  
    if (studentAge < 10) {  
        System.out.println("You did a great job");  
    } else {  
        System.out.println("You did pass");//test score >= 70  
                                           //and age >= 10  
    }  
} else { //test score < 70  
    System.out.println("You did not pass");  
}
```

Nested if Statements



Nested if Statements

- It is possible to write if tests in different ways to achieve the same result. For example, the preceding code can also be expressed as

```
if (testScore >= 70 && studentAge < 10) {  
    System.out.println("You did a great job");  
} else {  
    //either testScore < 70 OR studentAge >= 10  
  
    if (testScore >= 70) {  
        System.out.println("You did pass");  
    } else {  
        System.out.println("You did not pass");  
    }  
}
```

Nested if Statements

- Ex. This example shows a style of indentation accepted as standard for a nested if statement in which nesting occurs only in the else block. We will now display a letter grade based on the following formula:

Test Score	Grade
$90 \leq \text{score}$	A
$80 \leq \text{score} < 90$	B
$70 \leq \text{score} < 80$	C
$60 \leq \text{score} < 70$	D
$\text{score} < 60$	F

Nested if Statements

The statement can be written as

```
if (score >= 90)
    System.out.println("Your grade is A");
else
    if (score >= 80)
        System.out.println("Your grade is B");
    else
        if (score >= 70)
            System.out.println("Your grade is C");
        else
            if (score >= 60)
                System.out.println("Your grade is D");
            else
                System.out.println("Your grade is F");
```

Nested if Statements

However, the standard way to indent the statement is as

```
if (score >= 90)
    System.out.println("Your grade is A");
else if (score >= 80)
    System.out.println("Your grade is B");
else if (score >= 70)
    System.out.println("Your grade is C");
else if (score >= 60)
    System.out.println("Your grade is D");
else
    System.out.println("Your grade is F");
```

Nested if Statements

- We mentioned that indentation is meant for human eyes only. For example, we can clearly see the intent of a programmer just by looking at the indentation when we read

```
if (x < y)
    if (x < z)
        System.out.println("Hello");
else
    System.out.println("Good bye");
```

Indentation style A

Nested if Statements

However a Java compiler will interpret the previous code as

```
if (x < y)
    if (x < z)
        System.out.println("Hello");
    else
        System.out.println("Good bye");
```

Indentation style B

Nested if Statements

- Previous example has a 'dangling else problem'. The Java compiler matches an else with the previous unmatched if, so the compiler will interpret the statement by matching the else with the inner if (if (x < z)).
- To avoid 'dangling else problem' we recommend that always use the braces in the then and else blocks.

Nested if Statements

- Ex. Add test code inside the add() and deduct() methods of the Account class.

```
class Ch5Account {  
    ...  
    //Adds the passed amount to the balance  
    public void add(double amt) {  
        //add if amt is positive; otherwise, do nothing  
        if (amt > 0) {  
            balance = balance + amt;  
        }  
    }  
}
```


Nested if Statements

```
//Deducts the passed amount from the balance
public void deduct(double amt) {
    //deduct if amt is positive; do nothing otherwise
    if (amt > 0) {
        double newbalance = balance - amt;
        if (newbalance >= 0) {    //if a new balance is positive, then
            balance = newbalance; //update the balance; otherwise,
                                //do nothing.
        }
    }
    ...
}
```

Boolean Expressions and Variables

- The boolean expression is the expression which result is either 'true' or 'false'. We can declare a variable of data type boolean and assign a boolean value to it.

```
boolean pass, done;
```

```
pass = 70 < x;
```

```
done = true;
```

Boolean Expressions and Variables

- A 'boolean operator', also called a 'logical operator', takes boolean values as its operands and returns a boolean value.
- Three boolean operators are AND, OR, and NOT.
 - AND: &&
 - OR: ||
 - NOT: !

Boolean Expressions and Variables

- Combining boolean operators with relational and arithmetic operators, we can come up with long boolean expressions such as

```
(x + 150) == y || x < y && !(y < z && z < x)
(x < y) && (a == b || a == c)
a != 0 && b != 0 && (a + b < 10)
```

Boolean Expressions and Variables

- The AND operation (&&) results in true only if both P and Q are true.
- The OR operation (||) results in true if either P or Q is true.
- The NOT operation (!) is true if P is false and is false if P is true.

P	Q	P && Q	P Q	!P
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

Boolean Expressions and Variables

- Operator precedence rules. An operator with a higher precedence will be evaluated first.

Group	Operator	Precedence	Associativity
Subexpression	()	10 (If parentheses are nested, then innermost subexpression is evaluated first.)	Left to right
Postfix increment and decrement operators	++ --	9	Right to left

Boolean Expressions and Variables

Unary operators	-	8	Right to left
	!		
Multiplicative operators	*	7	Left to right
	/		
	%		
Additive operators	+	6	Left to right
	-		
Relational operators	<	5	Left to right
	<=		
	>		
	>=		

Boolean Expressions and Variables

Equality operators	<code>==</code> <code>!=</code>	4	Left to right
Boolean AND	<code>&&</code>	3	Left to right
Boolean OR	<code> </code>	2	Left to right
Assignment	<code>=</code>	1	Right to left

Boolean Expressions and Variables

- We can reverse the relational operator and switch the then and else blocks to derive the equivalent code.

```
if (age < 0) {  
    System.out.println("Invalid age is entered");  
} else {  
    System.out.println("Valid age is entered");  
}
```

Boolean Expressions and Variables

is equivalent to

```
if ( !(age < 0) ) {  
    System.out.println("Valid age is entered");  
} else {  
    System.out.println("Invalid age is entered");  
}
```

which can be written more naturally as

```
if (age >= 0) {  
    System.out.println("Valid age is entered");  
} else {  
    System.out.println("Invalid age is entered");  
}
```

Boolean Expressions and Variables

- Ex. If the temperature is greater than or equal to 65 degrees and the distance to the destination is less than 2 mi., we walk. Otherwise (it's too cold or too far away), we drive.

```
if (temperature >= 65 && distanceToDestination < 2) {  
    System.out.println("Let's walk");  
} else {  
    System.out.println("Let's drive");  
}
```

Boolean Expressions and Variables

We can rewrite the statement by negating the boolean expression and switching the then and else blocks as

```
if ( !(temperature >= 65 && distanceToDestination < 2) ) {  
    System.out.println("Let's drive");  
} else {  
    System.out.println("Let's walk");  
}
```

or more directly and naturally as

```
if (temperature < 65 || distanceToDestination >= 2) {  
    System.out.println("Let's drive");  
} else {  
    System.out.println("Let's walk");  
}
```

Boolean Expressions and Variables

The expression

```
!(temperature >= 65 && distanceToDestination < 2)
```

is equivalent to

```
!(temperature >= 65) || !(distanceToDestination < 2)
```

which, in turn, is equivalent to

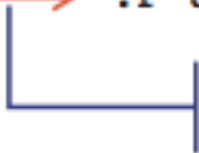
```
(temperature < 65 || distanceToDestination >= 2)
```

Boolean Expressions and Variables

- The logical equivalence is derived by applying the following DeMorgan's law:

Rule 1: $\neg(P \ \&\& \ Q) \iff \neg P \ || \ \neg Q$

Rule 2: $\neg(P \ || \ Q) \iff \neg P \ \&\& \ \neg Q$

 Equivalence symbol

Boolean Expressions and Variables

- `||` and `&&` operator are short-circuit evaluation.
 - `||` : if the left operand is true, then the right operand will not be evaluated, because the whole expression is true.

`y == 0 || x / y > z`

- `&&` : if the left operand is false, then the right operand will not be evaluated, because the whole expression is false.

`80 <= x && x < 90`

Boolean Expressions and Variables

- In mathematics, we specify the range of values for a variable as

$$80 \leq x < 90$$

In Java, we express it as

`80 <= x && x < 90`

You cannot specify it as

`80 <= x < 90`



Wrong

Boolean Expressions and Variables

- One possible use of boolean variables is to keep track of the program settings or user preferences. A variable (of any data type, not just boolean) used for this purpose is called a 'flag'.

Boolean Expressions and Variables

- Ex. We use variable 'displayShortDateFlag' to tell main() method, how to display date.

```
public class JavaApplication1 {  
    private static final String shortDatePattern = "d/M/yy";  
    private static final String longDatePattern = "d MMMM yyyy";  
  
    public static boolean displayShortDateFlag = false;  
  
    public static void main(String[] args) {  
        SimpleDateFormat format;  
  
        if(displayShortDateFlag){  
            format = new SimpleDateFormat(shortDatePattern);  
        }  
        else{  
            format = new SimpleDateFormat(longDatePattern);  
        }  
  
        Date today = new Date();  
        System.out.println("Today is " + format.format(today));  
    }  
}
```

Boolean Expressions and Variables

– If we set

```
public static boolean displayShortDateFlag = false;
```

Output: run:
 Today is 6 September 2015

– If we set

```
public static boolean displayShortDateFlag = true;
```

Output: run:
 Today is 6/9/15

Comparing Objects

- When two variables are compared (using ==), we are comparing their contents.
 - In the case of primitive data types, the content is the actual value.
 - In case of reference data types, the content is the address where the object is stored.

Comparing Objects

- Ex. Compare primitive data types.

```
int num1, num2;  
  
num1 = 15;  
num2 = 15;  
  
if (num1 == num2) {  
    System.out.println("They are equal");  
} else {  
    System.out.println("They are not equal");  
}
```

Output: They are equal

Comparing Objects

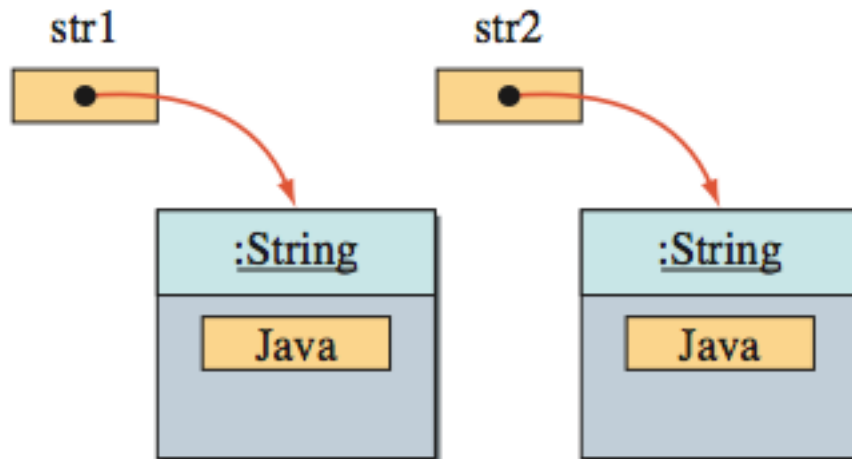
- Ex. Compare String objects 1.

```
String str1, str2;  
  
str1 = new String("Java");  
str2 = new String("Java");  
  
if (str1 == str2) {  
    System.out.println("They are equal");  
} else {  
    System.out.println("They are not equal");  
}
```

Output: They are not equal

Comparing Objects

Case A: Two variables refer to two different objects.



```
String str1, str2;
```

```
str1 = new String("Java");  
str2 = new String("Java");
```

```
str1 == str2 → false
```

Comparing Objects

- Ex. Compare String objects 2.

```
String str1, str2;
```

```
str1 = new String("Java");
```

```
str2 = str1;
```

No new object is created here. The content (address) of **str1** is copied to **str2**, making them both point to the same object.

```
if (str1 == str2) {
```

```
    System.out.println("They are equal");
```

```
} else {
```

```
    System.out.println("They are not equal");
```

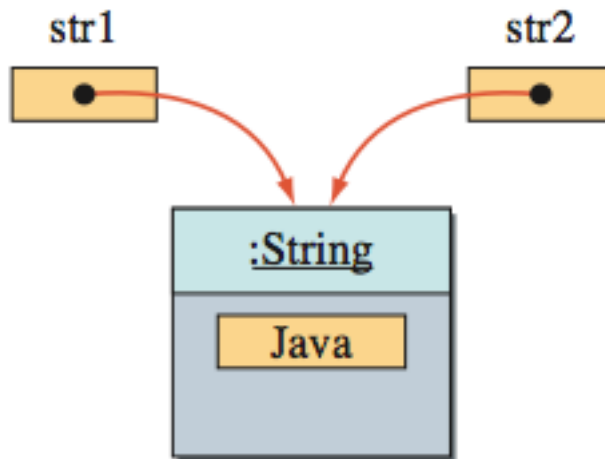
```
}
```

Output:

They are equal

Comparing Objects

Case B: Two variables refer to the same object.




```
String str1, str2;  
  
str1 = new String("Java");  
str2 = str1;
```

`str1 == str2` → true

Comparing Objects

- If we need to check whether two String objects have the same sequence of characters using equals() method

```
String str1, str2;  
  
str1 = new String("Java");  
str2 = new String("Java");  
  
if (str1.equals(str2)) {  
    System.out.println("They are equal");  
} else {  
    System.out.println("They are not equal");  
}
```



Use the **equals** method.

Output:

They are equal

Comparing Objects

- To compare objects from any classes, just create equals() method using this signature:
 - public boolean equals(ClassName object)

Comparing Objects

- Ex. To compare Fraction object

```
public class Fraction {  
    private int numerator;  
    private int denorminator;  
  
    public Fraction(int numerator, int denorminator) {  
        this.numerator = numerator;  
        this.denorminator = denorminator;  
    }  
  
    public double getValue(){  
        return 1.0 * numerator / denorminator;  
    }  
  
    public boolean equals(Fraction f) {  
        return getValue() == f.getValue();  
    }  
}
```

Comparing Objects

– Main Class

```
public class JavaApplication1 {  
    public static void main(String[] args) {  
        Fraction f1 = new Fraction(1, 2);  
        Fraction f2 = new Fraction(6, 12);  
  
        System.out.println(f1.equals(f2));  
    }  
}
```

– Output:

```
run:  
true
```

The switch Statement

- Another Java statement that implements a selection control flow is the switch statement. The syntax for the switch statement is

```
switch ( <integer expression> ) {  
    <case label 1> : <case body 1>  
    ...  
    <case label n> : <case body n>  
}
```

The <case label i> has the form

```
case <integer constant>    or    default
```

The switch Statement

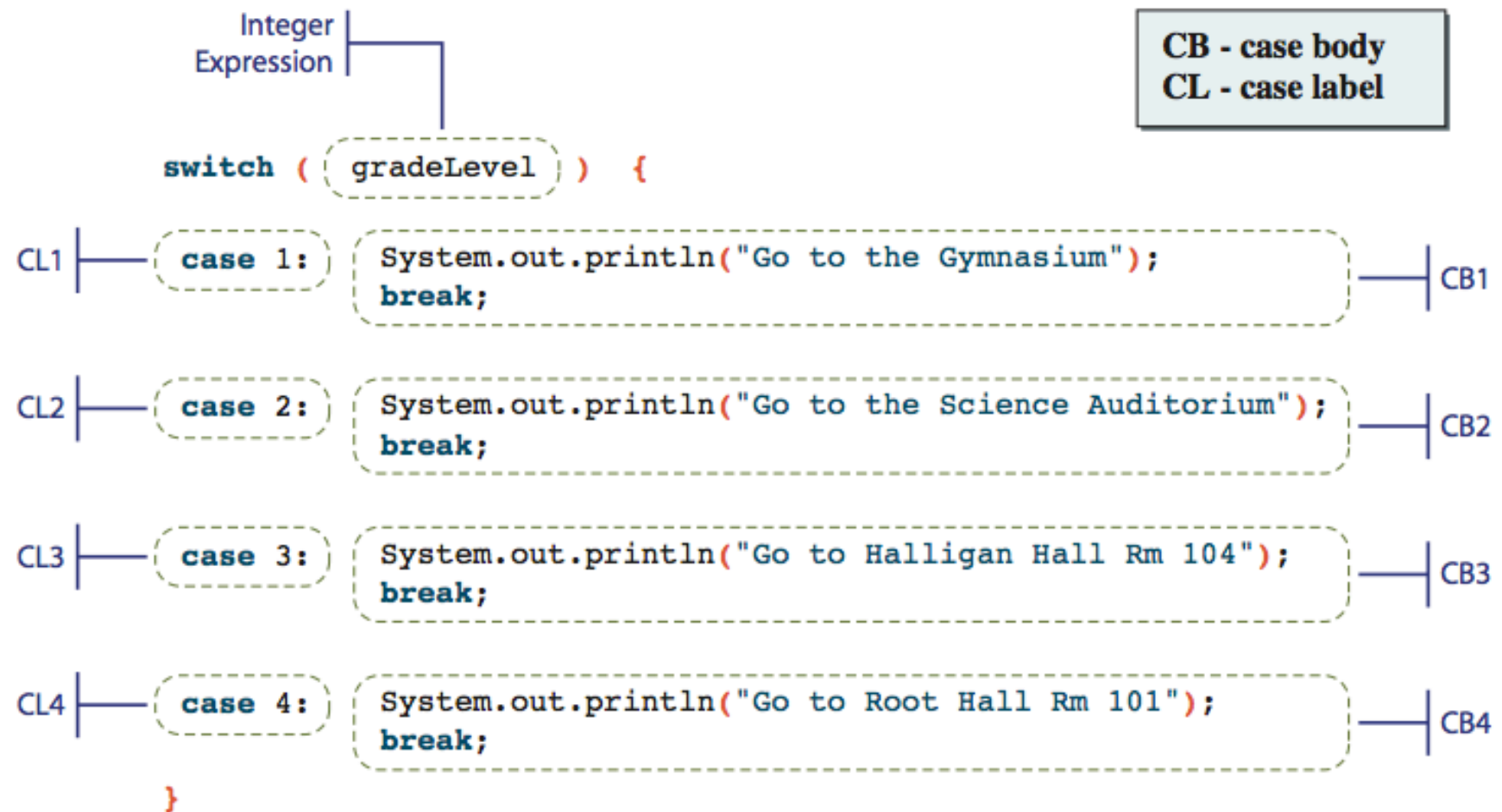
- Ex. Suppose we want to direct the students to the designated location for them to register for classes. The user enters 1 for freshman, 2 for sophomore, 3 for junior, and 4 for senior.

```
int gradeLevel;  
  
Scanner scanner = new Scanner(System.in);  
  
System.out.print("Grade (Frosh-1,Soph-2,...): ");  
  
gradeLevel = scanner.nextInt();
```

The switch Statement

```
switch (gradeLevel) {  
    case 1: System.out.println("Go to the Gymnasium");  
        break;  
    case 2: System.out.println("Go to the Science Auditorium");  
        break;  
    case 3: System.out.println("Go to Halligan Hall Rm 104");  
        break;  
    case 4: System.out.println("Go to Root Hall Rm 101");  
        break;  
}
```


The switch Statement



The switch Statement

- Notice that each case in the sample switch statement is terminated with the break statement. The following example illustrates how the break statement works:

```
selection = 1;

switch (selection) {
    case 0: System.out.println(0);
    case 1: System.out.println(1);
    case 2: System.out.println(2);
    case 3: System.out.println(3);
}
```

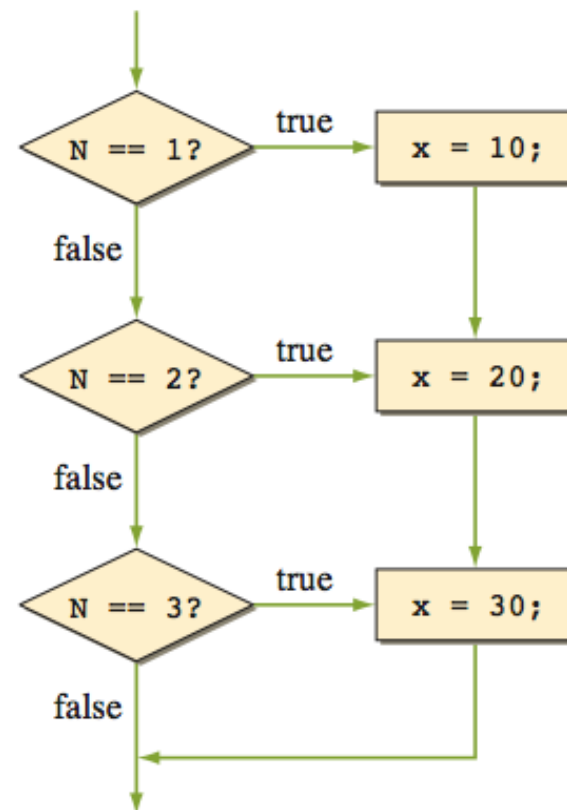
Output:

```
1
2
3
```

The switch Statement

- Control flow of the switch statement without the break statements.

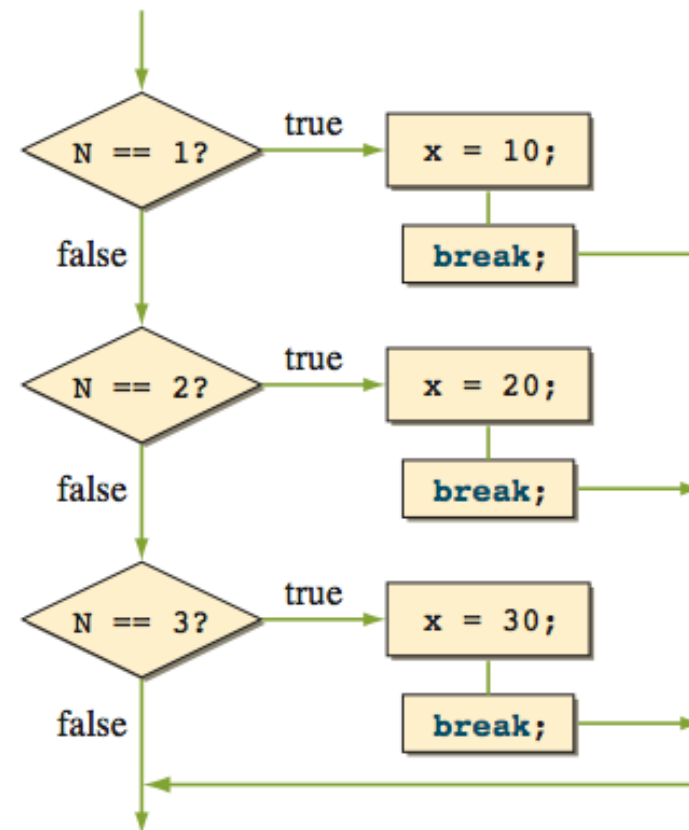
```
switch ( N ) {  
  case 1: x = 10;  
  case 2: x = 20;  
  case 3: x = 30;  
}
```



The switch Statement

- Control flow of the switch statement with the break statements.

```
switch ( N ) {  
  case 1: x = 10; break;  
  case 2: x = 20; break;  
  case 3: x = 30; break;  
}
```



The switch Statement

- We may include a default case that will always be executed if there is no matching case.

```
Scanner scanner = new Scanner(System.in);  
System.out.print("Input: ");  
int ranking = scanner.nextInt();
```

The switch Statement

```
switch (ranking) {  
    case 10:  
    case 9:  
    case 8: System.out.println("Master");  
            break;  
  
    case 7:  
    case 6: System.out.println("Journeyman");  
            break;  
  
    case 5:  
    case 4: System.out.println("Apprentice");  
            break;  
  
    default: System.out.println("Error: Invalid Data");  
            break;  
}
```

Enumerated Constants

- In this section, we will introduce a type of constant called 'enumerated constants'.
- Ex. Suppose we want to define a Student class and define constants to distinguish four student grade

```
class Student {  
    public static final int FRESHMAN = 0;  
    public static final int SOPHOMORE = 1;  
    public static final int JUNIOR = 2;  
    public static final int SENIOR = 3;  
    ...  
}
```

Enumerated Constants

- With the enumerated constants, this is how we can define the grade levels in the Student class:

```
class Student {  
    public static enum GradeLevel  
        {FRESHMAN, SOPHOMORE, JUNIOR, SENIOR}  
    ...  
}
```


Enumerated Constants

- The word 'enum' is a reserved word, and the basic syntax for defining enumerated constants is

```
enum <enumerated type> { <constant values> }
```

```
enum Month {JANUARY, FEBRUARY, MARCH, APRIL,  
            MAY, JUNE, JULY, AUGUST,  
            SEPTEMBER, OCTOBER, NOVEMBER, DECEMBER}
```

```
enum Gender {MALE, FEMALE}
```

```
enum SkillLevel {NOVICE, INTERMEDIATE, ADVANCED, EXPERT}
```

Enumerated Constants

- With enum, we can assign only possible values to enum variables

```
public class JavaApplication1 {  
    public static enum Fruit {APPLE, ORANGE, BANANA};  
  
    public static void main(String[] args) {  
        Fruit f1, f2, f3;  
        f1 = Fruit.APPLE;  
        f2 = Fruit.ORANGE;  
        f3 = f1;  
  
        System.out.println(f1 + ", " + f2 + ", " + f3);  
    }  
}
```

run:
APPLE, ORANGE, APPLE

Enumerated Constants

- Without enum, we can assign out of scope values to variables

```
public class JavaApplication1 {  
    private static final int APPLE = 1;  
    private static final int ORANGE = 2;  
    private static final int BANANA = 3;  
  
    public static void main(String[] args) {  
        int fOne, fTwo, fThree;  
        fOne = APPLE;  
        fTwo = ORANGE;  
        fThree = 30;  
  
        System.out.println(fOne + ", " + fTwo + ", " + fThree);  
    }  
}
```

run:
1, 2, 30

Enumerated Constants

- With enum, we can print informative output values.

```
public class JavaApplication1 {  
    private static final int APPLE = 1;  
    private static final int ORANGE = 2;  
    private static final int BANANA = 3;  
  
    private static enum Fruit{APPLE, ORANGE, BANANA};  
  
    public static void main(String[] args) {  
        Fruit f1 = Fruit.APPLE;  
        int f2 = APPLE;  
  
        System.out.println(f1);  
        System.out.println(f2);  
    }  
}
```

```
run:  
APPLE  
1
```

Enumerated Constants

- When referring to an enumerated constant, we must prefix it with its enumerated type name, for example

```
Fruit f = Fruit.APPLE;  
  
. . .  
if (f == Fruit.ORANGE) {  
    System.out.println("I like orange, too.");  
}  
  
. . .
```

Enumerated Constants

- A case label for a switch statement is the only exception, we can specify the case labels without the prefix as in

```
Fruit fruit = Fruit.APPLE;
switch (fruit) {
    case APPLE: . . . ;
                break;
    case ORANGE: . . . ;
                break;
    case BANANA: . . . ;
                break;
}
```

Summary

- A selection control statement is used to alter the sequential flow of control.
- The if and switch statements are two types of selection control.
- The two versions of the if statement are if–then–else and if–then.
- A boolean expression contains relational and boolean operators and evaluates to true or false.

Summary

- Three boolean operators in Java are AND (&&), OR (||), and NOT (!).
- An if statement can be a part of the then or else block of another if statement to formulate nested if statements.
- When the equality symbol == is used in comparing the variables of reference data type, we are comparing the addresses.

Summary

- The switch statement is useful for expressing a selection control based on equality testing between data of type int.
- The enumerated constants provide type safety and increase the program readability.

Reference

- C. Thomas Wu, An Introduction to Object-Oriented Programming with Java, 5th Edition
 - Chapter 5: Selection Statements

Question?