Outline

1. **Flow of Control**
2. **Conditional Statements**
3. **The if Statement**
4. **The if-else Statement**
5. **The Conditional operator**
6. **The Switch Statement**
7. **Useful Hints**

1. Flow of Control

- **The order of statement execution is called the *flow of control***
- **Unless specified otherwise, the order of statement execution through a method is linear (sequential): one statement after another in sequence**
- **Some programming statements allow us to:**
  - **decide whether or not to execute a particular statement**
  - **execute a statement over and over, repetitively**
- **These selection (decision) statements are based on *boolean expressions* (or *conditions*) that evaluate to true or false**

2. Selection Statements

- **A *Selection (conditional) statement* allows us to choose which statement (or block of statements) will be executed next.**
- **Java selection statements are:**
  - ***if statement -*** allows one option
  - ***if-else statement -*** allows two options
  - ***switch statement -*** allows multiple options

3. The if Statement

- **The *if statement* has the following syntax:**

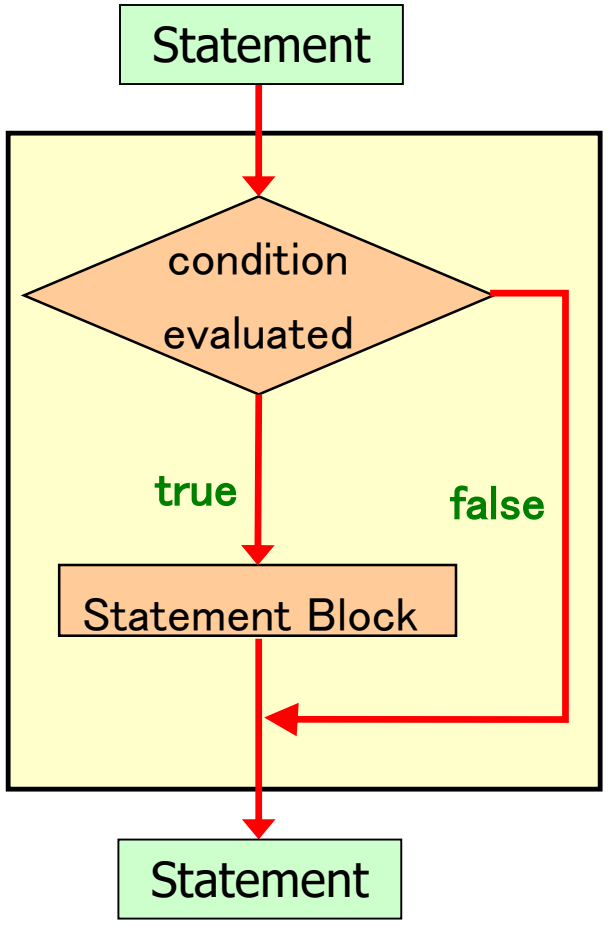
The ***condition*** must be a boolean expression. It must evaluate to either true or false.

`if` is a Java reserved word

```
if (condition)
{
    statementBlock;
};
```

If the ***condition*** is true, the ***statementBlock*** is executed.
If it is false, the ***statementBlock*** is skipped.

Logic of *if statement*




```
1 grade = 70;
2 If (grade>= 90)
3    System.out.println("You got an "A");
4 System.out.println("This is line 4");
```

```
1 grade = 95;
2 If (grade>= 90)
3    System.out.println("You got an "A");
4 System.out.println("This is line 4");
```

Boolean Expressions

- **A condition often uses one of Java's *equality operators* or *relational operators*, which all return boolean results:**

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

- **Note the difference between the equality operator (==) and the assignment operator (=)**

Example - if Statement

- **An example of an `if` statement:**

```
if (sum > MAX)
   delta = sum - MAX;
System.out.println ("The sum is " + sum);
```

- **First, the condition is evaluated -- the value of `sum` is either greater than the value of `MAX`, or it is not**
- **If the condition is true, the assignment statement is executed -- if it isn't (i.e., false), the assignment statement is skipped.**
- **Either way, the call to `println` is executed next**
- **See Age.java next slide**

Example - if Statement

```
// Age.java
import java.util.Scanner;
public class Age
{
  public static void main (String[] args)
  {
    final int MINOR = 21;
    Scanner scan = new Scanner (System.in);

    System.out.print ("Enter your age: ");
    int age = scan.nextInt();
    System.out.println ("You entered: " + age);

    if (age < MINOR)
        System.out.println ("Youth is a wonderful thing. Enjoy!");

    System.out.println ("Age is a state of mind.");
  }
}
```

Indentation

- **The statement controlled by the `if` statement is indented to indicate that relationship**
- **The use of a consistent indentation style makes a program easier to read and understand**
- **Although it makes no difference to the compiler, proper indentation is crucial for code readability and debugging**

Expressions

- **What do the following statements do?**

```
if (top >= MAXIMUM)
   top = 0;
//next statement starts here
```

Sets `top` to zero if the current value of `top` is greater than or equal to the value of `MAXIMUM`

```
if (total != stock + warehouse)
   inventoryError = true;
// next statement starts here
```

Sets a flag to true if the value of `total` is not equal to the sum of `stock` and `warehouse`

- **Note: the precedence of arithmetic operators is higher than the precedence of equality and relational operators.**

Logical Operators

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

!	**Logical NOT**
&&	**Logical AND**
||	**Logical OR**
^	**Logical XOR**

- **They all take boolean operands and produce boolean results**
- **Logical NOT is a unary operator (it operates on one operand)**
- **Logical AND, OR, and XOR are binary operators (each operates on two operands)**

Logical Operators

- **The *logical NOT* operation is also called *logical negation* or *logical complement***
- **If some boolean condition `a` is true, then `!a` is false; if `a` is false, then `!a` is true**
- **Logical expressions can be shown using a *truth table***

`boolean a`	`!a`
true	false
false	true

Logical Operators

- **The *logical AND* expression**

  `a && b`

  is true if both `a` and `b` are true, and false otherwise
- **The *logical OR* expression**

  `a || b`

  is true if `a` or `b` or both are true, and false otherwise
- **The *logical XOR* expression**

  `a ^ b`

  is true if and only if `a` and `b` are different.

Logical Operators

- **A truth table shows all possible true-false combinations of the terms**
- **Since `&&`, `||`, and `^` each have two operands, there are four possible combinations of conditions `a` and `b` (`boolean expressions`)**

`a`	`b`	`a && b`	`a || b`	`a ^ b`
true	true	true	true	false
true	false	false	true	true
false	true	false	true	true
false	false	false	false	false

Boolean Expressions

- **Expressions that use logical operators can form complex conditions**

```
if (total < MAX + 5 && !found)
   System.out.println ("Processing…");
```

- **Mathematical operators have higher precedence than the Relational and Logical operators**
- **Relational operators have higher precedence than Logical operators**

Boolean Expressions

- **Specific expressions can be evaluated using truth tables**
- **Given** `X = total < MAX && !found`

  What is the values of `x` **?**

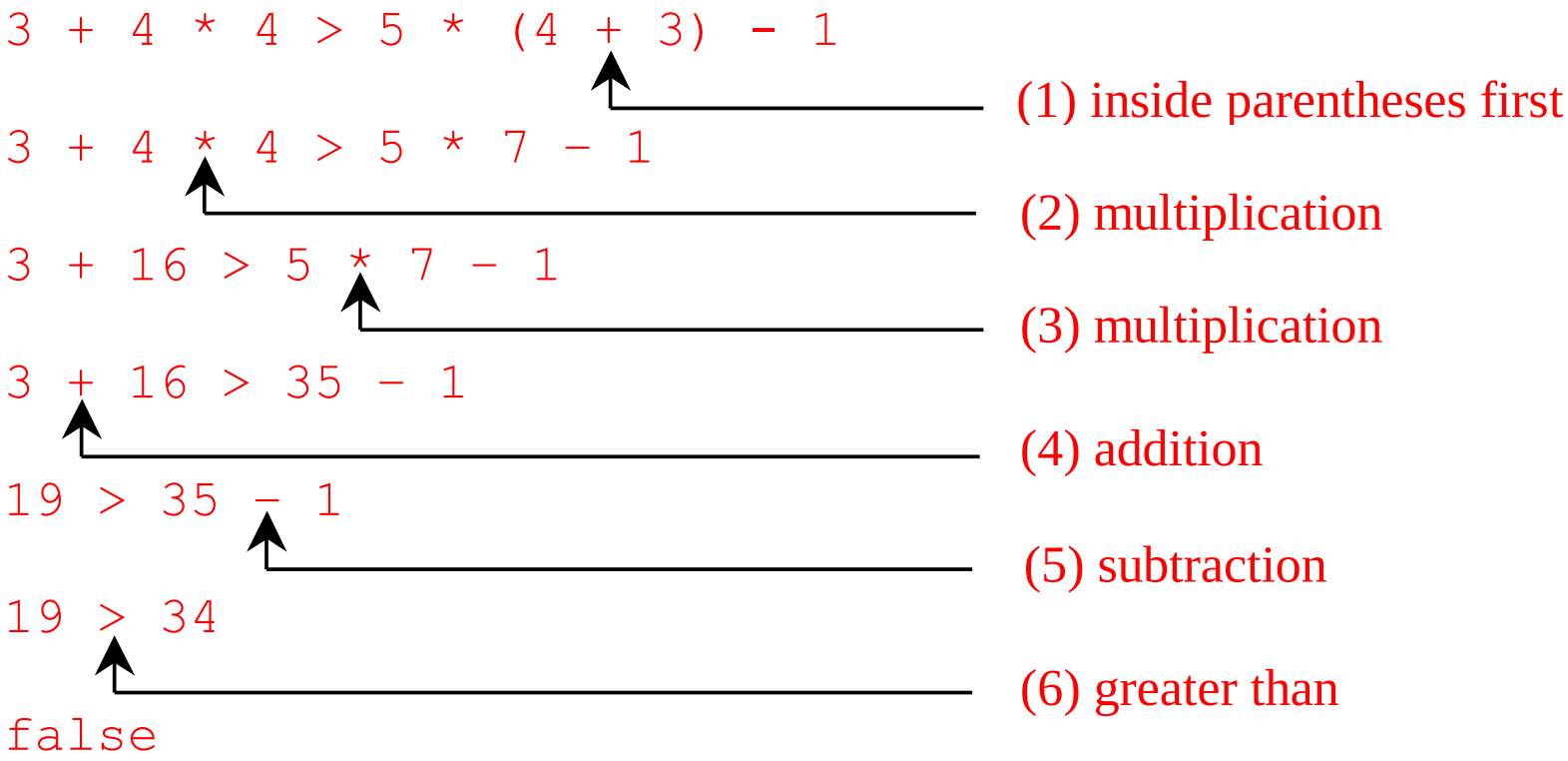
`total < MAX`	`!found`	`X = total < MAX && !found`
true	true	true
true	false	false
false	true	false
false	false	false

Operator Precedence

`var++, var--`	Postfix increment
`++var, --var`	Prefix increment
`+, -`	unary operators
`(type)`	Casting and parenthesis
`!`	Not
`*, /, %`	Math operators
`+, -`	Math operators
`<, <=, >, >=`	Relational operators
`==, !=`	Relational equality
`^`	Exclusive OR
`&&`	Logical AND
`||`	Logical OR
`=, +=, -=, *=, /=, %=`	Assignment operators

Operator Precedence

Applying operator precedence and associativity rule to the expression: `3 + 4 * 4 > 5 * (4 + 3) - 1`



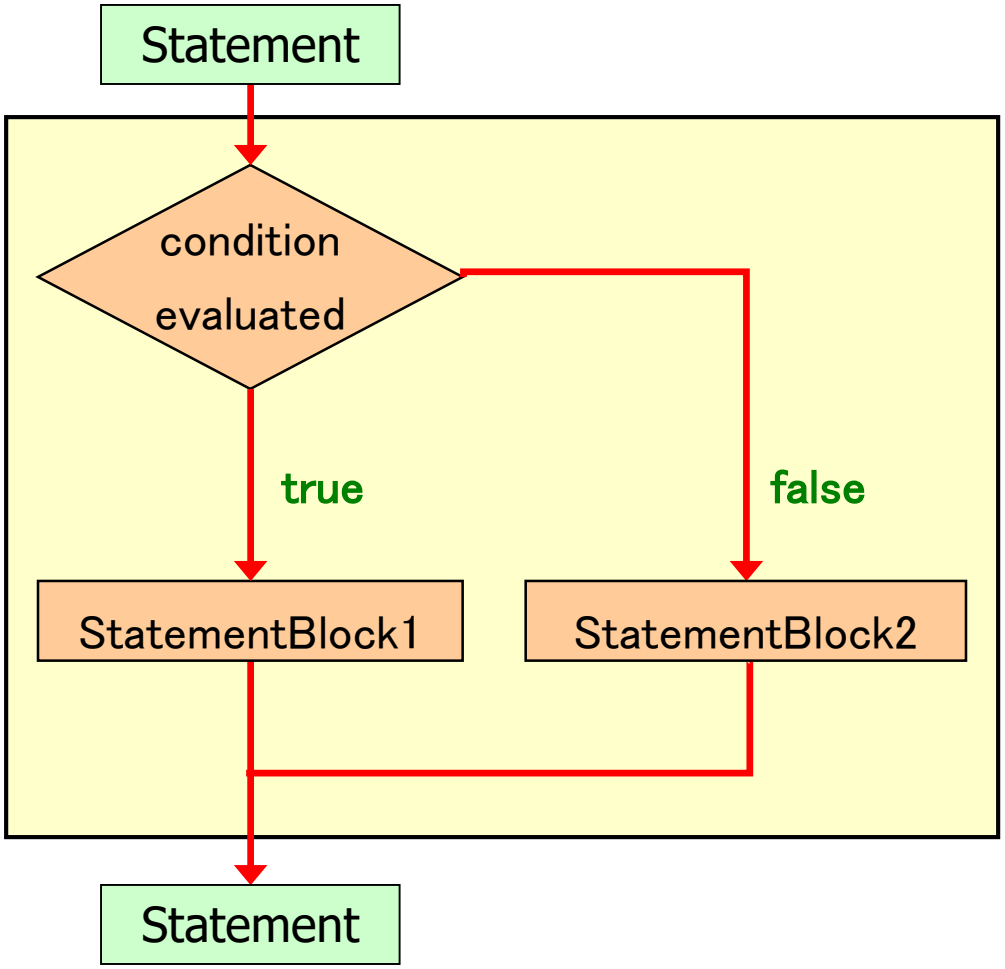
4. The if-else Statement

- **An *else clause* can be added to an `if` statement to make an *if-else statement***

```
if ( condition )
   statementBlock1;
else
   statementBlock2;
```

- **If the *`condition`* is true, *`statementBlock1`* is executed; if the condition is false, *`statementBlock2`* is executed**
- **One or the other will be executed, but not both**

Logic of an if-else statement



Trace if-else statement

Suppose score is 70.0

The condition is false

```
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");
```

Trace if-else statement

Suppose score is 70.0

The condition is false

```
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");
```

Trace if-else statement

Suppose score is 70.0

The condition is true

```
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");
```

Trace if-else statement

Suppose score is 70.0

grade is C

```
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");
```

Trace if-else statement

Suppose score is 70.0

Exit the if statement

```
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");
```

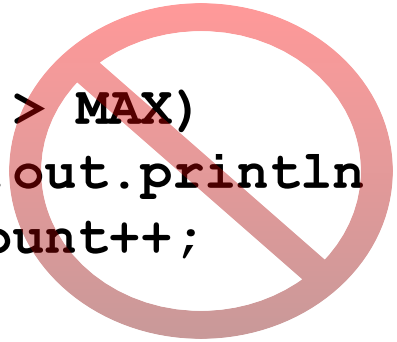
- See Wages.java example next slide.

Example

```
// Wages.java
import java.text.NumberFormat;
import java.util.Scanner;
public class Wages
{
  public static void main (String[] args)
  {
    final double RATE = 8.25; //regular pay rate
    final int STANDARD = 40; //weekly hours
    Scanner scan = new Scanner (System.in); //scanner object
    double pay = 0.0;  // initialization

    System.out.print ("Enter the number of hours worked: "); //prompt
    int hours = scan.nextInt(); //read input value
    System.out.println (); //print blank line

    // Pay overtime at "time and a half"
    if (hours > STANDARD)
        pay = STANDARD * RATE + (hours-STANDARD) * (RATE * 1.5);
    else
        pay = hours * RATE;

    NumberFormat fmt = NumberFormat.getCurrencyInstance();//format
    System.out.println ("Gross earnings: " + fmt.format(pay));//output
  }
}
```

Indentation - Revisited

- **Remember that indentation is for the human reader, and is ignored by the computer**



```
if (total > MAX)
   System.out.println ("Error!!");
   errorCount++;
```

Despite what is implied by the indentation, the increment will occur whether the condition is true or not

Block Statements

- **Several statements can be grouped together into a *block statement delimited by braces***

```
if (total > MAX)
{
   System.out.println ("Error!!");
   errorCount++;
   // more statements…
}
```

Block Statements

- **In an `if-else` statement, the `if` portion, or the `else` portion, or both, could be block statements**

```
if (total > MAX)
   {
    System.out.println ("Error!!");
    errorCount++;
   }
else
   {
    System.out.println ("Total: " + total);
    current = total * 2;
   }
```

- **See Guessing.java next slide.**

Example

```
// Guessing.java
import java.util.*;
public class Guessing
{
   public static void main (String[] args)
   {
     final int MAX = 10;
     int answer, guess;
     Scanner scan = new Scanner (System.in); //scanner object
     Random generator = new Random(); //number generator object
     answer = generator.nextInt(MAX) + 1; //generate a number
     System.out.print ("I'm thinking of a number between 1"
                       + "and " + MAX + ". Guess what it is: ");
     guess = scan.nextInt(); //read user input
     if (guess == answer)
         System.out.println ("You got it! Good guessing!");
     else
       {
         System.out.println ("That is not correct!");
         System.out.println ("The number was " + answer);
       }
   }
}
```

5. The Conditional Operator

Stop and Record…

5. The Conditional Operator

- **Java has a *conditional operator* that uses a boolean condition to determine which of two expressions is evaluated**
- **Its syntax is:**

```
condition ? expression1 : expression2
```

- **If the `condition` is true, `expression1` is evaluated; if it is false, `expression2` is evaluated**
- **The conditional operator is *ternary* because it requires three operands**

The Conditional Operator

- **The conditional operator is similar to an `if-else` statement, except that it is an expression that returns a value**
- **For example:**

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

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

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

The Conditional Operator

- **Another example:**

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

- **If `count` equals 1, then `"Dime"` is printed**
- **If `count` is anything other than 1, then `"Dimes"` is printed**

Nested if Statements

- **The statement executed as a result of an `if` statement or `else` clause could be another `if` statement**
- **These are called *nested if statements***
- **Java Rule: An `else` clause is matched to the last unmatched `if` (no matter what the indentation implies)**
- **Braces can be used to specify the `if` statement to which an `else` clause belongs**
- **See MinOfThree.java next slide**

Example

```
// MinOfThree.java
import java.util.Scanner;
public class MinOfThree
{
   public static void main (String[] args)
   {  int num1, num2, num3, min = 0;
      Scanner scan = new Scanner (System.in);
      System.out.println ("Enter three integers: ");
      num1 = scan.nextInt();
      num2 = scan.nextInt();
      num3 = scan.nextInt();
      if (num1 < num2)
          if (num1 < num3)
              min = num1;
          else
              min = num3;
      else
          if (num2 < num3)
              min = num2;
          else
              min = num3;
      System.out.println ("Minimum value: " + min);
   }
}
```

```
if (num1 < num2)
    min = num1;
else
    min = num2;

if (num3 < min)
    min = num3;
```

6. Switch Statement

- **The *switch statement* provides another way to decide which statement to execute next**
- **The `switch` statement evaluates an expression, then attempts to match the result to one of several possible *cases (options)***
- **Each case contains a value and a list of statements**
- **The flow of control transfers to statement associated with the first case value that matches**

Syntax

- **The general syntax of a `switch` statement is:**

`switch` and `case` are reserved words

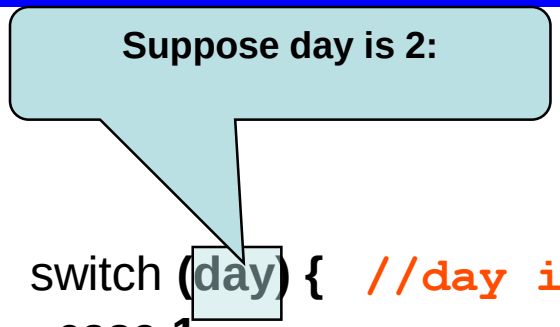
```
switch (expression)
{
   case value1:
        statement_List1
        break;
   case value2:
        statement_List2
        break;
   case value3:
        statement_List3
        break;
   case ...

   default:
        statement_List
}
```

If *expression* matches *value2*, control jumps to here

`break` Statement

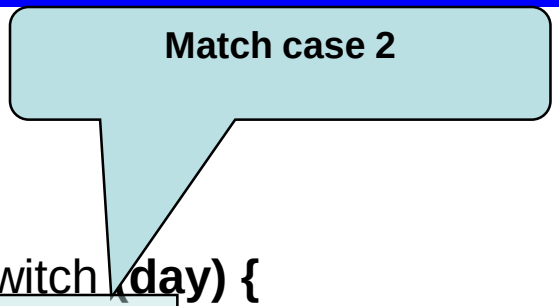
- **Often a *break statement* is used as the last statement in each case's statement list**
- **A `break` statement causes control to transfer to the end of the `switch` statement**
- **If a `break` statement is not used, the flow of control will continue into the next case**
- **Sometimes this may be appropriate, but often we want to execute only the statements associated with one case**

Trace switch statement



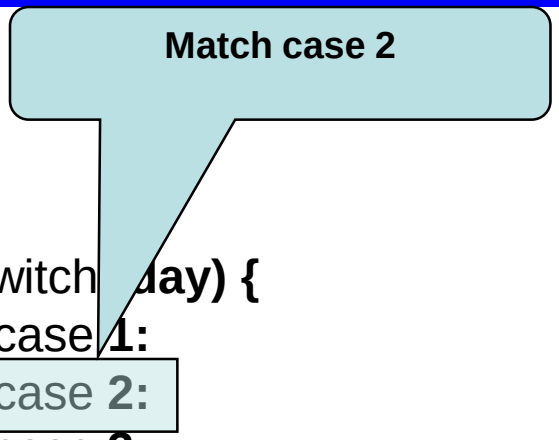

```
switch (day) {  //day is of type int
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 6:
  case 7: System.out.println("Weekend");
}
```

Trace switch statement



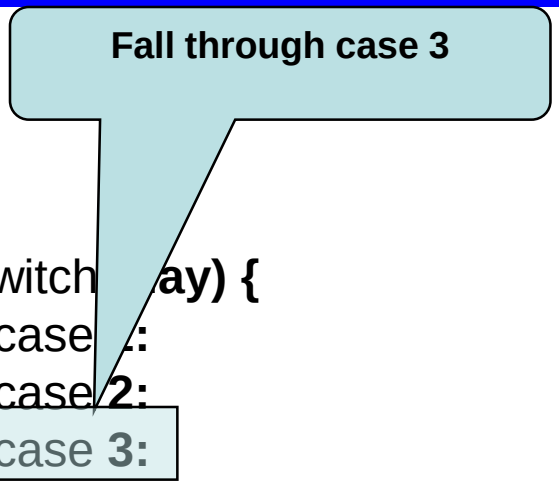

```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 6:
  case 7: System.out.println("Weekend");
}
```

Trace switch statement



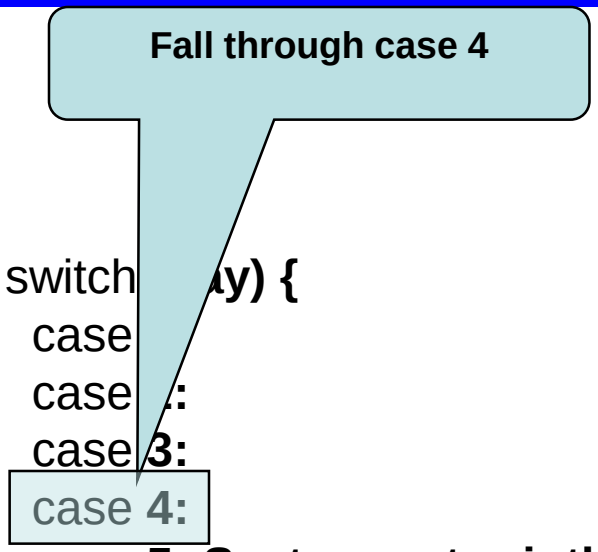

```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 6:
  case 7: System.out.println("Weekend");
}
```

Trace switch statement



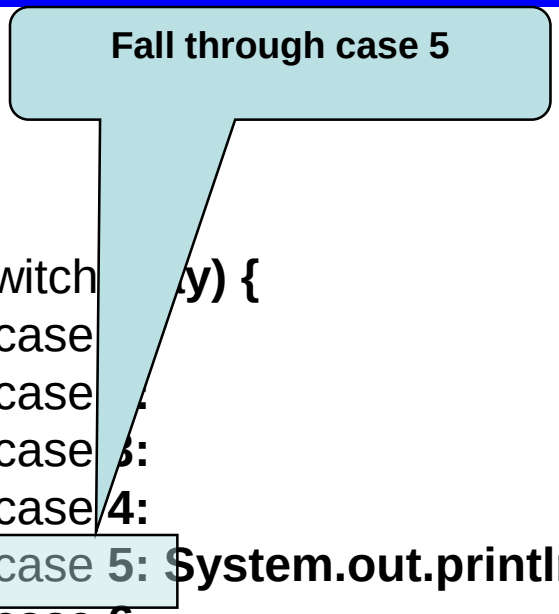

```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 6:
  case 7: System.out.println("Weekend");
}
```

Trace switch statement



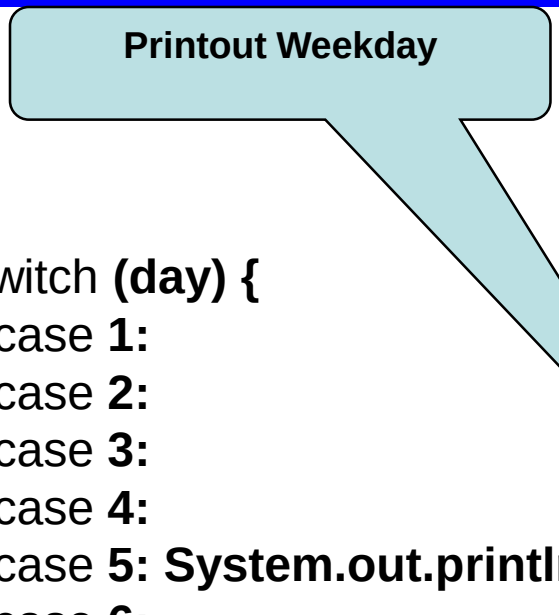

```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 6:
  case 7: System.out.println("Weekend");
}
```

Trace switch statement



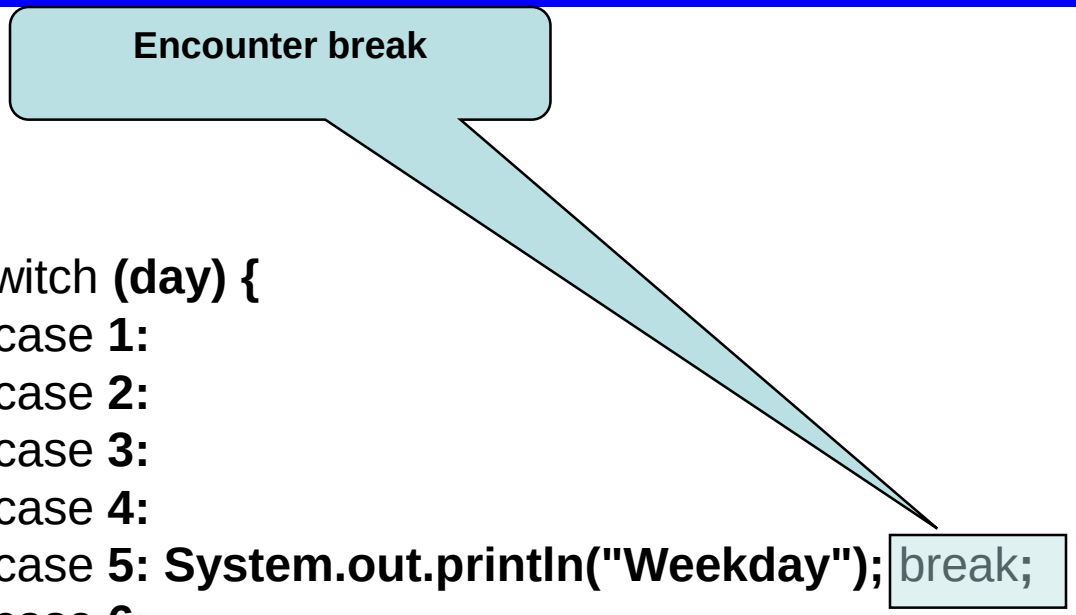

```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 6:
  case 7: System.out.println("Weekend");
}
```

Trace switch statement

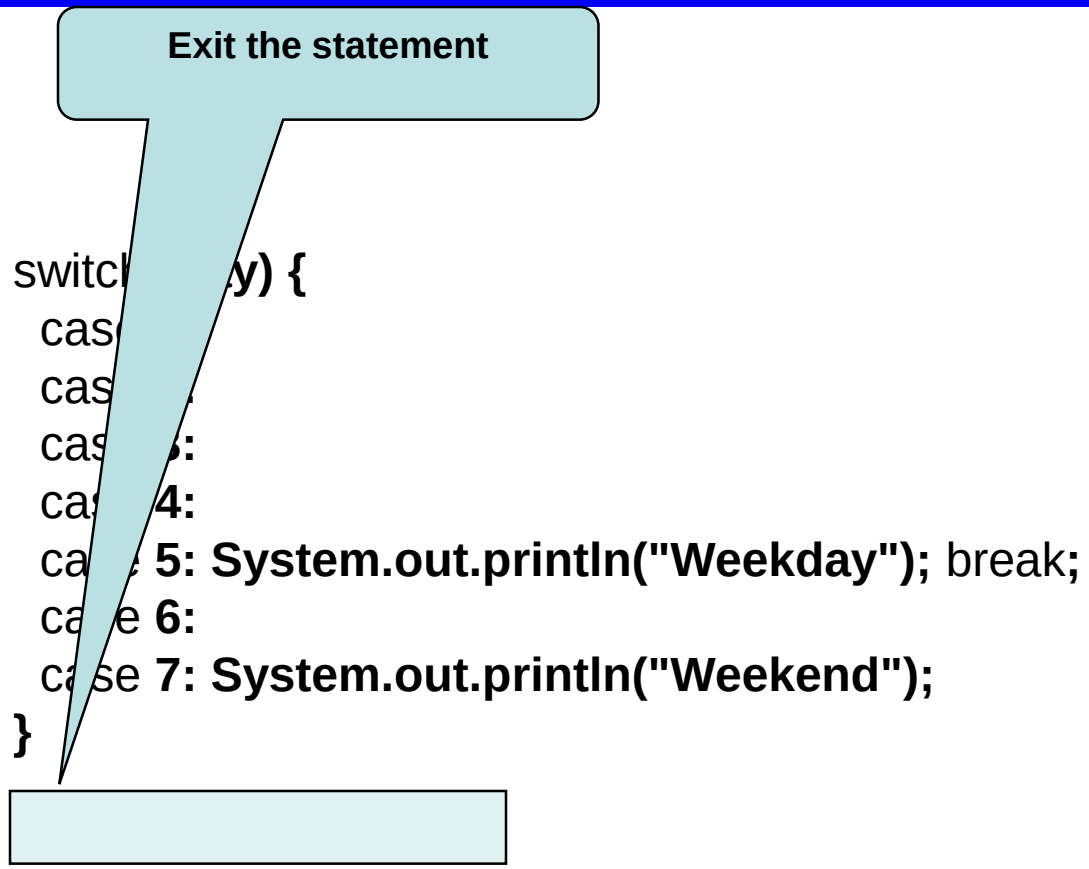



```
switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Weekday"); break;
  case 6:
  case 7: System.out.println("Weekend");
}
```

Trace switch statement



Trace switch statement



Default Case

- A `switch` statement can have an optional *default case*
- The default case has no associated value and simply uses the reserved word `default`
- If the default case is present, control will transfer to the default case if no other case value matches
- If there is no default case, and no other value matches, control falls through to the statement after the switch statement

Example

```
switch (option) //option is of type char
{
    case 'A':
        aCount = aCount + 1;
        break;
    case 'B':
        bCount = bCount + 1;
        break;
    case 'C':
        cCount = cCount + 1;
        break;
    default:
        System.out.println ("Invalid Option…")
}
```

Switch Statement Expression

- **The expression of a `switch` statement must result in an *integer type* (`byte`, `short`, `int`, `long`) or a *char type.***
- **It cannot be a `boolean` value or a floating point value (`float` or `double`)**
- **You cannot perform relational checks with a `switch` statement**
- **See GradeReport.java next slide**

Example

```
import java.util.Scanner;
public class GradeReport
{ public static void main (String[] args)
  {  ...  Some other code here
     grade = scan.nextInt();
     category = grade / 10;
     System.out.print ("That grade is ");
     switch (category)
     {
       case 10:
           System.out.println ("a perfect score, well done.");
           break;
       case 9:
           System.out.println ("well above average. Excellent.");
           break;
       case 8:
           System.out.println ("above average. Nice job.");
           break;
       case 7:
           System.out.println ("average.");
           break;
       case 6:
           System.out.println ("below average. Do better!");
           break;
       default:
           System.out.println ("not passing.");
    }
  }
}
```

7. Useful Hints

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

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

==========

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

Same as

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

Useful Hints

Nested if statements and style issue.

```
if (score >= 90.0)
  grade = 'A';
else
  if (score >= 80.0)
    grade = 'B';
  else
    if (score >= 70.0)
      grade = 'C';
    else
      if (score >= 60.0)
        grade = 'D';
      else
        grade = 'F';
```

Equivalen

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

Useful Hints

The else clause matches the most recent if clause in the same block.

```
int i = 1;
int j = 2;
int k = 3;

if (i > j)
   if (i > k)
     System.out.println("A");
else
    System.out.println("B");
```

Equivalent

```
int i = 1;
int j = 2;
int k = 3;

if (i > j)
   if (i > k)
     System.out.println("A");
   else
     System.out.println("B");
```

```
if (even == true)
  System.out.println(
    "It is even.");
```

Equivalent

```
if (even)
  System.out.println(
    "It is even.");
```

Useful Hints

Adding a semicolon at the end of an if clause is a common mistake.

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

This mistake is hard to find, because it is not a compilation error or a runtime error, it is a logical error.