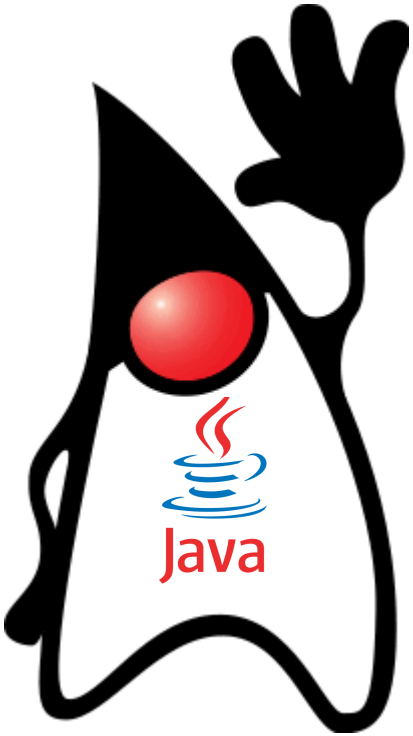
Control Statements: Selections



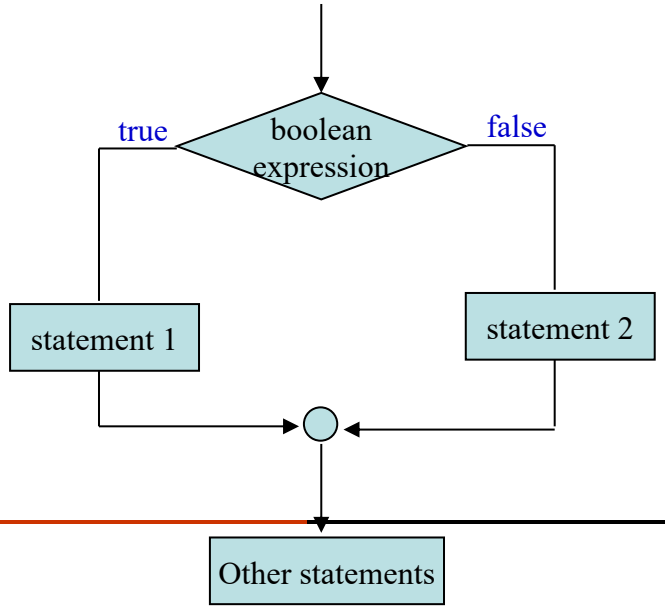
Dr. Praisan Padungweang


CHOICES

The if-else Statement

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

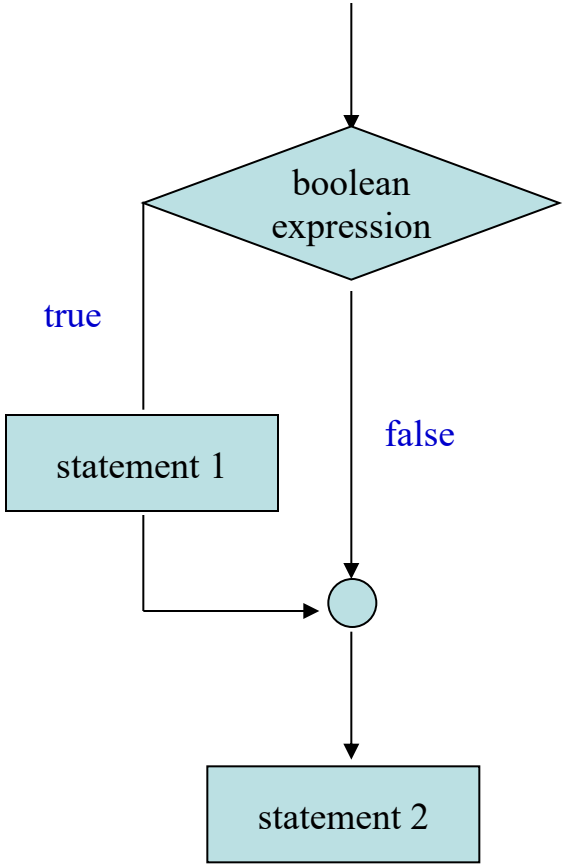
```
if (boolean expression)
   statement1;
else
   statement2;
```



Logic of an if statement

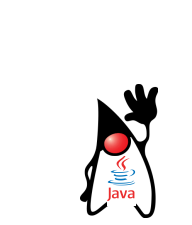
```
if (boolean expression)
   statement1;
Other statements;
```

```
if (boolean expression){
   statement1;
}
Other statements;
```

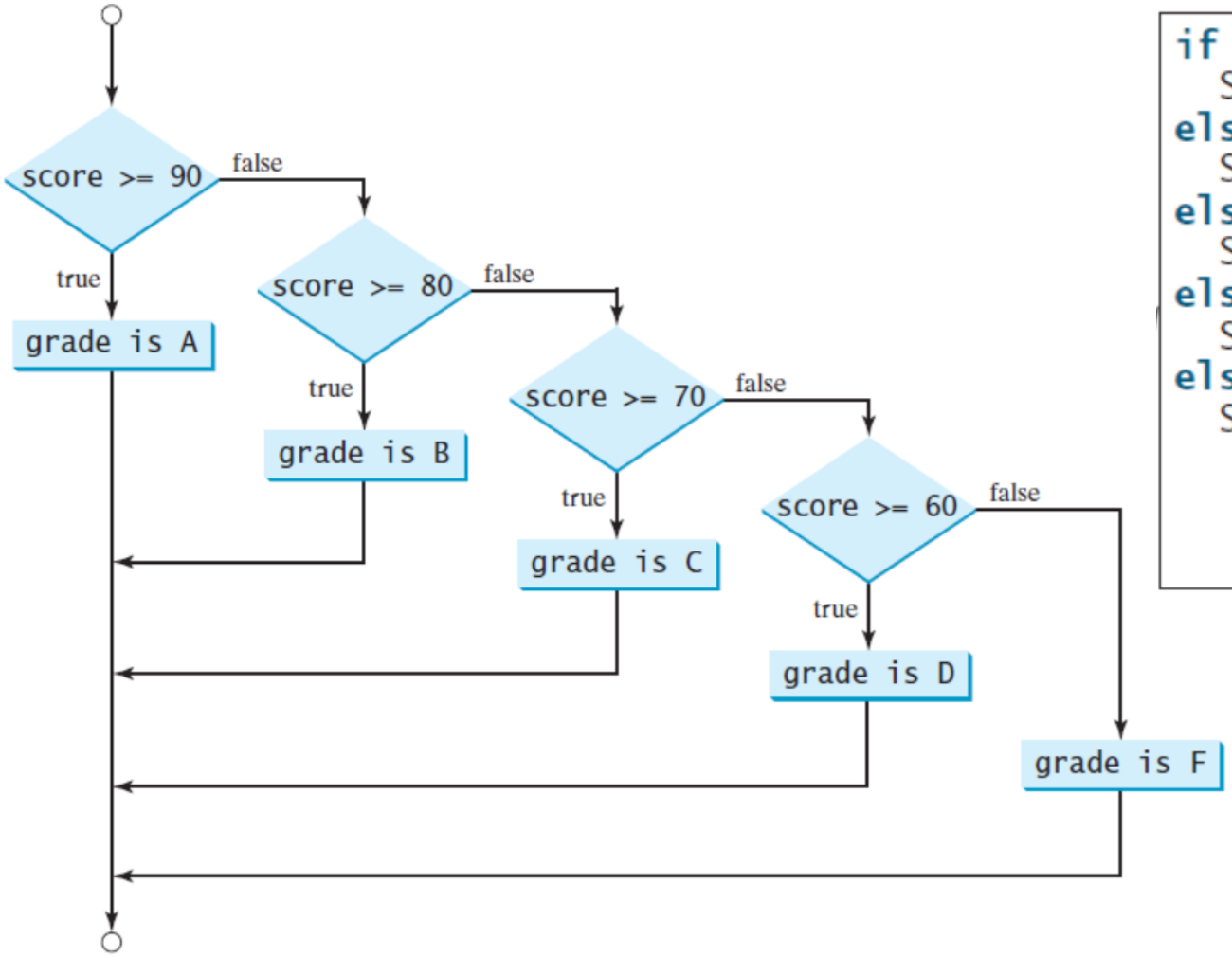


```java
import java.util.Scanner;

public class SimpleIfDemo {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.println("Enter an integer: ");
    int number = input.nextInt();

    if (number % 5 == 0)
      System.out.println("HiFive");

    if (number % 2 == 0)
      System.out.println("HiEven");
  }
}
```

☑Check point



Nested if Statements




```
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");
```

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

Logical Operators

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

!	`!a`	**Logical NOT**
&	`a && b`	**Logical AND (`&&` short-circuited AND)**
|	`a || b`	**Logical OR (`||` short-circuited OR)**

- They all take boolean operands and produce boolean results

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`

a	b	a && b	a || b
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

The & and | Operators

- If x is 1, what is x after this expression?

```
(x > 1) & (x++ < 10)
```

- If x is 1, what is x after this expression?

```
(1 > x) && ( 1 > x++)
```

- How about (1 == x) | (10 > x++)?

```
(1 == x) || (10 > x++)?
```

&& and || must be used carefully

Comparing Data

- Comparing Float Values
- Comparing Characters
- Comparing Strings
- Comparing Multiple value

Comparing Data

```
double x = 1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1;
System.out.println(x == 0.5);
```

```
double x = 1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1;
if (Math.abs(x - 0.5) < 0.0000001 )
   System.out.println(x + " is approximately 0.5");
```

```
final double EPSILON = 1E-14;
double x = 1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1;
if (Math.abs(x - 0.5) < EPSILON)
   System.out.println(x + " is approximately 0.5");
```

Danger with Comparing Floating point by using ==

- Floating point arithmetic is not exact.

```
class DecimalFraction {

   public static void main(String[] args) {
      float x = 1.0f;    // 1.0f means 1.0 float
      float y = 10.0f;

      if (x / y == 0.1) {
         System.out.println("x/y == 0.1");
      } else {
         System.out.println("x/y != 0.1");
      }
   }
}
```

Comparing Float Values

- To determine the equality of two floats, you may want to use the following technique:

```
if (Math.abs(f1 - f2) < TOLERANCE)
   System.out.println ("Essentially equal");
```

If the difference between the two floating point values is less than the tolerance, they are considered to be equal

The tolerance could be set to any appropriate level, such as 0.000001

Comparing Characters

- As we've discussed, Java character data is based on the Unicode character set
- Unicode establishes a particular numeric value for each character, and therefore an ordering
- We can use relational operators on character data based on this ordering
- For example, the character `'+'` is less than the character `'J'` because it comes before it in the Unicode character set

Comparing Characters

- In Unicode, the digit characters (0-9) are contiguous and in order
- Likewise, the uppercase letters (A-Z) and lowercase letters (a-z) are contiguous and in order

Characters	Unicode Values
0 – 9	48 through 57
A – Z	65 through 90
a – z	97 through 122

Comparing Characters

```
char ch1='B';
char ch2='a';
System.out.println(ch1<ch2); //true
System.out.println(ch1>ch2); //false
System.out.println(ch1==ch2); //false
```

Comparing Strings

- Remember that in Java a character string is an object
- The `equals` method can be called with strings to determine if two strings contain exactly the same characters in the same order
- The `equals` method returns a boolean result

```
if (name1.equals(name2))
   System.out.println ("Same name");
```

Comparing Strings

- We cannot use the relational operators to compare strings
- The `String` class contains a method called `compareTo` to determine if one string comes before another
- A call to `name1.compareTo(name2)`
  - returns zero if `name1` and `name2` are equal (contain the same characters)
  - returns a negative value if `name1` is less than `name2`
  - returns a positive value if `name1` is greater than `name2`

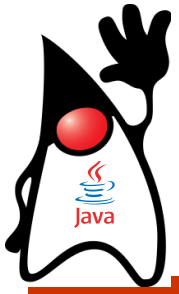
Comparing Strings

```
if (name1.compareTo(name2) < 0)
   System.out.println (name1 + "comes first");
else
   if (name1.compareTo(name2) == 0)
      System.out.println ("Same name");
   else
      System.out.println (name2 + "comes first");
```

Because comparing characters and strings is based on a character set, it is called a *lexicographic ordering*

Lexicographic Ordering

- Lexicographic ordering is not strictly alphabetical when uppercase and lowercase characters are mixed
- For example, the string `"Great"` comes before the string `"fantastic"` because all of the uppercase letters come before all of the lowercase letters in Unicode
- Also, short strings come before longer strings with the same prefix (lexicographically)
- Therefore `"book"` comes before `"bookcase"`

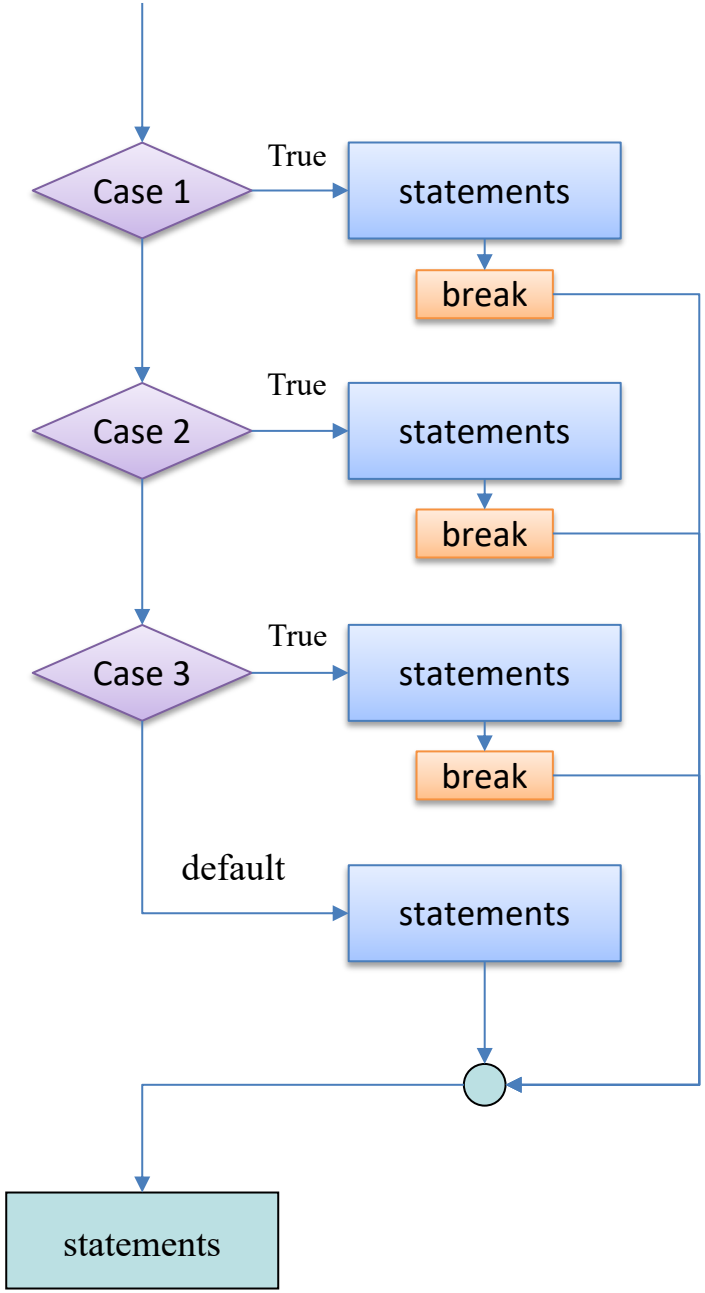


The switch Statement

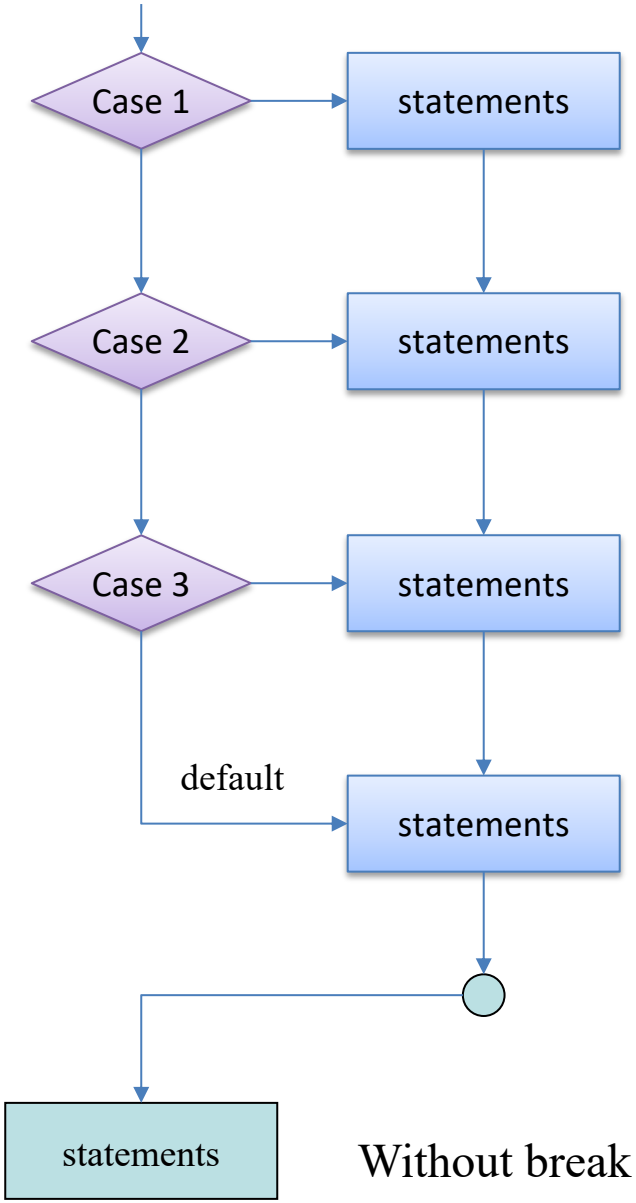
The switch Statement

- 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*
- 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

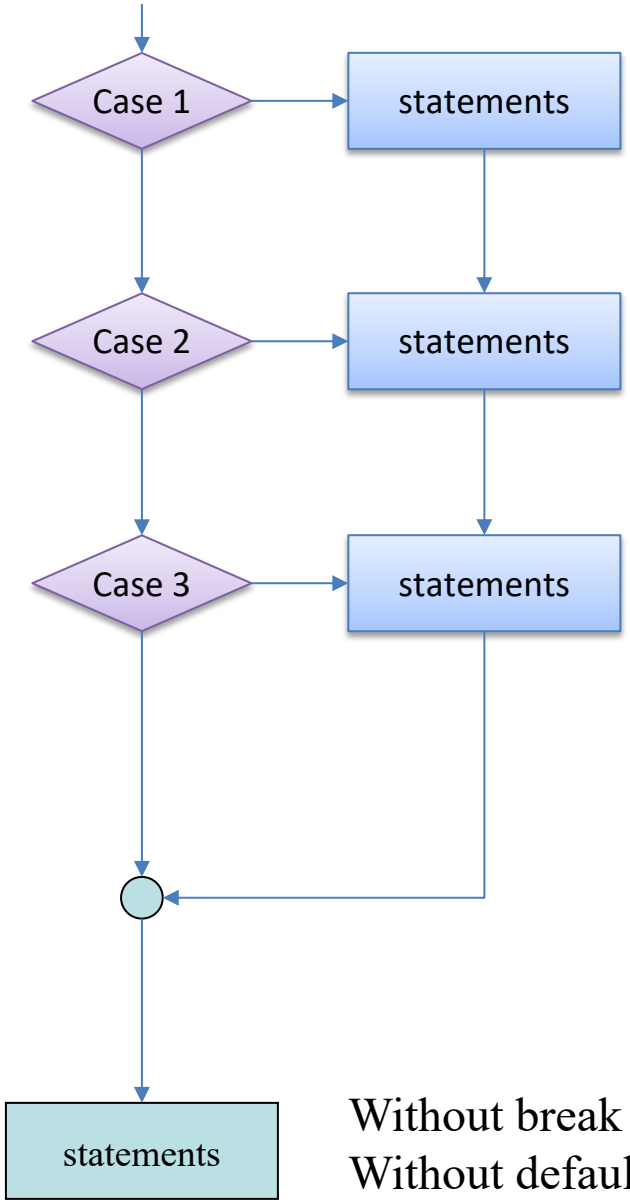
The switch Statement



The switch Statement



Without break



Without break
Without default

The switch Statement

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

```
switch (integer expression ){
    case value1 :
        statement-list1;
        break;
    case value2 :
        statement-list2;
         break;
    case value3 :
        statement-list3;
        break;
    default:
        statement-list;
}
```

Switch and **case** are reserved words

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

With break And default

The switch Statement

- The expression of a `switch` statement must result in an *integral type*, meaning an integer (`byte`, `short`, `int`) or a `char`
- In version 1.7 up, switch expression can be String type
- It cannot be a `boolean` value or a floating point value (`float` or `double`) or `long`
- The implicit boolean condition in a `switch` statement is equality
- You cannot perform relational checks with a `switch` statement

The switch Statement

- An example of a switch statement:

```
switch (option){
    case 'A':
        aCount = aCount+1;
        break;
    case 'B':
        bCount = bCount+1;
        break;
    case 'C':
        cCount = cCount+1;
        break;
}
```

3.31 What is `x` after the following `if-else` statement is executed? Use a `switch` statement to rewrite it and draw the flowchart for the new `switch` statement.

```
int x = 1, a = 3;
if (a == 1)
  x += 5;
else if (a == 2)
  x += 10;
else if (a == 3)
  x += 16;
else if (a == 4)
  x += 34;
```

3.32 Write a `switch` statement that displays Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, if `day` is `0`, `1`, `2`, `3`, `4`, `5`, `6`, accordingly.

☑Check point

Write a program to find out the Chinese Zodiac sign for a given year. The Chinese Zodiac is based on a twelve-year cycle, with each year represented by an animal-monkey, rooster, dog, pig, rat, ox, tiger, rabbit, dragon, snake, horse, or sheep-in this cycle, as shown in. Note that **year % 12** determines the Zodiac sign. 1900 is the year of the rat because **1900% 12** is **4**. Write a program that prompts the user to enter a year and displays the animal for the year.

year % 12 =
- 0: monkey
- 1: rooster
- 2: dog
- 3: pig
- 4: rat
- 5: ox
- 6: tiger
- 7: rabbit
- 8: dragon
- 9: snake
- 10: horse
- 11: sheep

```
Enter a year:  1963  Enter
rabbit
```

```
Enter a year:  1877  Enter
ox
```