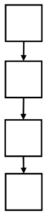
Choices and Selection

This lecture will

- Introduce control structures
- Explain the **`if`** and the **`if – else`** statement for making simple decisions
- Discuss the implications of swapping values
- Explain compound statements
- Introduce Boolean expressions and logical operators
- Explain how decisions between multiple alternatives can be made using **`switch`**
- Discuss the problems of comparing **`String`**s

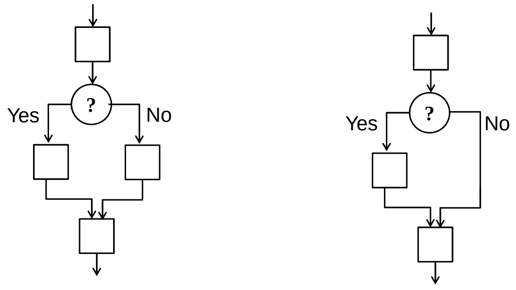
Flow of Control

- The way that Java moves from one statement to the next is called the **flow of control** in a program
- So far we have only seen **Sequence** - doing one statement after the next in order starting at the first statement in the main method



Selection

- In **Selection** the flow of control determined by a simple yes/no decision

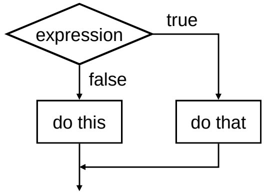


Simple selection

- Selection statements involve **Boolean expressions** that are either **`true`** or **`false`** (a binary decision). The action performed depends on the value of the expression.

Example (in pseudocode):

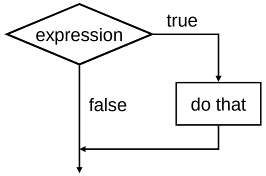
```
if I feel energetic then
   walk to work
else
   take a bus to work
```



Omitting the `else` clause

- Consider this selection (in pseudocode again):

```
if I feel hungry then
    buy a sandwich
else
    do nothing
```



- We can omit the ‘do nothing’ clause as follows:

```
if I feel hungry then
    buy a sandwich
```

Simple `if` statements in Java

```
if (  age >= 18  )
   System.out.println("Eligible for jury service");

if (  fruitAndVegPerDay < 5  )
   System.out.println("Eat more greens");

if (  numberOfKids == 3  )
   incomeSupport = incomeSupport*2;

if (  i != j  )
   System.out.println("i and j are not equal");
```

Notice there are no semicolons after **`if (...)`**

Relational operators

- The Boolean test in the **`if`** statement is performed using a **relational operator**:

Operator	Meaning
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to

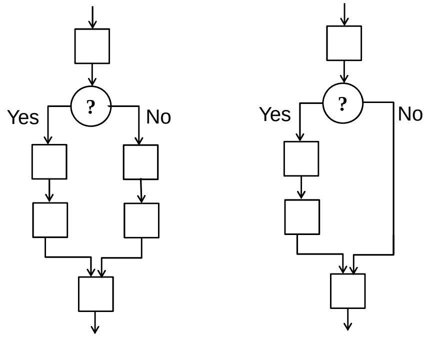
Simple `if-else` statements in Java

```
if (  age >= 18  )
   System.out.println("Eligible for jury service");
else
   System.out.println("Too young for jury service");

if (  age > 60 )
   benefit = (age-60)*annualrate;
else
   System.out.println("No benefit is payable");

if (  i != j  )
   System.out.println("i and j are not equal");
else
   System.out.println("i and j are equal");
```

Notice there are no semicolons after **`if (...)`** or after **`else`**

Selection

- Suppose we want to do more than a single thing after the Selection?



Swapping Values

- Consider the following fragment of psudocode:

  Read in two values and make sure the biggest is stored in a variable called biggest and the smallest in a variable called smallest.

```
int biggest = keyboard.readInt(
             "Please type a number ");
int smallest = keyboard.readInt(
             "Please type a number ");
if  (  smallest > biggest  )
   biggest = smallest;
if  (  biggest < smallest  )
   smallest = biggest;
```

✗

Swapping Values

- Consider the following fragment of psudocode:

  Read in two values and make sure the biggest is stored in a variable called biggest and the smallest in a variable called smallest.

```
    temporary = biggest;
    biggest = smallest;
    smallest = temporary;
```

Compound statements

- We can identify two kinds of statement in Java; single and compound .
- An example of a **single statement** is:

```
 sum = larger+smaller;
```

- A **compound statement** is a sequence of statements enclosed in curly brackets:

```
{
    temporary = larger;
    larger = smaller;
    smaller = temporary;
}
```

More about compound statements

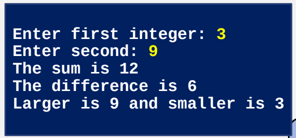
- We can use compound statements in an **if** construct in the same way that we use single statements:

```
if (  larger < smaller  ) {
    temporary = larger;
    larger = smaller;
    smaller = temporary;
}
```

- Indentation helps to clarify which statements are part of the same compound statement (or ‘block’)
- The program will work if you fail to indent statements within a compound statement **but you will lose marks for it**

Sorting via intermediate variables

```
EasyReader keyboard = new EasyReader();
int first = keyboard.readInt("Enter first integer: ");
int second = keyboard.readInt("Enter second: ");

int larger, smaller;
if (first < second) {
   smaller = first;
   larger = second;
}
else {
   smaller = second;
   larger = first;
}

System.out.println("The sum is "+(smaller+larger));
System.out.println("The difference is "+
                        (larger-smaller));
System.out.println("Larger is "+larger+
                        " and smaller is " +smaller);
```



```
Enter first integer: 3
Enter second: 9
The sum is 12
The difference is 6
Larger is 9 and smaller is 3
```

Note brackets

Sorting by swapping

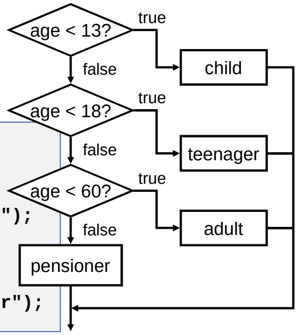
```
EasyReader keyboard = new EasyReader();
int larger = keyboard.readInt("Enter first integer: ");
int smaller = keyboard.readInt("Enter second: ");

if (larger < smaller) {
   int temporary = larger;
   larger = smaller;
   smaller = temporary;
}
int sum = larger+smaller;
int difference = larger-smaller;

System.out.println("The sum is "+sum);
System.out.println("The difference is "+difference);
System.out.println("Larger is "+larger+
                " and smaller is "+smaller);
```

temporary is declared in the compound statement and can only be used there

Multiple selection in Java

- Selections between multiple alternatives can be broken down into a sequence of binary decisions:

```
if (  age < 13  )
   System.out.println("child");
else if (  age < 18  )
   System.out.println("teenager");
else if (  age < 60  )
   System.out.println("adult");
else
   System.out.println("pensioner");
```



More about multiple selections

- No more than one statement is executed in a multiple-alternative **`if`** selection.
- The ordering of the tests is important.

❓ **What would be the result if we tested for higher ages first?**

```
if (  age < 60  )
   System.out.println("adult");
else if (  age < 18  )
   System.out.println("teenager");
else if (  age < 13  )
   System.out.println("child");
else
   System.out.println("pensioner");
```

The boolean type

- We can have variables of type **`boolean`** as well as Boolean expressions in Java:

```
boolean hasBigFeet = true;
```

- We can assign the result of a Boolean expression to a variable of type **`boolean`**:

```
hasBigFeet = shoeSize > 11;
```

- Boolean variables can themselves be compared using == and != but none of the other relational operators
- Boolean variables can be assigned the Boolean literal values **`true`** or **`false`**

Nested `if` statements

```
EasyReader keyboard = new EasyReader();
boolean rainTomorrow = keyboard.readBoolean(
      "Will it rain tomorrow? ");
boolean dryTomorrow  = keyboard.readBoolean(
      "Will it be dry tomorrow? ");

if  ( rainTomorrow  != dryTomorrow )

   if  (  rainTomorrow  )
      System.out.println("It will rain tomorrow");
   else
      System.out.println("It will be dry tomorrow");

else
   System.out.println("I don't know what the weather"+
                                    " will be like tomorrow");
```

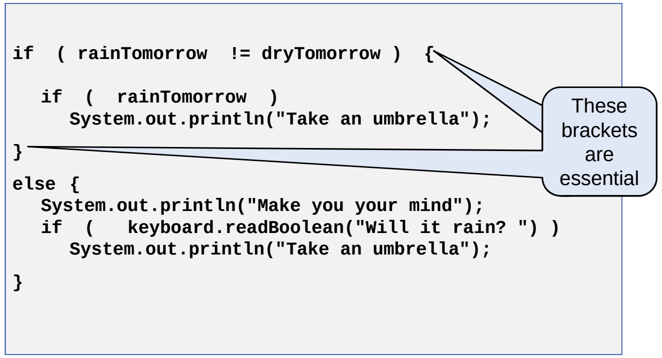
A Nested if

❓ **How does Java know which `else` goes with each `if`?**

Nested `if` statements

```
if  ( rainTomorrow  != dryTomorrow )

   if  (  rainTomorrow  )
      System.out.println("It will rain tomorrow");
   else
      System.out.println("It will be dry tomorrow");

else {

   System.out.println("Make you your mind");
   if  (   keyboard.readBoolean("Will it rain? ") )
      System.out.println("Take an umbrella");
}
```

Nested `if` statements



```
if  ( rainTomorrow  != dryTomorrow )  {

   if  (  rainTomorrow  )
      System.out.println("Take an umbrella");
}
else {
   System.out.println("Make you your mind");
   if  (   keyboard.readBoolean("Will it rain? ") )
      System.out.println("Take an umbrella");
}
```

These brackets are essential

The `boolean` operators

- A variable declared as a **`boolean`** can be either **`true`** or **`false`**
- We can make expressions using **`boolean`** values and the usual logical operators

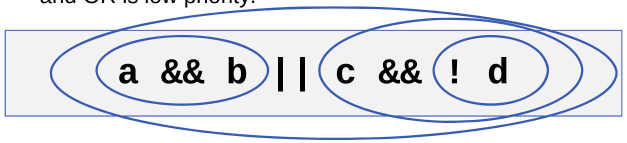
Operator	Symbol
And	&&
Or	||
Not	!
Equals	==
Not Equals	!=

And, Or and Not

- If we have two boolean variables `boolean a, b;`
- And `a && b`
- is true only if both **a** and **b** are true
- Or `a || b`
- is true if either **a** or **b** or both are true
- Not `! a`
- is true if **a** is false and false if **a** is true

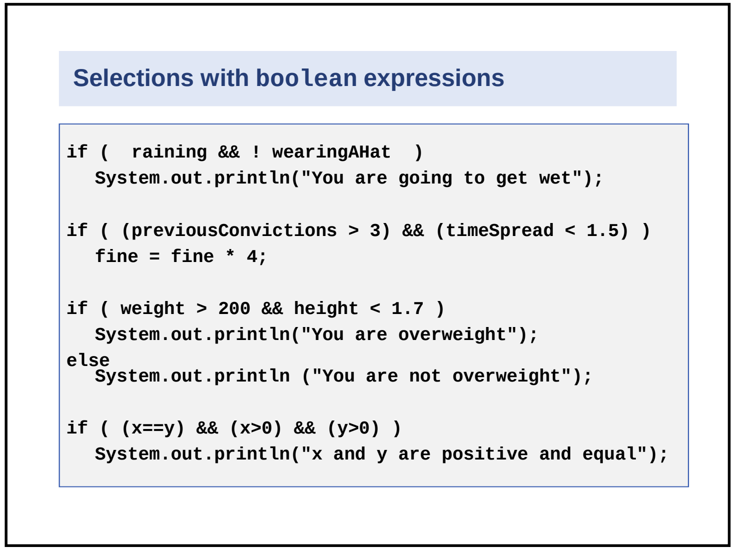
The Boolean operators priority

- Like arithmetic operators, these operators have different precedence; NOT is high priority, AND is medium priority and OR is low priority.



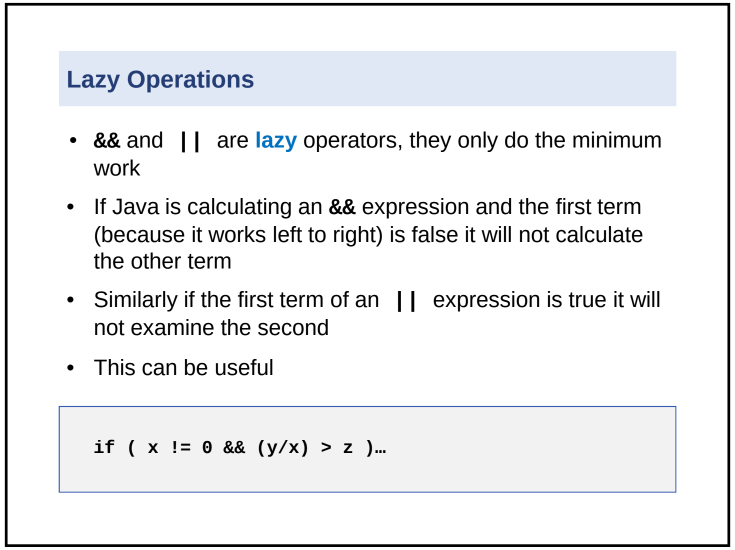
```
a && b || c && ! d
```

- As with any other sort of expression you can use brackets to alter the order of evaluation



Selections with boolean expressions

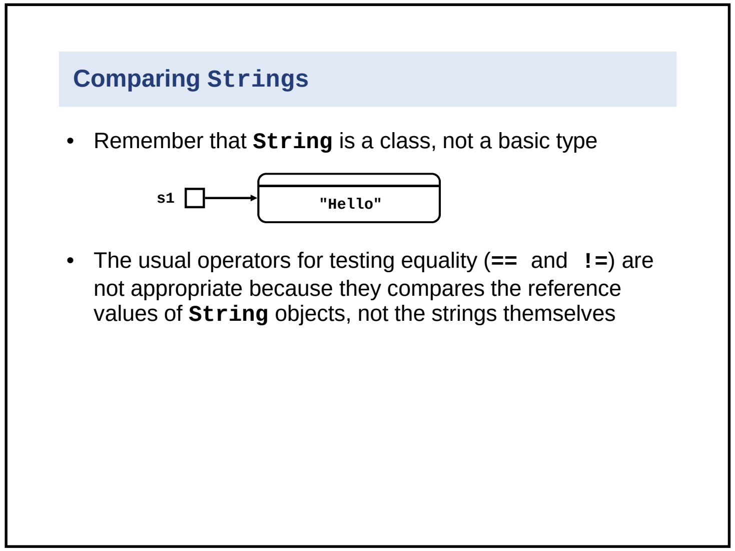
```
if (  raining && ! wearingAHat  )
   System.out.println("You are going to get wet");

if ( (previousConvictions > 3) && (timeSpread < 1.5) )
   fine = fine * 4;

if ( weight > 200 && height < 1.7 )
   System.out.println("You are overweight");
else
   System.out.println ("You are not overweight");

if ( (x==y) && (x>0) && (y>0) )
   System.out.println("x and y are positive and equal");
```



Lazy Operations

- && and || are lazy operators, they only do the minimum work
- If Java is calculating an && expression and the first term (because it works left to right) is false it will not calculate the other term
- Similarly if the first term of an || expression is true it will not examine the second
- This can be useful

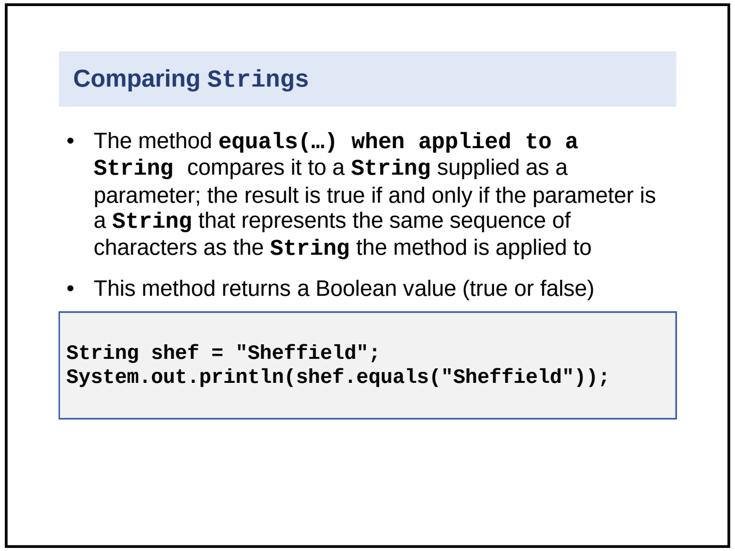
```
if ( x != 0 && (y/x) > z )…
```



Comparing Strings

- Remember that **String** is a class, not a basic type

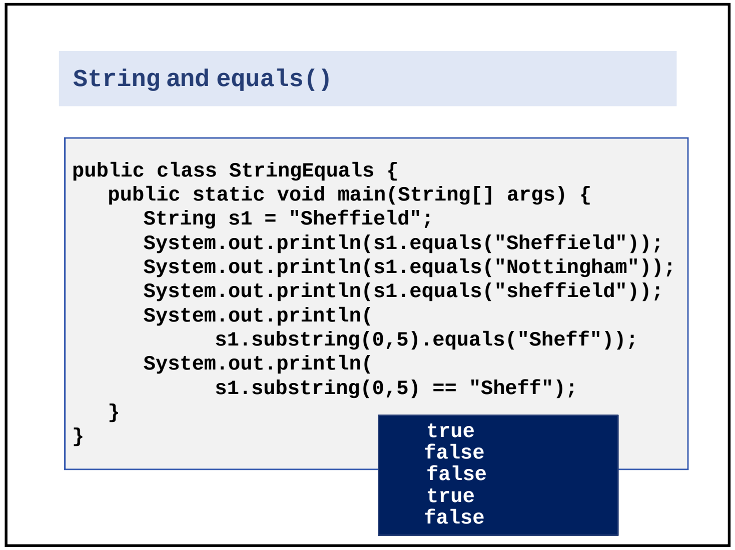
- The usual operators for testing equality (**==** and **!=**) are not appropriate because they compares the reference values of **String** objects, not the strings themselves



Comparing Strings

- The method **equals(…) when applied to a String** compares it to a **String** supplied as a parameter; the result is true if and only if the parameter is a **String** that represents the same sequence of characters as the **String** the method is applied to
- This method returns a Boolean value (true or false)

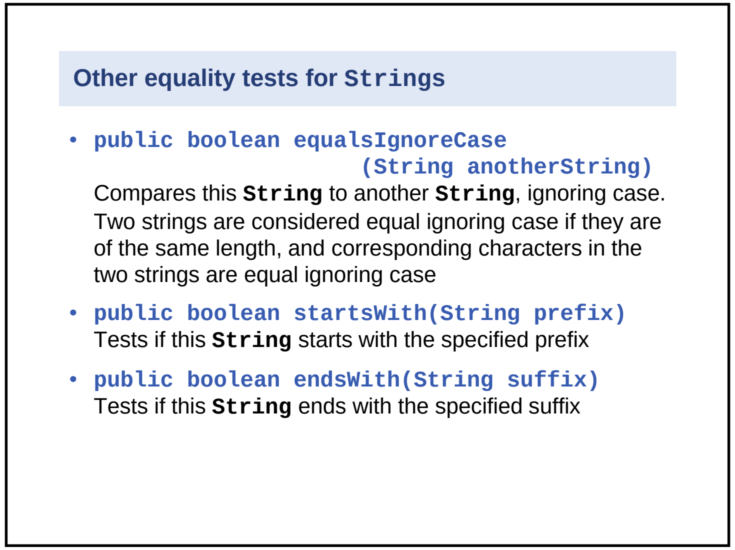
```
String shef = "Sheffield";
System.out.println(shef.equals("Sheffield"));
```



String and equals()

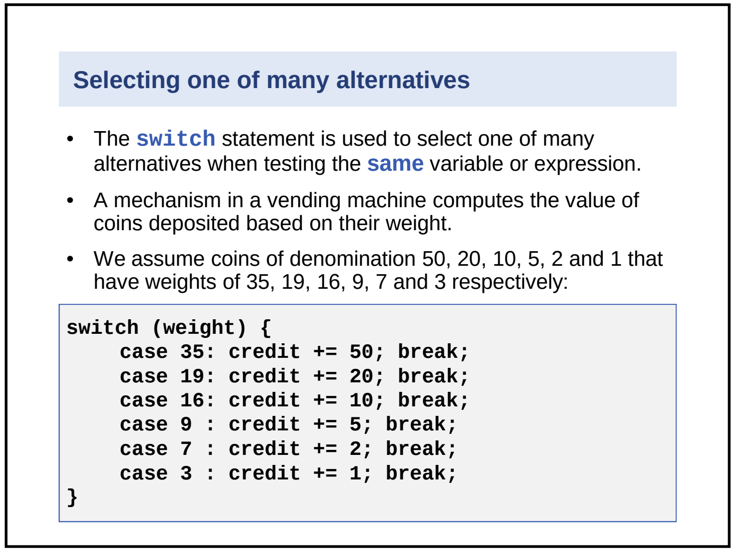
```
public class StringEquals {
   public static void main(String[] args) {
      String s1 = "Sheffield";
      System.out.println(s1.equals("Sheffield"));
      System.out.println(s1.equals("Nottingham"));
      System.out.println(s1.equals("sheffield"));
      System.out.println(
            s1.substring(0,5).equals("Sheff"));
      System.out.println(
            s1.substring(0,5) == "Sheff");
   }
}
```

```
true
false
false
true
false
```



Other equality tests for Strings

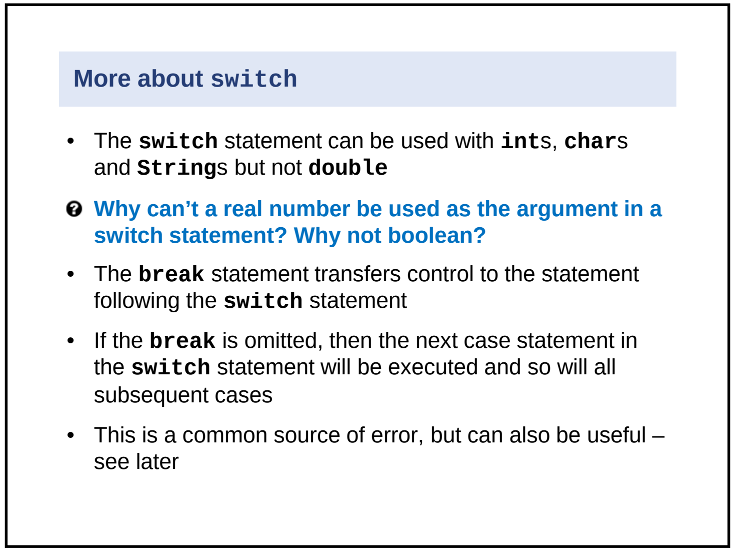
- **public boolean equalsIgnoreCase (String anotherString)**
  Compares this **String** to another **String**, ignoring case. Two strings are considered equal ignoring case if they are of the same length, and corresponding characters in the two strings are equal ignoring case
- **public boolean startsWith(String prefix)**
  Tests if this **String** starts with the specified prefix
- **public boolean endsWith(String suffix)**
  Tests if this **String** ends with the specified suffix



Selecting one of many alternatives

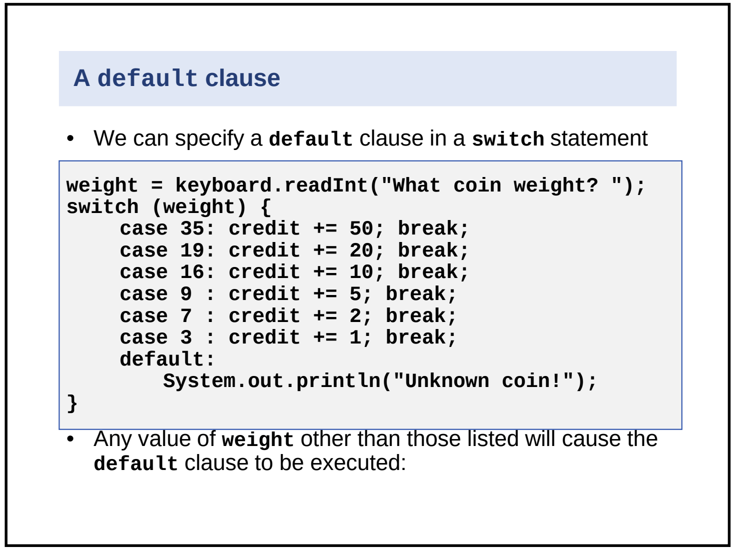
- The **switch** statement is used to select one of many alternatives when testing the **same** variable or expression.
- A mechanism in a vending machine computes the value of coins deposited based on their weight.
- We assume coins of denomination 50, 20, 10, 5, 2 and 1 that have weights of 35, 19, 16, 9, 7 and 3 respectively:

```
switch (weight) {
    case 35: credit += 50; break;
    case 19: credit += 20; break;
    case 16: credit += 10; break;
    case 9 : credit += 5; break;
    case 7 : credit += 2; break;
    case 3 : credit += 1; break;
}
```



More about switch

- The **switch** statement can be used with **int**s, **char**s and **String**s but not **double**
- **Why can't a real number be used as the argument in a switch statement? Why not boolean?**
- The **break** statement transfers control to the statement following the **switch** statement
- If the **break** is omitted, then the next case statement in the **switch** statement will be executed and so will all subsequent cases
- This is a common source of error, but can also be useful – see later

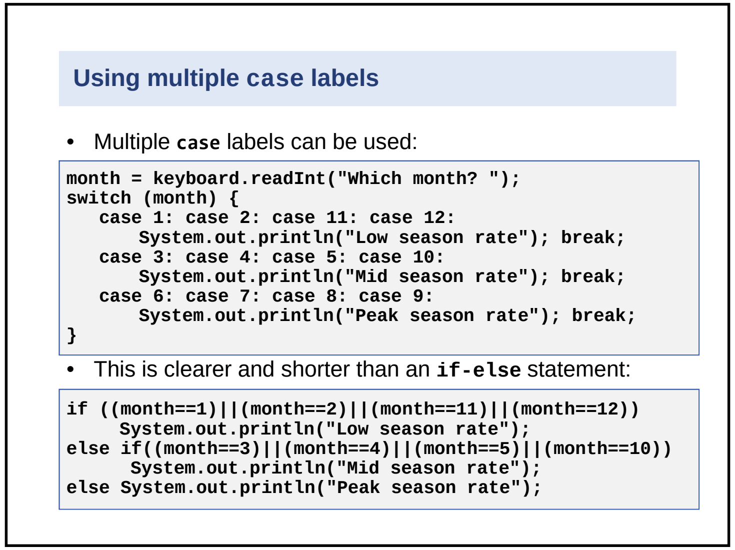


A default clause

- We can specify a **default** clause in a **switch** statement

```
weight = keyboard.readInt("What coin weight? ");
switch (weight) {
    case 35: credit += 50; break;
    case 19: credit += 20; break;
    case 16: credit += 10; break;
    case 9 : credit += 5; break;
    case 7 : credit += 2; break;
    case 3 : credit += 1; break;
    default:
        System.out.println("Unknown coin!");
}
```

- Any value of **weight** other than those listed will cause the **default** clause to be executed:



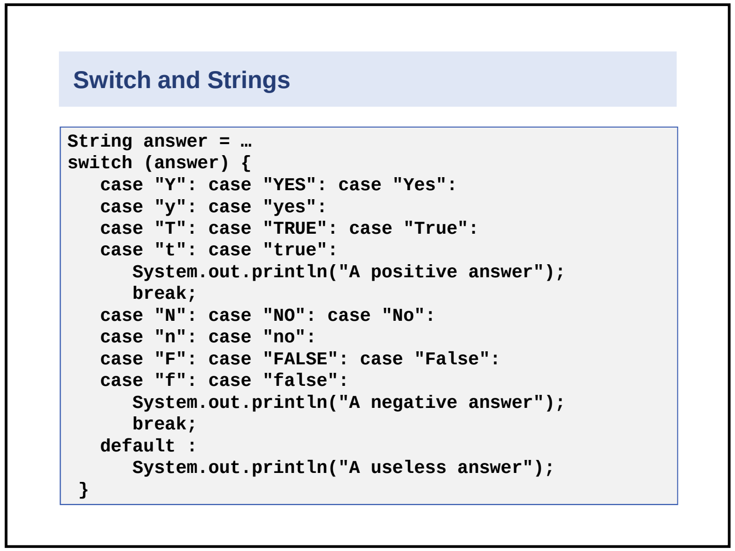
Using multiple case labels

- Multiple **case** labels can be used:

```
month = keyboard.readInt("Which month? ");
switch (month) {
   case 1: case 2: case 11: case 12:
      System.out.println("Low season rate"); break;
   case 3: case 4: case 5: case 10:
      System.out.println("Mid season rate"); break;
   case 6: case 7: case 8: case 9:
      System.out.println("Peak season rate"); break;
}
```

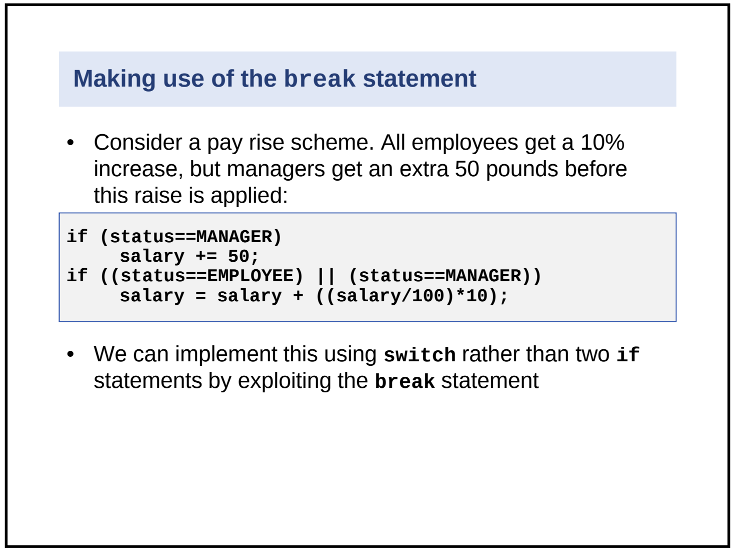
- This is clearer and shorter than an **if-else** statement:

```
if ((month==1)||(month==2)||(month==11)||(month==12))
     System.out.println("Low season rate");
else if((month==3)||(month==4)||(month==5)||(month==10))
      System.out.println("Mid season rate");
else System.out.println("Peak season rate");
```



Switch and Strings

```
String answer = …
switch (answer) {
   case "Y": case "YES": case "Yes":
   case "y": case "yes":
   case "T": case "TRUE": case "True":
   case "t": case "true":
      System.out.println("A positive answer");
      break;
   case "N": case "NO": case "No":
   case "n": case "no":
   case "F": case "FALSE": case "False":
   case "f": case "false":
      System.out.println("A negative answer");
      break;
   default :
      System.out.println("A useless answer");
 }
```

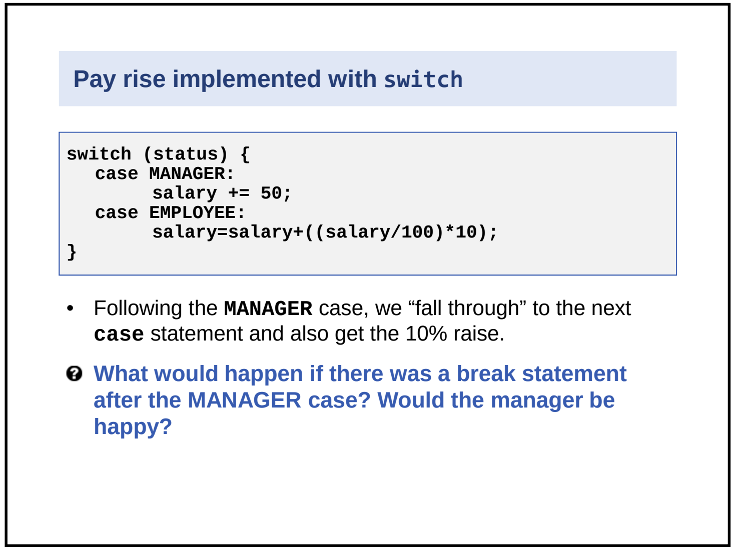


Making use of the break statement

- Consider a pay rise scheme. All employees get a 10% increase, but managers get an extra 50 pounds before this raise is applied:

```
if (status==MANAGER)
     salary += 50;
if ((status==EMPLOYEE) || (status==MANAGER))
     salary = salary + ((salary/100)*10);
```

- We can implement this using **switch** rather than two **if** statements by exploiting the **break** statement

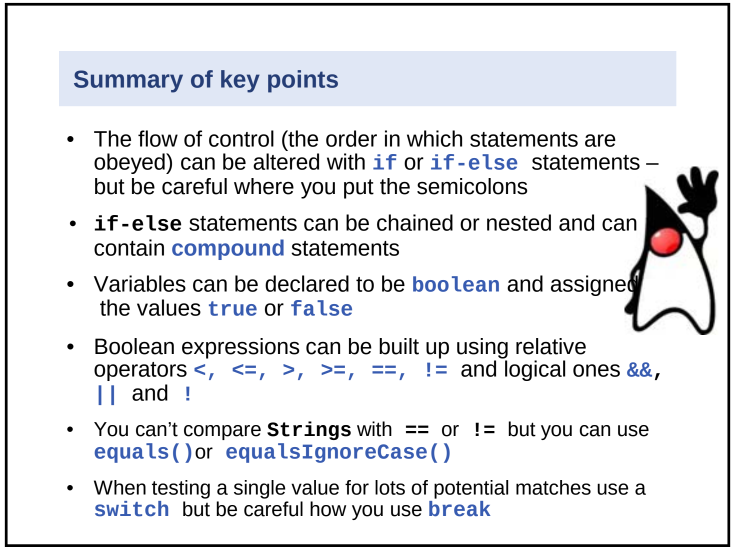


Pay rise implemented with switch

```
switch (status) {
   case MANAGER:
         salary += 50;
   case EMPLOYEE:
         salary=salary+((salary/100)*10);
}
```

- Following the **MANAGER** case, we “fall through” to the next **case** statement and also get the 10% raise.

❓ **What would happen if there was a break statement after the MANAGER case? Would the manager be happy?**



Summary of key points

- The flow of control (the order in which statements are obeyed) can be altered with **if** or **if-else** statements – but be careful where you put the semicolons
- **if-else** statements can be chained or nested and can contain **compound** statements
- Variables can be declared to be **boolean** and assigned the values **true** or **false**
- Boolean expressions can be built up using relative operators **<, <=, >, >=, ==, !=** and logical ones **&&, ||** and **!**
- You can’t compare **Strings** with **==** or **!=** but you can use **equals()**or **equalsIgnoreCase()**
- When testing a single value for lots of potential matches use a **switch** but be careful how you use **break**