Chapter 3

Selections
The if-else Statement
A flow chart is an easy way to graphically express your logic, without having to write any code. You are encouraged to develop your flowcharting skills as a means to express your solution graphically.

As a programmer you are constantly asked to solve problems, and using algorithms is how you present your step by step solution to the problem at hand, that does not require the use of code.

So, an algorithm, being your recipe of steps to follow in order to solve a particular problem can be graphically represented with a flowchart. It is another way to show the same results, one text base and the other graphical based.
In the example demonstrated above, we begin with the “Start” terminator as the beginning of our algorithm.

We then have to make a choice, is the income less than or equal to the no tax amount? This choice is shown with the use of the “Decision” symbol, and as shown, it has two possible outcomes, True or False.

If the income is less than or equal to the no tax amount, we execute the statement(s) on the left, i.e. we assign 0.0 to the variable taxesOwed. If the income is greater, we execute the statement(s) on the right, which assigns the quantity income \* TAX_RATE to the income variable.

As you can see from above, one of two outcomes is possible. We say that the choices are mutually exclusive, only one of the two will ever execute after a decision has been made.

Once a decision has been made and one of the two outcomes is executed, then execution continues with the steps that follow. This is indicated with the conversion of the flow lines into the “Connector” and ultimately the “Stop” terminator.
Java provides the if control structure for making decisions.

The above example shows a decision with two possible outcomes, thus an if-else is used. The statements after the if boolean expression make up the true block, and the statements after the else make up the false block.

Keep in mind that the else part is always optional and its use depends on how many possible outcomes you have. If you have one, then it is not used, if two it is used.
package taxesowed;

import java.util.Scanner;

public class TaxesOwed {
    public static final double NO_TAX_AMOUNT = 10000.0;
    public static final double TAX_RATE = 0.03;

    public static void main(String[] args) {
        double income;
        double taxesOwed;
        Scanner keyboard = new Scanner(System.in);

        System.out.print("Enter your income amount: ");
        income = keyboard.nextDouble();
    }
}
if (income <= NO_TAX_AMOUNT) {
    taxesOwed = 0.0;
} else {
    taxesOwed = income * TAX_RATE;
}// end else
System.out.println("You owe "+ taxesOwed + " on an income of "+ income);
}// end main()
Boolean expressions may use the following relational operators

<table>
<thead>
<tr>
<th>Operation</th>
<th>Operator</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>equality</td>
<td>==</td>
<td>a == b</td>
</tr>
<tr>
<td>inequality</td>
<td>!=</td>
<td>a != b</td>
</tr>
<tr>
<td>greater than</td>
<td>&gt;</td>
<td>a &gt; b</td>
</tr>
<tr>
<td>greater than or equal</td>
<td>&gt;=</td>
<td>a &gt;= b</td>
</tr>
<tr>
<td>less than</td>
<td>&lt;</td>
<td>a &lt; b</td>
</tr>
<tr>
<td>less than or equal</td>
<td>&lt;=</td>
<td>a &lt;= b</td>
</tr>
</tbody>
</table>

The relational operators or comparison operators allow you to perform a number of comparisons whose outcome is a boolean result, i.e. either true or false.

Warning: The equality is expressed with two equal signs, not one, so be careful as the latter will cause an error. For example, if (a = b) is not the same as if (a == b), and will make the compiler very unhappy.

The equality works as expected with integral values, but not so well with floating point and strings. Floating point variables may be simply approximations of the actual numbers, thus comparison may lead to undesirable results. It is best to compare the difference of the two against an acceptable margin of error.

Strings also pose a problem, since strings are actually objects. An object variable is nothing more than a reference (pointer) to the actual value of the object in memory, and thus two separate memory locations could have the exact same string value, but since their memory locations are different, the relational operators will result in an inaccurate outcome. An example of this is shown next.
Logical operators can be used to create compound expressions. The meaning of the operators is similar to what you may have learned while studying boolean algebra. Here is the low down on each one.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Operator</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>And</td>
<td>&amp;&amp;</td>
<td>(a &lt; b) &amp;&amp; (c &gt; d)</td>
</tr>
<tr>
<td>Or</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not</td>
<td>!</td>
<td>!fileIsOpen</td>
</tr>
</tbody>
</table>

&& : True only when both arguments are true, false otherwise.
|| : False only when both arguments are false, true otherwise.
! : The complement of its argument. This one should be avoided in expressions using the relational operators, since it leads to unreadable code. For example, !c < 0) is best expressed as (c >= 0), a more readable form.
package stringequality;
import java.util.Scanner;

public class StringEquality {
    public static void main(String[] args) {
        String str1;
        String str2;
        Scanner keyboard = new Scanner(System.in);

        System.out.print("Enter some text: ");
        str1 = keyboard.nextLine();
        System.out.print("Enter more text: ");
        str2 = keyboard.nextLine();

        System.out.println("str1 = " + str1);
        System.out.println("str2 = " + str2);
    }
}
System.out.println();
System.out.println("str1 = " + str1 + 
"\" : str2 = " + str2 + "\" ");
System.out.println();
System.out.println("str1 == str2 ? " + (str1 == str2));
System.out.println("str1.equals(str2) ? " + str1.equals(str2));
System.out.println("str2.equals(str1) ? " + str2.equals(str1));

Notice how the test str1 == str2 evaluates to false, even though the two string objects have the same value. This is important to understand because it underlies the fact that object variables (str1, str2 in this example) are merely references (pointers) to the actual objects, which both happen to have the same value ("Apples are red"), but are in different memory locations.

For proper string value comparisons, use the method .equals().
Since Java is case sensitive, if you want to compare two strings without accounting for the case, then use the method `str1.equalsIgnoreCase(str2)`, as shown above. This method treats uppercase the same as lowercase.
Application Deconstructed
<StringEquality.java>

```
; Output - StringEquality (run)

run:
Enter some text: Apples are red
Enter more text: Apples are red

str1 = "Apples are red" : str2 = "Apples are red"

str1 == str2 ? false
str1.equals(str2) ? true
str2.equals(str1) ? true

str1 = "Apples are red" : str2 = "apples are red"

str1.equals(str2) ? false
str1.equalsIgnoreCase(str2) ? true
BUILD SUCCESSFUL (total time: 11 seconds)
```
package multibranchif;
import java.util.Scanner;

public class MultiBranchIf {
    public static final int SUN = 1;
    public static final int MON = 2;
    public static final int TUE = 3;
    public static final int WED = 4;
    public static final int THU = 5;
    public static final int FRI = 6;
    public static final int SAT = 7;

    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        int weekDayNumber;
        String weekDayName = "";
    }

Notice the use of constants to aid readability and maintenance. If a change is needed later on to one of these values, then all you have to do is change one constant value. You will not have to search all of your code for each occurrence of the value itself.
System.out.print("Enter the day of the week [1-Sunday...7-Saturday]: ");
weekDayNumber = keyboard.nextInt();

if (weekDayNumber == SUN) {
    weekDayName = "Sunday";
} else if (weekDayNumber == MON) {
    weekDayName = "Monday";
} else if (weekDayNumber == TUE) {
    weekDayName = "Tuesday";
} else if (weekDayNumber == WED) {
    weekDayName = "Wednesday";
} else if (weekDayNumber == THU) {
    weekDayName = "Thursday";
} else if (weekDayNumber == FRI) {
    weekDayName = "Friday";
} else if (weekDayNumber == SAT) {
    weekDayName = "Saturday";
}
// end if

Recall that this if-else if structure is an if-else with nested if's inside the else. For instance,

if (...) {
    ...
    } else if (...) {
    ...
    } else {
    
} can be rewritten as:

if (...) {
    ...
} else {
    if (...) {
        ...
    } else {
        ...
    }
}
System.out.println("I see you chose " + weekDayName + ";");
} // end main()
} // end MultiBranchIf

Enter the day of the week [1-Sunday...7-Saturday]: 3
I see you chose Tuesday.
BUILD SUCCESSFUL (total time: 9 seconds)
The conditional operator is very handy

// The following if-else:
if (minute == 60) {
    minute = 0;
} else {
    minute += 1;
}

// can be written with a conditional operator as:
minute = (minute == 60)? 0 : minute + 1;

The conditional operator is just like an if-else. An expression is given in parentheses followed by a question mark; this constitutes your boolean expression in an if. If the expression evaluates to true then the first part between the ? and the : is executed. This is the equivalent of the true block in an if. Next comes the part between the : and the ;. This is the equivalent of the false block in an if.
Recap

- Use flowcharts to express algorithms.
- Use the `if` for decisions.
- Use `.equals` or `.equalsIgnoreCase` with strings
The Type
Boolean
True or False?
Boolean expressions evaluate to true or false

- **number > 0**
  - True if number > 0, false otherwise

- **3 > 4**
  - False

- **3**
  - True, since not 0

Every boolean expression must evaluate to either true or false.

Zero means false, and any non-zero value means true.
Boolean variables simplify

// Testing for leap year.
if ( (year % 100 != 0) &&
    (year % 4 == 0) ||
    (year % 400 == 0) ) {
    // leap year
}

// Testing for leap year.
boolean isLeap = (year % 100 != 0) &&
                (year % 4 == 0) ||
                (year % 400 == 0);

if (isLeap) {
    // leap year
}

The && has higher priority than ||

The second "if" is a lot easier to read and thus should be preferred over the first example, which is a lot hard to read and debug.

A boolean variable can receive the evaluation of a boolean expression, as in:

boolean isOfAge = (age > 21);

The variable isOfAge will be true if age > 21, false otherwise.
Be mindful of the order of evaluation

- Parentheses go first, then
- Arithmetic, then
- Relational, and then
- Logical
int x = 3 * 5 + (4 - 5) / 2 + 6 % 4;

bool goAhead = 5 > 3 || 3 + 5 < 3 * 2;

bool isRaining = true;
bool haveUmbrella = false;

bool gotWet = isRaining && !haveUmbrella;
Booleans use true or false

boolean isOld = true;

System.out.println("isOld = " + isOld);
// displays isOld = true

System.out.print("Answer true or false. This is fun: ");
boolean isOfAge = keyboard.nextBoolean();

// IO
Answer true or false. This is fun: true
Recap

- Booleans hold either true or false.

- (), arithmetic, relational, logical, in that order.

- Boolean variables simplify code.
The **switch** statement
```
switch (expression) {
    case label1: statements;
        [break];  // break is optional
    case label2: statements;
        [break];
    ...  //sdk 7: label can be a string
    default: statements;
        [break];
}
```
Switch Deconstructed
<DayOfTheWeek>

// ... code omitted

switch (dayOfTheWeek) {
case MON:  case TUE:
case WED:  case THU:
case FRI:  System.out.println("Bah, still a work
day!");
    break;
case SAT:  case SUN:  System.out.println("Woo hoo, weekend!");
    break;
default:    System.out.println("No comprendo!");
    break;
}

Evaluation will begin at the top of the switch, and as soon as one case is selected, its body will execute and the switch will exit.
If none of the cases are selected, then the default case will execute.
If the break is omitted from any case then after its body is executed, control will always continue with the next case that follows.
Brainteaser time!

Draw the flow chart for the DayOfTheWeek switch
package dayoftheweek;

import java.util.Scanner;

public class DayOfTheWeek {
    public static final int SUN = 1;
    public static final int MON = 2;
    public static final int TUE = 3;
    public static final int WED = 4;
    public static final int THU = 5;
    public static final int FRI = 6;
    public static final int SAT = 7;

    public static void main(String[] args) {
        int dayOfTheWeek;
        Scanner keyboard = new Scanner(System.in);
        dayOfTheWeek = keyboard.nextInt();
Application Deconstructed
<DateOfTheWeek.java>

    System.out.print("Enter the day of the week
    [Sun = 1...Sat = 7]: ");
dayOfTheWeek = keyboard.nextInt();

    switch (dayOfTheWeek) {
        case MON:   case TUE:
        case WED:   case THU:
        case FRI:   System.out.println("Bah, still a work
day!");
            break;

        case SAT:
        case SUN:   System.out.println("Woo hoo, weekend!");
            break;

        default:    System.out.println("No comprendo!");
            break;
    } // end switch
    } // end main()
} // end DayOfTheWeek