Conditionals

Contents

Overview

Conditionals in Java (and every other language) allow the developer to check values and make decisions based off the answer of that check. For instance, banks will use a very simple conditional whenever you withdraw money;

Is requested money smaller than or equal to available money?

If yes, dispense requested money and subtract from available money.

If no, reject the request.

If/Else

`if()`-statements are used in Java to build conditionals and when used the block of code inside the `if()`-statement is only run if the condition is met.

```
public class Conditionals {

    public static void main(String[] args) {
        boolean isLightOn = false;

        if(isLightOn) {
            System.out.println("The light is turned on");
        } else {
            System.out.println("The light is turned off");
        }
    }
}
```

In the above example we are declaring and initialising a boolean variable with the name `isLightOn` and the value `false`.

In the main method we are checking if `isLightOn` is `true`, as `if()`-statements will always return either true or false, if `isLightOn` is `true` then we will print `"The light is turned on"`.

Since in our example `isLightOn` is `false`, Java will instead go into the `else`-statement and print `"The light is turned off"`.

Operators

Within `if()`-statements we have more operators that we can use to check which path we should send the program down.

Operation Type	Operator	Function
Equality	`==`	Is equal to
Equality	`!=`	Not equal to
Equality	`<`	Less than
Equality	`>`	Greater than
Equality	`<=`	Less than or equal to
Equality	`>=`	Greater than or equal to
Conditional - Logical	`&&`	AND
Conditional - Logical	`||`	OR
Conditional - Boolean	`&`	Bitwise AND
Conditional - Boolean	`|`	Bitwise OR
Type Comparison	`instanceof`	Compares an object to a specified type

Is Equal to Example

```
public class Conditionals {

    public static void main(String[] args) {
        int number = 10;

        if(number == 10) {
            System.out.println("Number is equal to 10");
        } else {
            System.out.println("Number is not equal to 10");
        }
    }
}
```

In the above example we are declaring and initialising a variable as an `int` with the name `number` and value `10`.

In the main method, we are checking if `number` is equal to `10`, if it is we print `"Number is equal to 10"`.

If `number` is not equal to `10`, then the `else`-statement is run instead, which in this case prints `"Number is not equal to 10"`.

Not Equal to Example

```java
public class Conditionals {

    public static void main(String[] args) {
        int number = 8;

        if(number != 10) {
            System.out.println("Number is not equal to 10");
        } else {
            System.out.println("Number is equal to 10");
        }
    }
}
```

In the above example we are declaring and initialising a variable as an `int` with the name `number` and value `8`.

In the main method, we are checking if `number` is not equal to `10`, if it isn't then we print `"Number is not equal to 10"`.

If `number` is equal to `10` then we instead go into the `else`-statement and `"Number is equal to 10"` is printed.

Less Than Example

```java
public class Conditionals {

    public static void main(String[] args) {
        int number = 8;

        if(number < 10) {
            System.out.println("Number is less than to 10");
        } else {
            System.out.println("Number is greater than 10");
        }
    }
}
```

In the above example we are declaring and initialising a variable as an `int` with the name `number` and value `8`.

In the main method, we are checking if `number` is less than `10`, if it is then we go into the `if()`-statement and print `"Number is less than 10"`.

If `number` has a value of `10` or higher, then we will instead go into the `else`-statement and print `"Number is greater than 10"`.

Greater Than Example

```java
public class Conditionals {

    public static void main(String[] args) {
        int number = 11;

        if(number > 10) {
            System.out.println("Number is greater than to 10");
        } else {
            System.out.println("Number is less than 10");
        }
    }
}
```

In the above example we are declaring and initialising a variable as an `int` with the name `number` and value `11`.

In the main method we are checking if number is greater than `10`, if it is then we go into the `if()`-statement and print `"Number is greater than 10"`.

If `number` has a value of `10` or lower then we will instead go into the `else`-statement and print `"Number is less than 10"`.

Less Than or Equal to

```
public class Conditionals {

    public static void main(String[] args) {
        int number = 10;

        if(number <= 10) {
            System.out.println("Number is less than or equal to 10");
        } else {
            System.out.println("Number is greater than 10");
        }
    }
}
```

In the above example we are declaring and initialising a variable as an `int` with the name `number` and value `10`.

In the main method we are checking if `number` is less than or equal to `10`, if it is we print `"Number is less than or equal to 10"`.

If `number` is `11` or higher, we instead go into the `else`-statement and print `"Number is greater than 10"`.

Greater Than or Equal to

```
public class Conditionals {

    public static void main(String[] args) {
        int number = 10;

        if(number >= 10) {
            System.out.println("Number is greater than or equal to 10");
        } else {
            System.out.println("Number is less than 10");
        }
    }
}
```

In the above example we are declaring and initialising a variable as an `int` with the name `number` and value `10`.

In the main method we are checking if `number` is greater than or equal to `10`, if it is then we print `"Number is greater than or equal to 10"`.

If `number` is `9` or lower then we instead go into the `else`-statement and print `"Number is less than 10"`.

Logical And

```
public class Conditionals {

    public static void main(String[] args) {
        int number = 10;
        int number2 = 20;

        if(number >= 10 && number2 == 20) {
            System.out.println("Number is greater than or equal to 10 and Number
2 is equal to 20");
        } else {
            System.out.println("Number is less than 10, or Number 2 is not equal
to 20");
        }
    }
}
```

In the above example we are declaring and initialising two variables as `int`s, one with the name `number` and value of `10`, and one with the name `number2` and value of `20`.

In the main method we are checking if `number` is greater than or equal to `10`, if it is we then check if `number2` is equal to `20`, and if it is we then print `"Number is greater than or equal to 10 and Number 2 is equal to 20"`.

If either of these conditions fail, then we instead go into the `else`-statement and print `"Number is less than 10, or Number 2 is not equal to 20"`.

You can add to this and have as many logical and statements within the `if()`-statement as you want.
If the first condition in a logical and fails, then Java won't bother checking if the second one passes or fails since the entire condition will fail either way since we need both conditions to be true.

Logical Or

```
public class Conditionals {

    public static void main(String[] args) {
        int number = 9;
        int number2 = 20;

        if(number >= 10 || number2 == 20) {
            System.out.println("Number is greater than or equal to 10 or Number 
2 is equal to 20");
        } else {
            System.out.println("Number is less than 10, and Number 2 is not 
equal to 20");
        }
    }
}
```

In the above example we are declaring and initialising two variables as `int`s, one with the name `number` and value of `9`, and one with the name `number2` and value of `20`.

In the main method we are first checking if `number` is greater than or equal to `10`, if it is we skip the next condition and print `"Number is greater than or equal to 10 or Number 2 is equal to 20"`.

Because `number` is `9` and is therefore not greater than or equal to `10`, we check the second condition which is checking if `number2` is equal to `20`, which it is so we print `"Number is greater than or equal to 10 or Number 2 is equal to 20"`.

If both of these conditions fail then we would instead go into the `else`-statement and print `"Number is less than 10, and Number 2 is not equal to 20"`.

If the first condition in a logical or passes, then Java won't bother checking if the second one passes or fails since the entire condition will pass either way since we only need one of the two conditions to be true.

Bitwise-And

```
public class Conditionals {

    public static void main(String[] args) {
        int number = 10;
        int number2 = 20;

        if(number >= 10 & number2 == 20) {
            System.out.println("Number is greater than or equal to 10 and Number 
2 is equal to 20");
        } else {
            System.out.println("Number is less than 10, or Number 2 is not equal 
to 20");
        }
    }
}
```

In the above example we are declaring and initialising two variables as `int`s, one with the name `number` and value of `10`, and one with the name `number2` and value of `20`.

In the main method we are checking if `number` is greater than or equal to `10`, if it is we then check if `number2` is equal to `20`, and if it is we then print `"Number is greater than or equal to 10 and Number 2 is equal to 20"`.

If either of these conditions fail, then we instead go into the `else`-statement and print `"Number is less than 10, or Number 2 is not equal to 20"`.

You can add to this and have as many bitwise-and statements within the `if()`-statement as you want.
In a bitwise or if the first condition fails, Java will still check the second condition; even though we know the `if()`-statement will return false.

Therefore it is generally considered better practice to use a logical and as it is quicker and thus more efficient.

Bitwise-Or

```
public class Conditionals {

    public static void main(String[] args) {
        int number = 9;
        int number2 = 20;

        if(number >= 10 | number2 == 20) {
            System.out.println("Number is greater than or equal to 10 or Number
2 is equal to 20");
        } else {
            System.out.println("Number is less than 10, and Number 2 is not
equal to 20");
        }
    }
}
```

In the above example we are declaring and initialising two variables as `int`s, one with the name `number` and value of `9`, and one with the name `number2` and value of `20`.

In the main method we are first checking if `number` is greater than or equal to `10`, if it is we skip the next condition and print `"Number is greater than or equal to 10 or Number 2 is equal to 20"`.

Because `number` is `9` and is therefore not greater than or equal to `10`, we check the second condition which is checking if `number2` is equal to `20`, which it is so we print `"Number is greater than or equal to 10 or Number 2 is equal to 20"`.

If both of these conditions fail then we would instead go into the `else`-statement and print `"Number is less than 10, and Number 2 is not equal to 20"`.

If the first condition in a bitwise or returns `true` then Java will still check the second condition even though it doesn't matter what it will return since the `if()`-statement would still be `true`.

Therefore it is generally considered better practice to use a logical or as it is quicker and thus more efficient.

Logical vs Bitwise

If best practice is to use Logical operators, why does Java have Bitwise available?

Logical operators can speed up our program really easily by ignoring one of the conditions, this can sometime cause issues as an element we expect to be executed is instead skipped by Java.

```java
public class Conditionals {

    public static int variable = 0;

    public static void main(String[] args) {
        conditional();
    }

    public static void conditional() {
        if (true && setValue()) {
            System.out.println(variable);
        }
    }

    public static boolean setValue() {
        variable = 10;
        return true;
    }
}
```

We would expect the output of the above code to be 10, But because we have used a Logical AND, Java checks the first value, sees it is `true` and skips evaluating `setValue()`, this means the value of `variable` is still set to 0 and 0 will be printed to console.

```java
public class Conditionals {

    public static int variable = 0;

    public static void main(String[] args) {
        conditional();
    }

    public void conditional() {
        if (true & setValue()) {
            System.out.println(variable);
        }
    }

    public boolean setValue() {
        variable = 10;
        return true;
    }
}
```

In this example you can see we have used the Bitwise AND, this means Java will evaluate all of the statements, calling `setValue()` and allowing variable to be set to 10.

Use Bitwise when you **need** all conditions to be evaluated.

Type Comparison

```java
public class Conditionals {

    public static void main(String[] args) {
        Car car = new Car();

        if(car instanceof Car) {
            System.out.println("This is a Car");
        } else {
            System.out.println("This is not a Car");
        }
    }
}
```

In the above example we are declaring and instantiating a variable of type `Car` with the name `car` and a value of `new Car()`.

In the main method we are checking if the variable `car` is an instance of the class `Car`, if it is then we print `"This is a Car"`.

If it is not an instance of the class `Car` then we instead go into the `else`-statement and print `"This is not a Car"`.

If/Else If/Else

You can add more conditional paths using the `else if()`-statement, making the amount of paths we can send our code down seemingly limitless.

However each statement will need to be evaluated which can slow down the application.

```java
public class Conditionals {

    public static void main(String[] args) {
        int number = 50;

        if(number <= 20) {
            System.out.println("Number is less than or equal to 20");
        } else if(number < 40) {
            System.out.println("Number is between 20 and 40");
        } else {
            System.out.println("Number is greater than or equal 40");
        }
    }
}
```

In the above example we are declaring and initialising a variable of type `int` with the name `number` and value `50`.

In the main method we are then checking if `number` is less than or equal to `20`, if it is we would print `"Number is less than or equal to 20"`.

However in our example `number` has a value of `50`, so Java would go to the next statement, which is another `if()`-statement but because it is an `else if()`-statement it is only evaluated if the first `if()`-statement returns `false`.

When Java runs the else `if()`-statement it checks if `number` is less than `40`, if it is then it will print `"Number is between 20 and 40"`.

We know that if the conditional returns `true` that `number` will be between `20` and `40`, even though we aren't checking if it is greater than `20`, because the first condition returned `false` on whether or not `number` was less than or equal to `20` and the `else if()`-statement is evaluating if `number` is less than `40`.

However once again in our example the else `if()`-statement would return `false` as `number` has a value of `50`, so Java would then go into the `else`-statement and print `"Number is greater than or equal to 40"`.

Switch/Case Statements

As mentioned above if there are a number of conditions to evaluate multiple `if()-else()`-statements may not be efficient.

Therefore if we have a set number of options that are quite extensive then we can use `switch()-case:` statements.

If we have defined conditions or a large number of options then writing `switch()-case:` statement is much easier.

Once the `case` has been found, the block of code associated with it will execute.

```java
public class Conditionals {

    public static void main(String[] args) {
        int day = 5;

        switch(day) {
            case 1:
                System.out.println("Monday you can fall apart");
                break;
            case 2:
                System.out.println("Tuesday,");
                break;
            case 3:
                System.out.println("Wednesday break my heart");
                break;
            case 4:
                System.out.println("Oh, Thursday doesn't even start");
                break;
            case 5:
                System.out.println("It's Friday I'm in love");
                break;
            case 6:
                System.out.println("Saturdaaaay wait");
                break;
            case 7:
                System.out.println("Sunday always comes too late");
                break;
            default:
                System.out.println("You need to listen to The Cure");
                break;
        }
    }
}
```

In the above example we are declaring and initialising a variable of type `int`, with the name `day` and a value of `5`.

In the main method we have then specified that we are running a `switch()-case:` statement and in the parentheses told it which variable we want it to work with.

Each `case` is then evaluated to see which code should be run, so it first checks if `day` is equal to `1`, then if that is `false` it checks if `day` is equal to `2` and so on.

Once Java finds the `case` statement where the condition is `true` it will run the code inside that `case`.
The `break` keyword inside our `case`s will stop Java from evaluating the rest of the `case`s once it has found the `case` that evaluates to `true`.

If none of the `case`s evaluate to `true` then Java will instead run the code inside of the `default case`.

Tutorial

There is no tutorial for this module.

Exercises

Calculator Revisited

1. Edit the division method in the Calculator class, the sum should only execute if the first parameter is smaller than the second, otherwise it prints out a message saying that the division cannot be performed.

Results Revisited

1. Expand the Results class so that there is now a pass mark of 60%; if the person receives under 60% they get a fail message.

2. Expand the previous example so that even if the person gets higher than 60% overall, if they fail 1 or more of the exams (pass grade of 60% for each exam) they still fail overall.

   eg. if a student's scores were: Biology: 150, Chemistry: 150, Physics: 84

   they would have an overall grade of 85.3% (Pass)

   but in this case the student would fail because their physics grade is 54% (Fail)

3. Expand the above so that the message that is displayed varies depending on the number of subjects that they have failed.

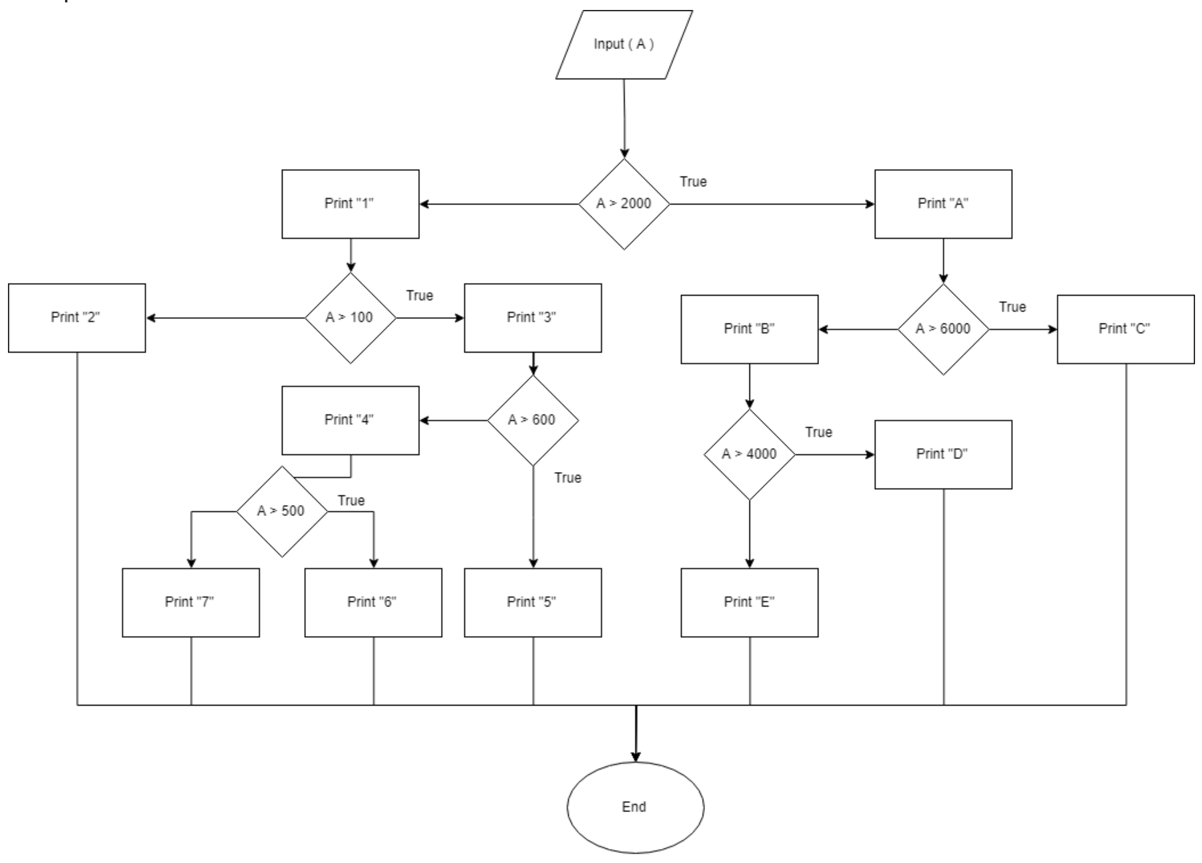
Flowcharts

1. Create a method which accepts 3 parameters, 2 integers and a boolean.

   - If the boolean is `true`, the method will `return` a sum of the two numbers, and it if is `false` it will `return` the multiplication of the two numbers.
     For example:

     ```
     Input (1, 2, true) -> 3
     Input (3, 3, false) -> 9
     Input (1, 1, true) -> 2
     ```

2. Recreate the following flowchart as a project. Ensure that your logic and outputs match that of the flowchart.



3. Create a method that takes a single integer `A` as a parameter.

4. First evaluate if `A` is greater than 2000. If true, print "A", if false, print "1"

5. Next, Directly under print 1 but not outside of the if statement, make another if statement that elaluates if A is greater that 100. If true, print "2", if false, print "3"

6. And so on. Be sure to check you are nesting the new if statements in the correct place.

BlackJack

1. Given 2 integer values greater than `0`, `return` whichever is closest to `21` without going over `21`. If they both go over `21` then `return 0`.

   ```
   play (10, 21) -> 21
   play (20, 18) -> 20
   play (1, 22) -> 1
   play (22, 23) -> 0
   ```

Unique Sum

1. Given 3 integer values, `return` their sum. If one value is the same as another value, they do not count towards the sum. In other words, only return the sum of unique numbers given.

```
Input(1, 2, 3) -> 6
Input(3, 3, 3) -> 0
Input(1, 1, 2) -> 2
```

Taxes

1. Create the tax class, it contains 2 methods.
    - Method 1 - which takes a salary and works out the percentage by which the salary will be taxed.
    - Method 2 - which works out the exact amount that the user will be taxed using a similar process to that in the first method. That amount should be `return`ed and output to the console.
    - *These 2 methods should be able to work independently, so it is fine in this case if some of the code is repeated.**
2. The salaries are taxed as below:
    - 0 - 14,999 : 0% tax
    - 15,000 - 19,999 : 10% tax
    - 20,000 - 29,999 : 15% tax
    - 30,000 - 44,999 : 20% tax
    - 45,000+ : 25% tax
3. Finally combine the 2 methods so that the second method utilizes the first method to `return` the correct result.

FizzBuzz

1. Create a method which returns 'Fizz' for multiples of three and 'Buzz' for the multiples of five.
2. Return 'FizzBuzz' for numbers which are multiples of both three and five.
3. Return the input number for numbers that are neither.