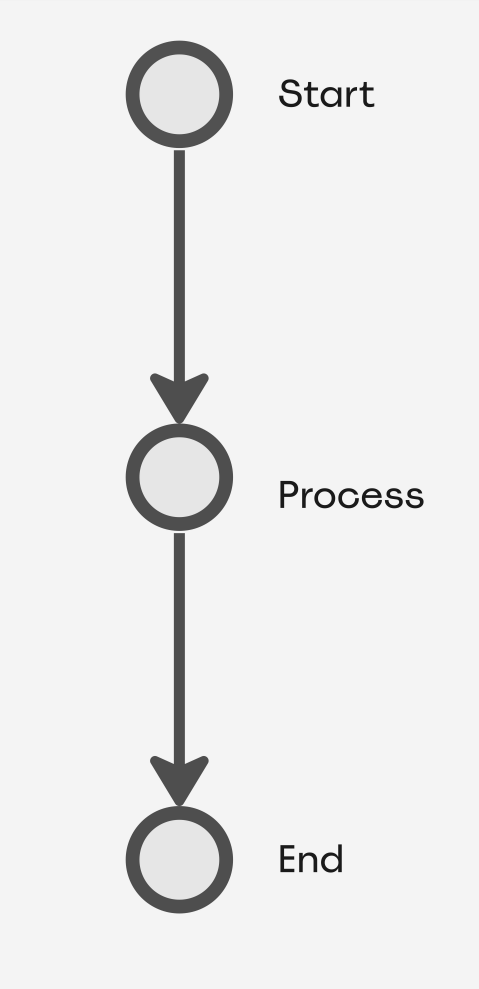
Module 3 - Selection Structures

One great way to think about programing is to think of our code as a linear sequence of events. Remember what I mentioned in previous labs, your computer will execute your program line by line, one after the other from “top to bottom” (although we will see in future modules there are some exceptions to this).



Our code must have a beginning or starting point, it should have some process, and our program must have an exit or end point.

For example, lets say that we are taking Physics I and we want to create a program to calculate acceleration. We know that the formula for acceleration is going to be:

$$Acceleration = \frac{Force}{Mass}$$

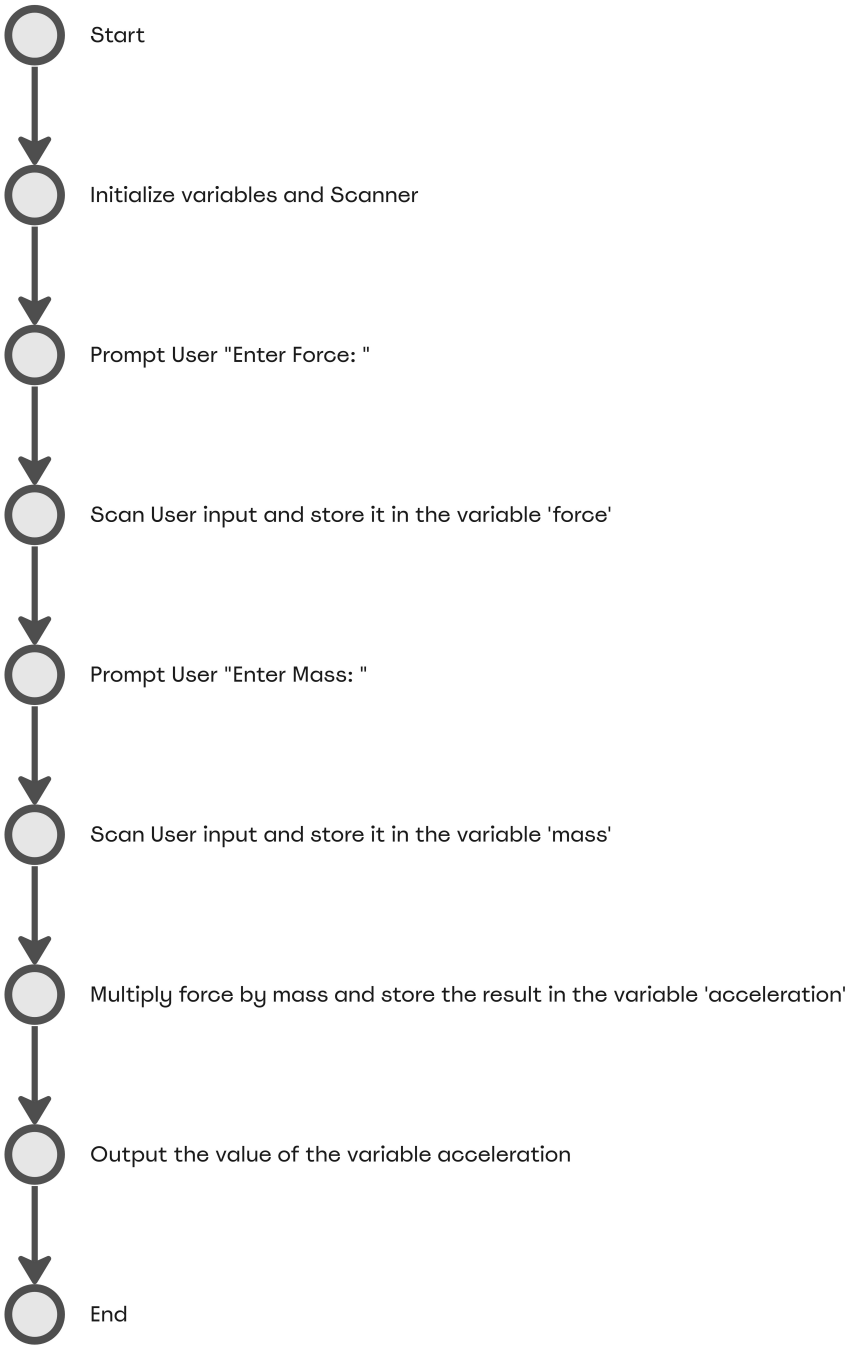
So our program should look something like this:

```
import java.util.Scanner;

public class ExampleA {
    public static void main(String[] args) {
        float acceleration;
        float force;
```

```
        float mass;

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter Force: ");
        force = sc.nextFloat();

        System.out.print("Enter Mass: ");
        mass = sc.nextFloat();

        acceleration = force / mass;

        System.out.println("The Acceleration is: " + acceleration);
    }
}
```

ExampleA.java

The diagram for this program would look like this



There are a few things to notice:

1. It is exactly the same how each task in the diagram and each line of code in our program is ordered.
2. The process from Start to End was a single straight path.

Regarding the second point, this makes our program not so dynamic and very simple. Most programs nowadays need to meet more complex requirements and therefore needs more complexity.

This week we are going to learn and implement a couple of fundamental concepts to make our programs more dynamic, complex, and more “conditional”.

If Statements

If statements have a simple and straight forward premise. IF some condition is TRUE, then we do something.

```
if(age >= 21){
  System.out.println("You can rent a car");
}
```

In this code snippet, we are instructing the computer “if the variable `age` is greater of equal to 21, print “You can rent a car”.

If the variable `age` is greater or equal to 21, the program will print the statement. If not, then nothing happens.

Else If

Now, lets add a bit more complexity to it,

```
if(age >= 21){
  System.out.println("You can rent a car");
}
else if(age >= 18){
  System.out.println("You can vote");
}
```

Now what happens is that if `age` is not greater than or equal to 21, BUT it is greater or equal to 18 (so we can assume `age` must be either 18, 19, or 20), then our program will not print `"You can rent a car"` but instead `"You can vote"`.

Notice that we also have a condition box (parentheses) after the `else if`.

Some more characteristics of the `else if` block:

- `else if` blocks are optional. Its implementation is purely based on the condition logic we need for our program.
- You can have an unlimited amount of `else if` statements.
- You need to have an If statement or `else if` statement above it.
- The else if block will belong to the `if` statement above it.

```
if(...){ // If statement #1
  ...
}
else if(...){ // This else if belongs to the If statement #1 structure.
  ...
}

if(...){ // If statement #2
  ...
}
else if(...){ // This else if belongs to the If statement #2 structure.
  ...
}
else if(...){ // This else if belongs to the If statement #2 structure.
  ...
}
```

Else

Finally, lets add a little bit of complexity to our logic,

```
if(age >= 21){
  System.out.println("You can rent a car");
}
else if(age >= 18){
  System.out.println("You can vote");
}
else{
  System.out.println("You cannot rent a car nor vote");
}
```

So, if `age` is not greater or equal to 21 OR not greater or equal to 18, we can safely assume then that the value of `age` must be from 0 to 17. If the `age` has a value 17 or lower, then we will print `"You cannot rent a car nor vote"`.

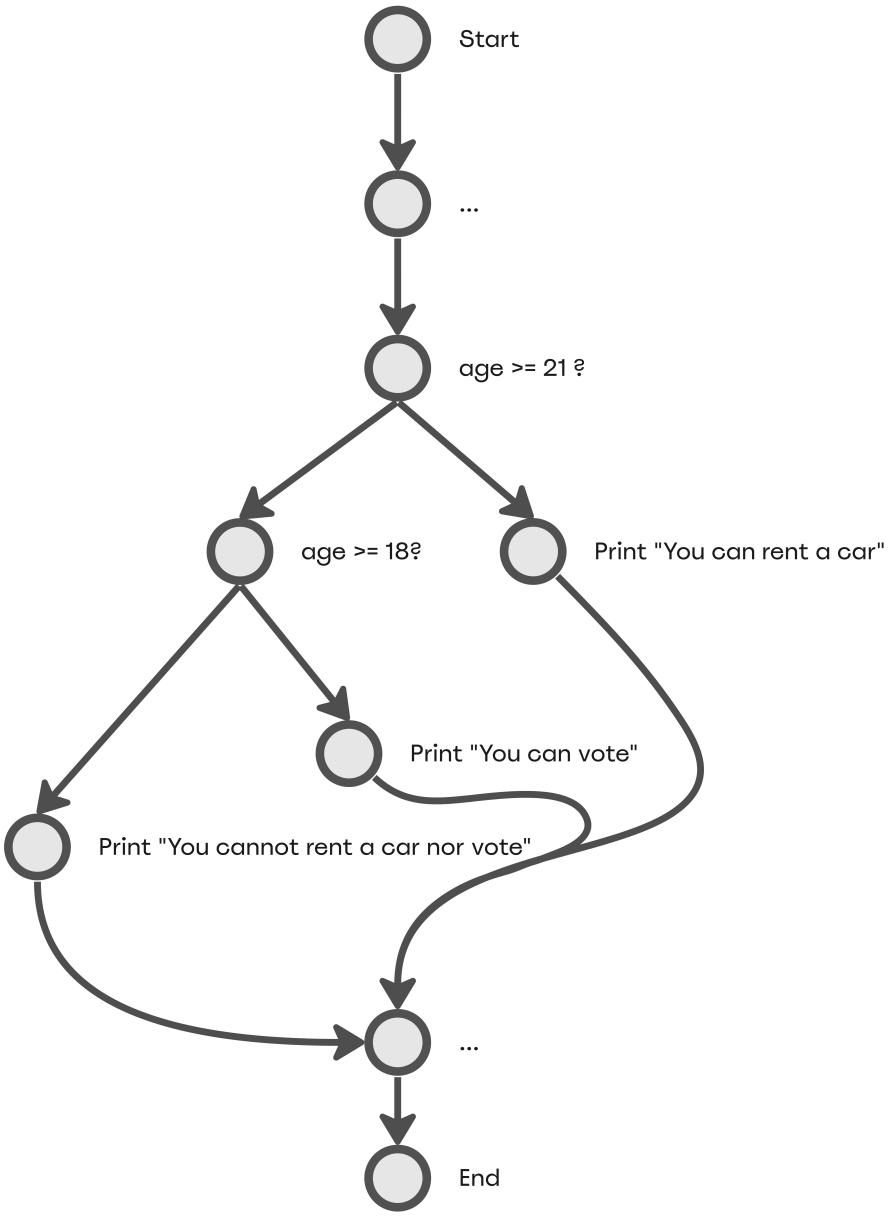
Notice how on the `else` we do not have a condition box (the parentheses). This is because in the logic of our program, we do not have anything special to print, so by default we will print `"You cannot rent a car nor vote"`.

Our program now would look something like this:

```
import java.util.Scanner;

public class ExampleB {
    public static void main(String[] args) {
        int age;

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter your age: ");
        age = sc.nextInt();

        if(age >= 21){
            System.out.println("You can rent a car");
        }
        else if(age >= 18){
            System.out.println("You can vote");
        }
        else{
            System.out.println("You cannot rent a car nor vote");
        }
    }
}
```

ExampleB.java

And the diagram of the flow of your program would look something like this,



Note that now instead of one path from start to end, we have three different paths.

Switch Case Statements

Switch Case Statements are very similar to If statements, and in some circumstances we can use them interchangeably.

The main differing point between switch case statements and if statements is that with a Switch Case, we evaluate the value of a variable and decide from multiple options, while with a if statement, we evaluate a condition for one particular case.

As an example, lets try make our program from *ExampleA.java* more interesting. We are going to first ask the user which value they want to calculate, either Force, Mass, or Acceleration. Given the User's choice, we will prompt and scan the proper values for each calculation and print the output.

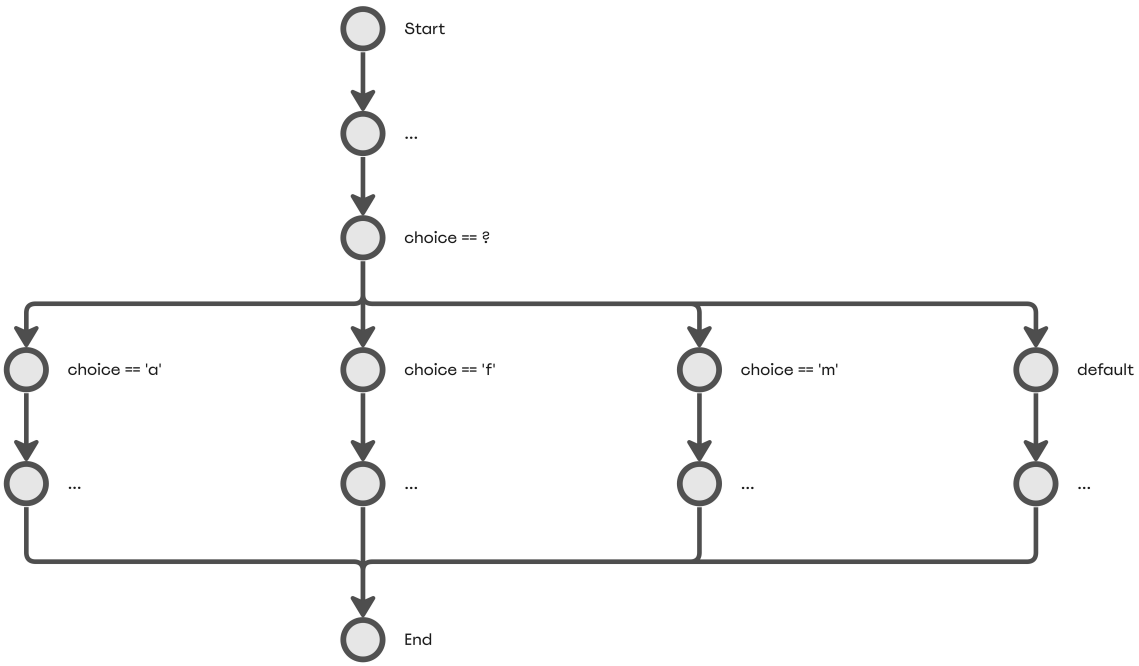
```
import java.util.Scanner;

public class ExampleC {
    public static void main(String[] args) {
        float acceleration;
        float force;
        float mass;
        char choice;

        Scanner sc = new Scanner(System.in);

        System.out.println("Choose an Option:");
        System.out.println("a - Calulate Acceleration");
        System.out.println("f - Calculate Force");
        System.out.println("m - Calculate Mass");
        System.out.println("Option: ");
        choice = sc.nextLine().charAt(0);

        switch(choice){
            case 'a':
                System.out.print("Enter Force: ");
                force = sc.nextFloat();

                System.out.print("Enter Mass: ");
                mass = sc.nextFloat();

                acceleration = force / mass;

                System.out.println("The Acceleration is: " + acceleration);
                break;
            case 'f':
                System.out.print("Enter Mass: ");
                mass = sc.nextFloat();

                System.out.print("Enter Acceleration: ");
                acceleration = sc.nextFloat();

                force = mass * acceleration;

                System.out.println("The Force is: " + force);
```

```
                break;
            case 'm':
                System.out.print("Enter Force: ");
                force = sc.nextFloat();

                System.out.print("Enter Acceleration: ");
                acceleration = sc.nextFloat();

                mass = force / acceleration;

                System.out.println("The Mass is: " + mass);
                break;
            default:
                System.out.println("That is not an option. Bye.");
        }


    }
}
```

ExampleC.java

In this program, we first ask the user for a choice, and based on the value of the choice, we will perform a certain calculation. If the choice is ‘a’, we will do case ‘a’. If the choice is ‘f’, we will do case ‘f’.

The flow diagram of this program would look something like this,



Cases

Inside the switch case statement, we have different cases, each case will depend on the value that we are evaluating. In case of ExampleC.java, we have three choices/cases, perform the calculation of Acceleration, Force, or Mass.

Note that after each case we have a `break;`. This is because after we are done working through our case block, we want the computer to move on the whatever is after the switch case statement. If you do not have a break in a case, it will keep on working on the code you have for the next case below it.

Default

Default cases are optional and they act like the `else` statement in the If statement block. In ExampleC.java, it means that if the user inputs any other character other than a, f, or m, it will print out `"That is not an option. Bye.”`.

Logic Operators

<	Lesser than

>	Greater than
<=	Lesser or equals
>=	Greater or equals
==	Equals
!=	Not equals

In Java, when we are comparing two strings, we cannot use `==`. We must use the `equals()` or `equalsIgnoreCase()` functions.

```
if(stringOne.equals(stringTwo)){
  ...
}

if(stringOne.equalsIgnoreCase(stringTwo)){
  ...
}
```

📢 `equalsIgnoreCase()` will perform a comparison but will disregard capitalization. So, “Hello” will not be equals to “hello” if we use `equals()` but it will be true with `equalsIgnoreCase()`.

Compound Logic

We can implement more complex logic by compounding different evaluations into a single expression.

To do this we have two sets of symbols

&&	AND
||	OR

So, if we want to evaluate the value of a variable to be (3, 5) (exclusive 3 and exclusive 5) we would have the following if statement

```
if(a > 3 && a < 5){
  ...
}
```

Or, if we want to evaluate that if a variable is equal to 3 or another variable is equal to 5

```
if(a == 3 || a == 5){
  ...
}
```