A Very Simple Java Program

```java
public class HelloWorld {
    public static void main(String[] args) {
        int a = 5;
        int b = 10;
        a = a + b;
        System.out.println(a);
    }
}
```

In Java, all code is **inside** some class

**Execution** starts in `main()`

Allocate memory for an `int` variable named `a`, and initialize that memory to the value 5.

1) **Gets** the value stored in memory named `a`.
2) **Gets** the value stored in memory named `b`.
3) **Adds** these two values.
4) **Overwrites** whatever value was in `a` with the newly calculated sum: `a+b`.

Output current value of `a`
Java's Primitive Types

- **byte**: 8-bit, [-128, 127].
- **short**: 16-bit, [-32,768, 32,767].
- **int**: 32-bit, [-2,147,483,648, 2,147,483,647].
- **float**: 32-bit, [1.4x10^{-45}, 3.4028235x10^{38}]
- **double**: 64-bit, [4.9x10^{-324}, 1.7976931348623157x10^{308}]
- **boolean**: Only two possible values: true and false.
- **char**: 16-bit, \[\backslash u0000\] (0), \[\backslash uffff\] (65,535).

Basic Syntax: Part 1

```java
public class HelloWorld
{
    public static void main(String[] args)
    {
        final double PROB_WIN = 18/38;
        final double PROB_LOSE = 20.0/38.0;
        System.out.println("Probabilities: Win=\n" + PROB_WIN + ", Lose=\n" + PROB_LOSE);
    }
}
```

The System class is a package or a library that contains many useful methods and fields.

Method call: `println()`

Statements end with a ;

Block structure: {} with correct indenting
Basic Syntax: Part 2

```java
public class HelloWorld {
    public static void main(String[] args) {
        final double PROB_WIN = 18/38;
        final double PROB_LOSE = 20.0/38.0;

        System.out.println("Probabilities: Win= " +
                           PROB_WIN + ", Lose= " + PROB_LOSE);
    }
}
```

Basic Syntax: Part 3

```java
public class HelloWorld {
    public static void main(String[] args) {
        final double PROB_WIN = 18/38;
        final double PROB_LOSE = 20.0/38.0;

        System.out.println("Probabilities: Win= " +
                           PROB_WIN + ", Lose= " + PROB_LOSE);
    }
}
```
What is the Output?

```java
public class HelloWorld {
    public static void main(String[] args) {
        final double PROB_WIN = 18/38;
        final double PROB_LOSE = 20.0/38.0;

        System.out.println(
            "Probabilities: Win=", PROB_WIN + ", Lose=", PROB_LOSE);
    }
}
```

Probabilities: Win=0.0, Lose=0.5263157894736842

---

Short-Cut Operators: ++, +=

```java
public class HelloWorld {
    public static void main(String[] args) {
        int a = 5;
        int b = 3;
        int c = a+b;
        a++; //same as a=a+1;
        b += 10; //same as b=b+10;
        System.out.println("a="+a);
        System.out.println("b="+b);
        System.out.println("c="+c);
    }
}
```

Output: a=6 b=13 c=8

Notice that c is the sum of what a and b were when the calculation was preformed.
Program State

- The State of a single threaded program is
  - The value of the execution pointer, and
  - The value of all memory accessible by the program at that execution moment.

```java
1. public class HelloWorld
2. { public static void main(String[] args)
3. { int x = 5; int a = 0;
4. if (x < 10) a=1;
5. if (x < 6) a=2;
6. System.out.println(a);
7. }
8. }
```

<table>
<thead>
<tr>
<th>Line</th>
<th>x</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

Output: 2

Control Flow and Program State: if

```java
1. public class HelloWorld
2. { public static void main(String[] args)
3. { int x = 5;
4. int a = 0;
5. if (x < 10) a=1;
6. if (x < 6) a=2;
7. if (x < 1) a=3;
8. System.out.println(a);
9. }
10. }
```

<table>
<thead>
<tr>
<th>Line</th>
<th>x</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

Table of program state at the start of each line in the order of execution.
Control Flow and Program State: if

1. public static void main(String[] args)
2. {
3.     int x = 5;
4.     int a = 0;
5.     if (x < 10)
6.         { a=1;
7.         }
8.     if (x < 6)
9.         { a=2;
10.        }
11.    if (x < 1)
12.    { a=3;
13.    }
14.    System.out.println(a);
15.}

Output: 2

Control Flow: if and else if

1. public static void main(String[] args)
2. {
3.     int x = 5;
4.     int a = 0;
5.     if (x < 10)
6.         { a=1;
7.         }
8.     else if (x < 6)
9.         { a=2;
10.        }
11.    else if (x < 1)
12.    { a=3;
13.    }
14.    System.out.println(a);
15.}

Output: 1
What is the Output?

```java
public class HelloWorld {
    public static void main(String[] args) {
        boolean cool = false;
        if (cool = true) {
            System.out.print("BAD: ["+cool);  
        }  
        if (cool = false)  
            System.out.print("XOX: ["+cool); 
        System.out.println("] -> END: ["+cool +"]");
    }
}
```

BAD: [true] -> END: [false]

¿What is going on?

Integers Divisible by 7 – Explicit Prints

```java
public class IntegersDivisibleBy7v1 {
    public static void main(String[] args) {
        //Print the integers less than 100 that are divisible by 7.
        System.out.println("7");
        System.out.println("14");
        System.out.println("21");
        System.out.println("28");
        System.out.println("35");
        ...
    }
}
```
Properties of Explicit Print Solution

- The code is long and repetitive.
- The programmer must know all parts of the solution.
- Difficult to modify the functionality.
  - For example, to change the program from printing all integers less than 100 that are divisible by 7, to all integers less than 100 divisible by 11, the programmer would need to change almost every line of the code.
- Not a scalable solution.

Integers Divisible by 7 – for loop & if

```java
public class IntegersDivisibleBy7v2 {
    public static void main(String[] args) {
        for (int i=1; i<100; i++) {
            if (i % 7 == 0) {
                System.out.println(i);
            }
        }
    }
}
```

Create the local field i and it initialize to a value of 1. If i is less than 100, execute the for loop body.

When end of loop body is reached, add 1 to i. Then, return to the top of the loop block.
```java
public class IntegersDivisibleBy7v2 {
    public static void main(String[] args) {
        for (int i = 1; i < 100; i++) {
            if (i % 7 == 0) {
                System.out.println(i);
            }
        }
    }
}
```

This block is executed if the logical expression evaluates to true.

1. The code is short.
2. The programmer need not know all parts of the solution.
3. Easy to modify.
4. Scalable.
5. However, i walks through many unneeded values (values that are not divisible by 7).
Integers Divisible by 7 – Leverage Pattern

```java
public class IntegersDivisibleBy7v3 {
    public static void main(String[] args) {
        for (int i=7; i<100; i=i+7) {
            System.out.println(i);
        }
    }
}
```

1. The code is even shorter.
2. The programmer need not know all parts of the solution.
3. Easy to modify.
4. Scalable.
5. Leverages a pattern to skip values of i that are not needed.

Compile and Run!