3.5: Boolean Expressions (&& , || , !)
$\uparrow$ and $\uparrow$ or $\uparrow$ not

. And operator :

ex:
```
public static void main (String[] args) {

        boolean isVehicle = true;
        int year= 2005;
        if (isVehicle && year > 2000) {
                System.out.println("This car is not super old);
        }
        else {
                System.out.println ("This is not a vehicle or it is super old);
        }
}
```
&& operator makes sure to execute the "if statement" if both elements are true. current execution will return true.

← can also be written as if ((isVehicle == true) && (year > 2000))

ex 2:
```
public static void main (String[] args) {
        int a= 5;
        int b= 6;
        int c: 7;

        if ((c>a) && (c>b) && (c%2 != 0) {
                System.out.println( c + "is the highest number." + c + " is also odd");
        }
        else if ((c>a) && (c>b) && (c%2== 0) {
                System.out.println(c + " is the highest number." + c + " is also even.");

        }
        else
                System.out.println(c + " is not the highest number.");
}
```
← using && multiple times here.

The above example outputs "7 is the highest number. 7 is also odd".
It goes through the first "if" since everything is true (true && true && true) which equates to "true".

: Or operator.

ex 3:
```
public static void main (String[] args) {
        int a= 5;
        int b= 6;
        int c: 6;

        if ((c >b) ||(c > a)
                System.out.println( c + "is either higher than " a "or" b);
        }
        else if ((c>a) && (c>b)) {
                System.out.println(c + " is the highest number.");
        }
        else {
                System.out.println (c + " is the lowest number");
        }
}
```
What will be the output if
(i) a=5, b=6, c=6
(ii) a=7, b=6, c=5
(iii) a=7, b=8, c=9

- not operator

ex4
```
public static void main (String[] args) {
        int a= 5;
        int b= 6;

        if (!(a>b) {
            System.out.println( a + " is not the highest number");
        }

}
```

- Truth tables: Truth table is a great way to verify your logic with trial + error.

ex. checking logic of if (a>b) && (b>c) .

Sample Value	a>b	b>c	(a>b) && (b>c)
a: 2, b: 3, c: 4	false	false	false
a: 3, b=2, c: 4	true	false	false
a=2, b=3, c=1	false	true	false
a=5, b=4, c=3	true	true	True

that is what we wanted.

We walk through the code with our sample values and verify if the output matches the truth table.

Note: we will be using alias in the future to make things simpler.

ex: P is an alias for a>b
Q is an alias for b>c

refering using a symbol instead of $\boxed{a>b}$

P && Q is True when P is true and Q is true
(a>b) && (b>c) (a>b) (b>c)

- Short circuit evaluation.

ex:
```
if ((a>b) && (a==0)){
    print ("Hello");
}
```

← when a=5, b=4, it checks both statements.
but
when a=4, b=5, it checks only the first statement.

It doesn't matter if (a==0) since a is less than b. To go to inside the if statement, both statements need to be true. If the first one is false, it will skip the if and go to the next line after if block.

ex:
```
if ((a>b) || (a==0)){
        print("World");
}
```
← if a=4 and b=3, it will just go to the if block after the first statement (a>b).

Since a>b and it is or, it doest matter if a==0 since regardless of that being True or False, it will execute the if block.

However, when a=3, b=4, it will go and check if a==0. Sine first is false, it checks if (a==0) sine it will execute if any one statement was true.

3.6: Equivalent Boolean Expressions (DeMorgan's Laws)

Law: not (a and b) => (not a) or (not b)

The law states that not (a and b) is equivalent to (not a) or (not b).

Example.

Consider: x=4, y=3, z=4

a = (x>y) ← in this case, true

b = (x>z) ← in this case, false

not (a and b)	(not a) or (not b)
not (True and False)	(not true) or (not false)
not (False)	(false) or (True)
$\boxed{\text{True}}$	$\boxed{\text{True}}$

This example verifies the first DeMorgan's Law.
You can make a truth table to verify the Law with various combinations

a	b	a and b	not (a and b)
T	T	T	F
T	F	F	T
F	T	F	T
F	F	F	T

a	b	not a	not b	(not a) or (not b)
T	T	F	F	F
T	F	F	T	T
F	T	T	F	T
F	F	T	T	T

Law: Similarly not (a or b) => (not a) and (not b)

The law also states not (a or b) is equivalent to (not a) and (not b)

Note: a and b can be any logic that returns boolean values like True/False.
a can be output of (5>6) or (6==6) etc.

Q: Make a truth table to compare if not (a or b) is equivalent to (not a) and (not b).

- !(c==d) is equivalent to c!=d.
- !(c!=d) is equivalent to c==d
- !(c<d) is equivalent to c>=d
- !(c>d) is equivalent to c<=d.
- !(c<=d) is equivalent to c>d.
- !(c>=d) is equivalent to c<d.

3.7 Comparing Objects

- We visited this earlier. We treat object differently than primitive values like boolean, int, double.
- String is an object so we use "compareTo" and "equals" method to compare string values instead of "==".
- We can use "equals" operator to check if two objects are equal.

Note: "==" will return true if two variables refer to the same object. These variables are called object references or aliases for the same object.

ex:
```
public static void main(String[] args) {
        String s1 = new String("Hello");   } creating a new
        String s2 = new String("Hello");   } String objects

false ↗  System.out.println(s1 == s2);   ← checking if both objects are equal
true  ↘  System.out.println(s1.equals(s2));   ← checking if the contents of the object are equal.

}
```

ex:
```
public static void main(String[] args) {
        String s1 = new String("Hello");   ← creating a new object.
        String s2 = s1;   ← creating an alias for s1. s2 is pointing to s1 object.

        System.out.println(s1 == s2);   ← prints true
        System.out.println(s1.equals(s2));   ← prints true

}
```

- Comparing with null:

ex:
```
public static void main(String[] args) {
        String str = null;
        if ((str != null) && (str.indexOf("a") >= 0)) {
                System.out.println( s + " contains an a") ;
        }
}
```

That is the only place you should use "==" or "!=" instead of equals. It checks if the string/object really exists.

Note: if we don't add the null check and if the string is null, execution of this code will result in NullPointerException.