```
irror_mod = modifier_ob.
 mirror object to mirror
irror_mod.mirror_object
peration == "MIRROR_X":
irror_mod.use_x = True
irror_mod.use_y = False
irror_mod.use_z = False
 operation == "MIRROR_Y":
irror_mod.use_x = False
irror_mod.use_y = True
irror_mod.use_z = False
 operation == "MIRROR_Z":
irror_mod.use_x = False
irror_mod.use_y = False
```

Topic 6

Strings and Conditions

What is a String?

String

A `String` is a group of characters. Lets look at the following example:

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

    String game = "Super Mario World";

}
```

```java
public static void main(String[] args) {
    String game = "Super Mario World";
}
```

DATA	S	u	p	e	r		M	a	r	i	o		W	o	r	l	d	\0
INDEX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Keep in mind two things:

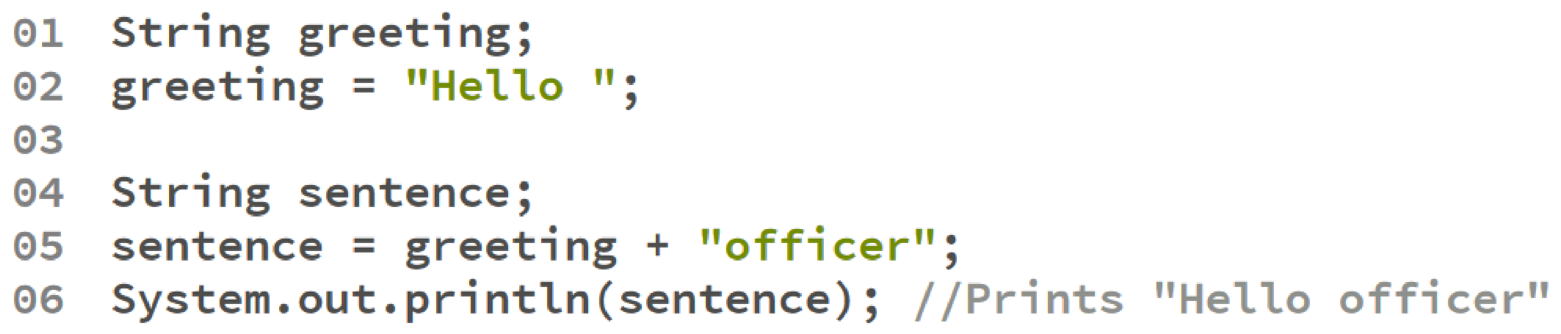
- We start counting positions at 0. Character `‘S’` in stored at index 0.
- To indicate that a String ends, Java stores a NULL character (`\0`) at the end of the character chain.

```java
String g1;
g1 = "Hello!";

String g2 = "Hello!";

String g3 = new String("Hello!");
```

String concatenation

Two strings can be concatenated using the addition operator **(+)**.

We can concatenate different data types together in this way (Strings, ints, doubles, chars).



```
String greeting;
greeting = "Hello ";

String sentence;
sentence = greeting + "officer";
System.out.println(sentence); //Prints "Hello officer"
```

Iniatilize String

We can initialize an empty String using the following statement.

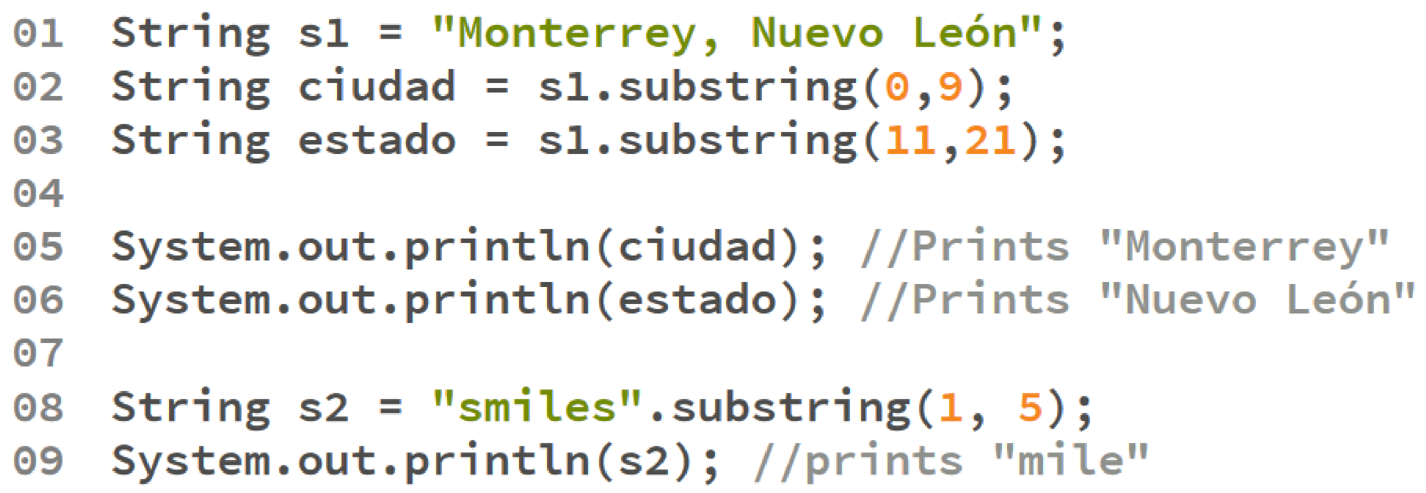
```
String s1 = "";
```

substring() method

The `substring(int beginIndex, int endIndex)` method returns a smaller set of a given original String.

The substring will begin in the given specified index (`beginIndex`) and will extend until (`endIndex – 1`).

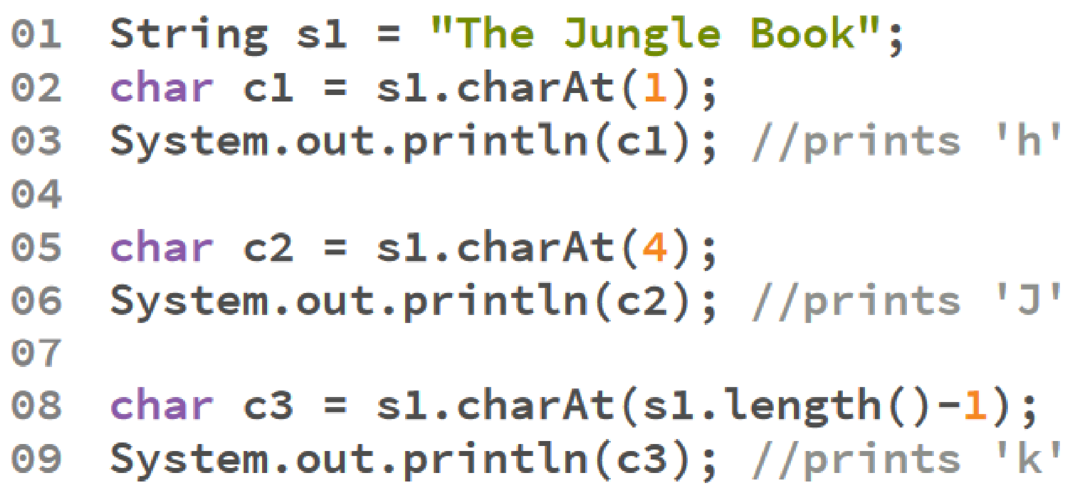
The size of the resulting string will be `endIndex - beginIndex`



```java
String s1 = "Monterrey, Nuevo León";
String ciudad = s1.substring(0,9);
String estado = s1.substring(11,21);

System.out.println(ciudad); //Prints "Monterrey"
System.out.println(estado); //Prints "Nuevo León"

String s2 = "smiles".substring(1, 5);
System.out.println(s2); //prints "mile"
```

charAt() method

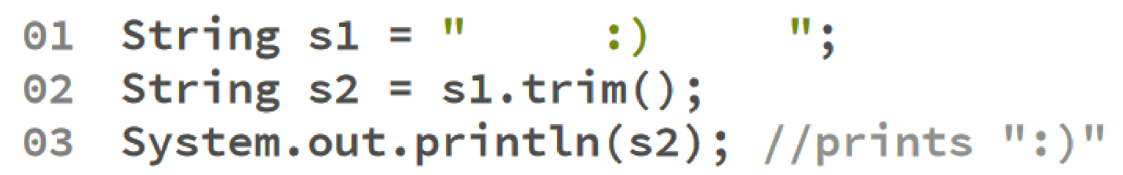
The charAt(int index) method allows you to retrieve the character at the given specified position.



```java
String s1 = "The Jungle Book";
char c1 = s1.charAt(1);
System.out.println(c1); //prints 'h'

char c2 = s1.charAt(4);
System.out.println(c2); //prints 'J'

char c3 = s1.charAt(s1.length()-1);
System.out.println(c3); //prints 'k'
```

trim() method

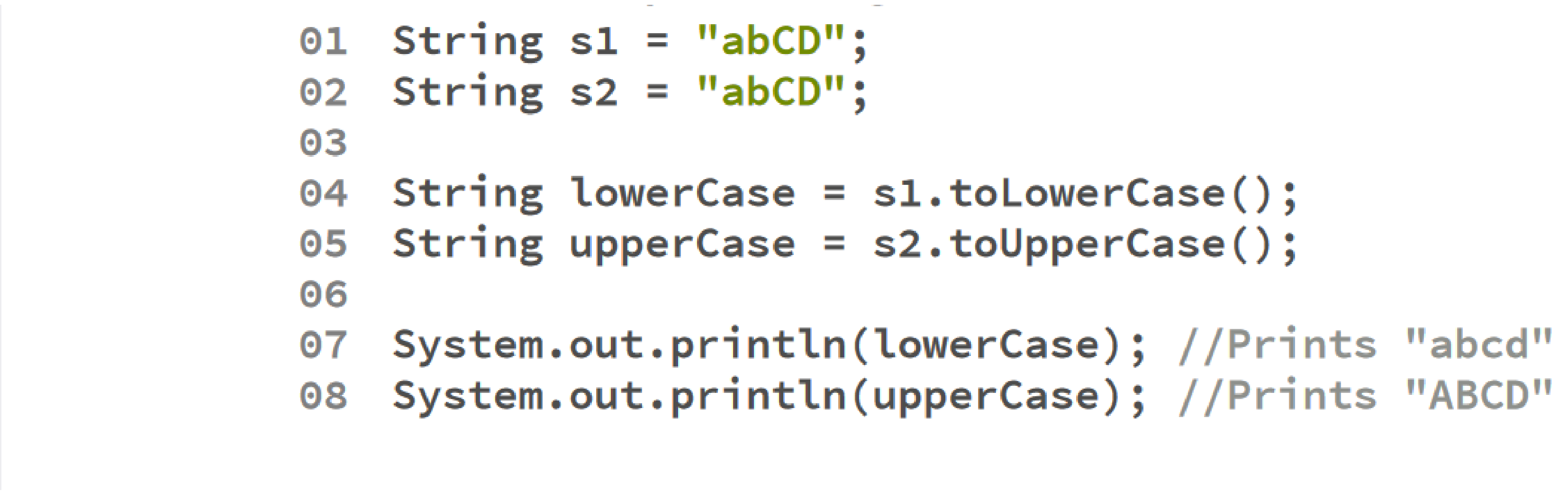
The trim() method creates a new string without the empty blank spaces at the beginning and the end of the String.



```
String s1 = "     :)     ";
String s2 = s1.trim();
System.out.println(s2); //prints ":)"
```

toLowerCase() and toUpperCase() method

Methods toLowerCase() and toUpperCase() allow us to convert a given String into upper or lowercase respectively. This methods create new Strings, so we usually assign their result to a new String variable.



```java
String s1 = "abCD";
String s2 = "abCD";

String lowerCase = s1.toLowerCase();
String upperCase = s2.toUpperCase();

System.out.println(lowerCase); //Prints "abcd"
System.out.println(upperCase); //Prints "ABCD"
```

replace() method

The replace() method allows you to replace a character inside of a given text for another.

```java
String s1 = "Bienvenido a la ciudad!";
String s2 = s1.replace('e','x');
System.out.println(s2); //Prints "Bixnvxnido a la ciudad!"

s2 = s2.replace('a','x');
System.out.println(s2); //Prints "Bixnvxnido x lx ciudxd!"
```

Conditions

IF

The `if` statement is a flow control instruction that allows some statements to be executed selectively.

```
int numerator = 10;
int denominator = 2;
int result;

if (denominator != 0){
  result = numerator / denominator;
}
```

When the boolean expression inside of the if statement evaluates true, then the instructions inside of the block `{ }` will execute.

IF-ELSE

The `if` block can be paired with an `else`, which will only be executed when the boolean expression evaluates as false.

```
int numerator = 5;
int denominator = 0;
int result;

if (denominator != 0){
  result = numerator / denominator;
} else {
  System.out.println("Error, you cannot divide by zero");
}
```

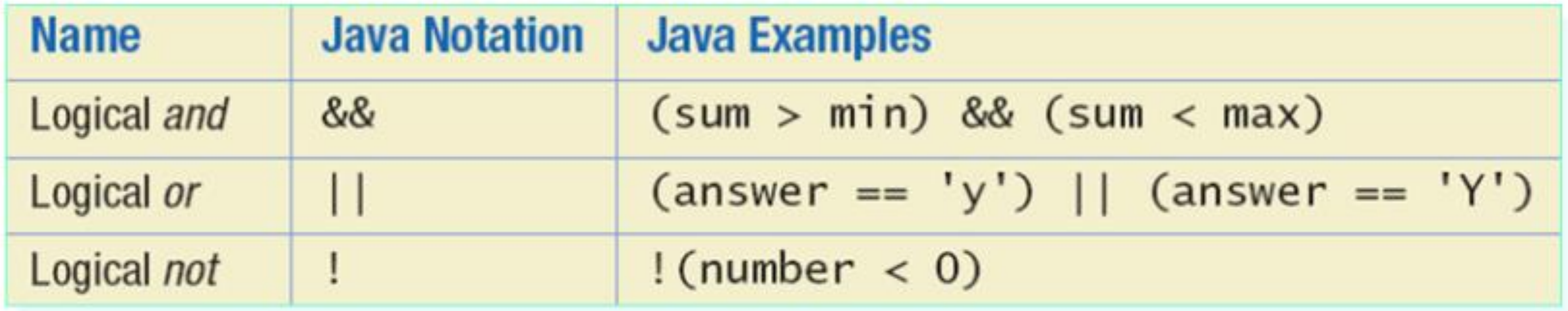
Boolean Expression

Inside of an `if` statement, we put a boolean expression. This can be a simple boolean expression:

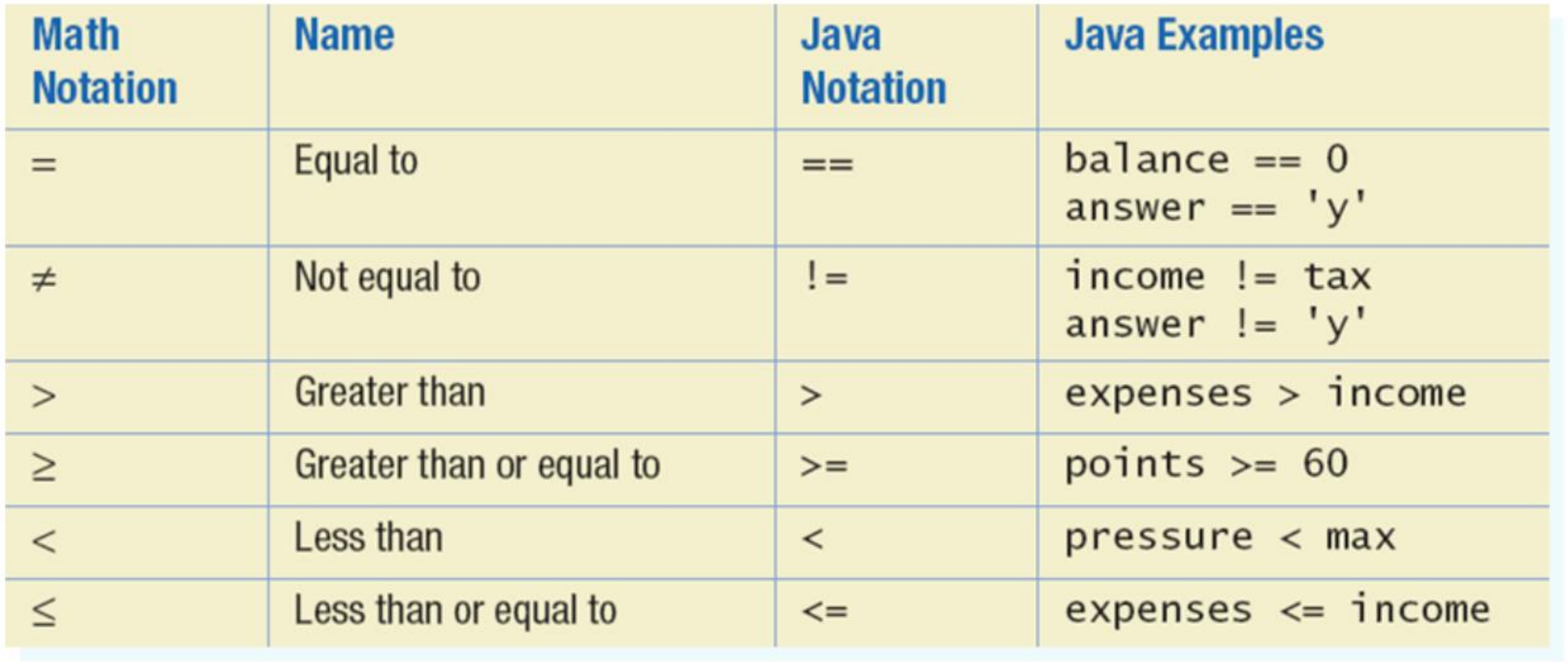
```
if (minutes > 60){
    hour++;
}

if (letter == 'c') {
    System.out.println("You selected option c");
}
```

Or complex boolean expressions, which are made up of more than one condition

```
if ((hour  >= 8) && (hour <= 22)){
    System.out.println("The store is open!");
}

if (letter == 'a' || letter == 'e' || letter == 'i' || letter == 'o' || letter == 'u' ){
    System.out.println("Letter is a vocal!");
}
```



Name	Java Notation	Java Examples
Logical *and*	`&&`	`(sum > min) && (sum < max)`
Logical *or*	`||`	`(answer == 'y') || (answer == 'Y')`
Logical *not*	`!`	`!(number < 0)`



Math Notation	Name	Java Notation	Java Examples

=	Equal to	`==`	`balance == 0`
			`answer == 'y'`

≠	Not equal to	`!=`	`income != tax`
			`answer != 'y'`

>	Greater than	`>`	`expenses > income`
≥	Greater than or equal to	`>=`	`points >= 60`
<	Less than	`<`	`pressure < max`
≤	Less than or equal to	`<=`	`expenses <= income`

Value of *A*	**Value of *B***	**Value of *A* && *B***	**Value of *A* || *B***	**Value of ! (*A*)**
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Comparing primitive variables

To compare two primitive variables, we can use the equality operator (==).

For example:

```
if (a == 3) {
  System.out.println("a equals 3");
}
```

Comparing Strings

Strings, on the other hand, are not primitive variables and can only be compared using the equals() method.

For example:

```
String s1 = "hello";
String s2 = "goodbye";

if (s1.equals(s2)){
  System.out.println("The strings are equal");
} else{
  System.out.println("The strings are different");
}
```

Comparing Strings

Imagine the following Strings s1 and s2. Can you think of a way to test if they are equal without considering lower and upper cases?

```
String s1 = "hello";
String s2 = "HELLO";
```

Option 1

We can compare two Strings using the equalsIgnoreCase():

```
String s1 = "hello";
String s2 = "HELLO";

if (s1.equalsIgnoreCase(s2)){
  System.out.println("The strings are equal");
} else{
  System.out.println("The strings are different");
}
```

Option 2

Convert both strings to upper case / lower case, then compare them using the equals() method:

```
String s1 = "hello";
String s2 = "HELLO";

s1 = s1.toLowerCase();
s2 = s2.toLowerCase();

if (s1.equals(s2)){
  System.out.println("The strings are equal");
} else{
  System.out.println("The strings are different");
}
```

Excercise!

Monkey Trouble

There are two monkeys in a cage. We are in trouble if:

- Both monkeys are smiling
- None of the monkeys are smiling

Design a Java program to model this interaction

```java
import java.util.Scanner;

public class MonkeyTrouble {
    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);

        System.out.print("Is the first monkey smiling? (true/false): ");
        boolean monkey1 = teclado.nextBoolean();

        System.out.print("Is the second monkey smiling? (true/false): ");
        boolean monkey2 = teclado.nextBoolean();

        if ((monkey1 == true && monkey2 == true) || (monkey1 == false && monkey2 == false)) {
            System.out.println("Look out! The monkeys are planning something!");
        }
        else {
            System.out.println("Don't worry, everything is OK");
        }
        teclado.close();
    }
}
```