Decisions decisions decisions

#3

CS 0007
Introduction to
Computer Programming

Luís Oliveira

Summer 2020

Logic

Or is it?

Booleans again

- Boolean represent truthiness of statements

```
boolean condition = false;
condition = true;
```

- Booleans can store the result of comparisons

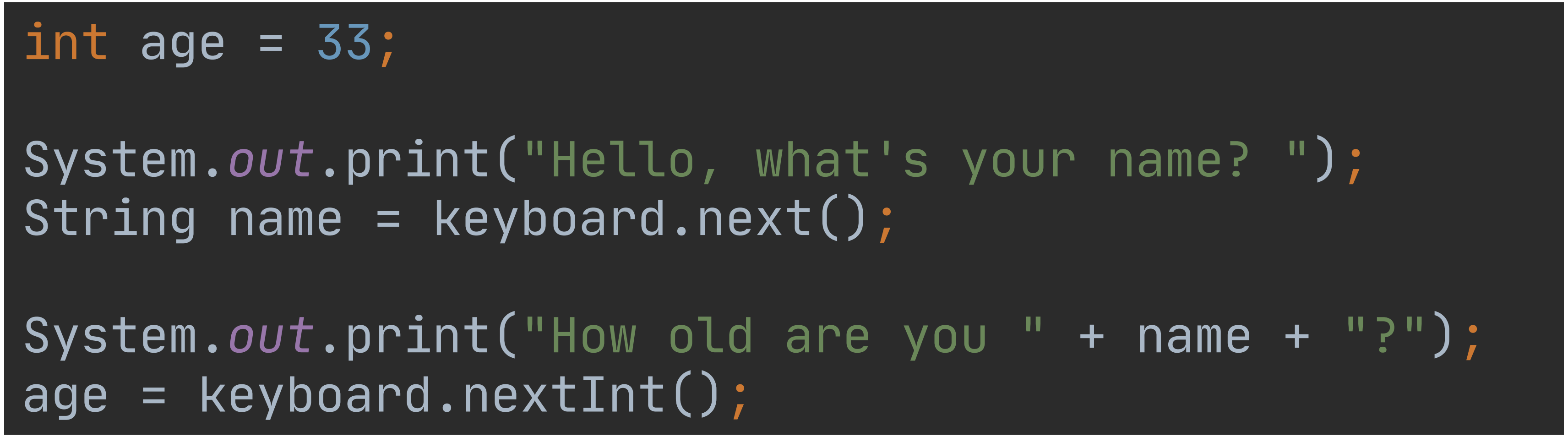
```
int x=-2, y=10;
boolean isXgreaterThanY = x>y;    // false
boolean isXNegative = x<0;        // true
```

Decisions

Beyond the simple calculator

So far....

- Code runs sequentially



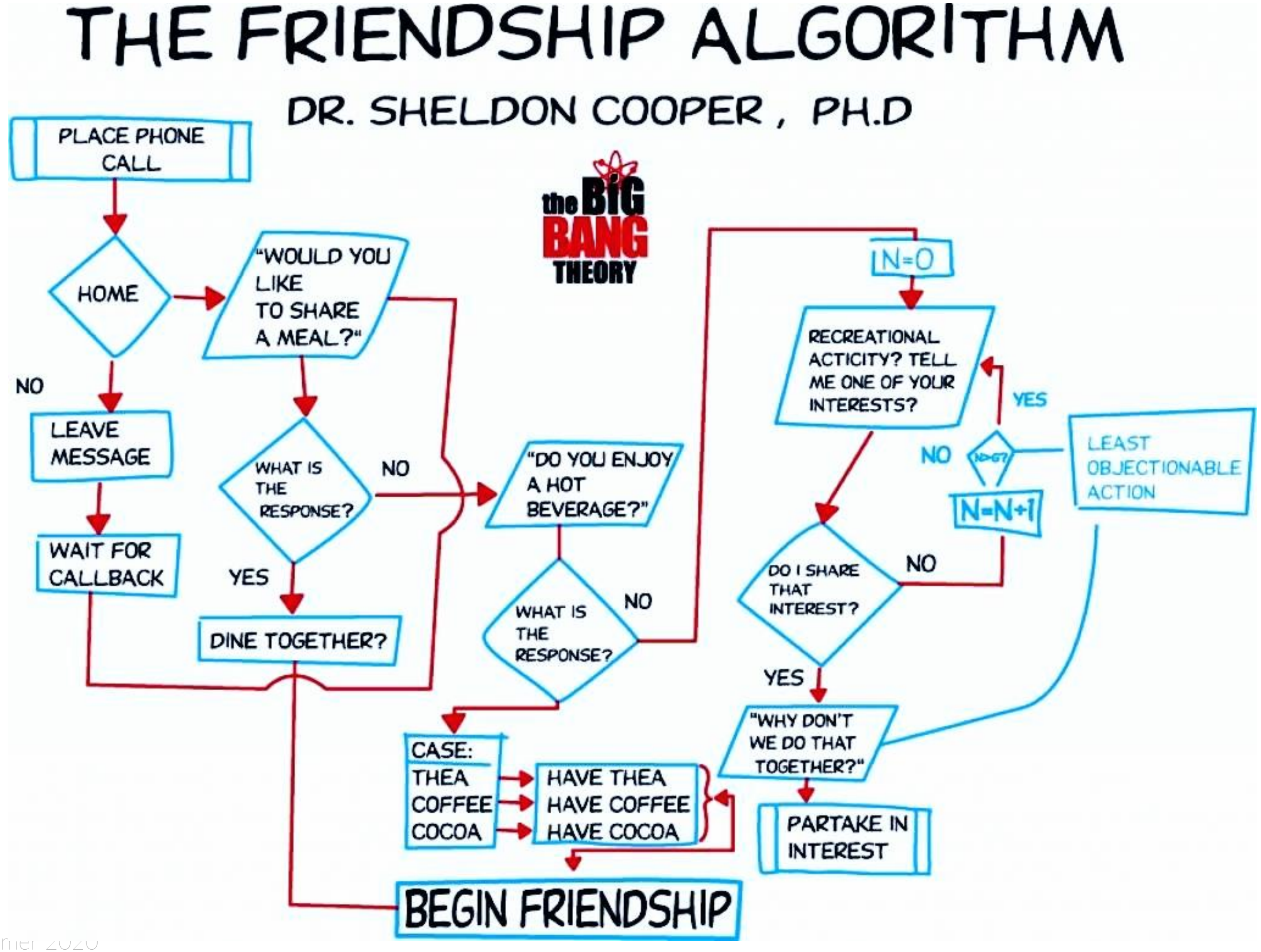
```java
int age = 33;

System.out.print("Hello, what's your name? ");
String name = keyboard.next();

System.out.print("How old are you " + name + "?");
age = keyboard.nextInt();
```

- This only takes us so far ☹

Algorithms with choices



Making decisions

- If statements
  - A structure that allows us to make decisions!

```
boolean isHome = true;
if(isHome)
{
    System.out.println(“Share a meal?");
}
System.out.println("This always runs!");
```

Making different decisions

- Else allows us to do something *else (ah!)* when the condition is false

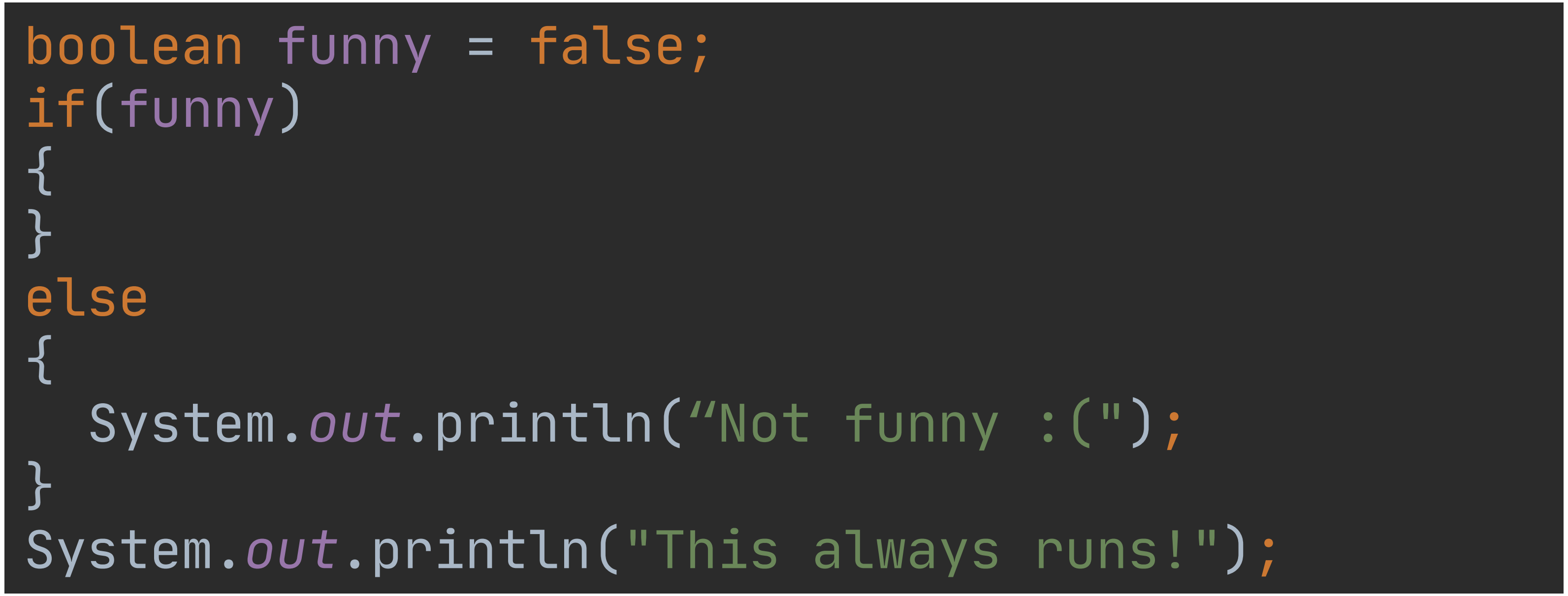
```
boolean isHome = true;
if(isHome)
{
    System.out.println("Share a meal?");
}
else
{
    System.out.println(“Leave a message!");
}
System.out.println("This always runs!");
```

Multiple choices

```java
boolean enjoyHotBeverage = true;
boolean enjoyActivities = true;
if( enjoyHotBeverage ) {
    System.out.println(“Which beverage?");
    System.out.println(“Don’t know if likes activities!");
}
else if( enjoyHotBeverage ) {
    System.out.println(“Doesn’t like hot beverages!");
    System.out.println(“But likes activities!");
}
else {
    System.out.println(“No beverages, no activities!");
}
System.out.println("This always runs!");
```

Don’t use the else without the if

- Don’t do this!



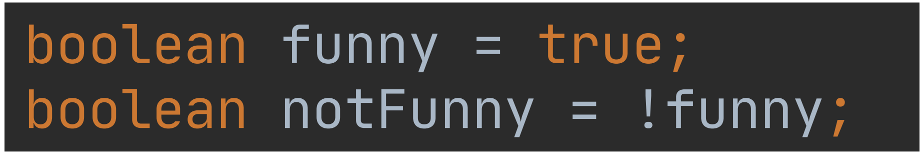
```
boolean funny = false;
if(funny)
{
}
else
{
  System.out.println(“Not funny :(");
}
System.out.println("This always runs!");
```

This is funny…. NOT!

- If you need to negate a condition, you have the NOT operator

funny	NOT funny
Yes	No
No	Yes

funny	!funny
true	false
false	true



```
boolean funny = true;
boolean notFunny = !funny;
```

Negate the condition

- If you negate the condition, you can remove the empty if statement

```
boolean funny = false;
if(!funny)
{
  System.out.println(“Not funny :(");
}
System.out.println("This always runs!");
```

Advanced conditions

Ready OR Set AND Go!

AND and OR – Going to the beach

Can I go with my car?

Car	Fuel	Going to t ch
No	No	
No		
		No
Ye	es	Yes

To go to the beach I need both: Car AND Fuel

A	B	Result
False	False	False
False	True	False
True	False	False
True	True	True

Can I go using public transportation?

Bus	Train	Going to t ch
No	No	
No		
		Yes
Ye	es	Yes

To go to the beach I need either: Bus OR Train

A	B	Result
False	False	False
False	True	True
True	False	True
True	True	True

Going to the beach with Java

- Can I go with my car?

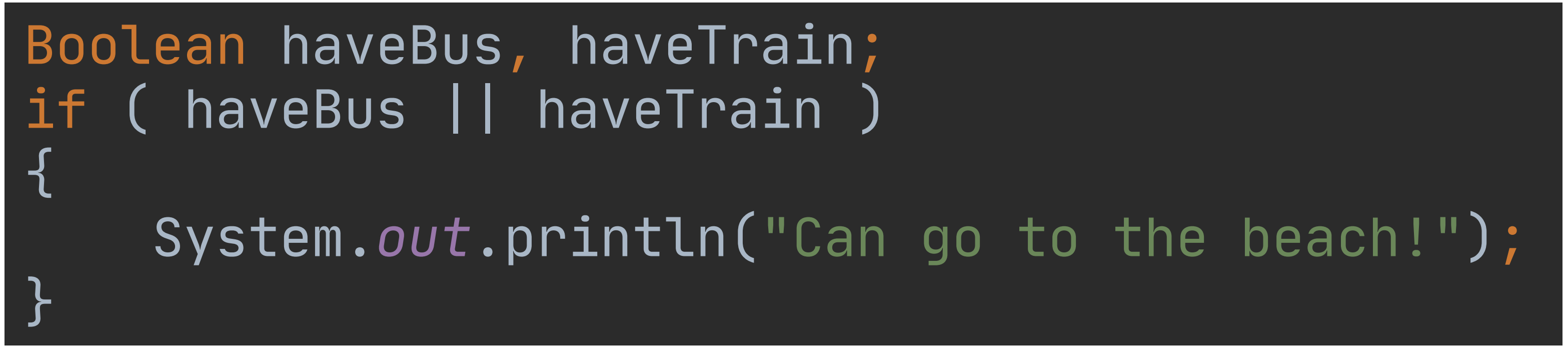
```java
boolean haveCar, haveFuel;
if ( haveCar && haveFuel ){
    System.out.println("Can go to the beach!");
}
```

- Can I go using public transportation?

```java
boolean haveBus, haveTrain;
if ( haveBus || haveTrain ){
    System.out.println("Can go to the beach!");
}
```

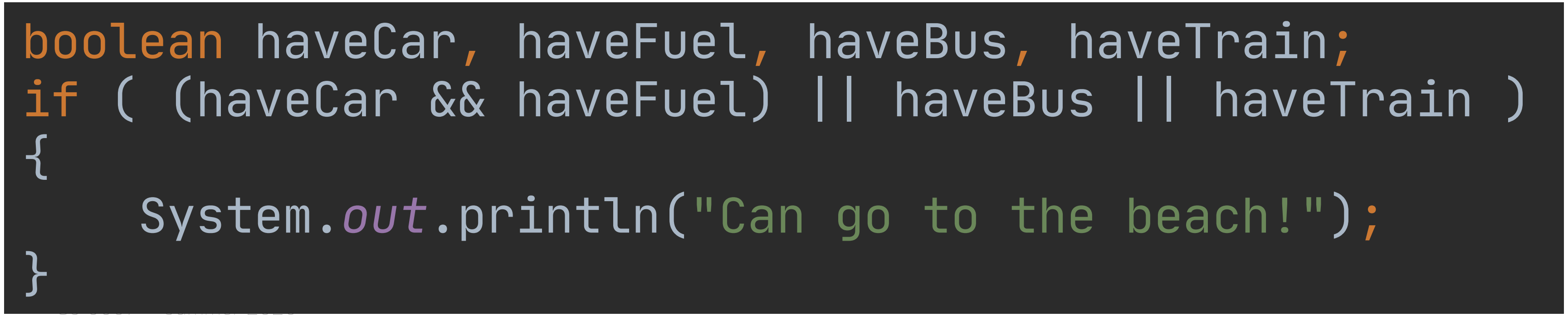
Short-circuits

- Short-circuit: decide before evaluating everything
  - E.g. if I have a bus that I can take it doesn't matter if I have a train



```
Boolean haveBus, haveTrain;
if ( haveBus || haveTrain )
{
    System.out.println("Can go to the beach!");
}
```

  - E.g. if I have a car and fuel, doesn't matter if I have a bus or a train



```
boolean haveCar, haveFuel, haveBus, haveTrain;
if ( (haveCar && haveFuel) || haveBus || haveTrain )
{
    System.out.println("Can go to the beach!");
}
```

Order again

- So… () go first, */% go second, and +- go third
  - Where do the boolean operators fit in this?
- So what goes before/after that?
  - NOT goes before
  - Relational operators go after
  - Logical operators go last
- Last thing done is always assignment

Operator	Associativity
-(negation) ! (NOT)	Right to left
* / %	Left to right
+ -	Left to right
< > <= >=	Left to right
== !=	Left to right
&&	Left to right
||	Left to right
= += -= *= /= %=	Right to left

Soooooo.....

- Some thing like this

```
age > 30 && height < 70
```

- Is equivalent to this

```
(age > 30) && (height < 70)
```

- But the second one is WAYYYY more clear ☺
  - So use parentheses
  - Clarity over character economy!!!

Switches get stitches

Or something like that

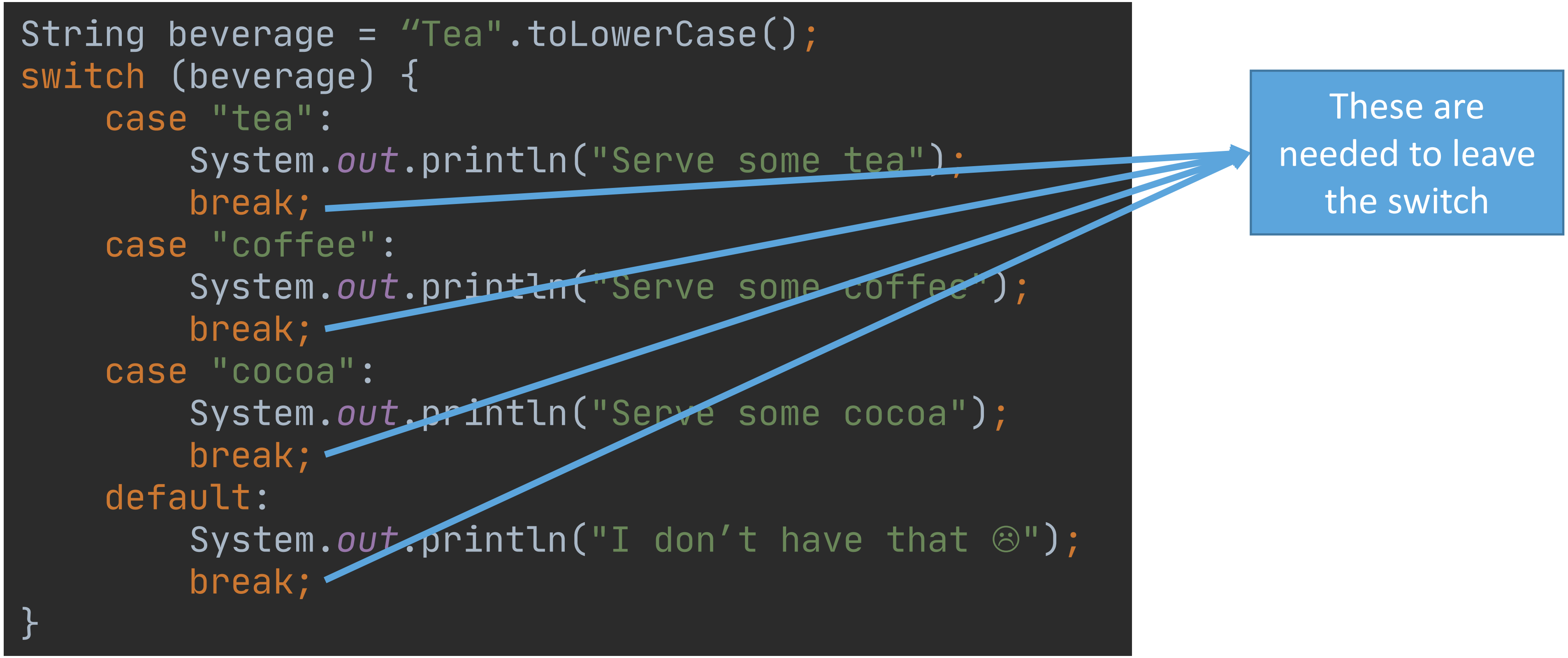
When all conditions are equal

- This is possible! And there is nothing wrong with it.
  - However…

```
String beverage = “Tea".toLowerCase();
if (beverage.equals(“tea"))
{
   System.out.println(“Serve some tea");
}
 else if (beverage.equals(“coffee"))
{
   System.out.println(“Serve some coffee");
}
else if (beverage.equals(“cocoa"))
{
   System.out.println(“Serve some cocoa");
}
else
{
    System.out.println(“I don’t have that ☹");
}
```

Switches

- There is another Java decision structure that you can use

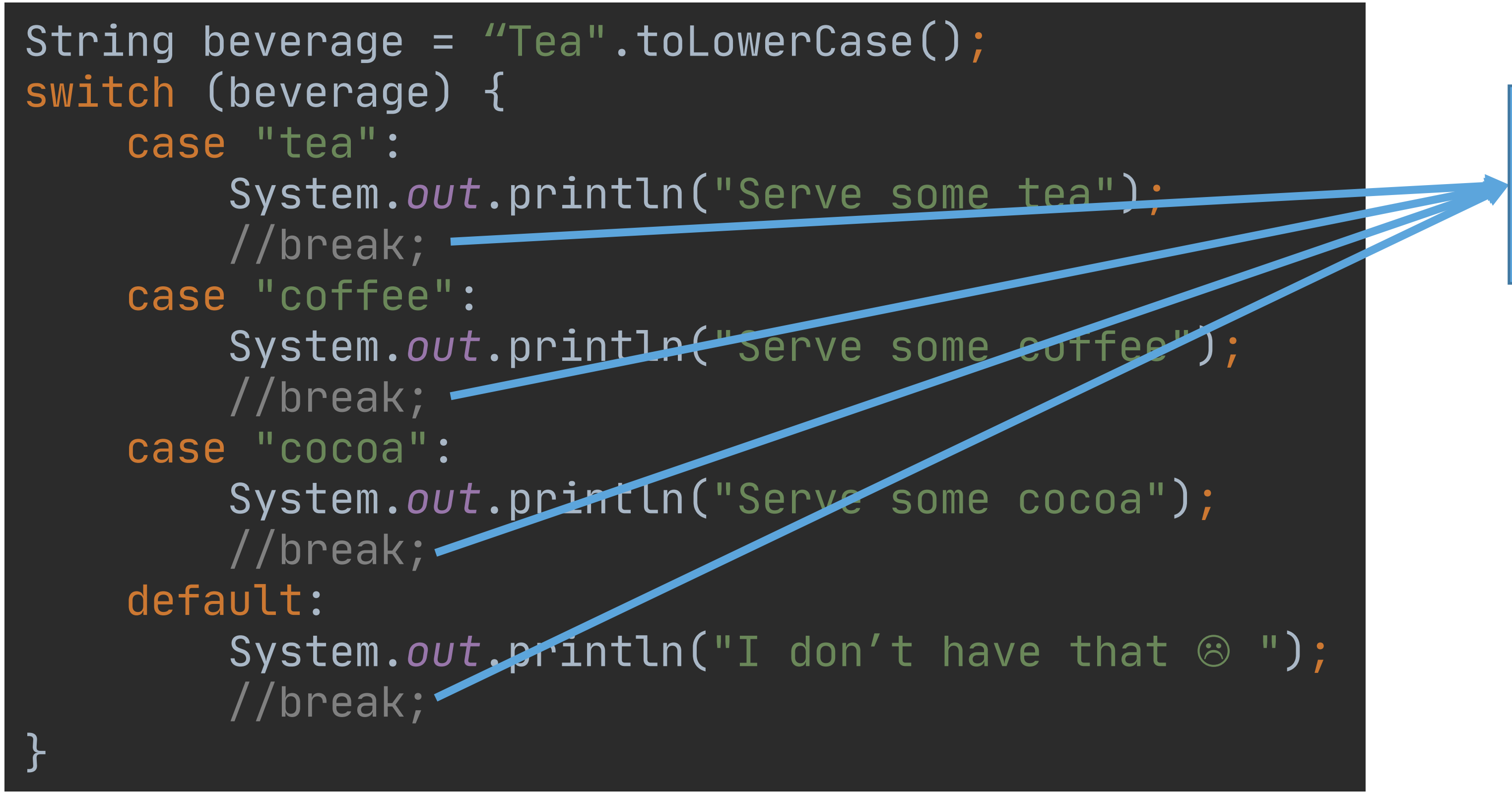


```
String beverage = “Tea".toLowerCase();
switch (beverage) {
    case "tea":
        System.out.println("Serve some tea");
        break;
    case "coffee":
        System.out.println("Serve some coffee");
        break;
    case "cocoa":
        System.out.println("Serve some cocoa");
        break;
    default:
        System.out.println("I don’t have that ☹");
        break;
}
```

These are needed to leave the switch

Switches

- If you remove the breaks, you have the grandmother switch
  - *“You are not eating properly, have everything!”*

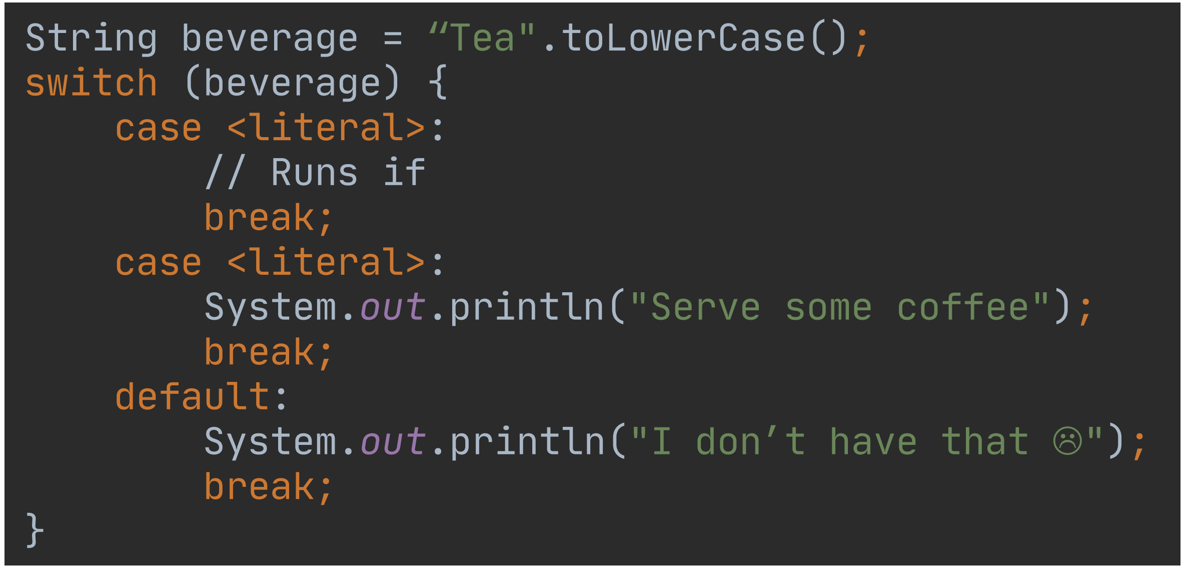


```
String beverage = “Tea".toLowerCase();
switch (beverage) {
    case "tea":
        System.out.println("Serve some tea");
        //break;
    case "coffee":
        System.out.println("Serve some coffee");
        //break;
    case "cocoa":
        System.out.println("Serve some cocoa");
        //break;
    default:
        System.out.println("I don’t have that ☹ ");
        //break;
}
```

Remove them.
See what
happens

Switches

- Switches only work with some types:
  - Integer types (byte, short, int, long)
  - String
  - char
- The case must be a literal!
  - No variables
  - If that is needed use `if`s
- No comparisons
  - Either equal or not-equal
  - No greater/less than, etc.



```
String beverage = “Tea".toLowerCase();
switch (beverage) {
    case <literal>:
        // Runs if
        break;
    case <literal>:
        System.out.println("Serve some coffee");
        break;
    default:
        System.out.println("I don’t have that ☹");
        break;
}
```

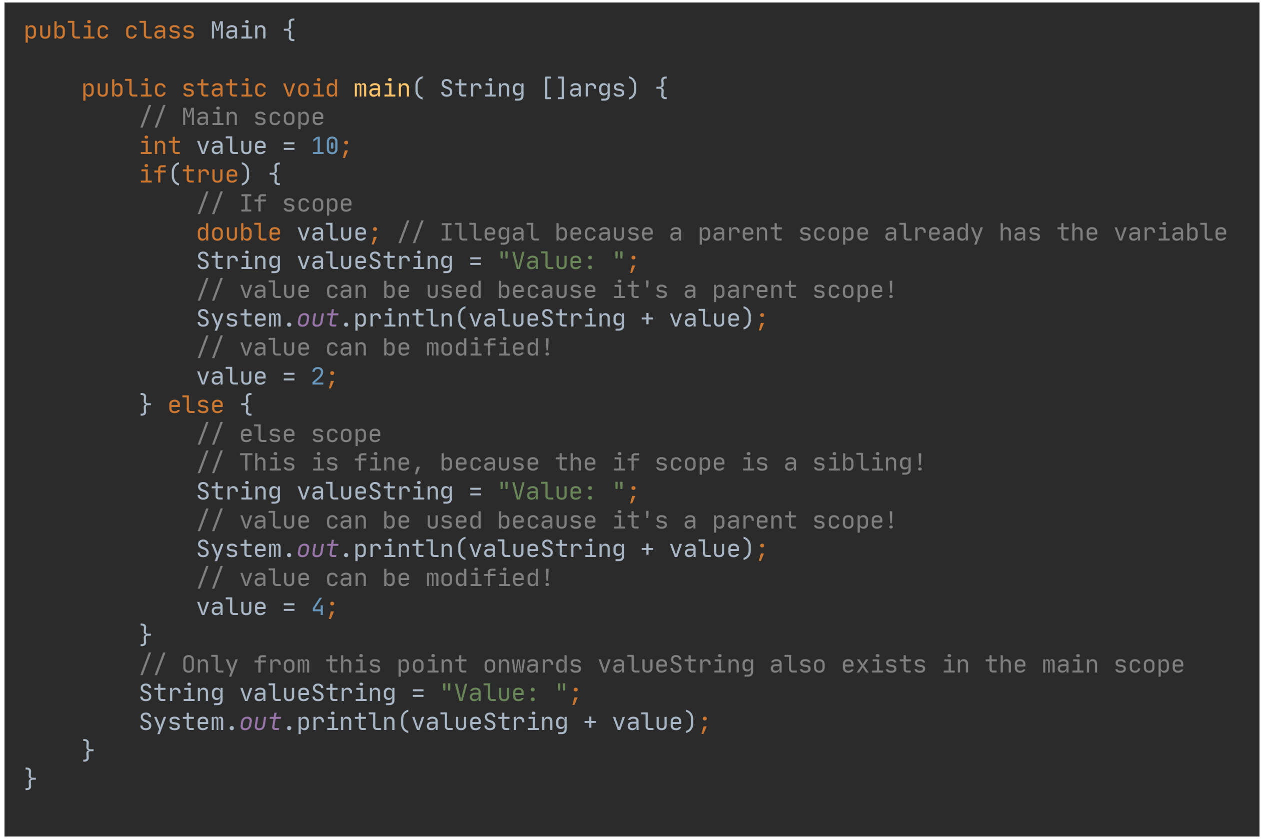
- `default` is the default behaviour (i.e. if nothing else matches)

Scopes

Can you see them?

Blocks and scopes

- Blocks start with { and end with } – each defines its own scope
  - They can be stacked
  - Parent scopes are visible in children scopes
  - Sibling scopes are not visible to each other
  - Variables with same name cannot exist in children scopes
  - Variables with same name can exist in sibling scopes



```java
public class Main {

    public static void main( String []args) {
        // Main scope
        int value = 10;
        if(true) {
            // If scope
            double value; // Illegal because a parent scope already has the variable
            String valueString = "Value: ";
            // value can be used because it's a parent scope!
            System.out.println(valueString + value);
            // value can be modified!
            value = 2;
        } else {
            // else scope
            // This is fine, because the if scope is a sibling!
            String valueString = "Value: ";
            // value can be used because it's a parent scope!
            System.out.println(valueString + value);
            // value can be modified!
            value = 4;
        }
        // Only from this point onwards valueString also exists in the main scope
        String valueString = "Value: ";
        System.out.println(valueString + value);
    }
}
```