Programming 1 / Object Oriented Programming

Introduction to Java: Lecture #2: Conditional and loop constructs / Arrays

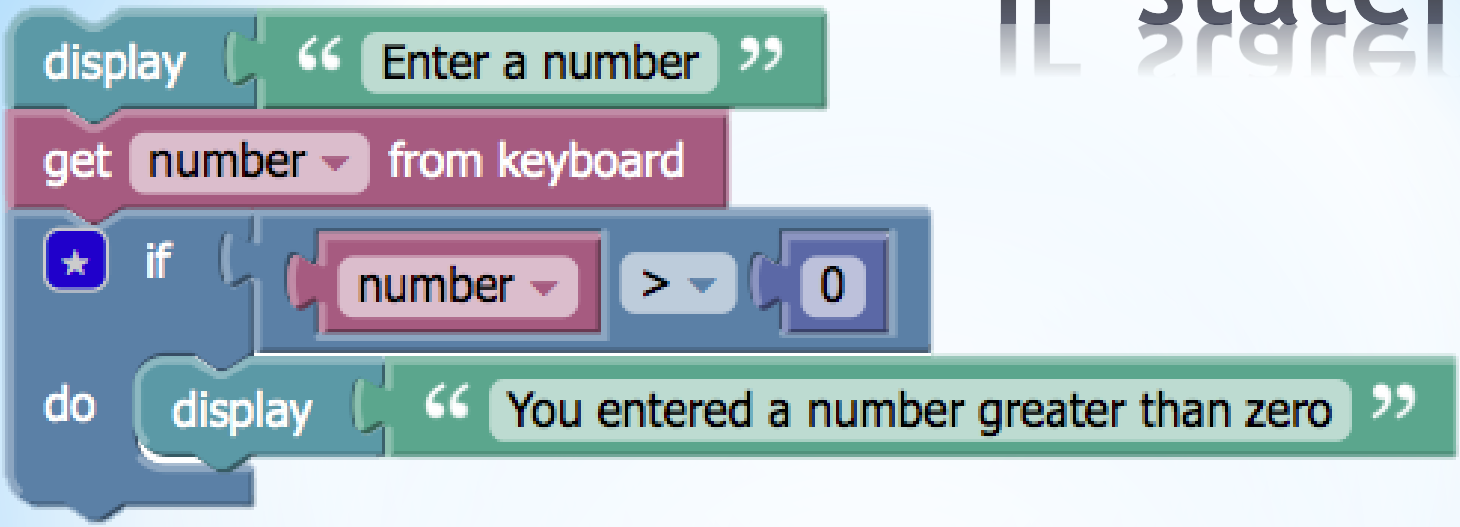
CLICKER CHANNEL: 82

Consider this specification

- The program asks the user to enter a number
- If the user enters a number greater than zero, the program displays a message: “You entered a number greater than zero”
- Otherwise, the program does nothing

- The action of the program depends on the input
- We can create this program using an `if` statement

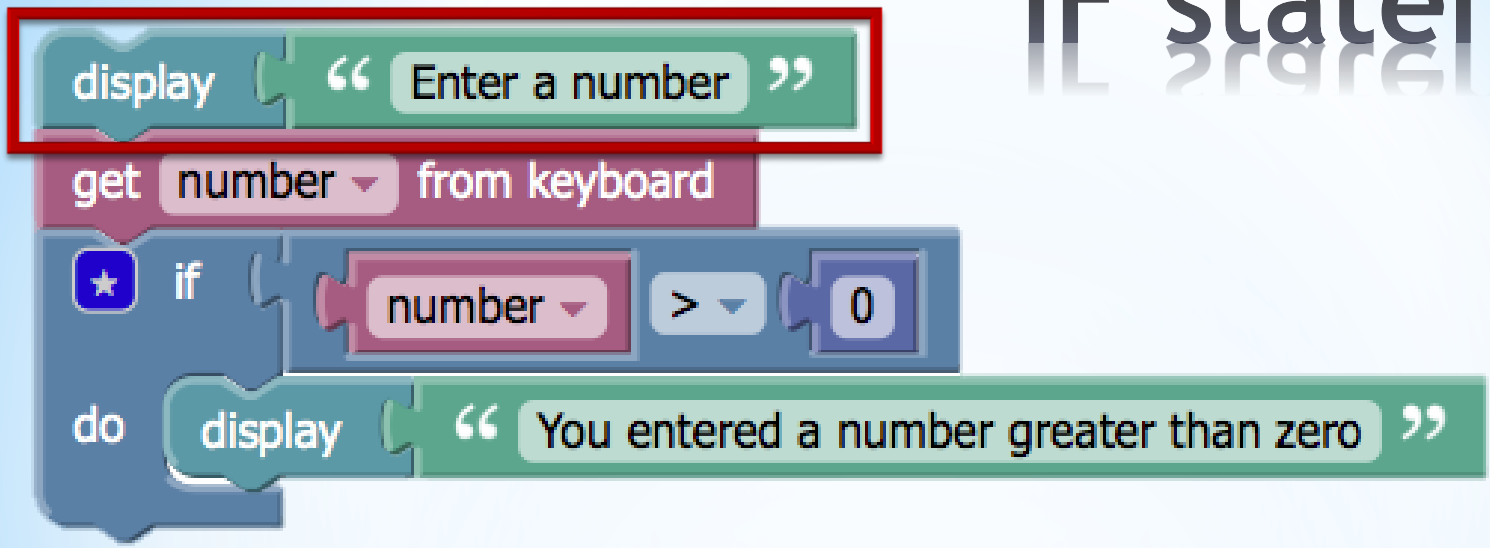
IF statement




```
display "Enter a number"
get number
if number > 0
  display "You entered a number greater than zero"
endif
```

```
System.out.println("Enter a number");
Scanner keyboard = new Scanner(System.in);
int number = keyboard.nextInt();
if (number > 0)
{
    System.out.println("You entered a number greater than zero");
}
```

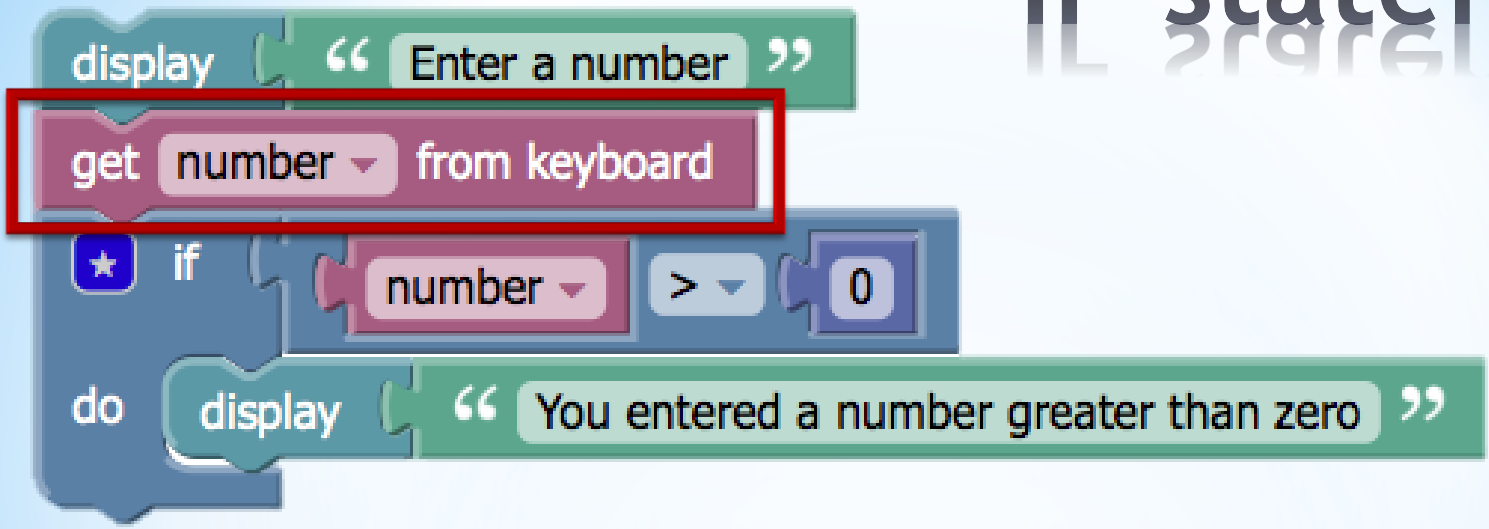
IF statement




```
display "Enter a number"
get number
if number > 0
  display "You entered a number greater than zero"
endif
```

```
System.out.println("Enter a number");
Scanner keyboard = new Scanner(System.in);
int number = keyboard.nextInt();
if (number > 0)
{
    System.out.println("You entered a number greater than zero");
}
```

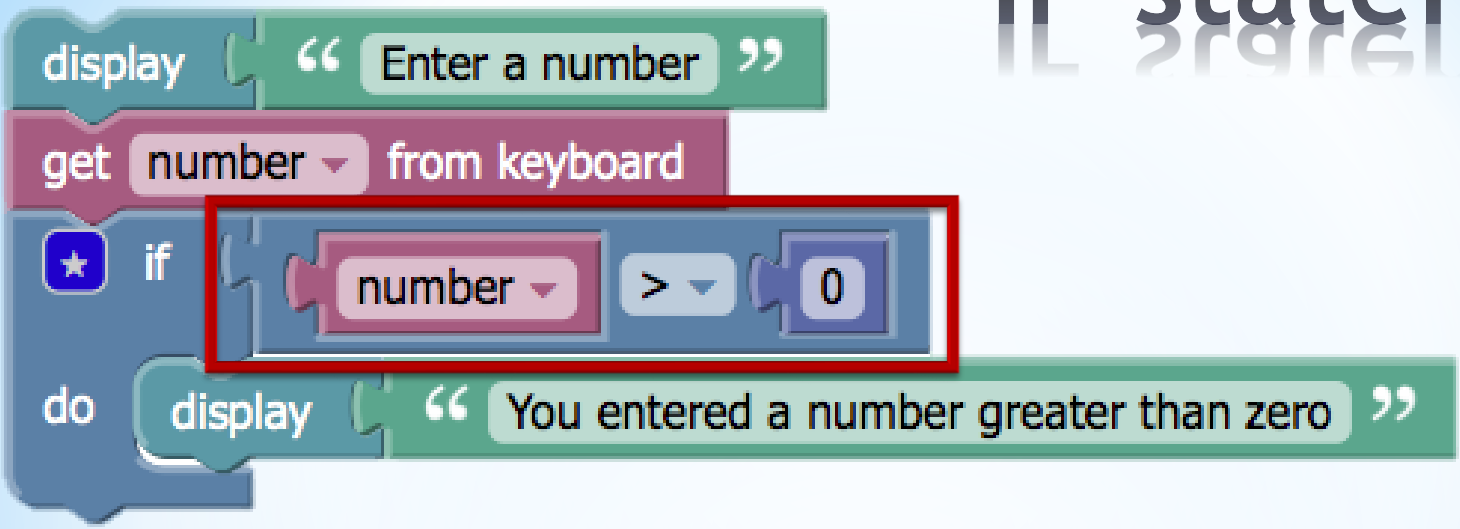
IF statement




```
display "Enter a number"
get number
if number > 0
  display "You entered a number greater than zero"
endif
```

```
System.out.println("Enter a number");
Scanner keyboard = new Scanner(System.in);
int number = keyboard.nextInt();
if (number > 0)
{
    System.out.println("You entered a number greater than zero");
}
```

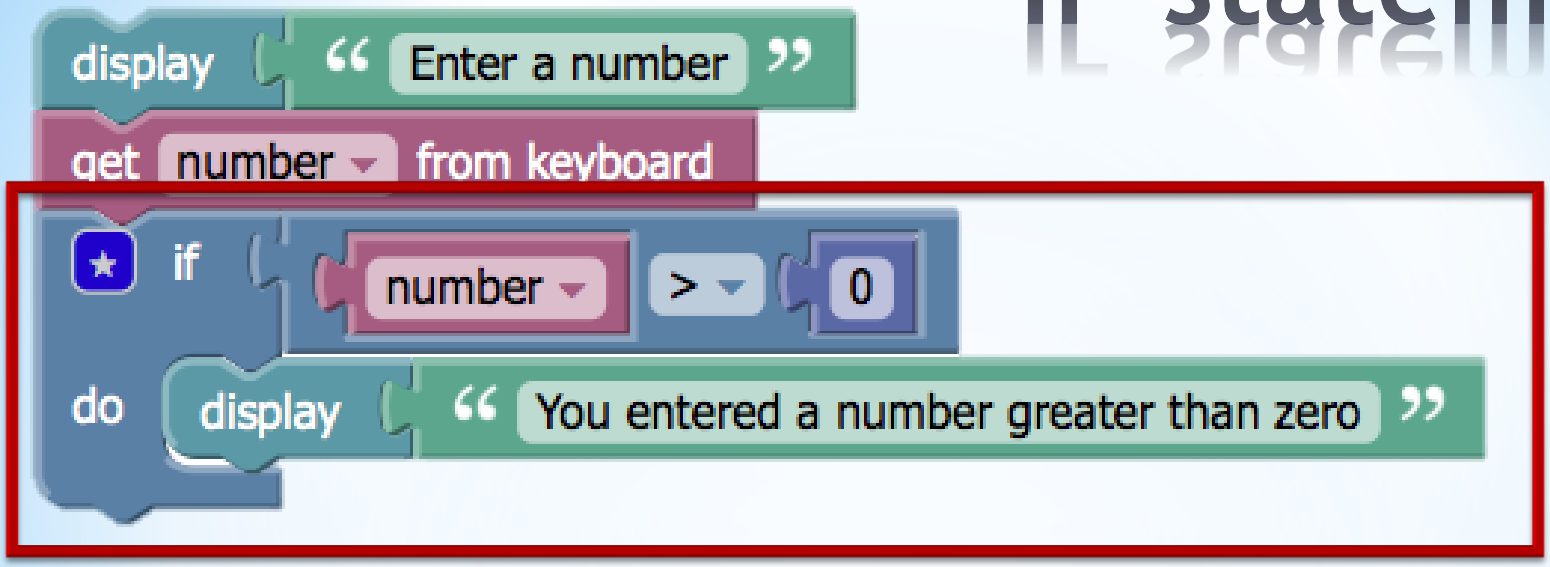
IF statement




```
display "Enter a number"
get number
if number > 0
  display "You entered a number greater than zero"
endif
```

```
System.out.println("Enter a number");
Scanner keyboard = new Scanner(System.in);
int number = keyboard.nextInt();
if (number > 0)
{
    System.out.println("You entered a number greater than zero");
}
```

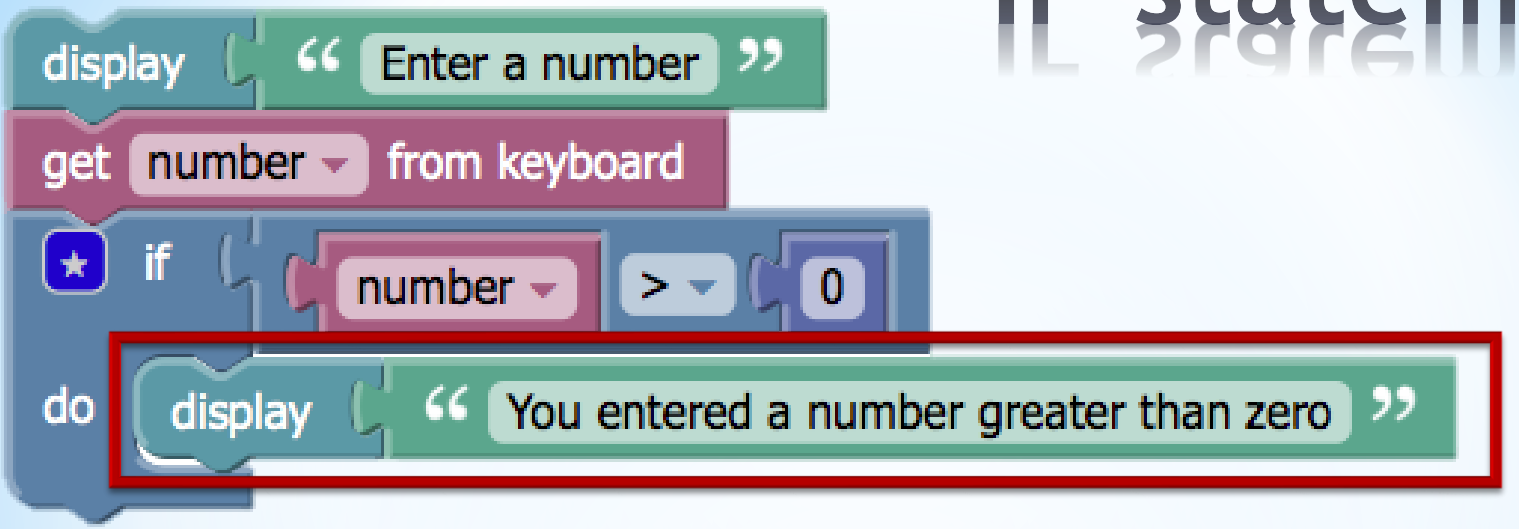
IF statement




```
display "Enter a number"
get number
if number > 0
  display "You entered a number greater than zero"
endif
```

```
System.out.println("Enter a number");
Scanner keyboard = new Scanner(System.in);
int number = keyboard.nextInt();
if (number > 0)
{
    System.out.println("You entered a number greater than zero");
}
```

IF statement

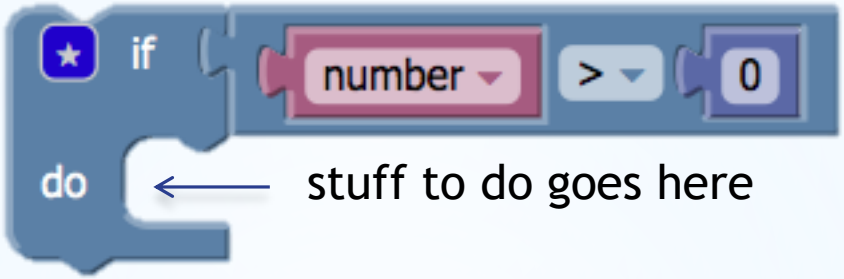



```
display "Enter a number"
get number
if number > 0
  display "You entered a number greater than zero"
endif
```

```
System.out.println("Enter a whole number");
Scanner keyboard = new Scanner(System.in);
int number = keyboard.nextInt();
if (number > 0)
{
    System.out.println("You entered a number greater than zero");
}
```

Remember the blocks

Blocks



Banana/Pseudocode

```
if number > 0
   (stuff to do goes here)
endif
```

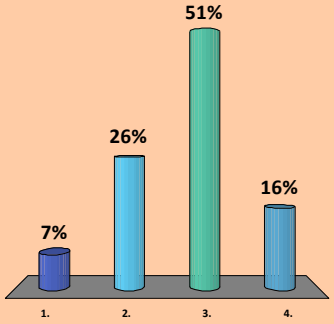
Java

```
if (number > 0)
{
    (stuff to do goes here)
}
```

What would be the output of this program if the user typed -1?

```
Scanner keyboard = new Scanner(System.in);
System.out.println("Enter a whole number");
String input1 = keyboard.nextLine();
int number = Integer.parseInt(input1);
if (number > 0);
{
    System.out.println("You entered a number greater than zero");
    System.out.println("Hooray!");
    System.out.println("We like numbers greater than zero!");
}
```

1. The program would not compile/run
2. The program would run but there would be an error
3. There would be no output
4. It would display the message about the number being greater than zero



Recap: A common mistake

```
Scanner keyboard = new Scanner(System.in);
System.out.println("Enter a whole number");
String input1 = keyboard.nextLine();
int number = Integer.parseInt(input1);
if (number > 0);
{
    System.out.println("You entered a number greater than zero");
    System.out.println("Hooray!");
    System.out.println("We like numbers greater than zero!");
}
```

- Can you see the problem yet?

Recap: A common mistake

```
Scanner keyboard = new Scanner(System.in);
System.out.println("Enter a whole number");
String input1 = keyboard.nextLine();
int number = Integer.parseInt(input1);
if (number > 0);
{
    System.out.println("You entered a number greater than zero");
    System.out.println("Hooray!");
    System.out.println("We like numbers greater than zero!");
}
```

Relational operators

Operator	Meaning	Example
>	greater than	`if (number > 40)`
<	less than	`if (height < 1.5)`
==	equals	`if (counter == 0)`
!=	not equals	`if (records != 1)`
>=	greater than or equal to	`if (students >= 10)`
<=	less than or equal to	`if (result <= -5)`

Relational operators and boolean values

- Relational operators result in a **boolean** value
  - A boolean value has two possible states – TRUE or FALSE
- If an integer variable `value` contains 7

```
value < 5          false
value > 5          true
value == 5         false
value != 5         true
value == 7         true
```

The grammar of relational operators and boolean values

- An **expression** involving relational operators will result in a single value, just as with mathematical operators
- The single value that results from an expression with relational operators can only be TRUE or FALSE – i.e. a boolean
- So our IF statements, grammatically speaking, expect something that will end up as a single boolean value

Which of these code fragments is grammatically incorrect?

```
int value = 7;
if (value == 7)
{
  value = 0;
}
```

1

```
int value = 7;
if (value > 5)
{
  value = 0;
}
```

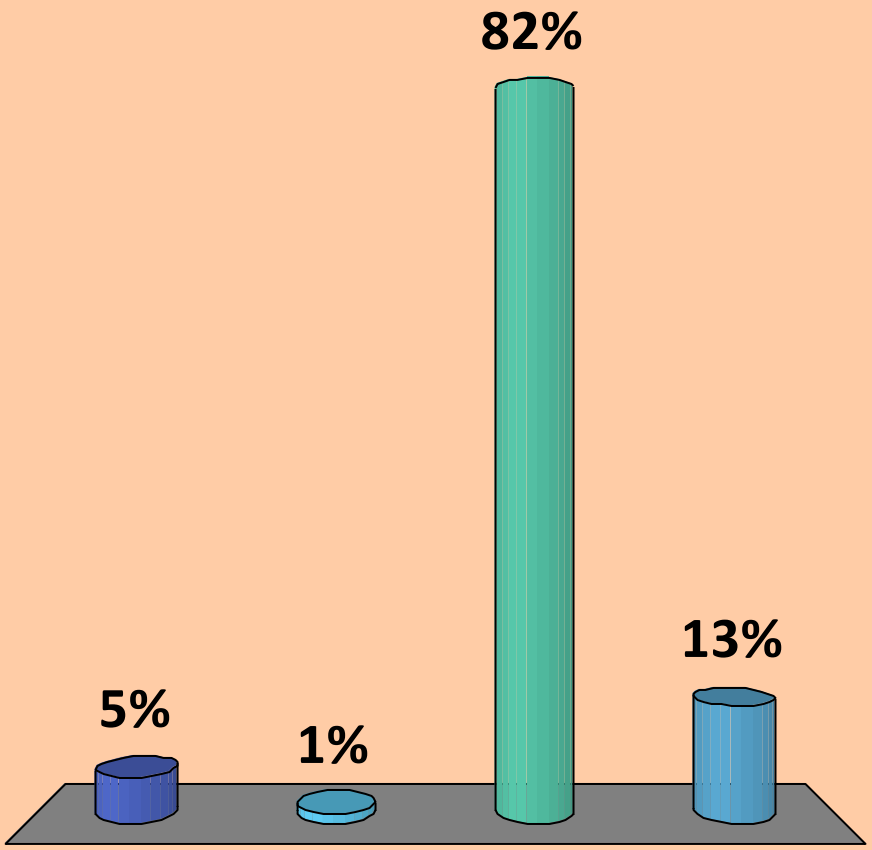
2

```
int value = 7;
if (value + 5)
{
  value = 0;
}
```

3

```
int value = 7;
if (value + 5 == 13)
{
  value = 0;
}
```

4



1.
2.
3.
4.

The grammar of relational operators and boolean values

```
if (an expression that results in true or false)
{
   …do something
}
```

```
int value = 7;
if (value == 7)
{
  value = 0;
}
```

```
int value = 7;
if (value > 5)
{
  value = 0;
}
```

```
int value = 7;
if (value + 5)
{
  value = 0;
}
```

```
int value = 7;
if (value + 5 == 13)
{
  value = 0;
}
```

The boolean data type

```
boolean status = true;
if (status == true)
{
    System.out.println("Status is true");
}
```

The boolean data type

```
boolean status = true;
if (status == true)    <- Not needed - why?
{
    System.out.println("Status is true");
}
```

Pseudocode - Compare Strings

Display “Enter first word”

Get *str1*

Display “Enter second word”

Get *str2*

If *str1* == *str2*

    Display “Your two words are the same”

EndIf

String comparison

- Strings are *different* (irritatingly…)

```
if (str1.equals(str2))
{
    System.out.println("The two words are the same");
    System.out.println("The word you entered was: " + str1);
}
```

- Strings are compared with the .equals method
- Comparison is **case sensitive**
  - "flibble" is not the same as "FLIBBLE"
- **Do NOT use str1 == str2 with strings**
  - …that might not work as expected

String comparison

```
if (str1.equalsIgnoreCase(str2))
{
    System.out.println("The two words are the same");
    System.out.println("The word you entered was: " + str1);
}
```

- Can also are compare with the .equalsIgnoreCase method
- In this case the comparison is NOT **case sensitive**
  - "flibble" IS the same as "FLIBBLE"

Pseudocode

Display “Enter a number”

Get *number*

If *number* < 0

    Display “You entered a number less than zero”

Else

    Display “You entered a number that was zero or greater”

EndIf

General structure of `if-else` statement

```
if (boolean expression)
{
   statements;
}
else
{
   statements;
}
```

evaluate this expression to get a value of `true` or `false`

if the expression evaluates to `true`, execute these statements

if `false`, execute these statements

Using if-else

```
display “Enter a number”
get number
if number < 0
    Display “You entered a number less than zero”
else
    Display “You entered a number that was zero or greater”
endIf
```

```
Scanner keyboard = new Scanner(System.in);
System.out.println("Enter a whole number");
int number = keyboard.nextInt();
if (number < 0)
{
    System.out.println("You entered a number less than zero");
}
else
{
    System.out.println("You entered a number zero or greater");
}
```

Chained `if` statements

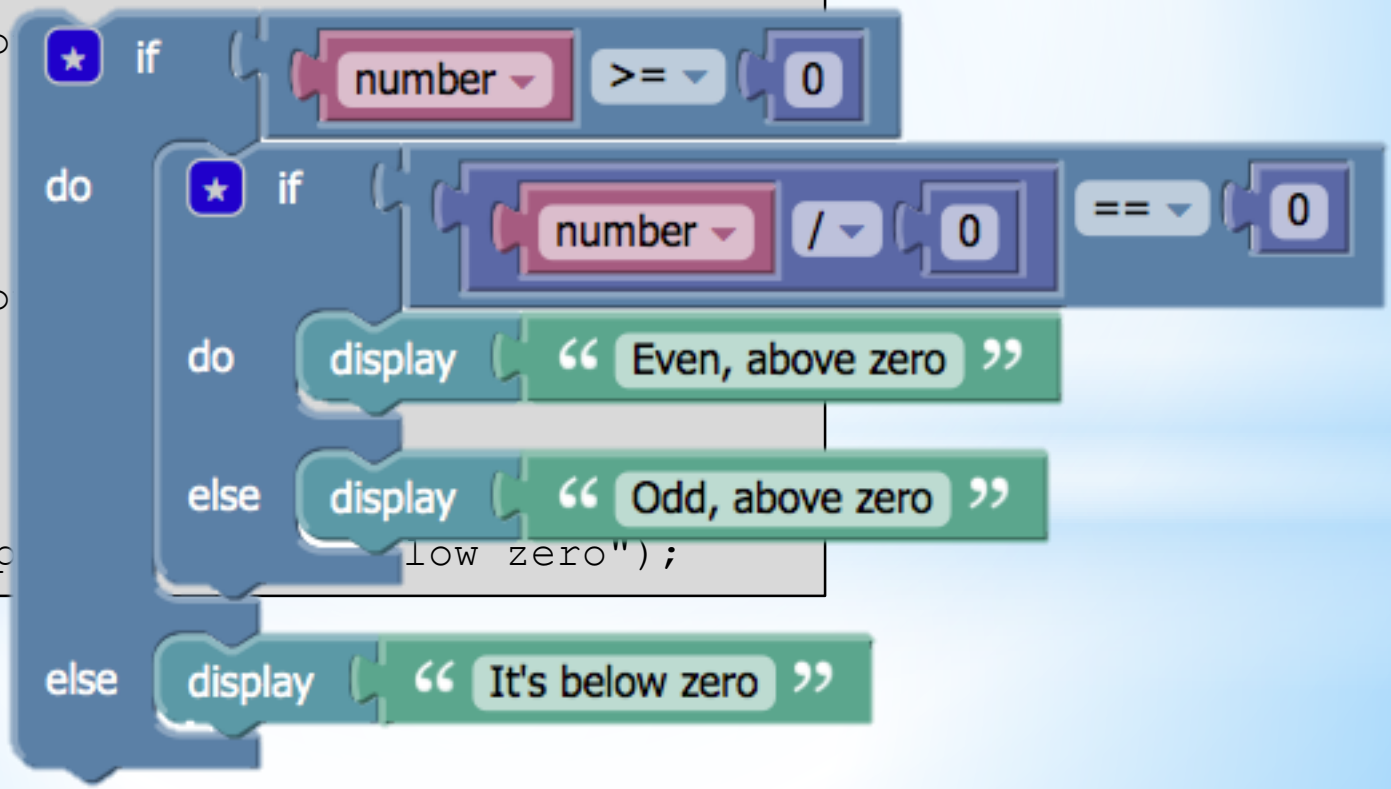
```
if (number > 0)
{
    System.out.println("It's greater than zero");
}
else if (number == 0)
{
    System.out.println("It's equal to zero");
}
else
{
    System.out.println("It's below zero");
}
```

Nested `if` statements - see the blocks

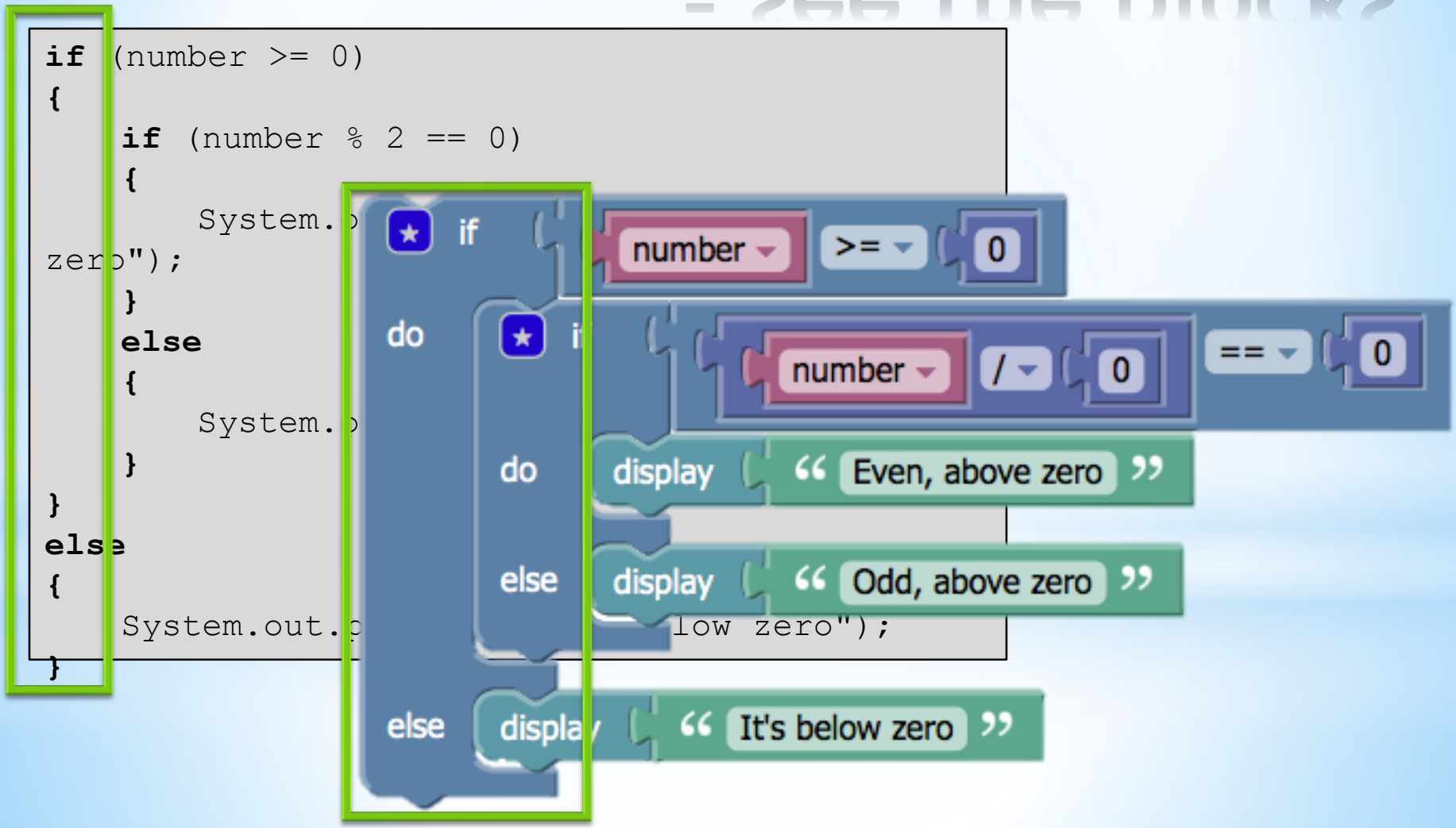
```
if (number >= 0)
{
    if (number % 2 == 0)
    {
        System.out.println("Even, above
zero");
    }
    else
    {
        System.out.println("Odd, above zero");
    }
}
else
{
    System.out.println("It's below zero");
}
```

Nested `if` statements - see the blocks

```
if (number >= 0)
{
    if (number % 2 == 0)
    {
        System.o
zero");
    }
    else
    {
        System.o
    }
}
else
{
    System.out.p                low zero");
}
```

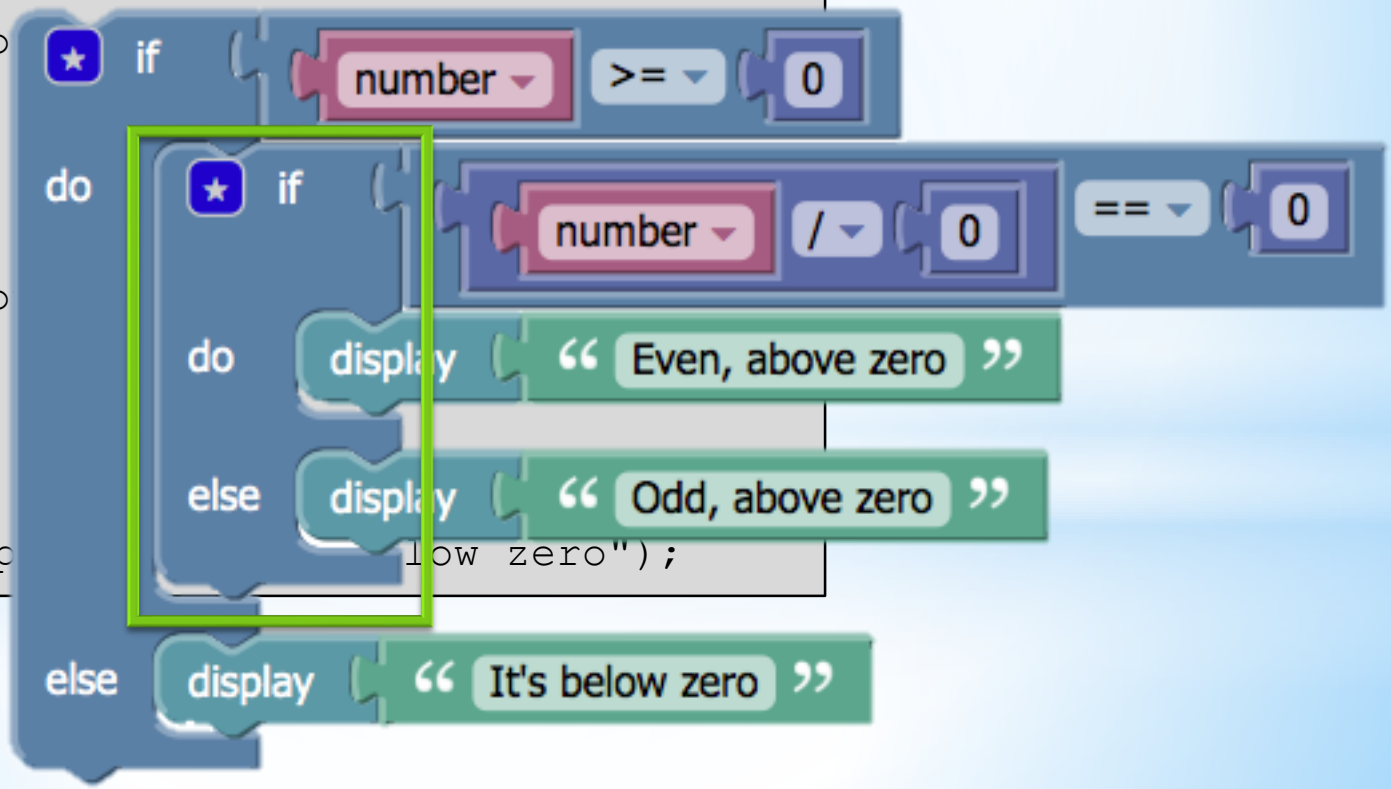


Nested `if` statements - see the blocks



Nested `if` statements - see the blocks

```
if (number >= 0)
{
    if (number % 2 == 0)
    {
        System.o
zero");
    }
    else
    {
        System.o
    }
}
else
{
    System.out.p                low zero");
}
```



Logical operators

Operator	Name	Description
||	OR	If ANY of the conditions are true, this operator will return TRUE. If ALL of the conditions are false, it will return FALSE
&&	AND	If ALL of the conditions are true, this operator will return TRUE. If ANY of the conditions are false, it will return FALSE.
!	NOT	Reverses a condition – so if the original condition was true, this will return FALSE. If the original condition was false, this will return TRUE

Logical operators

- Logical operators let you combine several conditions

```
Scanner keyboard = new Scanner(System.in);
System.out.println("Enter your name");
String name = keyboard.nextLine();
if (name.equals("Paul") || name.equals("Fred"))
{
   System.out.println("I was looking for you, Paul or Fred");
}
else
{
   System.out.println("I was looking for someone else");
}
```

Logical operators

- Assume we type “Fred”…

```
Scanner keyboard = new Scanner(System.in);
System.out.println(“Enter your name”);
String name = keyboard.nextLine();
if (name.equals(“Paul”) || name.equals(“Fred”))
{         false     ||          true
   System.out.println(“I was looking for you”);
}
else
{
   System.out.println(“I was looking for someone else”);
}
```

Logical operators

- Assume we type “Fred”…

```
Scanner keyboard = new Scanner(System.in);
System.out.println(“Enter your name”);
String name = keyboard.nextLine();
if (name.equals(“Paul”) || name.equals(“Fred”))
{        false     ||        true
   System.out.println(“I was looking for you”);
}
else
{
   System.out.println(“I was looking for someone else”);
}
```

TRUE

Introducing the FOR loop in Java

```
for (int i = 0; i < 5; i++)
{
    System.out.println("Java is awesome");
}
```

Declares and initialises the counter variable – in this case, the counter is *i* and it starts at zero.

Specifies the *condition* under which the loop continues to repeat – in this case it keeps going while it's less than 5… i.e. it does it UNTIL it gets to 4…

Introducing the FOR loop in Java

```
for (int i = 0; i <= 4; i++)
{
    System.out.println("Java is awesome");
}
```

equivalent to the pseudocode

```
FOR I = 0 TO 4
DISPLAY "Java is awesome"
ENDFOR
```

Declares and initialises the counter variable – in this case, the counter is *i* and it starts at zero.

Specifies the *condition* under which the loop continues to repeat – in this case it keeps going UNTIL it gets to 5.

What happens to the counter variable every time we repeat the loop – in this case, it is increased by 1.

Variable scope in Java

- Depending on where a variable gets declared, only certain parts of a program will be able to see the variable
- As a rule of thumb, if a variable is declared within a code block, it can only be seen from INSIDE that code block
- Variables declared inside a code black CANNOT be seen outside the code block
- The below would NOT WORK:

```
Scanner keyboard = new Scanner(System.in);
int x = keyboard.nextInt();
if (x > 10)
{
    int y = 27;
}
System.out.println("Y is "+y);
```

Variable scope in Java

- Depending on where a variable gets declared, only certain parts of a program will be able to see the variable
- As a rule of thumb, if a variable is declared within a code block, it can only be seen from INSIDE that code block
- Variables declared inside a code black CANNOT be seen outside the code block
- The below would NOT WORK:

```
Scanner keyboard = new Scanner(System.in);
int x = keyboard.nextInt();
if (x > 10)
{
    int y = 27;
}
System.out.println("Y is "+y);
```

Breaks variable scope…
This statement cannot see "inside" the code block

Variable scope in Java

- This WOULD work though:

```
Scanner keyboard = new Scanner(System.in);
int x = keyboard.nextInt();
if (x > 10)
{
    int y = 27;
    if (x > 20)
    {
        System.out.println("Y is "+y);
    }
}
```

Variable scope in Java

- This WOULD work though:

```
Scanner keyboard = new Scanner(System.in);
int x = keyboard.nextInt();
if (x > 10)
{
    int y = 27;
    if (x > 20)
    {
        System.out.println("Y is "+y);
    }
}
```

this...

Variable scope in Java

- This WOULD work though:

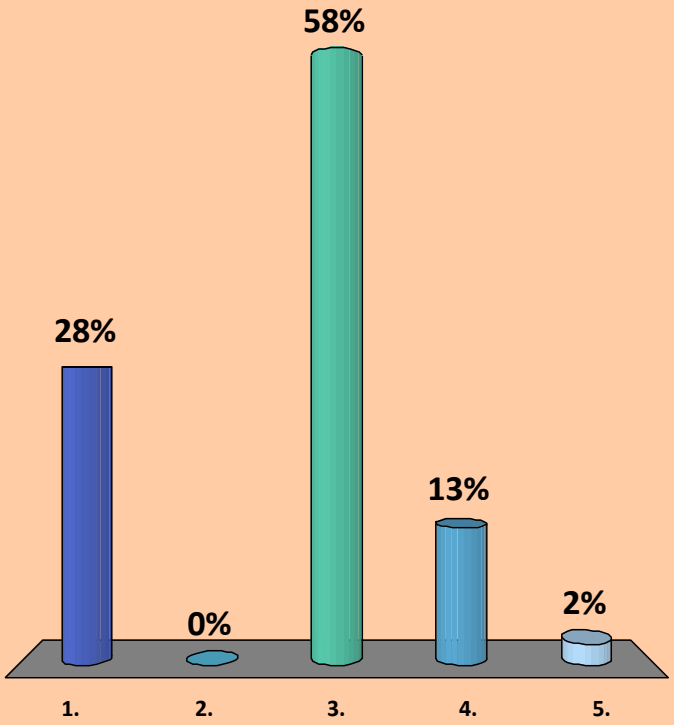
```
Scanner keyboard = new Scanner(System.in);
int x = keyboard.nextInt();
if (x > 10)
{
    int y = 27;
    if (x > 20)
    {
        System.out.println("Y is "+y);
    }
}
```

…is still inside this

How could you fix this code?

```
for (int i = 0; i < 10; i++)
{
  System.out.println("I is now "+i);
}
System.out.println("I finished up as "+i);
```

1. Fix it? It's fine. Leave it alone!
2. Remove the word **int** from the first line
3. Change the first two lines to read
   ```
   int i;
   for (i = 0; i < 10; i++)
   ```
4. Declare a new variable inside the loop and assign the value of i to it, then display that new variable in the final line of the program
5. Something else



Variable scope and for loops

```
for (int i = 0; i < 10; i++)
{
  System.out.println("I is now "+i);
}
System.out.println("I finished up as "+i);
```

- So above, *i* is declared as part of the FOR statement…
  - …therefore it is only visible in the code block of the FOR loop.
- In the final line *i* is out of scope
- We must declare the counter various outside of the loop if we wish to use it outside of the loop:

```
int i; // declare our variable first
for (i = 0; i < 10; i++)
{
  System.out.println("I is now "+i);
}
System.out.println("I finished up as "+i);
```

Remember your grammar

- You don't have to use a literal value in your FOR loop condition – you can use a variable or, indeed, any expression

```
System.out.println("How awesome is Java?");
System.out.println("Give a number between 1 and 10");
Scanner keyboard = new Scanner(System.in);
int repetitions = keyboard.nextInt()
for (int i = 0; i < repetitions; i++)
{
   System.out.println("Java is awesome x "+i);
}
```

- or even…

```
System.out.println("How awesome is Java?");
System.out.println("Give a number between 1 and 10");
Scanner keyboard = new Scanner(System.in);
int repetitions = keyboard.nextInt()
for (int i = 0; i < repetitions*10; i++)
{
   System.out.println("Java is awesome x "+i);
}
```

Nested loops (loops within loops)

```
for (int i = 0; i < 10; i++)
{
    String stars = "*";
    for (int j = 0; j < i; j++)
    {
        stars = stars + " *";
    }
    System.out.println(stars);
}
```

The WHILE loop

```
int i = 0;
while (i < 5)
{
   System.out.println("Java is awesome");
   i++;
}
```

equivalent to the pseudocode

```
SET i = 0;
WHILE i < 5
  DISPLAY "Java is awesome"
  i = i + 1;
ENDWHILE
```

The WHILE loop

```
int i = 0;
while (i < 5)
{
    System.out.println("Java is awesome");
    i++;
}
```

equivalent to the pseudocode

```
SET i = 0;
WHILE i < 5
  DISPLAY "Java is awesome"
  i = i + 1;
ENDWHILE
```

The **condition** specifies the scenario under which the loop continues to repeat - in this case it keeps going WHILE the variable i is less than 5.

While the **condition** evaluates to true, these statements will repeatedly run.

DO-WHILE

```
int i = 0;
do
{
   System.out.println("Java is awesome");
   i++;
} while (i != 5);
```
<- note the semicolon

equivalent to the pseudocode

```
SET i = 0;
REPEAT
  DISPLAY "Java is awesome"
  i = i + 1;
UNTIL i == 5
```

DO-WHILE

```
int i = 0;
do
{
    System.out.println("Java is awesome");
    i++;
} while (i != 5);
```

The **condition** specifies the *condition* under which the loop continues to repeat – in this case it keeps going WHILE the variable i is not equal to 5.

equivalent to the pseudocode

```
SET i = 0;
REPEAT
  DISPLAY "Java is awesome"
  i = i + 1;
UNTIL i == 5
```

While the **condition** evaluates to true, these statements while repeatedly run.

IMPORTANT: note the fundamental difference between the REPEAT/UNTIL construct you saw in pseudocode and Java's DO/WHILE.

- REPEAT/UNTIL repeats **until** the final condition is TRUE.
  - So when it is true it stops!
- DO/WHILE in Java repeats **while** the condition is true
  - So when it is true it repeats!

WHILE y DO-WHILE in a nutshell

```
Scanner keyboard = new Scanner(System.in);
System.out.println("Do you want to be insulted?");
String choice = keyboard.nextLine();
while (choice.equals("yes"))
{
  System.out.println("You are ugly");
  System.out.println("Go again?");
  String choice = keyboard.nextLine();
}
```

```
Scanner keyboard = new Scanner(System.in);
System.out.println("Do you want to be insulted?");
String choice = keyboard.nextLine();
do
{
  System.out.println("You are ugly");
  System.out.println("Go again?");
  String choice = keyboard.nextLine();
} while (choice.equals("yes"));
```

What happens if you say "no" right at the start?

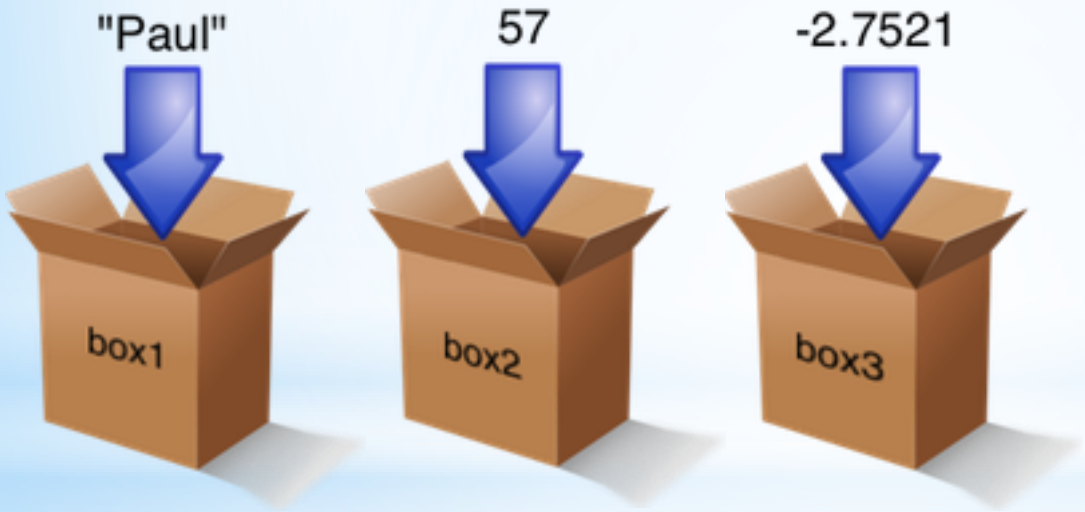
The moral of the story: deciding which loop to use and when

- `for` loop - when the number of repetitions can be determined *before the loop is entered*
- `while` loop - if the number of repetitions cannot be determined before the loop is entered
- `do-while` loop - same as a `while` loop, but *the statements are executed at least once*

Arrays

Variables - a recap

- A variable is like a box – although with certain rules and restrictions
  - Our boxes have labels on the side
  - We can put **one** thing **only** into each box



In pseudocode, this would be

```
set box1 = "Paul";
set box2 = 57;
set box3 = -2.7521
```

In Java, this would be

```
String box1 = "Paul";
int box2 = 57;
double box3 = -2.7521
```

Variables - a recap

- If you try to put a value into a variable that is already defined, and already contains a value, the new value **replaces** the old one

```
String name = "Prince";
System.out.println(name);
name = "The Artist Formerly Known As";
System.out.println(name);
```

Introducing arrays

- An array is a special type of variable in that it can contain many values
- If a standard variable is like a box, think of an array as being like a box with compartments:

- One of these "compartments" is more correctly referred to as an *element* of the array
  - Each element has a unique number (or *index*)
  - In most programming languages element indexes start at 0

Arrays in Java

- To Create an Array in Java
  - Use the **new** operator

```
// 3 ints
int[] arr;
arr =  new int[3];
```

create array of 3 ints:

arr[0], arr[1], arr[2]

or

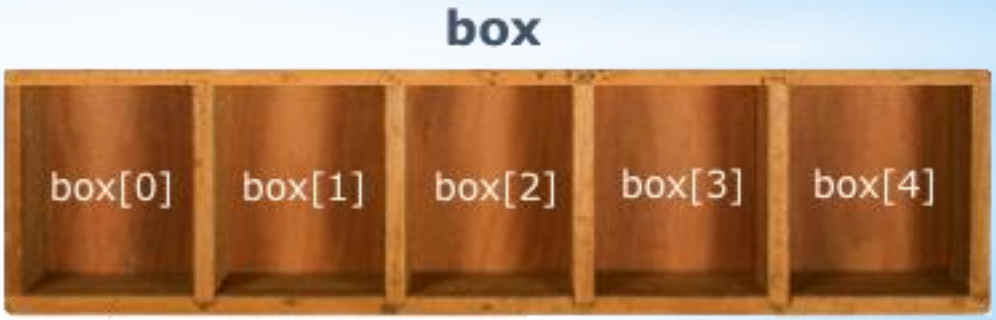
```
int[] arr = new int[3];
```

Can be combined in one statement

Arrays in "boxspeak"

- If a variable is like a box, then an array is like a box with numbered compartments…

```
String[] box = new String[5];
```



Arrays in "boxspeak"

- If a variable is like a box, then an array is like a box with numbered compartments…

```
String[] box = new String[5];
```

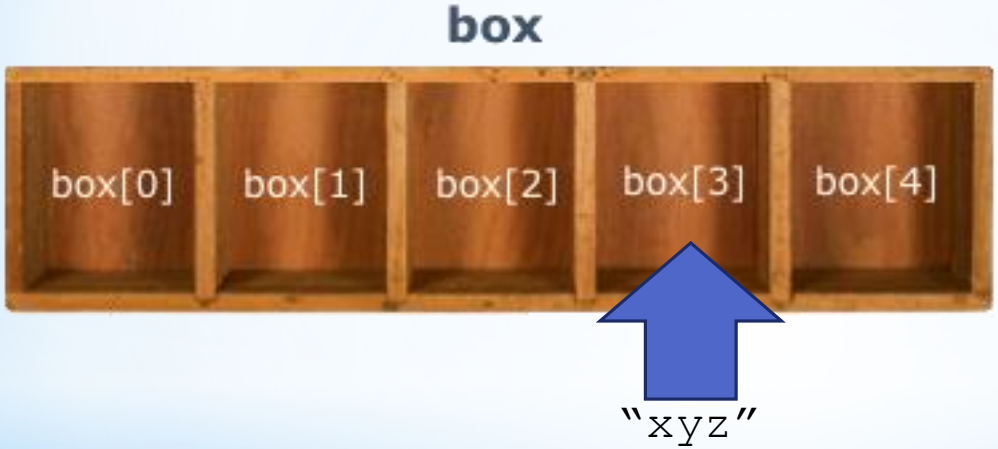
specify what data type the new array will have

give it a name (in this case, *box*)

allocate 5 compartments (*elements*)

Accessing the compartments

- Place elements into a “compartment” of the array by specifying the compartment number:
  - `box[3] = “xyz”;`



- Until you assign something to an array element, it will contain the default value for that data type
  - Numeric primitives (e.g. int, double) will be zero
  - Strings will be *null*

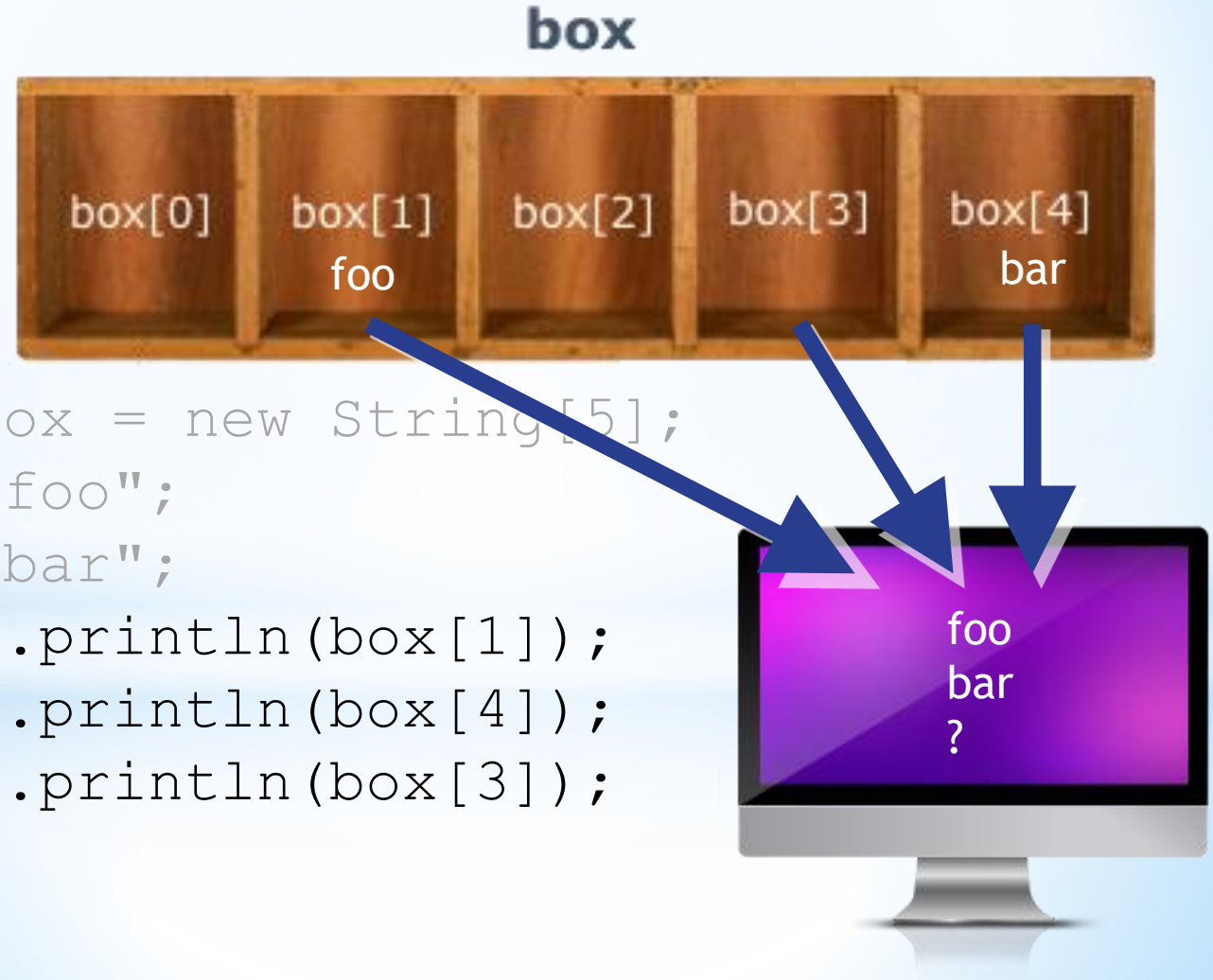
Accessing the compartments

- We access an array by using its name, just like any other variable
- However, usually we will want to be accessing not only the array but a specific element of the array
- We specify the element we're dealing with by putting the element number in square brackets after the array name

Accessing the compartments


```
String[] box = new String[5];
box[1] = "foo";
box[4] = "bar";
System.out.println(box[1]);
System.out.println(box[4]);
System.out.println(box[3]);
```

Accessing the compartments




```
String[] box = new String[5];
box[1] = "foo";
box[4] = "bar";
System.out.println(box[1]);
System.out.println(box[4]);
System.out.println(box[3]);
```

Initialising the compartments

- Until you assign something to an array element, it will contain the default value for that data type
  - Numeric primitives (e.g. int, double) will be zero
  - Strings will be *null*
- Translation: if you leave an array element empty, then depending on what kind of data type your array stores, you'll get a starting value
  - An array that stores numbers will have zero in all the elements
  - An array that stores objects (such as Strings) will start out with the elements being *null*
    - *Null* is a special value which means nothing – like undefined in Banana or Javascript

Arrays: what's the point?

- Why bother?
- Why not just do

```
String box1 = "foo";
String box2 = "bar";
String box3 = "fubar";
```

- instead of

```
String box[] = new String[3]
box[0] = "foo";
box[1] = "bar";
box[2] = "fubar";
```

- ?

Arrays: what's the point?

- What about if you wanted to print out all the values in an array?
- If we didn't use an array and just used lots of individual variables, we could only access each variable by explicit name
- We would need one line of code to print each one

```
String box1 = "foo";
String box2 = "bar";
String box3 = "fubar";
System.out.println(box1);
System.out.println(box2);
System.out.println(box3);
```

- What if we had 10 of these? 100? 1000?!

Arrays: what's the point?

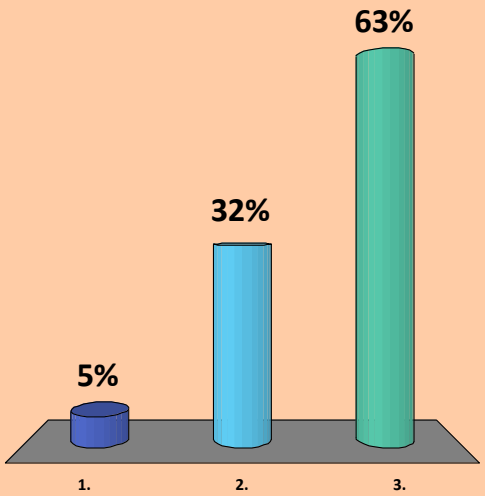
- Contrast the following:

```
String[] box = new String[3];
box[0] = "foo";
box[1] = "bar";
box[2] = "fubar";
for (int count = 0; count < 3; count++)
{
   System.out.println(box[count]);
}
```

- What would this do?
- **Remember your grammar!** ☺
- What if we had 10 elements in the array? 100? 1000?

What would this code do?

```
String[] words = new String[5]
words[0] = "one";
words[1] = "two";
words[2] = "three";
words[3] = "four";
System.out.println("Enter a number");
Scanner keys = new Scanner(System.in);
int num = keys.nextInt();
System.out.println("You typed "+words[num]);
```



1. It would convert whatever word the user typed in into the corresponding number (so if they typed ***three*** they'd get ***3***)
2. It would convert whatever number the user typed in into the corresponding word (so if they typed ***3*** they'd get ***three***)
3. It would do something else

Declaring arrays in one hit

- You can declare and populate an array in a single line of code:

```
String[] words = { "zero", "one", "two", "three", "four" };
System.out.println("Enter a number");
Scanner keys = new Scanner(System.in);
int num = keys.nextInt();
System.out.println("The number in words is "+words[num]);
```

The power of arrays

- Here's a good one ☺

```
System.out.println("How many numbers do you want to
store?");
Scanner keys = new Scanner(System.in);
int max = keys.nextInt();
int[] numbers = new int[max];
for (int count = 0; count < max-1; count++)
{
   System.out.println("Enter number "+count);
   numbers[count] = keys.nextInt();
}
System.out.println("The numbers you entered were");
for (int count = 0; count < max-1; count++)
{
   System.out.println(numbers[count]);
}
```

The power of arrays

- We can access elements of arrays by number
- The *grammar* of programming means that anything that results in a number is treated the same as a number…
  - variables that contain numbers
  - calculations
- We can write constructs that access arrays programmatically
  - A for loop that iterates through all the elements of an array
  - An array that's declared based on the value of a variable
  - A program that asks the user which element they want to view
  - …the sky's the limit!
- You CANNOT do this with conventional variables

The power of arrays

- We might use two arrays to store associated information

```
Scanner keys = new Scanner(System.in);
String[] friends = new String[5];
String[] colours = new String[5];
for (int count = 0; count < 5; count++) {
  System.out.println("Enter a friend's name");
  friends[count] = keys.nextLine();
  System.out.println("Enter their favourite colour");
  colours[count] = keys.nextLine();
}
```

Restrictions of arrays

- Most languages define arrays to be of specific and fixed length
  - declare myArray[6]
  - myArray[12] = "ooops"
  - would not work – the element 12 would be beyond the end of the array which is only six elements long
    - Q: What number would the *last* element be?
  - You would most likely get an error message along the lines "array element index out of range"
  - Plus, you can't make an array bigger once it's declared

Declaring an array in one hit

- If you know the values that the array needs to store at the outset you can use the short array notation

```
String[] friends = { "Fred","Carol","Nigel","Kevin","Bill" };
```

- You can also use variables (the values don't have to be constants), for example

```
String[] friends = { bestFriend, enemy, stranger, x, banana };
```

- (assuming of course that **`bestFriend`**, **`enemy`**, **`stranger`**, **`x`** and **`banana`** were variables of type **String**)

Java problem solving exercise

Problem solving example

- Write a program that asks me to enter the current balance on my credit card. Each month I pay back £100, but I am then charged 10% interest on the remaining balance.
- The program should tell me:
  - How long does it take to clear my credit card balance?
  - How much did I pay in total?

"be the computer…"

- Let's say we a balance of £500 to start with.
- Month one:
  - Starting Balance is £500
  - We pay back £100
    - Remaining balance is £400
    - But we are charged 10% interest
    - 10% of £400 is £40
    - So remaining balance after month one is £440

"be the computer…"

- Month two
  - Starting balance is £440
  - We pay back £100
    - Remaining balance is £340
    - But we are charged 10% interest
    - 10% of £340 is £34
    - So remaining balance after month two is £374

"be the computer…"

- Month three
  - Starting balance is £374
  - We pay back £100
    - Remaining balance is £274
    - But we are charged 10% interest
    - 10% of £274 is £27.40p
    - Remaining balance is £301.40p

"be the computer…"

- Month four
  - Starting balance is £301.40
  - We pay back £100
    - Remaining balance is £201.40
    - But we are charged 10% interest
    - 10% of £201.40p is £20.14p
    - Remaining balance is £221.54p

"be the computer…"

- Month five
  - Starting balance is £221.54
  - We pay back £100
    - Remaining balance is £121.54
    - But we are charged 10% interest
    - 10% of £121.54 is £12.15
    - Remaining balance is £133.69

"be the computer…"

- Month six
  - Starting balance is £133.69
  - We pay back £100
    - Remaining balance is £33.69
    - But we are charged 10% interest
    - 10% of £33.69 is £3.37
    - Remaining balance is £37.06

"be the computer…"

- Month seven
  - Starting balance is £37.06
  - We DON'T pay £100
    - We pay off the remaining balance of £37.06!
    - We celebrate our lack of debt! ☺

- What process did we follow to figure that out?
  - We calculated how much the balance would be with £100 taken off
  - We calculated what the interest would be on that remaining balance
  - We added the interest to the remaining balance
  - We repeated this until the balance remaining was less than £100

- How would we know how many months it took to pay the loan off?
- How might we calculate the total amount paid?

Convert to pseudocode

```
set months = 1
set balance = 500
set totalpaid = 0;
while balance > 100
  balance = balance - 100;
  set interest = balance * 0.019
  balance = balance + interest
  totalpaid = totalpaid + 100
  months = months + 1
endwhile
totalpaid = totalpaid+balance
display ”You paid ”+totalpaid
display ”It took you ”+months+” months”
```

Introducing the *algorithm*

- What we did here was to determine the *algorithm* for this problem by working through it step by step
  - An algorithm is a step by step procedure for performing a calculation or solving a problem
- Our pseudocode solutions can all be said to be algorithms in their own right – they are themselves step by step procedures
- Computer programs are simply implementations of algorithms stated in a programming language such as Java
- The design processes we've been encouraging you to use are simply aids to helping you formulate an algorithm
- Sometimes (as in this case) you know instinctively what the algorithm is – but it can also help to work through the problem by hand and observe *how* you're solving it

Convert to Java…

- That's a job for you  during your practical session…!

Summary

- The basic programming constructs for Java (i.e. **`if`**, **`for`**, **`while`**, **`do/while`**) follow C-like syntax
  - A code block in C-like syntax uses curly brackets { and } to mark the beginning and the end of the block
- In most cases, there is a one-to-one equivalent to the Banana code (or pseudocode) you'll have previously learned
  - The exception is **`do/while`** versus **repeat/until**
    - do/while continues WHILE the condition is true
    - repeat/until continued UNTIL the condition is true
      - You will therefore need to invert your condition when porting a pseudocode **`repeat/until`** to a Java **`do/while`**.

Summary

- If a variable is like a box, an array is a box with compartments
- Each compartment (or *element*) has a number
  - The first element is number 0
- Just as with any other variable, an array must be declared the first time it is used, e.g.

```
String[] names = new String[5];
```

  - Just as with any other variable in Java, when you declare an array you need to specify the data type
  - Put [] after the data type to indicate it's an array
  - Use the keyword new immediately after the equals to indicate it's a new object
  - Then specify the data type and the length of the array

Summary

- Access a specific array element by giving the array variable name and the element number in brackets:
  - `name[3] = "Paul";`
    - …would place the string *Paul* into the 4th array element (remember the first one is number zero!)
  - `System.out.println(name[2]);`
    - …would print the value of the 3rd array element

Summary

- Remember your grammar and that programming is like lego!
  - You can use a (numeric) variable instead of an integer literal to refer to an array element
    - `System.out.println(name[num]);`
      - In this case
        - name is an array
        - num is an integer variable
  - You can therefore use user input or other variables to access a specific element of an array
  - It also means you can use constructs like for loops to iterate through the values in an array

Summary

- The basic programming constructs for Java (i.e. **`if`**, **`for`**, **`while`**, **`do/while`**) follow C-like syntax
  - A code block in C-like syntax uses curly brackets { and } to mark the beginning and the end of the block
- In most cases, there is a one-to-one equivalent to the Banana code (or pseudocode) you'll have previously learned
  - The exception is **`do/while`** versus **repeat/until**
    - do/while continues WHILE the condition is true
    - repeat/until continued UNTIL the condition is true
      - You will therefore need to invert your condition when porting a pseudocode **`repeat/until`** to a Java **`do/while`**.