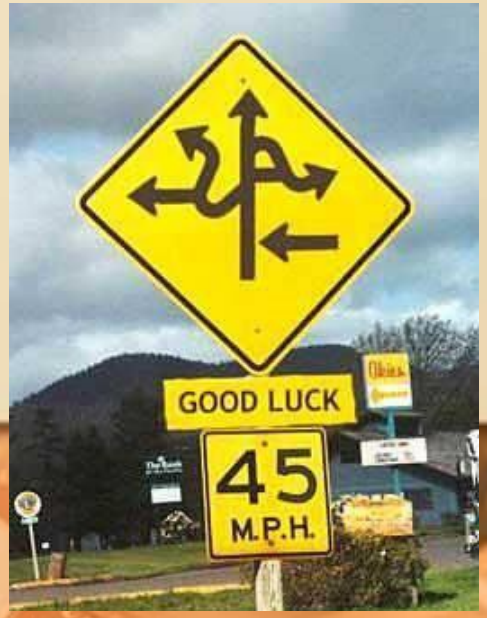
Lecture 2-1

Conditional and Iterative Processing

Part I



The big picture

any program you may want to write

objects

handling graphics, sound, and images

arrays

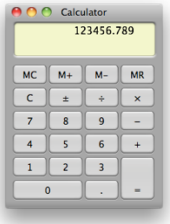
functions

conditionals and loops

this lecture

math | text I/O

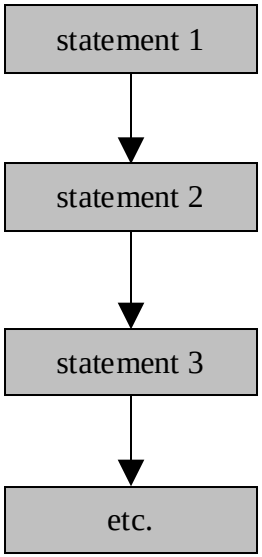
primitive data types | variables



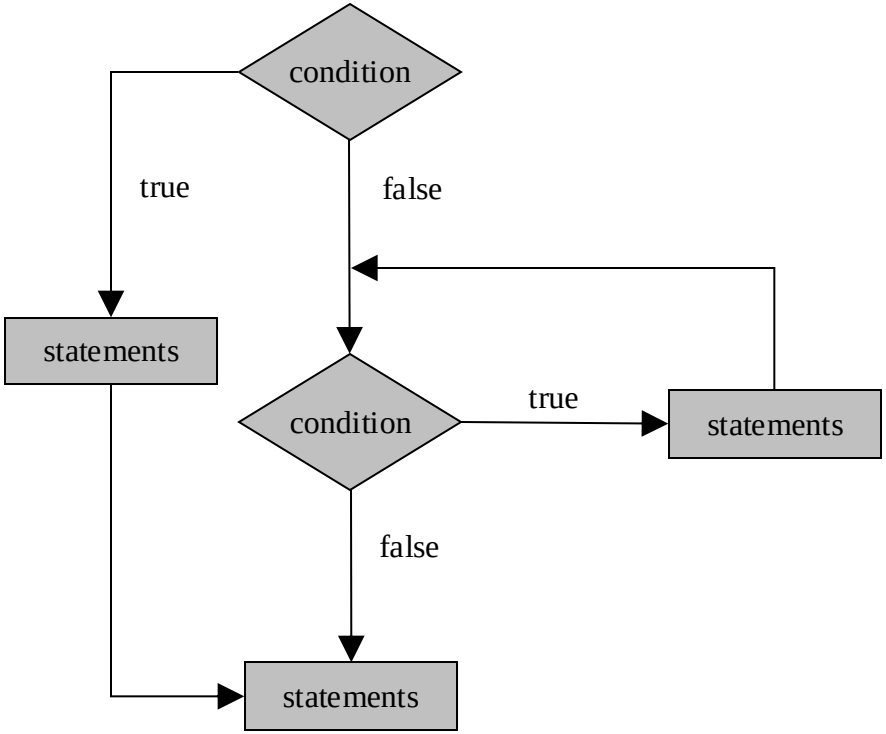
last week:
equivalent
to a fancy
calculator

Program flow

sequential flow:



conditional and iterative flow (example):



Program flow: *Actual* order of execution, during a program execution (runtime)

Conditionals and loops: Used to control the program flow.

Lecture plan

- Conditional logic:
  - `if`
  - `switch`
- Strings
- Iterative logic:
  - `while`
  - `for`
  - `do ... while`

IF example

```
public static void main(String args[]) {
    // computes and prints the absolute value of args[0]
    int x = Integer.parseInt(args[0]);
    if (x < 0) {
        x = -x;
    }
    System.out.println("Absolute value: " + x);
}
```

```
% java AbsValue 5
Absolute value: 5

% java AbsValue -12
Absolute value: 12
```

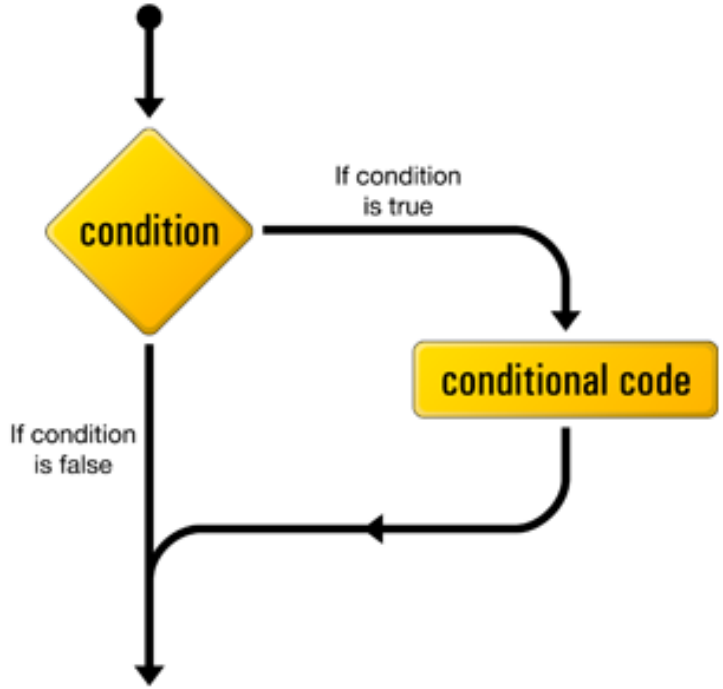
IF statement

Syntax:

Boolean expression;

```
if (condition) {
    // we get here if condition is true
    conditional code
}
// we get here always
code continues
```

One or more statements



(flow charts images are taken from http://articles.sitepoint.com/article/mysql-3-getting-started-php/3)

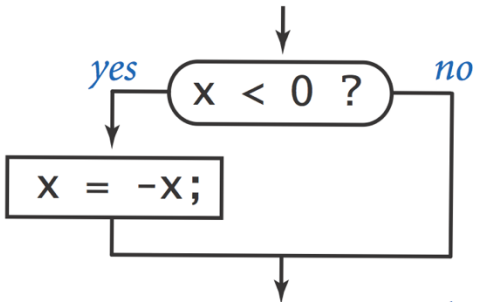
IF statement

```
// Sets x to abs(x)
```

```
if (x < 0) {
   x = -x;
}
```

```
if (x < 0)
   x = -x;
```

```
if (x < 0) x = -x;
```



Coding style:

- Different styles, same semantics
- Which style to use? The style that promotes *safety* and *readability*

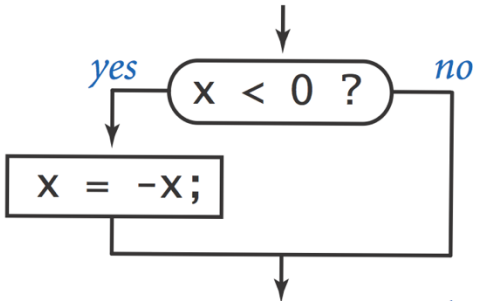
IF statement

```
// Sets x to abs(x)
```

```
if (x < 0) {
   x = -x;
}
```

```
if (x < 0)
   x = -x;
```

```
if (x < 0) x = -x;
```



Coding style:

- Different styles, same semantics
- Which style to use? The style that promotes *safety* and *readability*
- Convention: Use {*blocked code*} with braces,
  even when *blocked code* is only one statement
- Read our Java Code Style Guidelines.

IF … ELSE statement

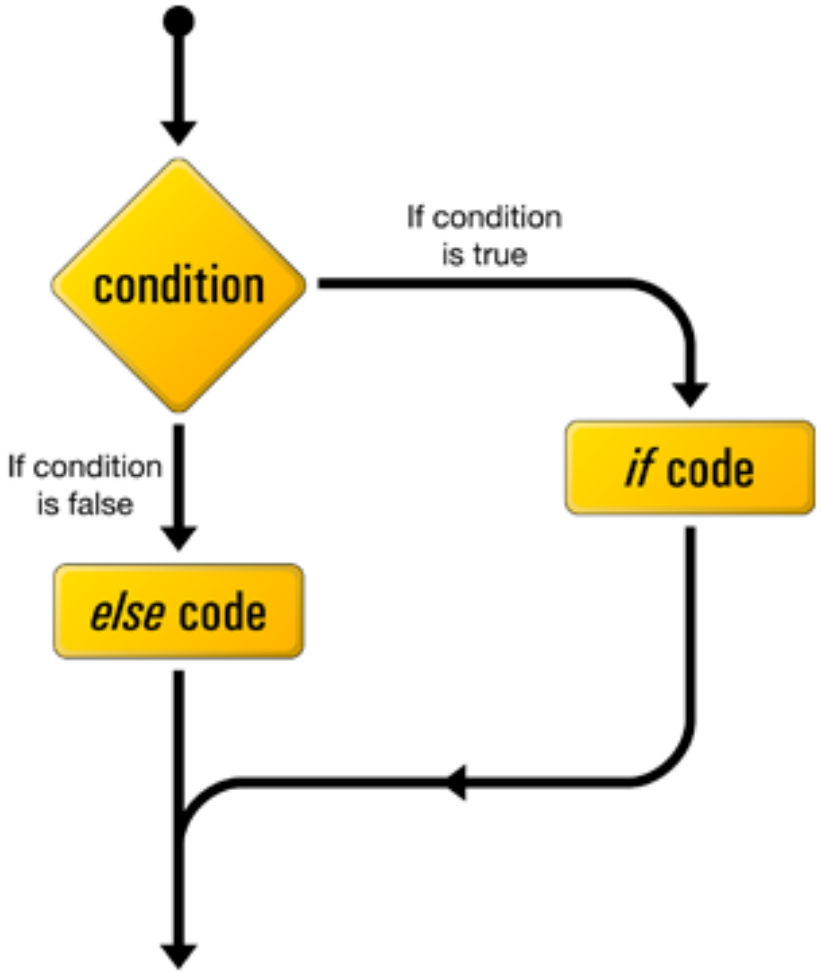
Syntax:

Boolean expression

```
if (condition)
    if-code
else
    else-code
```

One or more statements

One or more statements



IF … ELSE examples

```
public class Flip {
   public static void main(String[] args) {
      if (Math.random() < 0.5) {
         System.out.println("Heads");
      } else {
         System.out.println("Tails");
      }
   }
}
```

```
% java Flip
Heads

% java Flip
Heads

% java Flip
Tails

% java Flip
Heads
```

IF … ELSE examples

```
// Computes max(x,y)
int max;
if (x > y) {
   max = x;
} else {
   max = y;
}
```

```
// Computes max(x,y)
int max = x;
if (y > max) {
   max = y;
}
```

```
// Computes the roots of the quadratic equation x^2 + b*x + c = 0
double discriminant = b * b - 4.0 * c;
if (discriminant < 0.0) {
   System.out.println("no real roots");
} else {
   System.out.println("x1 = " + (-b + Math.sqrt(discriminant)) / 2.0);
   System.out.println("x2 = " + (-b - Math.sqrt(discriminant)) / 2.0);
}
```

Nested code

```
// Compares variables a and b
if (a == b) {
    System.out.println("a equals b");
} else {
    if (a > b) {
        System.out.println(a + " is greater");
    } else {
        System.out.println(b + " is greater");
   }
}
```

nested if

```
if (a == b) { System.out.println("a equals b"); }
if (a > b)  { System.out.println(a + " is greater"); }
if (a < b)  { System.out.println(b + " is greater"); }
```

What's the difference between the two code segments?

Example: tax calculation

Task: compute the applicable tax rate, according to income level.

```
if (income < 5280) {
   rate = 0.10;
} else {
  if (income < 9010) {
      rate = 0.14;
  } else {
    if (income < 14000) {
       rate = 0.21;
    } else {
      if (income < 20000) {
          rate = 0.31;
      } else {
          rate = 0.34;
      }
    }
  }
}
```

גובה ההכנסה	שיעור המס למדרגה
מ-0 ₪ ועד 5,280 ₪	10%
מ-5,281 ₪ ועד 9,010 ₪	14%
מ-9,011 ₪ ועד 14,000 ₪	21%
מ-14,001 ₪ ועד 20,000 ₪	31%
מ-20,001 ₪ ואילך	34%

Same results, different styles

```
if      (income < 5280)  rate = 0.10;
else if (income < 9010)  rate = 0.14;
else if (income < 14000) rate = 0.21;
else if (income < 20000) rate = 0.31;
else                     rate = 0.34;
```

The conditional operator: a shorthand if/else

Example

```
max = (a > b) ? a : b;
```

Equivalent to:

```
if (a > b) max = a; else max = b;
```

Syntax

condition ? *expression*1 : *expression*2 ;

If the ***condition*** is true,
the expression evaluates to ***expression1***;
else,
the expression evaluates to ***expression2***

Example

```
System.out.println("Thanks! you bought " + qty + ((qty > 1) ? "items" : "item"));
```

- The conditional (also called *ternary if*) is similar to an if / else statement
- But, the conditional is not a statement; it's an *expression* that returns a value
- In some cases, makes the code more readable; in other cases, more obscure
- Use your judgement.

Lecture plan

- Conditional logic:

  ✓ `if`

  `Switch` (will be covered in the recitation)

➡ Strings

- Iterative logic:

  `While`

  `For`

  `do ... while`

Data types (revisited)

Primitive types

type	set of values	example values	typical operations
`int`	integer numbers	`17` `-5034`	add, subtract, multiply, divide
`double`	floating point numbers	`3.1415` `-171.19`	add, subtract, multiply, divide
`boolean`	truth values	`true` `false`	and, or, not
`char`	characters	`'c' 'K' '?' '5' '+'`	compare
`...`			

- Represent "scalar" / single values
- The Java language features eight primitive data types
- Built-into the language

Object types

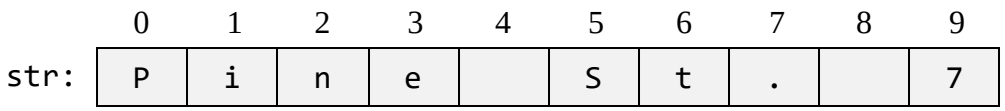
`String`	sequences of characters	`"xckd"` `"hello world"`	concatenate
`Date`	dates	`03/11/2018`	equals, greater than, less than
`...`			

- Represent "aggregates" of values
- Java libraries feature many object types
- More object types can be created as needed

Strings

Java program

```
...
// Creates and initializes a variable of type String
String str = "Pine St. 7";   // Street address
...
```



Abstraction

Implementation

Memory

	...	
2017	80	'P'
2018	105	'i'
2019	110	'n'
2020	101	'e'
2021	32	' '
2022	83	'S'
2023	116	't'
2024	46	'.'
2025	32	' '
2026	55	'7'
	...	

(the addresses are an arbitrary example)

A `String` is an indexed sequence of *characters* (`char` values), each being a nonnegative integer in the range 0 to 65535.

String operations

Examples

```
// Examples of string processing operations:
// (will be explained more fully later in the course)
String s = "Herzliya";
System.out.println(s.charAt(0)); // 'H'
System.out.println(s.charAt(1)); // 'e'
System.out.println(s.charAt(7)); // 'a'
System.out.println(s.length()); // 8
System.out.println(s.charAt(s.length() - 1)); // 'a'
System.out.println(s.charAt(s.length())); // error
...
```

A sequence of 8 `char` values, indexed 0, 1, 2, ..., 7

`s.length()`: A function call, returns the length of `s`

`s.charAt(i)`: A function call, returns the character at index `i` in `s`

(We'll have more to say about the `char` and `String` types later in the course)

Lecture plan

- Conditional logic:
  - `if`
  - `switch`
- Strings (Intro, more later)
- Iterative logic:
  - `While`
  - `For`
  - `do ... while`

WHILE loop

Example

```
// Prints 0 to N-1
int count = 0;
while (count < N) {
    System.out.println(count);
    count = count + 1;
}
```

What will the program print when . . .

N = 3 ?

N = 1 ?

N = 0 ?

N = -3 ?

N = 1000 ?

N	COUNT	COUNT<N	PRINT
3	0	TRUE	0
3	1	TRUE	1
3	2	TRUE	2
3	3	FALSE	

Trace table:

An important debugging aide

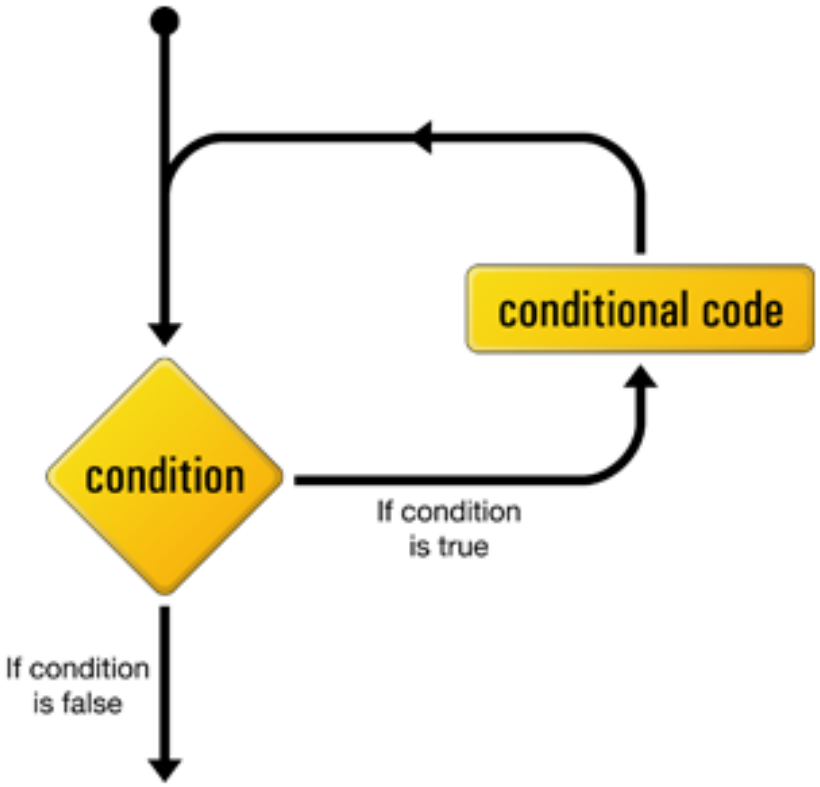
Important:

Always think about, and test, "edge cases"

WHILE loop

Example

```
// Prints 0 to N-1
int count = 0;
while (count < N) {
    System.out.println(count);
    count = count + 1;
}
```



Boolean expression

Syntax:

```
while (condition) {
    conditional code
}
```

One or more statements

Note

A while loop executes zero or more times.

Example: String processing

```
// Deletes all the vowels (a, e, i, o, u) from a given string
public class DeleteVowels {
    public static void main(String[] args) {
        String s = args[0];
        String sOut = "";
        String vowels = "aeiou";
        int i = 0;
        while (i < s.length()) {
            char c = s.charAt(i);
            if (vowels.indexOf(c) == -1) {
                sOut = sOut + c;
            }
            i++; // i = i + 1;
        }
        System.out.println(sOut);
    }
}
```

```
% java DeleteVowels router
rtr

% java DeleteVowels sync
sync

% java DeleteVowels example
xmpl
```

Algorithm

`sIn` = the input string

`sOut` = an empty, output string

`vowels` = `"aeiou"`

for each character in `sIn`:

if the character is not in `vowels`, add it to `sOut`

`s.length()`: Returns the length of `s`

`s.charAt(`*int*`)`: Returns the character at index *int* in `s`

`s.indexOf(`*char*`)`: Returns the first index at which *char* appears in `s`, or `-1` if not found

Example: String processing

```
// Deletes all the vowels (a, e, i, o, u) from a given string
public class DeleteVowels {
    public static void main(String[] args) {
        String s = args[0];
        String sOut = "";
        String vowels = "aeiou";
        int i = 0;
        while (i < s.length()) {
            char c = s.charAt(i);
            if (vowels.indexOf(c) == -1) {
                sOut = sOut + c;
            }
            i++; // i = i + 1;
        }
        System.out.println(sOut);
    }
}
```

```
% java DeleteVowels router
rtr

% java DeleteVowels sync
sync

% java DeleteVowels example
xmpl
```

Algorithm

`sIn` = the input string

`sOut` = an empty, output string

`vowels` = `"aeiou"`

for each character in `sIn`:

if the character is not in `vowels`, add it to `sOut`

Note

In this program we use the **+** operator (concatenation) to build the string incrementally;

There are more efficient ways to build strings incrementally (later).

Example: Powers of 2

Task: Print powers of 2: $2^0, 2^1, 2^2, 2^3, \ldots, 2^N$

```
public class PowersOfTwo {
  public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      int i = 0;
      int v = 1;
      while (i <= N) {
        System.out.println(i + " " + v);
        i++;     // i = i + 1;
        v *= 2;  // v = v * 2;
      }
  }
}
```

```
% java PowersOfTwo 3
0 1
1 2
2 4
3 8

% java PowersOfTwo 6
0 1
1 2
2 4
3 8
4 16
5 32
6 64
```

Algorithm

N = *input*
i = 0
v = 1
while $i \leq N$:
    print (*i v*)
    increment *i*
    double *v*

Example: Powers of 2 / buggy version

Task: Print powers of 2: $2^0, 2^1, 2^2, 2^3, \ldots, 2^N$

```
public class PowersOfTwo {
  public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      int i = 0;
      int v = 1;

      while (i <= N)
        System.out.println(i + " " + v);
        i++;     // i = i + 1;
        v *= 2;  // v = v * 2;

  }
}
```

```
% java PowersOfTwo 6
0 1
0 1
0 1
0 1
0 1
0 1
0 1
0 1
0 1
0 1
0 1
...
```

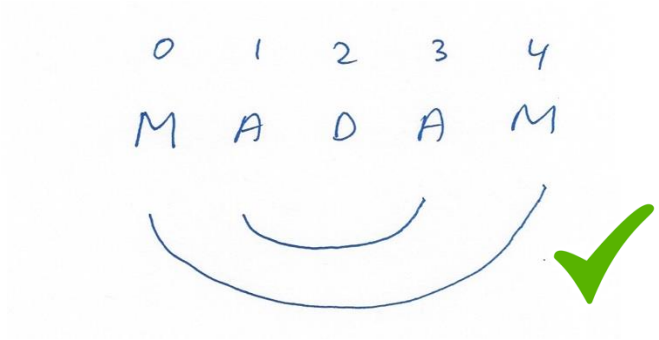
Bug: The curly braces around the body of the loop are missing.

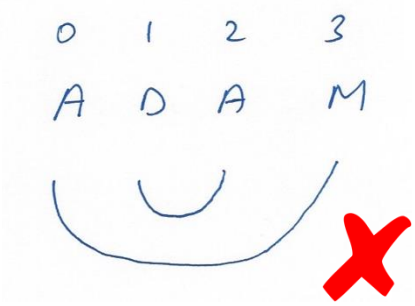
Example: Palindromes

- radar
- kayak
- racecar
- drab bard
- Rise to vote, sir
- Dennis and Edna sinned
- Doom an evil deed, liven a mood
- Go hang a salami; I'm a lasagna hog
- A man, a plan, a canal, Panama

Rules of the game:

- Ignore: white space, upper/lower case, punctuation characters
- A single character string is a palindrome
- An empty string is a palindrome.

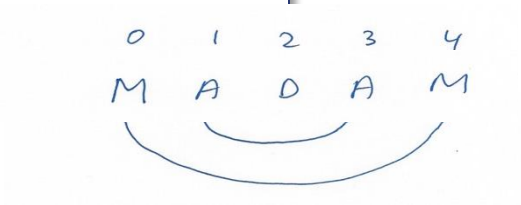




Example: Palindromes

```
public class Palindrome {
  public static void main(String[] args) {
      String s = args[0];
      int left = 0;
      int right = s.length() - 1;
      // sanity check: Makes sure that we have a good start
      System.out.println(s);
      System.out.println(left);
      System.out.println(right);
}
```

```
% java Palindrome adam
adam
0
3
```



Relevant string processing functions

`s.length()`: Returns the length of `s`

`s.charAt(`*int*`)`: Returns the character at index *int* in `s`

`s.indexOf(`*char*`)`: Returns the first index at which *char* appears in `s`, or `-1` if not found

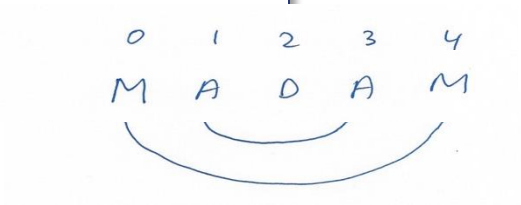
Example: Palindromes

```
public class Palindrome {
  public static void main(String[] args) {
      String s = args[0];
      int left = 0;
      int right = s.length() - 1;
      while ((s.charAt(left) == s.charAt(right)) && (left < right)) {
         left++;
         right--;
      }

      if (left < right) {
          System.out.println(s + " is not a palindrome");
      } else {
          System.out.println(s + " is a palindrome");
      }
   }
}
```

```
% java Palindrome adam
adam is not a palindrome

% java Palindrome madam
madam is a palindrome
```



Relevant string processing functions

`s.length()`: Returns the length of `s`

`s.charAt(`*int*`)`: Returns the character at index *int* in `s`

`s.indexOf(`*char*`)`: Returns the first index at which *char* appears in `s`, or `-1` if not found

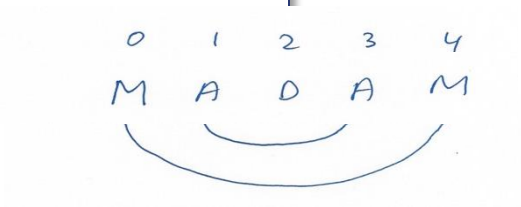
Example: Palindromes

```
% java Palindrome adam
adam is not a palindrome

% java Palindrome madam
madam is a palindrome
```

```
public class Palindrome {
  public static void main(String[] args) {
      String s = args[0];
      int left = 0;
      int right = s.length() - 1;
      while ((s.charAt(left) == s.charAt(right)) && (left < right)) {
         left++;
         right--;
      }

      if (left < right) {
          System.out.println(s + " is not a palindrome");
      } else {
          System.out.println(s + " is a palindrome");
      }
   }
}
```



MADAM:

LEFT	RIGHT	=	LEFT < RIGHT
0	4	T	T
1	3	T	T
2	2	T	F

ADAM:

LEFT	RIGHT	=	LEFT < RIGHT
0	3	F	T

Example: Print square

```
% java PrintSquare 3
***
***
***
```

```
public class PrintSquare {
  public static void main(String[] args) {
      int n = Integer.parseInt(args[0]);

      int i = 0;
      // Iterates through square rows
      while (i < n) {
          // Iterates through square columns
          int j = 0;
          while (j < n) {
               System.out.print("*");
               j++;
          }
          System.out.println();
          i++;
      }
   }
}
```

← nested while

Relevant printing functions

`System.out.print(`*str*`):`

Prints the string *str* at the *cursor position*,
and moves the cursor just after *str*

`System.out.println(`*str*`):`

Prints *str* at the cursor position,
and moves the cursor to the beginning on the next line

`System.out.println():`

Moves the cursor to the beginning on the next line

Self practice

Write these programs:

```
% java PrintSquare 3
***
***
***
```

(done)

```
% java PrintLeft 3
*
**
***
```

```
% java PrintRight 3
   *
  **
 ***
```

```
% java PrintCentered 3
  *
 ***
*****
```

Relevant printing functions

`System.out.print(`*str*`):`

Prints the string *str* at the *cursor position*,
and moves the cursor just after *str*

`System.out.println(`*str*`):`

Prints *str* at the cursor position,
and moves the cursor to the beginning on the next line

`System.out.println():`

Moves the cursor to the beginning on the next line

Lecture plan

- Conditional logic:
  - ✓ `if`
  - `Switch` (will be covered in the recitation)
- ✓ Strings (Intro, more later)
- Iterative logic:
  - ✓ `While`
  - `For`
  - `do ... while`