#### Week 3

Methods and Strings

#### This week...

- Methods: functions vs procedures
- String functions
- Process: break it down, build it up.

## Methods

#### Methods

We consider two kinds of methods:

- 1. A **procedure** does something. It's name is a verb.
- 2. A **function** returns something. It's name is a noun.

#### **Procedures**

- A procedure is a method that does an action / has some "effect".
   e.g. prints a value, changes a value
- A procedure may take parameters, but should return nothing.
- The name of a procedure is a verb describing the goal.

```
public static void showCircleArea(double radius) {
    double area = Math.PI * radius * radius;
    System.out.println("The area of the circle is " + area);
}
```

 A procedure may use local variables. A local variable is temporary. It is deleted when the method exits.

#### **Functions**

- A function is a method that returns a value.
- A function should **not** have any side effects.
   e.g. It should **not** print a value. It should **not** change a value.
- A function may take parameters.
- The name of a function is a noun describing what is returned.

```
public static double circleArea(double radius) {
    double area = Math.PI * radius * radius;
    return area;
}
```

A function may also use local variables.

#### Side effects

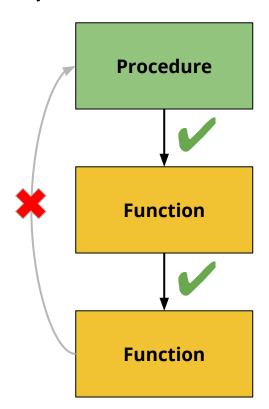
- Function design rule
   A function returns a value and changes nothing
- If a function changes something, this is called a "side effect"
- Side effects are bad:
  - the reader assumes the function changes nothing
  - the reader does not look inside the function
  - because a function changes nothing
- Avoid programming by side effect.
   Unless it is a known pattern.

#### Interaction between procedures and functions

- A procedure can call a function.
- A function can call a function.
- But a function should not call a procedure

Functions should not have side effects

Calling a procedure may introduce side effects



# String functions

#### Strings

In java, String is a class, providing a set of useful functions.

int length()	returns the length of the string
char charAt(int i)	returns the character at position i
String[] split(String separator)	returns an array of substrings split by the separator

#### e.g.

#### The "string loop" pattern

**Goal**: Loop over the characters in a string.

#### **Example:**

Count the number of l's in the word "hello".

```
String s = "hello";
int count = 0;
for (int i = 0; i < s.length(); i++) {
   if (s.charAt(i) == 'l')
        count++;
}
System.out.println("Number of l's = " + count);</pre>
```

#### The "for-each" loop

```
Create an array of values
    String[] array = { "car", "truck", "bus", "van" };
These two code fragments do the same thing:
    for (int i = 0; i < array.length; i++)
                                                      -Array loop
         System.out.println(array[i]);
    for (String word : array)
                                                       -For-each loop
         System.out.println(word);
```

**Read**: For each word in array, print that word.

#### Looping over words in a string using "split"

#### **Program**:

#### Output:

```
Next word = Eat
Next word = your
Next word = vegetables
```

#### Split by one or more spaces

If you have a string with extra spaces between words:

```
String sentence = "Eat your vegetables";
```

Use the regular expression " +" as the separator

```
for (String word : sentence.split(" +"))
    System.out.println("Next word = " + word);
```

#### **Output**:

```
Next word = Eat
Next word = your
Next word = vegetables
```

#### More String functions...

boolean contains(String s)	does this string contain s?	
int indexOf(String s)	returns the position where s is found	
boolean startsWith(String s)	does this string start with s?	
boolean endsWith(String s)	does this string end with s?	
boolean equals(String s)	does this string equal s?	
String substring(start, end)	returns a substring from start to end	
String toUpperCase() / String toLowerCase()	returns the string in upper/lower case	
String trim()	returns the string without leading/trailing spaces.	

For more, see the documentation: https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

# Functional patterns

#### When to use a function

If it produces a value, make it a function.

#### Read functions

The read pattern returns a value, so it is a function.
 The name has the form read<X>

```
int readAge() {
    System.out.print("Age: ");
    return In.nextInt();
}
String readName() {
    System.out.print("Name: ");
    return In.nextLine();
}
```

#### The "old" read loop pattern

**Specification**: Read and ages until the user enters -1.

**Problem**: There is repeated code.

Don't repeat code. Put it in a method.

#### Read loop with read functions

```
int age = readAge();
while (age !=-1) {
    <use age>
    age = readAge();
int readAge() {
    System.out.print("Age: ");
    return In.nextInt();
```

**Problem**: There is still repeated code: age = readAge();

#### Merged read loop

**Key**: call readAge() inside the while condition.

Exercise: can you make this into a pattern?

#### Merged read loop

- Whenever you need a read loop, always use the merged read loop.
- Example: reading characters:

Example: reading strings:

#### The "any" pattern

**Goal**: Determine if any item in a collection passes <test>

```
<for each item>
    if (<item passes test>)
        return true;
return false;
```

**Example**: Test if any number in an array is negative:

```
boolean anyNegative(int[] array) {
    for (int item : array)

    if (item < 0)
        return true;
    return false;
}</pre>
Key idea:

If any item is negative
    return true.
```

# Homework: Two variations of the "any" pattern for you to work out...

#### 1. The "every" pattern

**Goal**: Determine if all items in a collection pass <test>

```
<for each item>
    if (! <item passes test>)
        return false;
return true;
```

#### 2. The "none" pattern

**Goal**: Determine if no items in a collection pass <test>

```
<for each item>
    if (<item passes test>)
        return false;
return true;
```

# Boolean functions

#### Bad boolean tests

- Bad: if (matches == true)
- Good: if (matches)
  - There is no need to compare a boolean to true or false
  - A boolean **is** true or false

- Bad: if (c == 'x' | | 'y' | | 'z')
- Good: if (c == 'x' || c == 'y' || c == 'z')
  - o Each OR component must be a full boolean expression

← won't compile

#### Boolean functions

A boolean function returns a boolean value:

```
boolean isDry(int rain) {
   return rain == 0;
}
```

• The name of a boolean function is an adjectival phrase:

```
boolean dry(int rain)
boolean isDry(int rain)
boolean hasDry()
```

#### Bad boolean functions

• Bad:

```
boolean isDry(int rain) {
    if (rain == 0)
        return true;
    else
        return false;
}
Good:
boolean isDry(int rain) {
    return rain == 0;
```

No need to test if (rain == 0). It is a boolean. Just return it.

# Process: Break it down, build it up.

#### Break down a program into functions

**Specification**: Read in a sentence. Show the number of words that contain a lowercase vowel.

**Remember**: Each goal goes in a separate method.

### Levels of processing

Read a sentence. Show the number of words that contain a lowercase vowel.

Level	Goal	Pattern
Sentence level	Goal: How many matching words in this sentence?	count
Word level	Goal: Are there any lowercase vowels in this word?	any
Character level	Goal: Is this character any of these: a/e/i/o/u?	any

#### What is the form of each method?

Read a sentence. Show the number of words that contain a lowercase vowel.

Level Goal Form Goal: How many matching words param: String sentence Sentence level in this sentence? result: int Goal: Are there any lowercase param: String word Word level vowels in this word? result boolean Goal: Is this character any of param: char c Character level result boolean these: a/e/i/o/u?

#### DEM<sub>0</sub>

#### DEM<sub>0</sub>

#### DEM<sub>0</sub>

#### A complete program still needs procedures!

- Functions don't have any "effect".
- To cause something to "happen" we need procedures.
- e.g. "show" the number of matching words in the terminal:

```
public static void showMatchingWords (String sentence) {
    System.out.println("Matching words = " + matchingWords (sentence));
}
```

Every program must have a main also have a main method:

#### Interaction between procedures and functions

- The functions do all the grunt work
  - split a sentence into words
  - count the vowel words
  - test if a word contains a vowel
  - test if a character is a vowel
- The procedures just present the result of that hard work.

