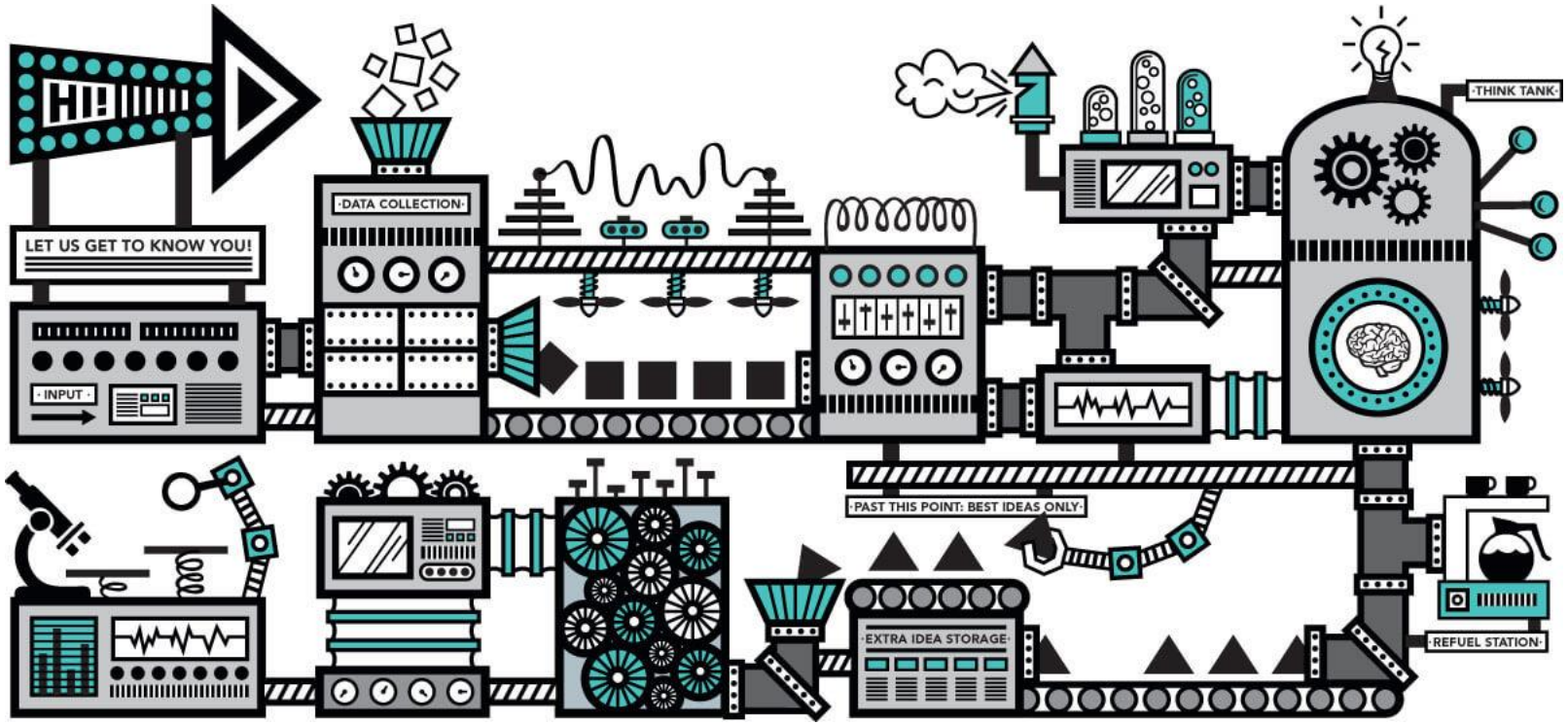
Defining Methods Part I

Sect. 3.3, 8.2

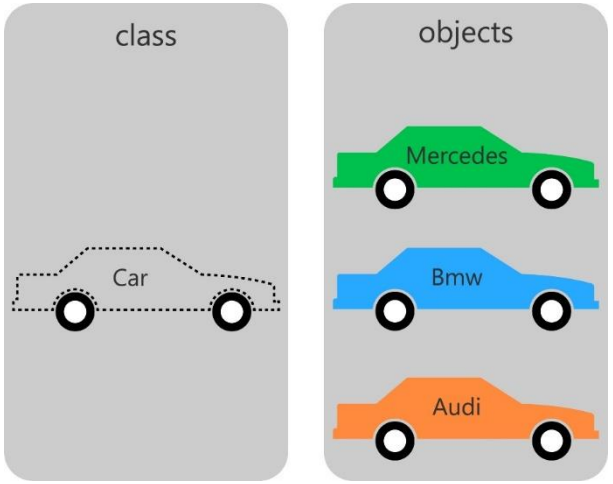
There's a method in my madness.



Example Class: Car

How Cars are Described

- Make
- Model
- Year
- Color
- Owner
- Location
- Mileage

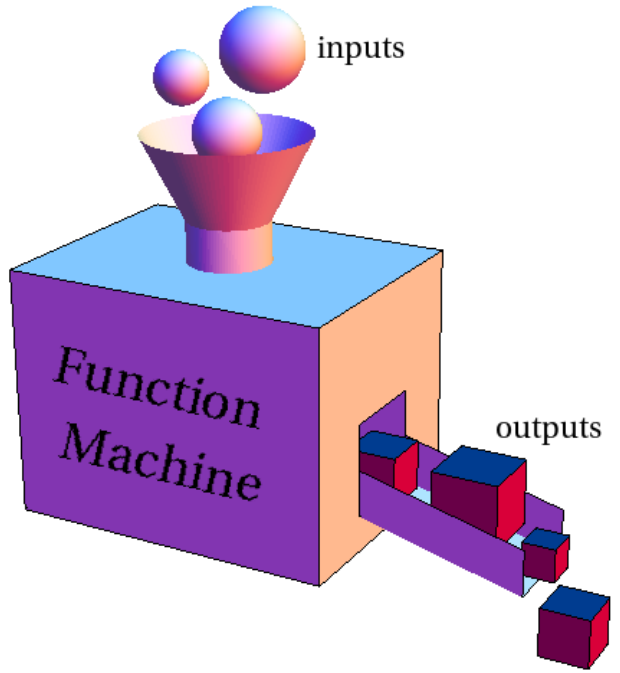


Actions that can be applied to cars

- Create a new car
- Transfer ownership
- Move to a new location
- Repaint
- Delete a car

Method Invocation

- Much more about method invocation in the next lecture
- For now, think of method invocation like a function machine…
  - You need to provide inputs to the method
  - You need to start the method
  - The method will produce outputs
- This lecture looks inside the method



Method Definition Syntax

Sect. 3.2, 3.3

```
class classname {
        modifiers returnType method_name (parameters) throws {
                body
        }
}
```

modifiers : access: public private protected
also: static abstract final synchronize native strictfp

returnType : Any (primitive or reference so far) type or void

method_name : Any valid identifier (starts w/ lower case by convention)

parameters : Next page or so.

throws : later… ignore for now

body : Java instructions to perform the action

Example Method Headers

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

int getIndex(int n) { …

public void println(String x) { …

public GridLayout(int rows, int cols, int hgap, int vgap) {…

public static int sum(int… numbers) { …
```

Modifiers

Return Type

name

Parameters

Method Modifiers

- Access Modifiers:
  - public – method can be used by all java code
  - protected – method is visible in super-classes* and in the package
  - package-private – method is visible only to java code in the package
    - This is the default if nothing else is specified
  - private – method is visible only within the class
- static vs. dynamic : Later this lecture
- abstract, final, …. we'll talk about later

* *We'll talk about super-classes later*

Return Type

- Type of data which the method returns to the caller
  - The actual value returned will be specified in the body of the method
- Required for all methods!
  - Use void to indicate method does not return anything
- The only data outside the method that the method can change EXCEPT for any objects referenced in the parameters

Parameters

- Comma separated list of things that look like variable declarations
  - Inside the method, treated like a variable
- Each parameter is a positional place-holders to hold copies of the input values
  - When a method is invoked, the invocation contains one argument for each parameter… that becomes the initial value of the parameter
- The **last** parameter in the list may specify a variable length list of values. If so, it must be specified as

  type **...** *name*

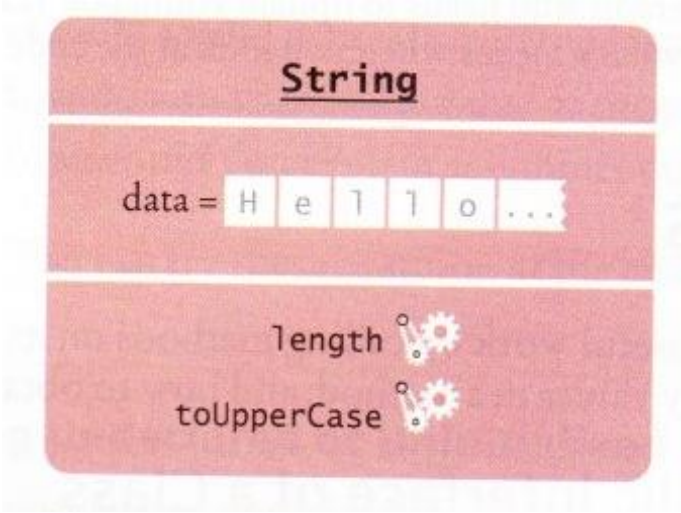
  where *name* is interpreted as an array within the method.

Method Body

- The instructions used to perform the desired action(s)
- This is that actual Java code!
- Java instructions are primarily:
  - Local declares
  - Assignment statements with expressions
  - Control Flow
- This course assumes you will learn the details of these on your own (They are similar to other languages.)

Note: No Code in Objects

- In section 2.3 of the text, there are diagrams like:
- Despite these pictures, there is no actual CODE stored in the object!
- The object ONLY contains the values of the fields for that object
- The code is stored in memory only once for the entire class



Local Variables

- Any value used temporarily within a method, like a counter for a loop, is kept in a ***local variable***
- Local variables are not available outside the method. (local scope)
- Local Variable Syntax:

```
class classname {
        modifiers returnType method_name (parameters) {
                type variable_name;
                type variable_name = initial_value;
        }
}
```

Declaring Variables

Sect. 2.2

- Variables must be ***declared*** and ***initialized*** before they are used
  - Java is an imperative language, and this is typical
  - Enables compiler to check to make sure variables are used correctly
  - Declaration and initialization can be combined

- Declaration
  - specifies "type" – how this variable will be used and what values are valid
  - specifies "name" – how you will refer to this value from now on
- Initialization
  - specifies the first value that the variable will have.

Local Variables Schematically

```
double precipAug = 3.59;
```