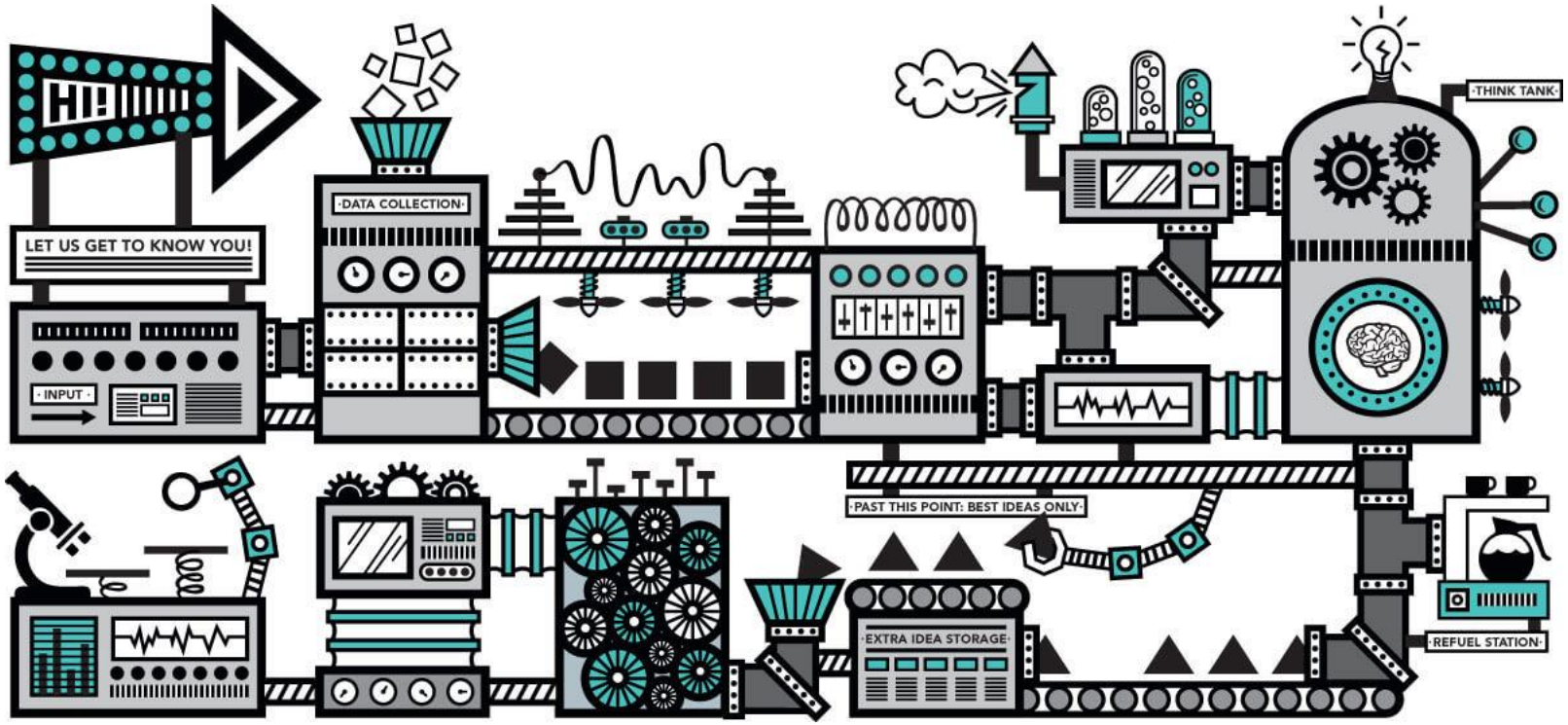
Invoking Methods

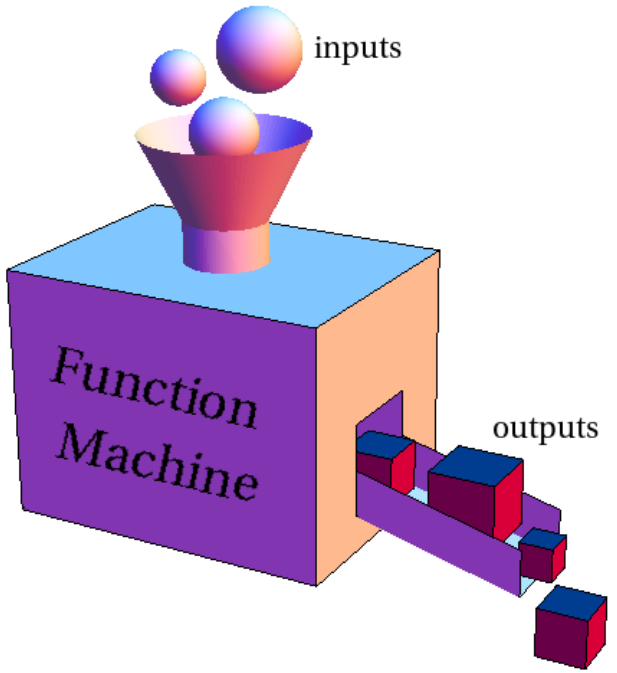
Sect. 3.3, 8.2

There's a method in my madness.



Method Invocation

- Much more about method invocation in the this 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 at the outside of the machine.



Method Invocation Terminology

- **Invocation** : Running a method one time with specific argument values

- **Arguments** : The values (result of expressions) passed in to a method invocation
- **Parameters**: The "variables" initialized with the argument values inside of the method.

- **Return Type** : The data type of the output value from the method
- **Return Value** : The value created by a specific invocation of a method

Invoking "main"

- Run the Java Virtual Machine (JVM) by executing the "java" command.
  - The first argument is the name of a package/class that has a public static "main" method
  - The rest of the arguments become an argument to "main" – an array of blank-delimited strings
  - The JVM loads the package/class, then invokes "main" for us
  - Since "main" is static, it has no receiver object
- The return from main is "void" – no return values
  - Errors are handled using "exceptions" instead of return codes.

Invoking a Method (theory)

- To invoke a non-static method, you need to start with an object
- That object will "invoke" (execute) a specific method (code)
- The methods are defined in the class…
  - The list of actions that can be performed on any object in this class
- When you invoke a method, you are telling Java to perform a specific action on a specific object, the receiver of that action
- You may want to specify extra information (parameters) that control how that action is performed.
  - e.g. I want to move the soccer ball 10 meters to the right.

Instance Method Invocation Syntax

Sect. 2.3

receiver.method (arguments)

receiver: A reference to the receiver object

method : The name of a non-static method defined in the class of the receiver

arguments : a list of expressions that evaluate to the parameter values

Note: Parenthesis required, even when the method has no parameters

```
box.trans(4,5);
```

Reference to an object in the Rectangle class

Action: Method from the Rectangle class

Argument to be passed to the trans method

Arguments

- The positional list of values or references passed in to a method (to be kept in a parameter)
- An argument may be an expression that evaluates to a value or a reference

```
System.out.println("Hello");

… myList.getIndex(i+4);

… panel.setLayout(new GridLayout(2, 2, 5, 5));
```

Method Invocation in an Expression

- When a method is invoked in an expression…
  - The argument expression(s) are evaluated
  - The method is invoked with the argument values
  - The method returns a return value
- …the invocation is “replaced” by the return value
  - Special case for “void”

```
int boxArea = box.area();
System.out.println(“My box has area “ + boxArea);
```

Class Method Invocation Syntax

Sect. 2.3

class.method (arguments)

class: A Java class

method: The name of a static method defined in the class of the receiver

arguments : a list of expressions that evaluate to the parameter values

Note: Parenthesis required, even when the method has no parameters

```
Rectangle.strans(box,3,4);
```

Name of a class

Static Method from the Rectangle class

Argument to be passed to the strans method

Method Invocation Internals

When a method is invoked, the JVM:

1. Creates an “activation record” to keep track of the current invocation
2. Puts the caller's return address in the activation record
3. Evaluates argument expressions, and saves the results in the activation record (including receiver “this”)
4. Reserves space in the activation record for local variables
5. Executes the method
6. When a return statement is encountered, saves the return value
7. Goes back to the return address – the caller of the method
8. The caller uses the return value to replace the invocation
9. ‘Deletes’ the activation record

Activation Record Implications

- Parameters are copied, not passed by reference
  - However, if a reference is copied, you can change the referenced object!
- New local variables with each invocation
- No access to local variables before or after invocation
  - But objects referenced can be accessed after invocation as long as you still have a handle.