OBJECT

You keep using that word.
I do not think it means
what you think it means.

Exercise code at

https://github.com/dastels/oo-session-code

Simula67

- for writing simulations
- In Norway by Ole-Johan Dahl and Kristen Nygaard
- Superset of ALGOL60
- objects, classes, inheritance, subclasses,virtual methods, and garbage collection
- The first appearance of OO concepts. Some argue it is the first OOPL

Alan Kay

- Alan Kay
- Inspired by cellular biology, networking, & Simula67
- “I thought of objects being like biological cells and/or individual computers on a network, only able to communicate with messages”
- Cells -> objects

Smalltalk

- Developed by Kay & team at Xerox PARC in the 70s
- Released as Smalltalk-80
- A pure OO language, library, runtime, tools, environment
- First true OO system
- All OO languages/systems since are heavily influenced by Smalltalk

All Objects, All the Time

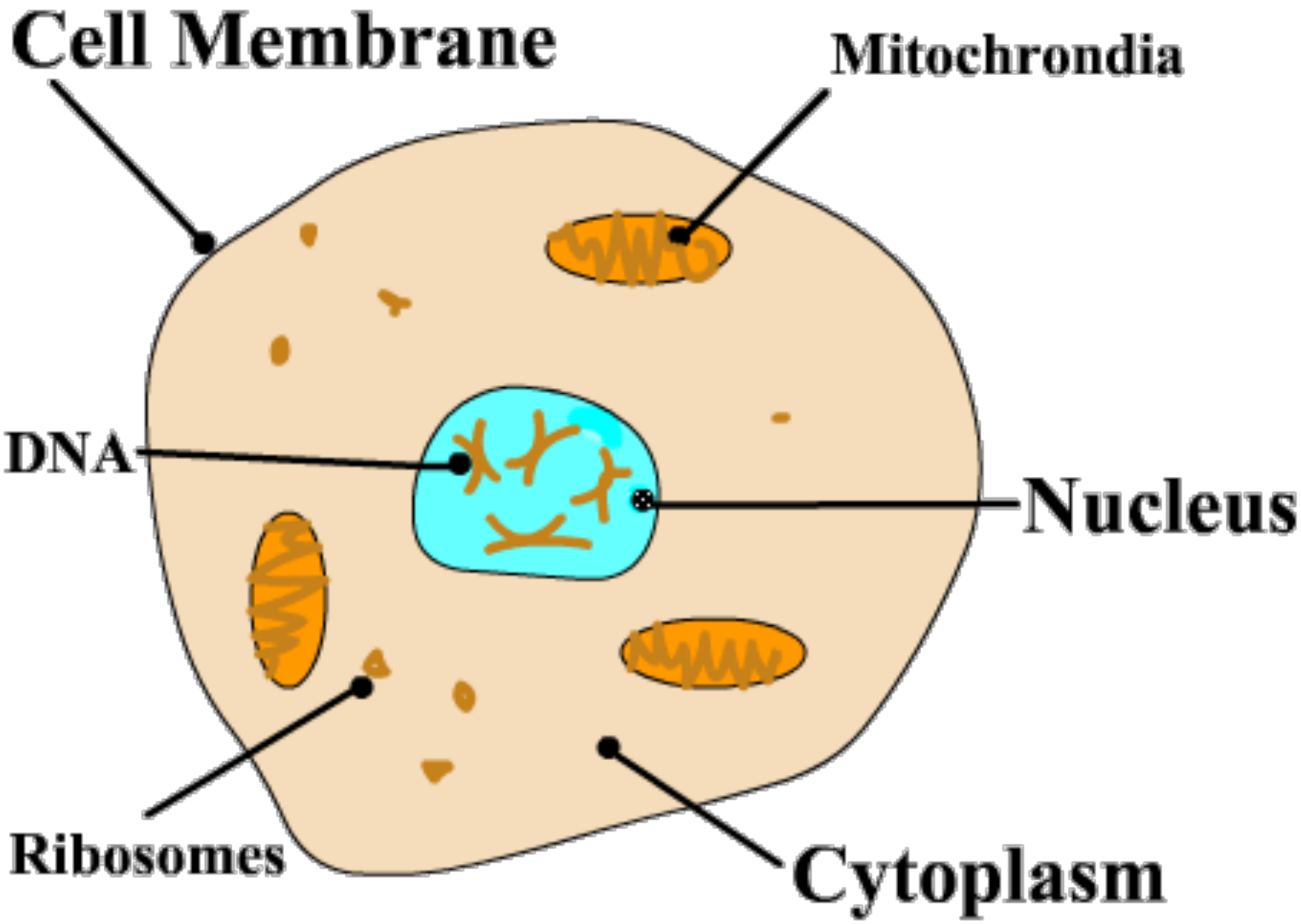
- The Smalltalk environment/image is a collection of communicating, interacting objects.

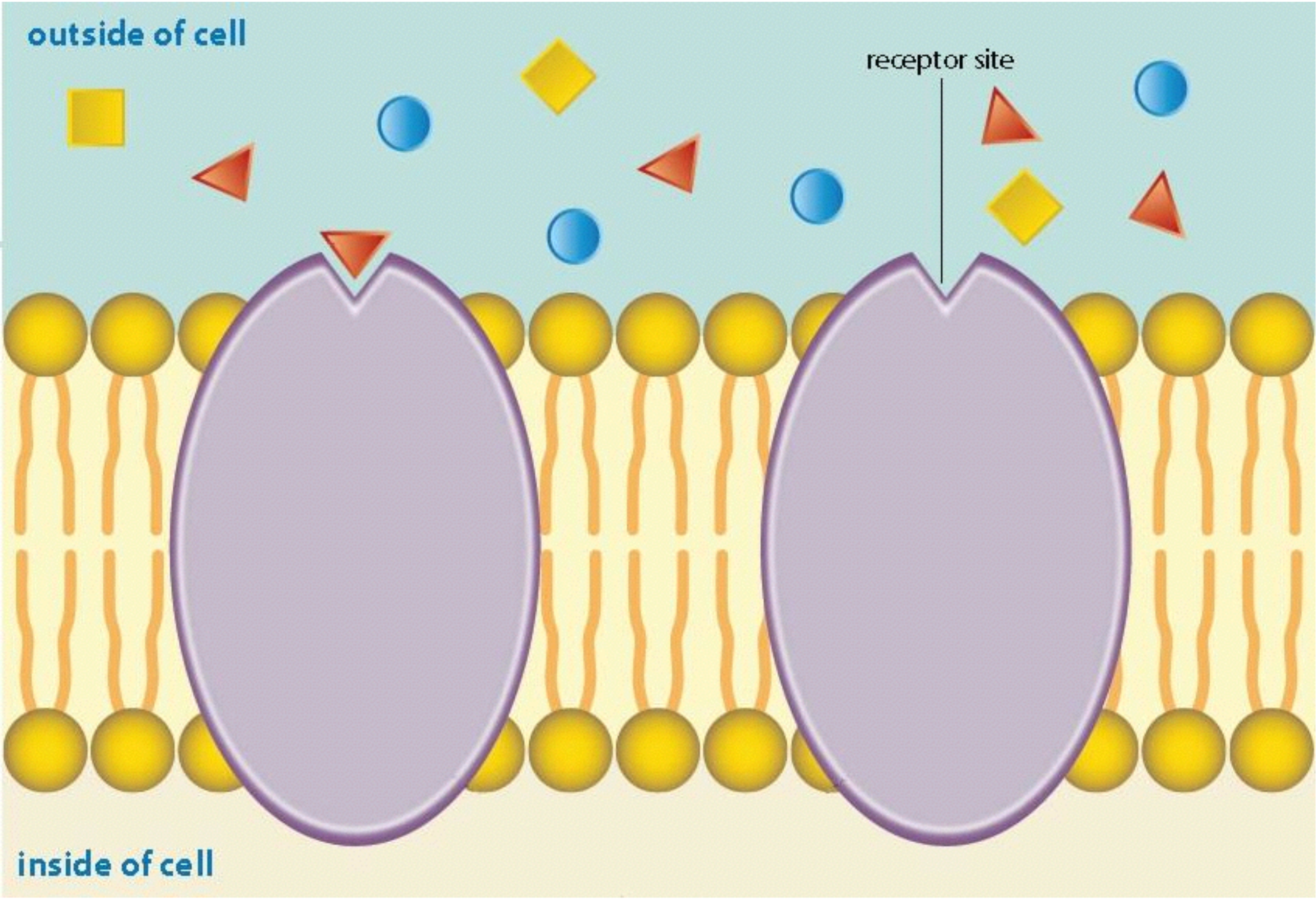
- Modeless: always able to edit code, always at runtime, anywhere you can type you can execute code. At any time.

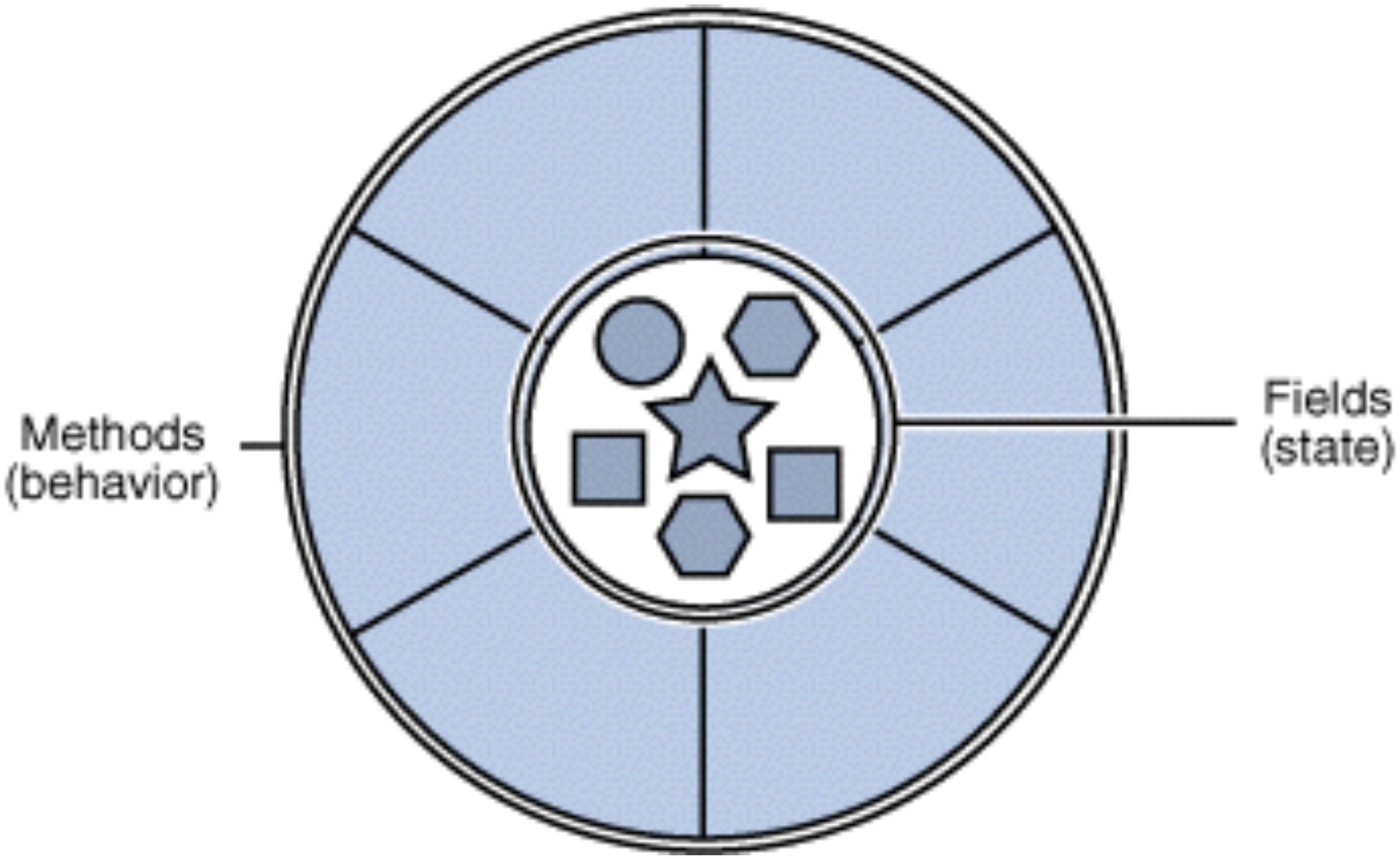
- When you save, you save a snapshot of the runtime environment… everything. Really a snapshot.

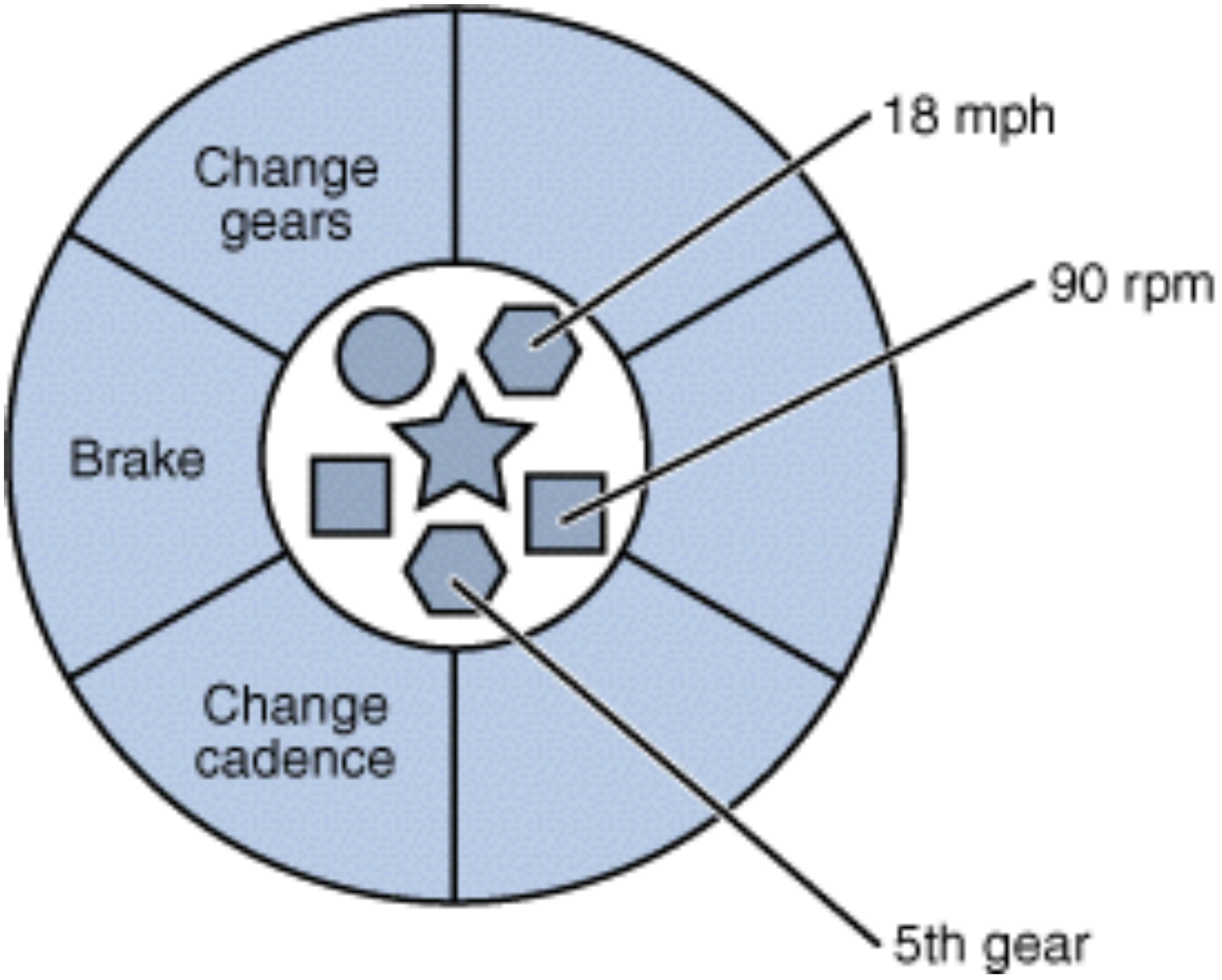
Smalltalk Demo

Objects as Cells


Cell Membrane
Mitochrondia
DNA
Nucleus
Ribosomes
Cytoplasm


outside of cell
receptor site
inside of cell


Methods
(behavior)
Fields
(state)


Change gears
Brake
Change cadence
18 mph
90 rpm
5th gear

Encapsulation

- Objects are like cells
- There's a bunch of stuff inside that is nobody else's business
- Anyone else can only use what's publicly exposed/ exported
- All you can do is send messages to request some behavior

Implementation Details

- Anything inside an object is an implementation detail
- Private data & methods
- Implementation details should be inaccessible from outside
- This lets you evolve the class without impacting clients of it

Abstraction

- Let’s say you’re making a car class.
- Which do you want?
  - `setSteeringWheelAngle(`
    `getSteeringWheelAngle() +`
    `Math.rad(-1.5)`
    `)`
  - `turn(-1.5)`

Abstraction

- Push details into the object
- Make a clean, abstract interface
- Clients shouldn’t care about **HOW**, just **WHAT**

Exercise

```
class  Node
  def initialize
    @labels = []
  end
end
```

- Add to this and write client code to add labels and print out each label, capitalized.

- Do it without accessing instance vars from outside the object.

- 10 minutes

Coupling

- Coupling is dependance on other objects
- Coupling complicates refactoring and testing
- You want to minimize coupling

Coupling

- Inheritance is tight coupling
- Composition is looser
- Composition through an interface is even looser.
- Composition in a dynamically typed language can be very loose.

Coupling

- If the only thing a class A knows about another class B is it's public interface, coupling is low.

- If A knows more about B's internals than that, coupling is high.

- High coupling means that if B's internal details change, A might need to change. This is bad.

Exercise

- Take some tightly coupled code and make it looser
- Inheritance -> composition
- 20 minutes

Exercise

- Extending collection classes is seldom a good idea. Fix this.

- Add a method to return a member by name ‘member_named(name)'.

- Refactor to make name lookup more efficient.

- Add country to Member and make old_enough_to_drink? vary based on country.

- Make it easy to add new contries without having to edit Member for each.

Cohesion

- A class has variables and methods
- Cohesion is a measure of how well the methods use the variables.
- All methods using all variables is 100% cohesion.
- You want to maximize cohesion.

Cohesion

- If a class has a single focus, thus a single reason to change, it has high cohesion.

- Low cohesion often means a class needs to be split.

Exercise

- Take a class that isn't very cohesive, split it up into more cohesive classes.

- 15 minutes

CRC Cards

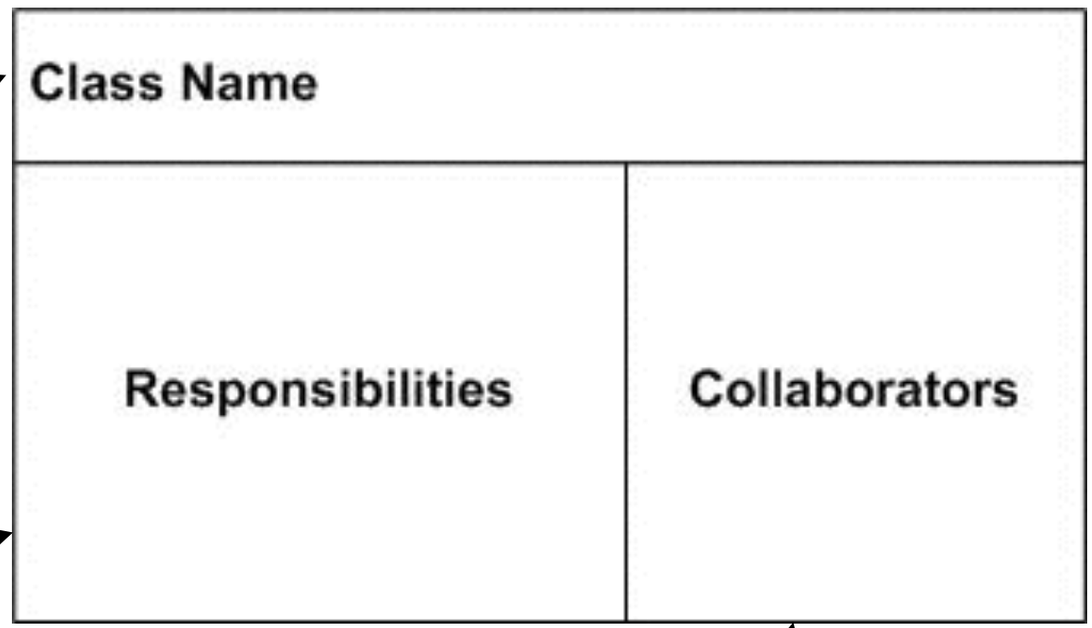
Use

- brainstorming tool for doing OO design

- http://c2.com/doc/oopsla89/paper.html

Parts

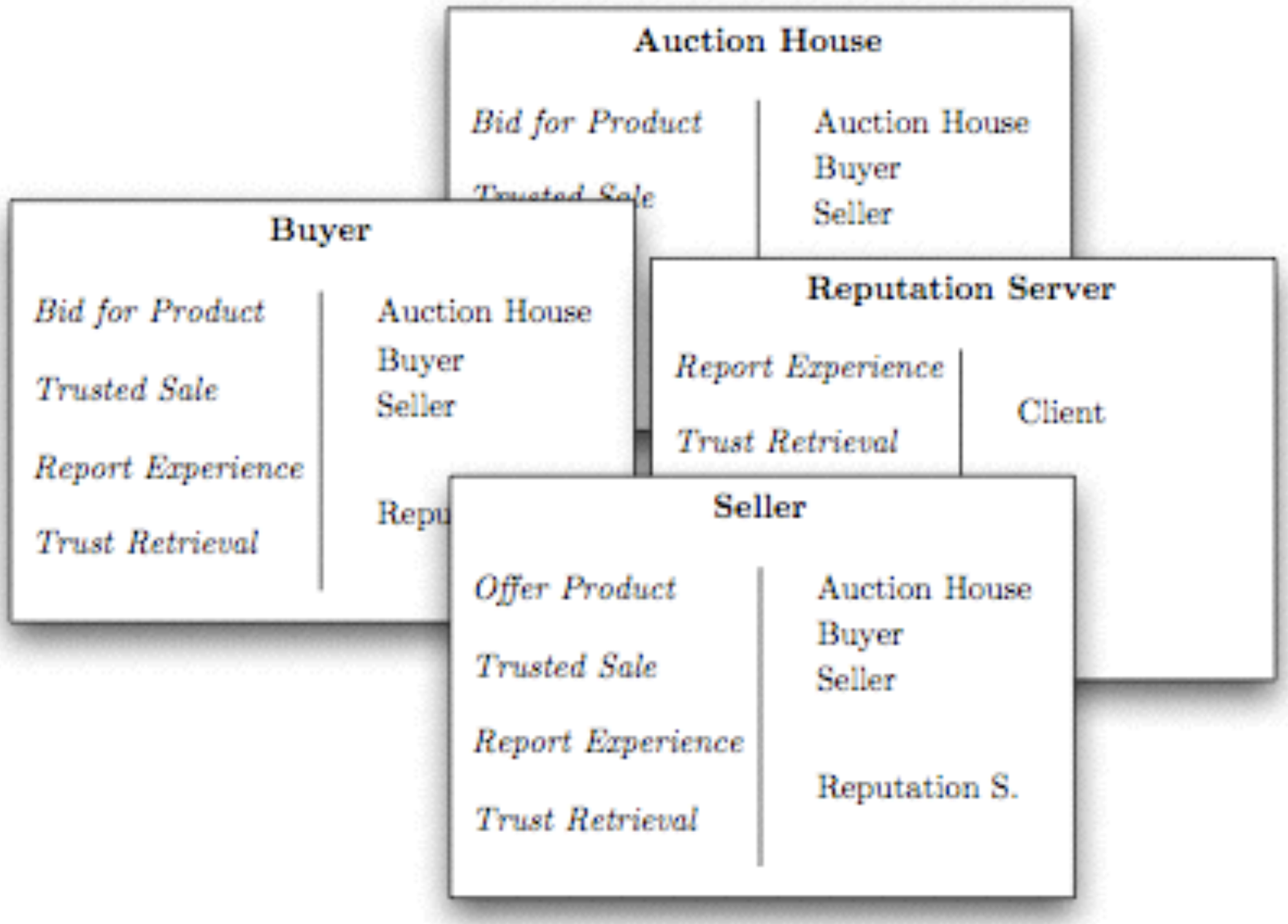
- On top of the card, the **class** name
- On the left, the **responsibilities** of the class
- On the right, **collaborators** (other classes) with which this class interacts to fulfill its responsibilities



Guidelines

- Use small cards to control complexity.
- Focus on the essentials of the class.
- Ignore implementation.
- Minimize responsibilities.

Example


Auction House
Bid for Product
Trusted Sale
Auction House
Buyer
Seller
Buyer
Bid for Product
Trusted Sale
Report Experience
Trust Retrieval
Auction House
Buyer
Seller
Repu
Reputation Server
Report Experience
Trust Retrieval
Client
Seller
Offer Product
Trusted Sale
Report Experience
Trust Retrieval
Auction House
Buyer
Seller
Reputation S.