Object-Oriented Concepts

Object-Oriented Concepts

Upon completion of this lesson you will be able to

- Define encapsulation, polymorphism and inheritance
- Define object, class, attribute, method, message
- Explain what it means to create an object-oriented system

Computing in the late 60's -70's

What did computing look like: 1967 - 1972?

- Mainframes, command-line, text-based
- Punched Cards
- COBOL
- FORTRAN
- Assembler
- ALGOL, Pascal, C/Unix
- Little/no discipline or structure
- No OO

Smalltalk to the Rescue!

- First “real” OO language, invented by Alan Kay, Adele Goldberg and others at Xerox Palo Alto Research Center (PARC) (early 1970’s)
- Reaction to command-line, text-based, centralized, monolithic mainframe mentality of the time
- Designed to mimic Kay's biological model of individual entities, or "cells," communicating with each other via messages
- System was easily extendible, with a simple syntax, a self-contained development environment, and objects, which more closely resembled the real world things being simulated

Alan Kay: Object Oriented Programming

At Utah sometime after Nov ‘66 when, influenced by Sketchpad, Simula, the design for the ARPAnet, the Burroughs B5000, and my background in Biology and Mathematics, I thought of an architecture for programming. It was probably in 1967 when someone asked me what I was doing, and I said: "It's object-oriented programming".

The original conception of it had the following parts.

- I thought of objects being like biological cells and/or individual computers on a network, only able to communicate with messages (so messaging came at the very beginning -- it took a while to see how to do messaging in a programming language efficiently enough to be useful).

- I wanted to get rid of data. The B5000 almost did this via its almost unbelievable HW architecture. I realized that the cell/whole-computer metaphor would get rid of data, and that "<-‟ would be just another message token (it took me quite a while to think this out because I really thought of all these symbols as names for functions and procedures.

OOP to me means only messaging, local retention and protection and hiding of state-process, and extreme late-binding of all things. It can be done in Smalltalk and in LISP.

Object Oriented Defined

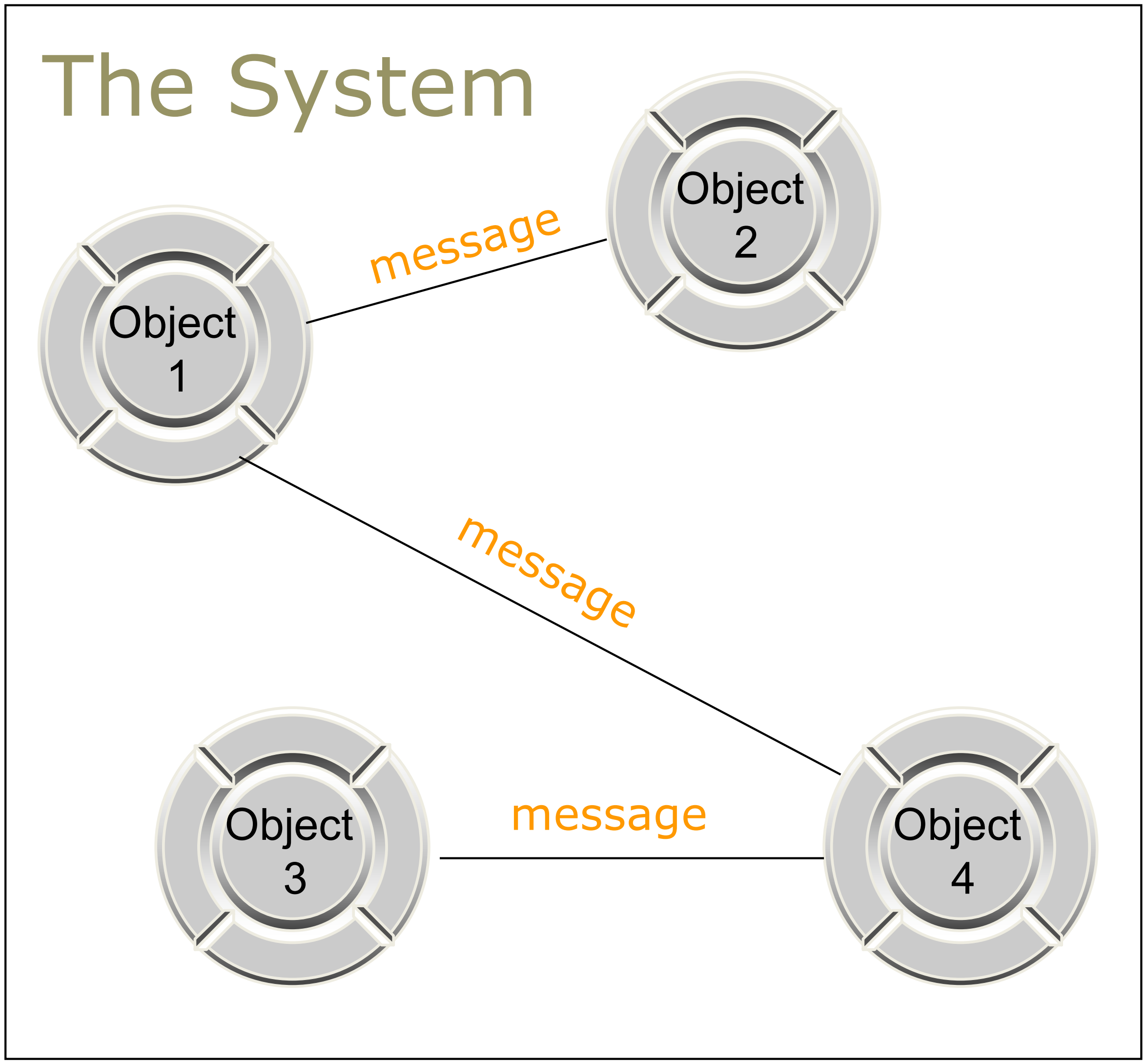
Alan Kay, Sept 14, 2016:

- Part of the problem here is that I made a mistake with how the term was coined — I should have picked something else — in hindsight: “server-oriented programming”?
- For example, a “server” could choose to allow its encapsulation to be violated — e.g. by making its services to closely resemble data structures acted on by procedures. Here, in my opinion, we would be simulating quite the wrong kinds of things, and devolving back into weak and fragile programming styles. (That is my view of what has mostly happened with “objects” — “real objects” never showed up because most people wanted to retain their data oriented style, etc.)
- For example — today and tomorrow — we should be programming in terms of “requirements and goals” that can manifest a workable system (possibly needing a super-computer).
- We should be able to optimize a system like this without touching the requirements and goals part, etc. In other words, we want to devote most of our attention into “the whats” rather than “the hows”, use most of our energy for design, and we’d like to “ship the design!” (that would be a good slogan for the next few years).

Eric des Courtis, Jun 29, 2016:

- *An object is simply a virtual server. In other words an encapsulated computer which uses messages to communicate with other virtual or physical servers.*
  It's simple and brilliant. All other aspects of OOP are secondary.
  He also said he believed every object should have an IP (sort of like a pointer).

Alan Kay’s Vision



What is This OO Stuff, Anyway?

- OO is about ***simulating the real world*** in a computer
- In the real world we interact with objects everyday without thinking about how they work or why
- We tell someone or something to do something and it does it (mostly)

What do *we know* that a Cat and Dog can do?

What Cats and Dogs Can Do

Eat	Eat
Sleep	Sleep
Meow	Speak
Scratch	Chase cars
Walk	Fetch (thing)
	Walk

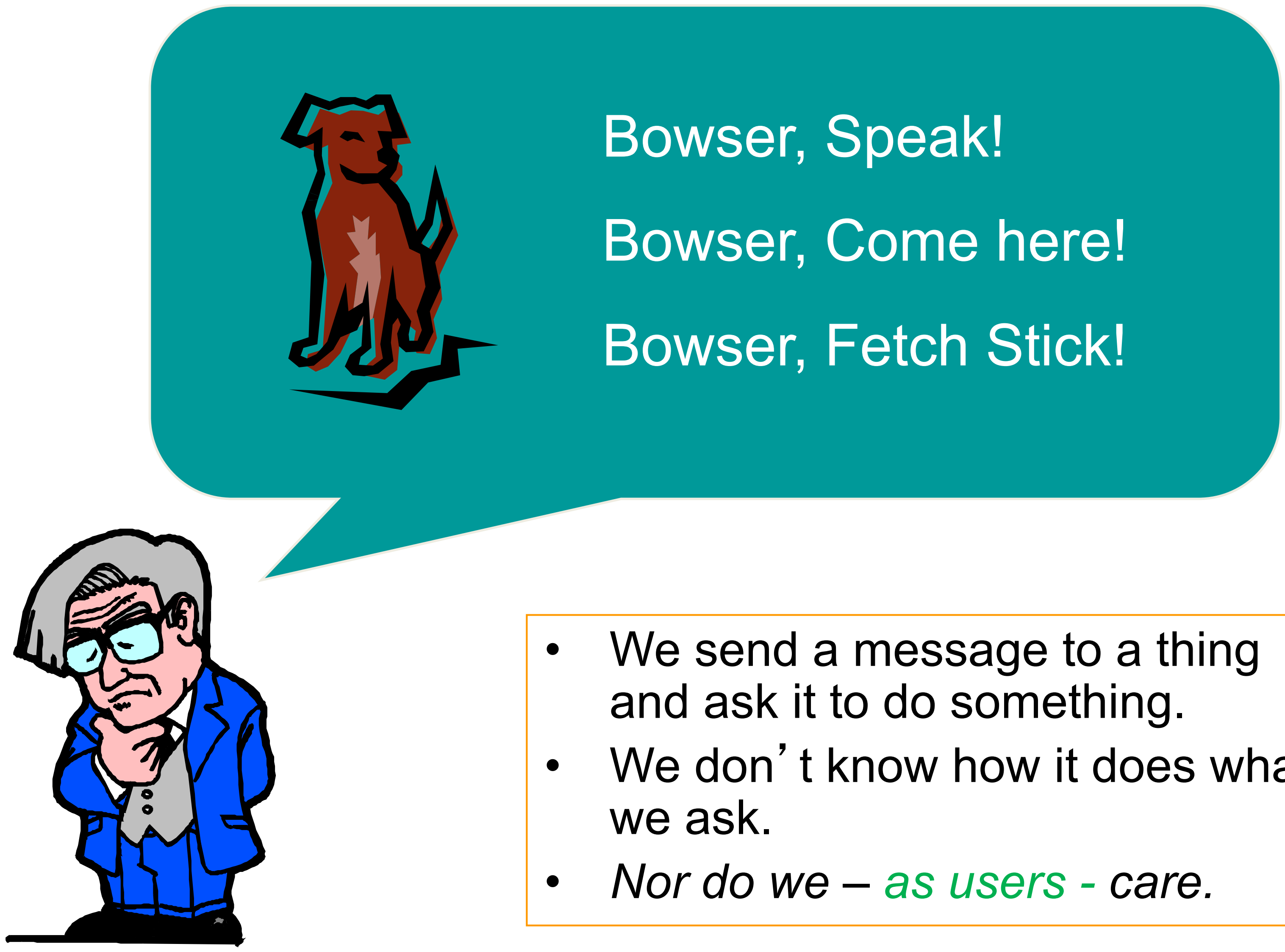
Cats and Dogs, like all things, have *behavior*

What Cats and Dogs Know

name	name
weight	weight
whiskerCount	bonesCount
length	legCount
napCount	Owner
dateLastMeal	dateLastMeal

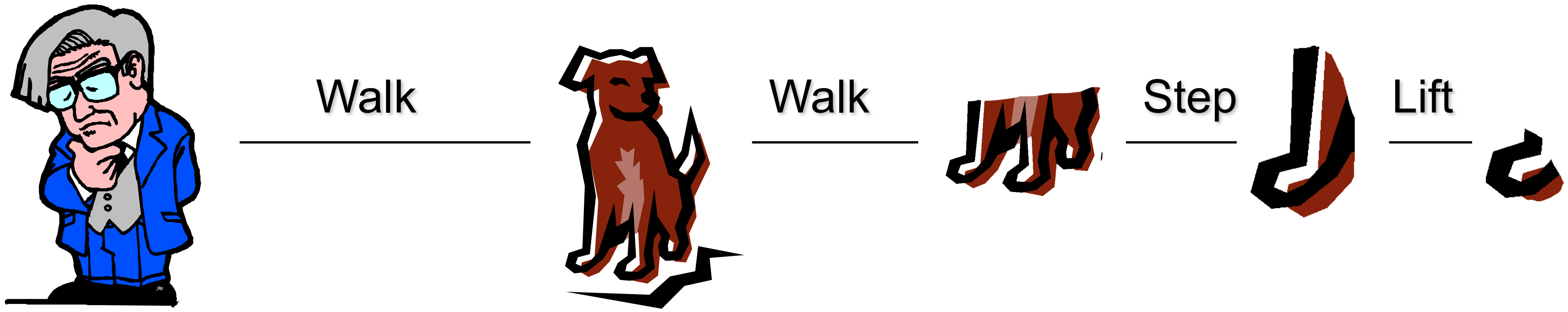
Cats and Dogs, like all things, know about their *internal state*

How We Interact with Dogs and Cats



- We send a message to a thing and ask it to do something.
- We don't know how it does what we ask.
- *Nor do we – as users - care.*

How Dogs Respond to Our Message

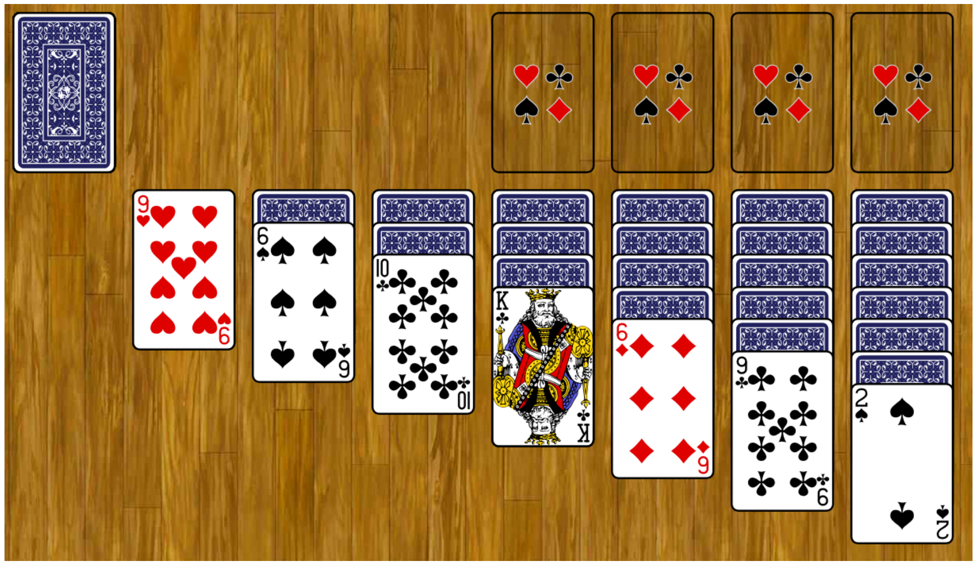


- Objects are *responsible* for doing things
- Objects *collaborate* with other objects to perform tasks
- Object use their internal state to carry out the behavior

Solve a Problem Using Objects

Problem Domain: Create a Solitaire Engine

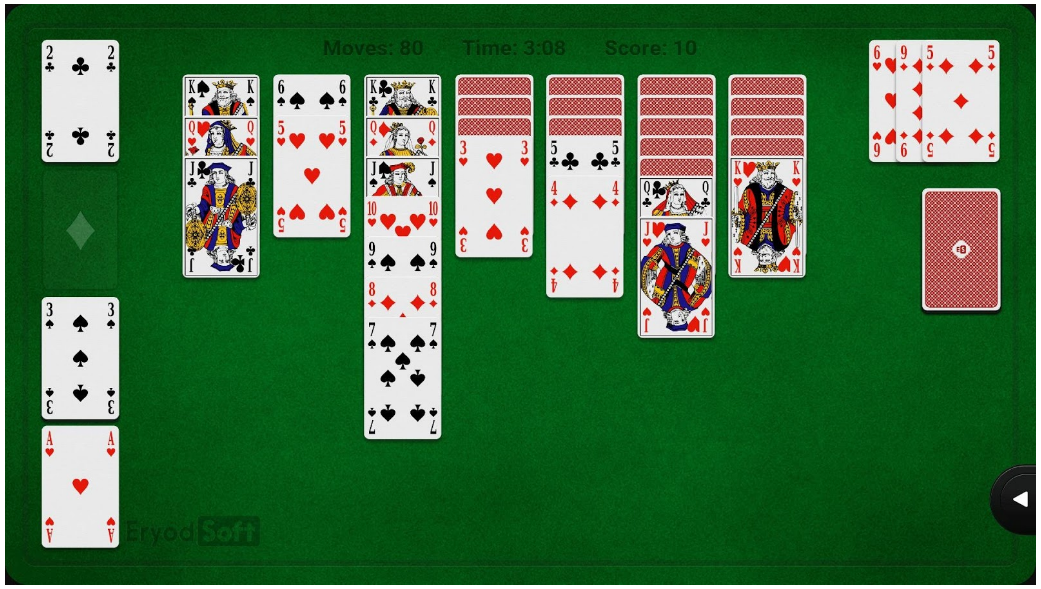
- You have been tasked with creating an “engine” to play games of Solitaire and record the results for later analysis
- What things do you typically work with in this domain?



What Objects are Used?

In the Solitaire (Card Game) *Domain* we work with

- Card (e.g. Ace of Clubs)
- Deck of Cards (a Collection of individual Cards)
- Pile (of cards)
- Player (scores)
- Tableau (e.g. the main layout)



How are the Objects Used

The user of the system describes for us how the things will be used:

"To form the tableau, seven piles need to be created. Starting from left to right, place the first card face up to make the first pile, deal one card face down for the next six piles. Starting again from left to right, place one card face up on the second pile and deal one card face down on piles three through seven. Starting again from left to right, place one card face up on the third pile and deal one card face down on piles four through seven. Continue this pattern until pile seven has one card facing up on top of a pile of six cards facing down.

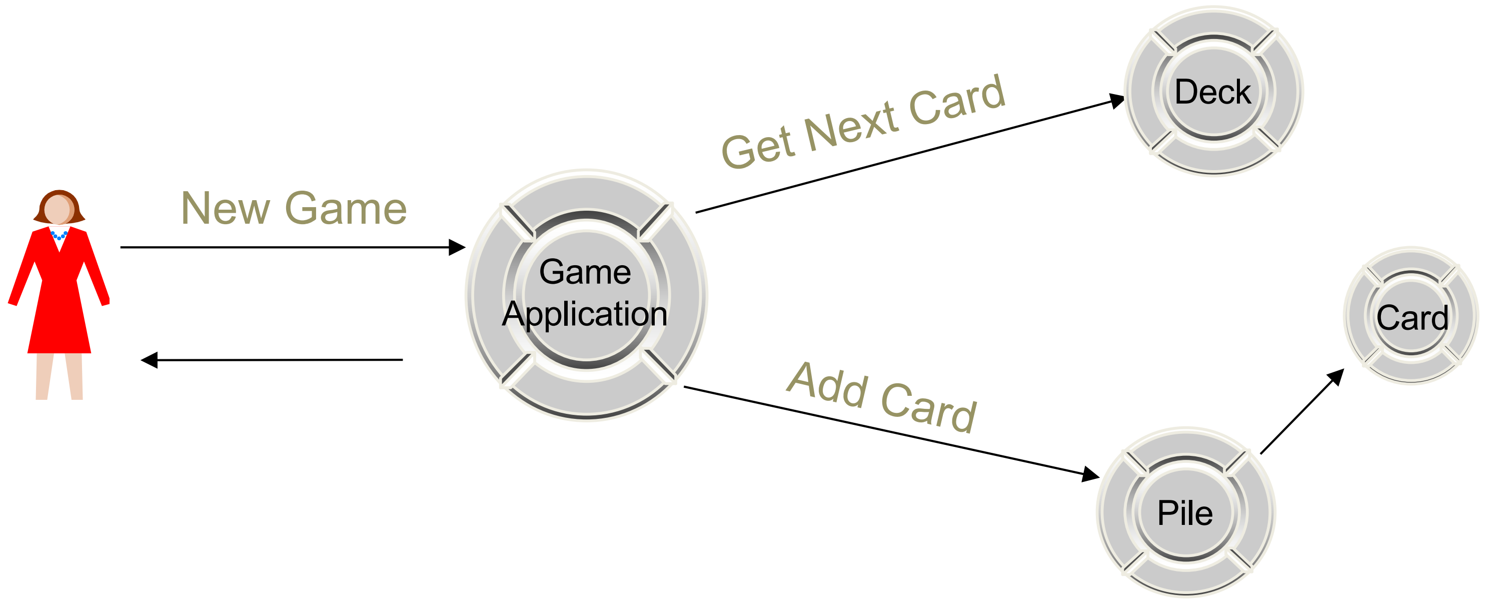
The remaining cards form the stock (or "hand") pile and are placed above the tableau.

When starting out, the foundations and waste pile do not have any cards."

This scenario is called a Use Case. It is where Object Orientation starts. A Use Case describes the ***objects that will be used to create the application***. It also describes how the objects will be used (their behavior).

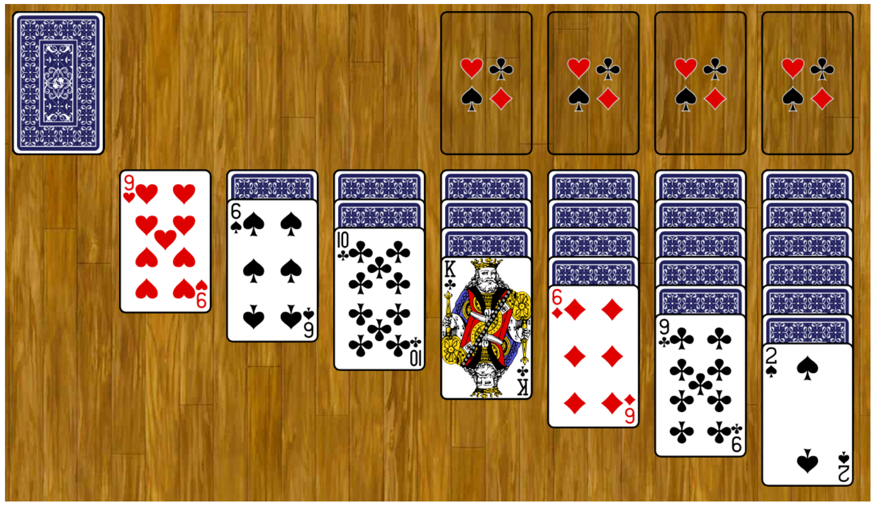
How This Works with Objects

1. Get the next Card from the Deck
2. Add that Card to the Pile



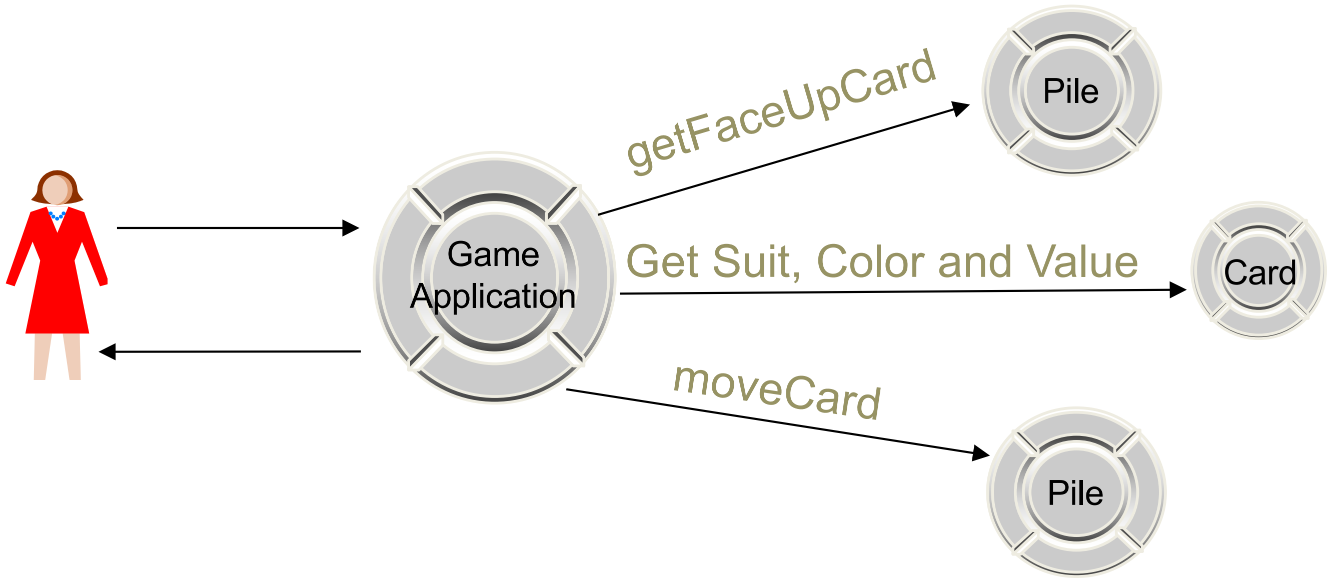
Start Game:

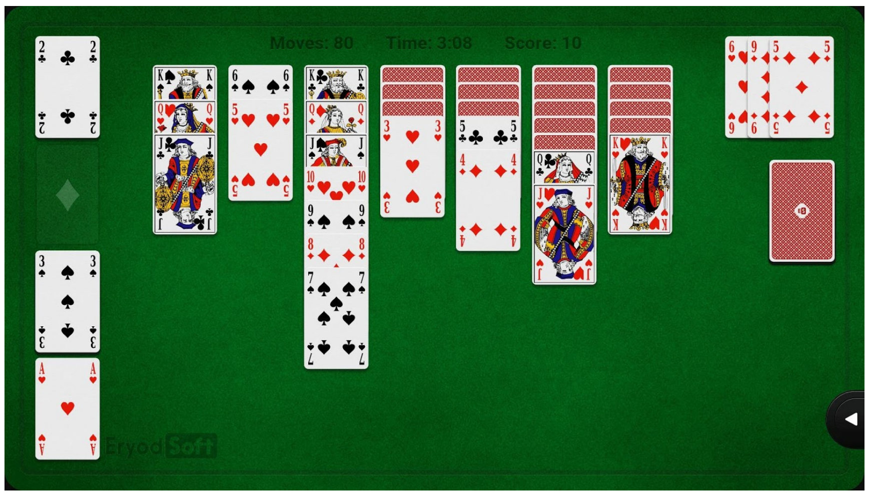
1. Zero out totals
2. Shuffle Deck of Cards
3. Get a Card and Add to the Next Pile
4. Repeat until Piles are complete



How This Works with Objects

1. For each Pile, get the playable Card's Suit and Color.
2. Is that card Transferable? If so, transfer it to the other Pile.

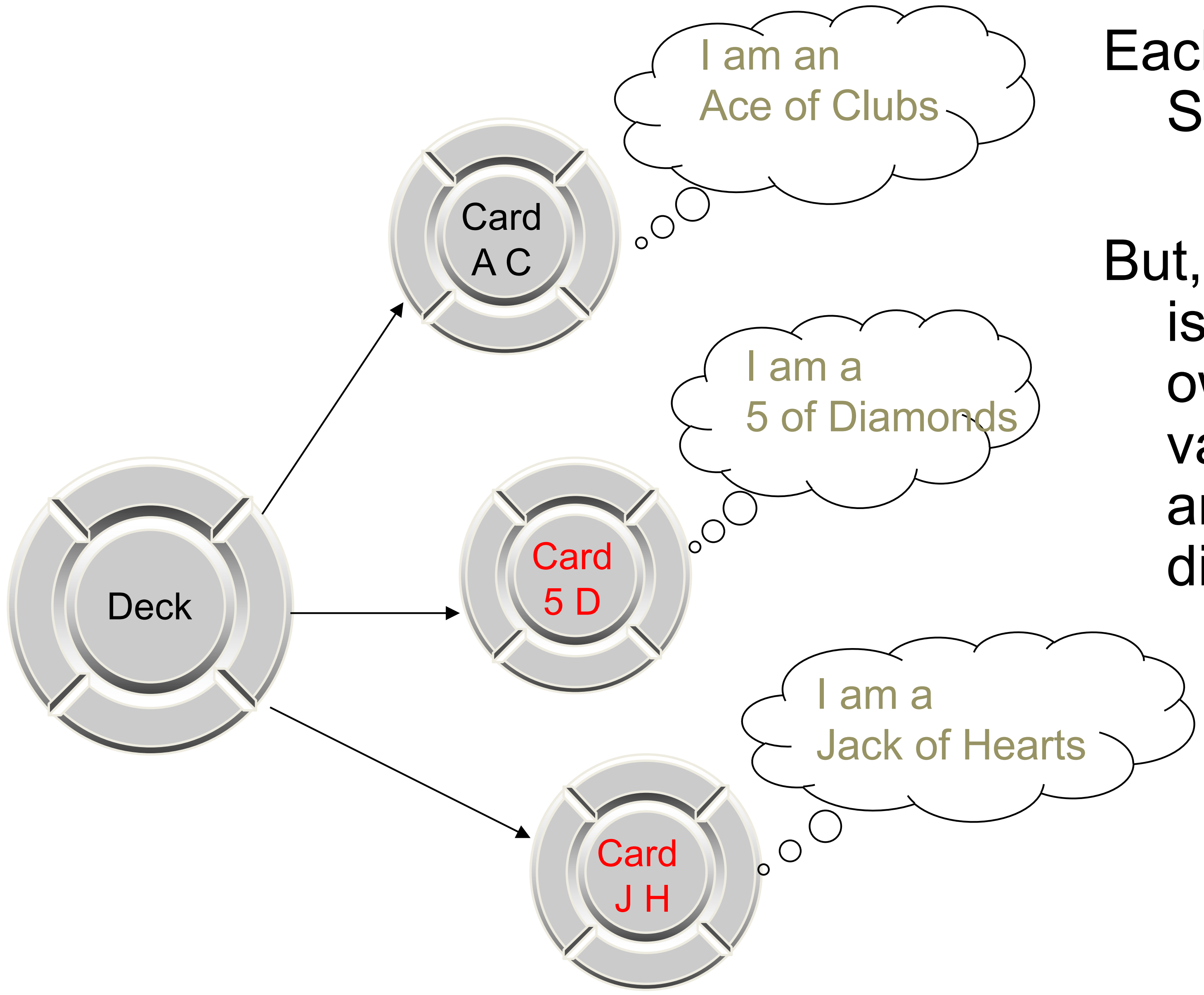




Objects “Know” How to Do Things

- An object contains information about its internal state and what behaviors it can perform:
- A Customer object can “tell you” its name and address
- A Home object can “tell you” its address, year built and type (single family, barn, skyscraper)
- What can a Card object tell you?
- What can a Deck object tell you?

Objects “Know” How to Do Things (cont.)



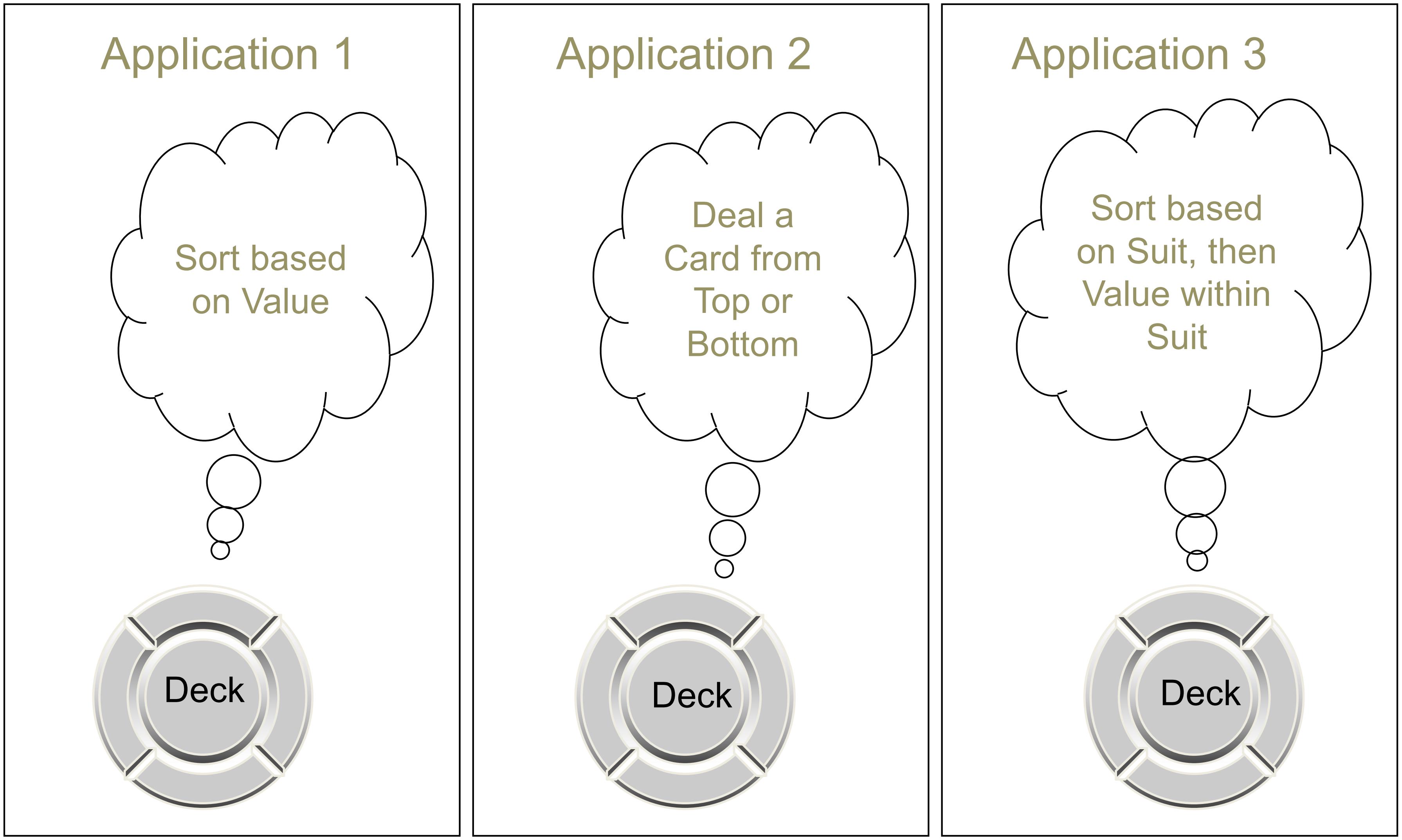
Each object knows its Suit and Rank

But, since each object is different (has its own data/state) the values of the rank and suit may be different

The Deck Object: From the User's View

- The user sees the Deck object as helping to solve a problem
- The user only needs the functions of a Deck that help to solve the problem (***abstraction***)
- The same Deck may be used differently to solve different problems in different applications. How the objects are used (called a Use Case) determines what behaviors must be created for them.
  - One application needs to be able to deal from the top and bottom of the deck
  - One application may need to sort the Deck by suit, then rank within suit
  - A different application may need to sort the Deck based on value without regard to suit

The Deck Object in Use

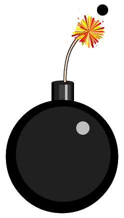

Application 1
Sort based on Value
Deck
Application 2
Deal a Card from Top or Bottom
Deck
Application 3
Sort based on Suit, then Value within Suit
Deck

From the Deck Designer’s View

The **designer** must use variables (other objects) to represent the policy object’s state:

- `String deckType = “Rummy”`
- `int totalCards = 52`
- `int cardsRemaining =  52`

- What happens if we decide to change the implementation and use a different name or way to represent the value for `cardsRemaining`? Or read it from a database or Web Service?
- If the implementation (data structure and algorithm) is visible to the user of the object:
  - Every place that referred to the Deck’s `cardsRemaining` variable would have to be changed, recompiled, and debugged
  - Sometimes called a *side effect*



- If the implementation (data structure and algorithm) is hidden from the user:
  - If the implementation changes, the user of the object is not impacted

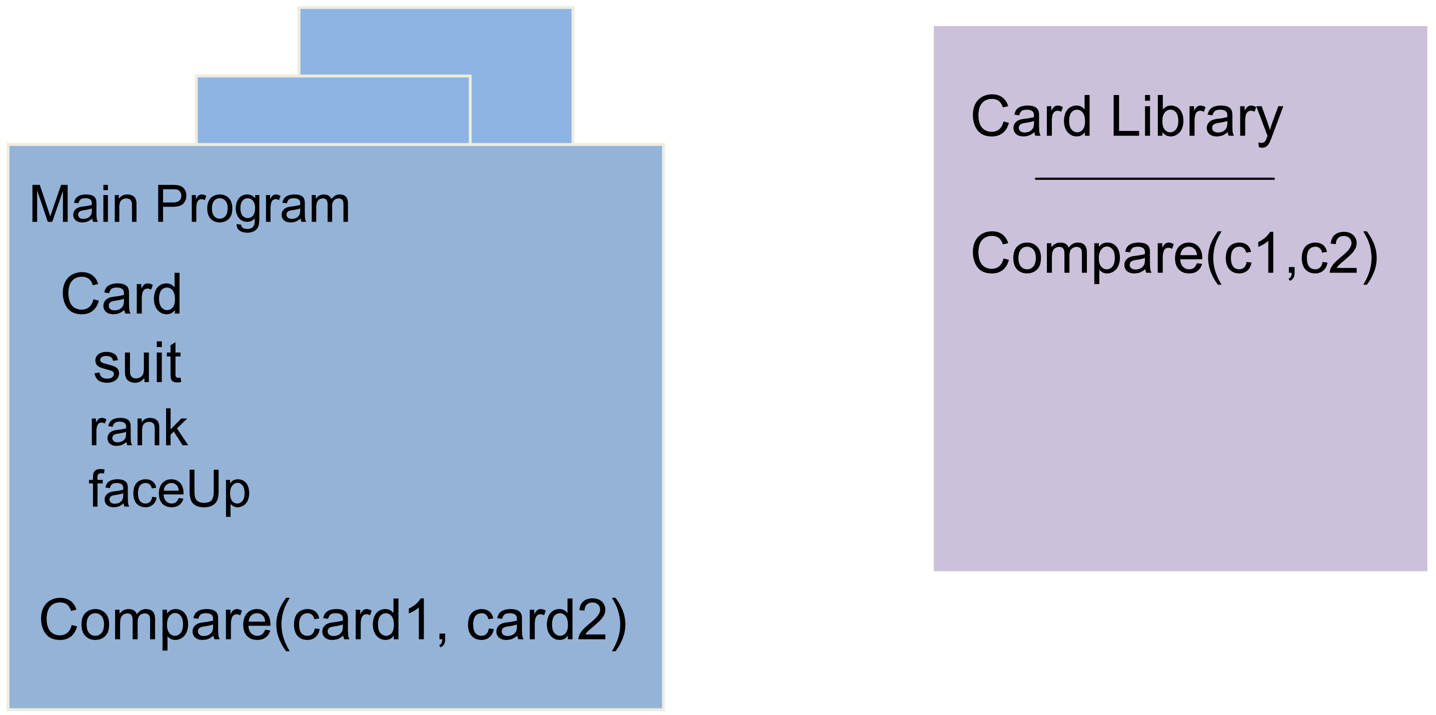
Which is **better**: more flexible, easier to maintain and enhance?

Abstraction

- The Problem Domain determines what subset of behaviors and information you need to capture
- Different problem domains require different behaviors, even in the same object
- Notice how the airplane’s ***behavior*** is dependent on how it is used
- Ex: Given a Airplane Object
  - How does an Airport application use the Airplane object?
  - How does a Travel Agent application use the Airplane object ?
  - How does an Airline application use the Airplane object?

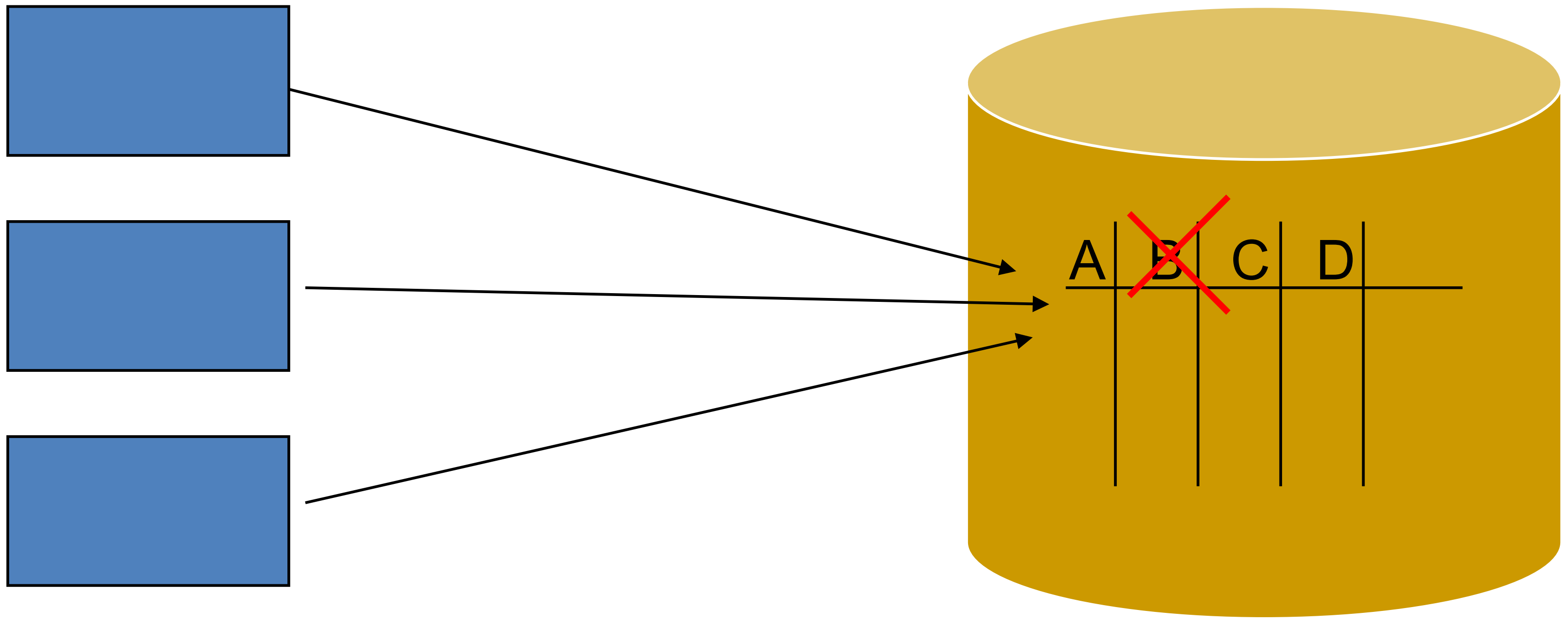
Procedural Programming

- Procedural programs put the data structure into the main program and call a function and pass the data structure
- What happens when the data structure changes?

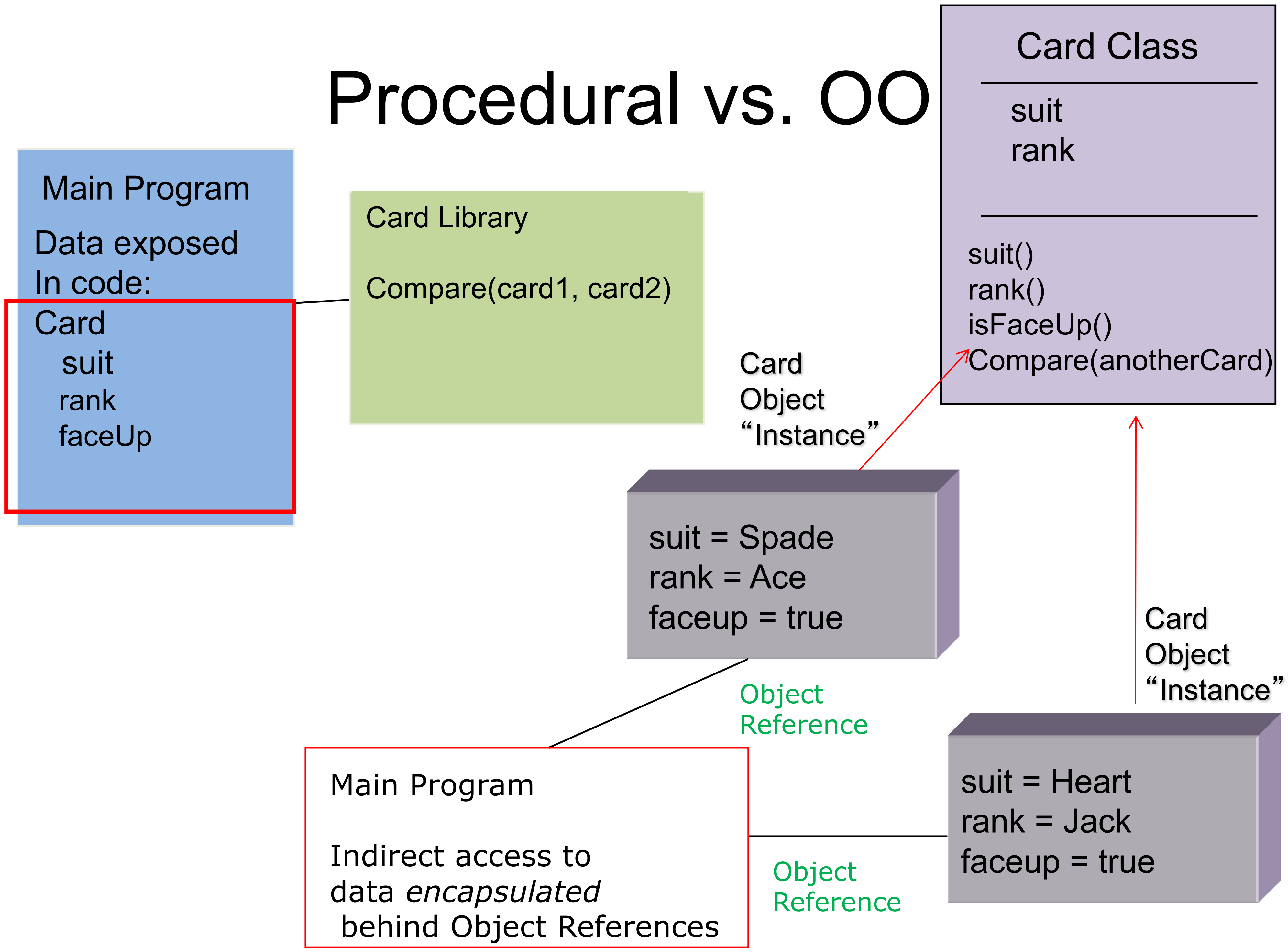


Access Databases

- The problem is more evident with databases
- If all programs directly access tables, what happens when the table structure changes ?



Procedural vs. OO



Objects and Classes

suit = Club
rank = 2
faceUp = true

suit = Heart
rank = Jack
faceUp = false

suit = Diamond
rank = Queen
faceUp = true

What do these objects all have in common?

- They are all types of Card objects
- They have the same attributes (`suit, rank, faceUp`)
- They have the same behavior: they can all tell you their suit, rank and if they are face up

We use a **Class** to specify the design of our objects

An Object is a **specific instance** of a Class at runtime

Ex: Table and row

Ex: Recipe and meal

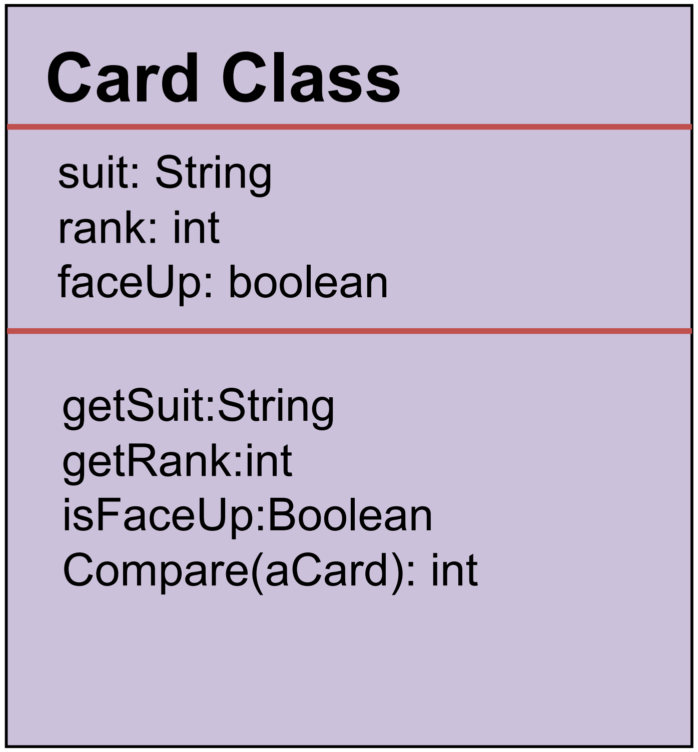
Ex: Blueprint and building

Ex: Integer and 23

What is a Class

A template (factory) for creating new objects:

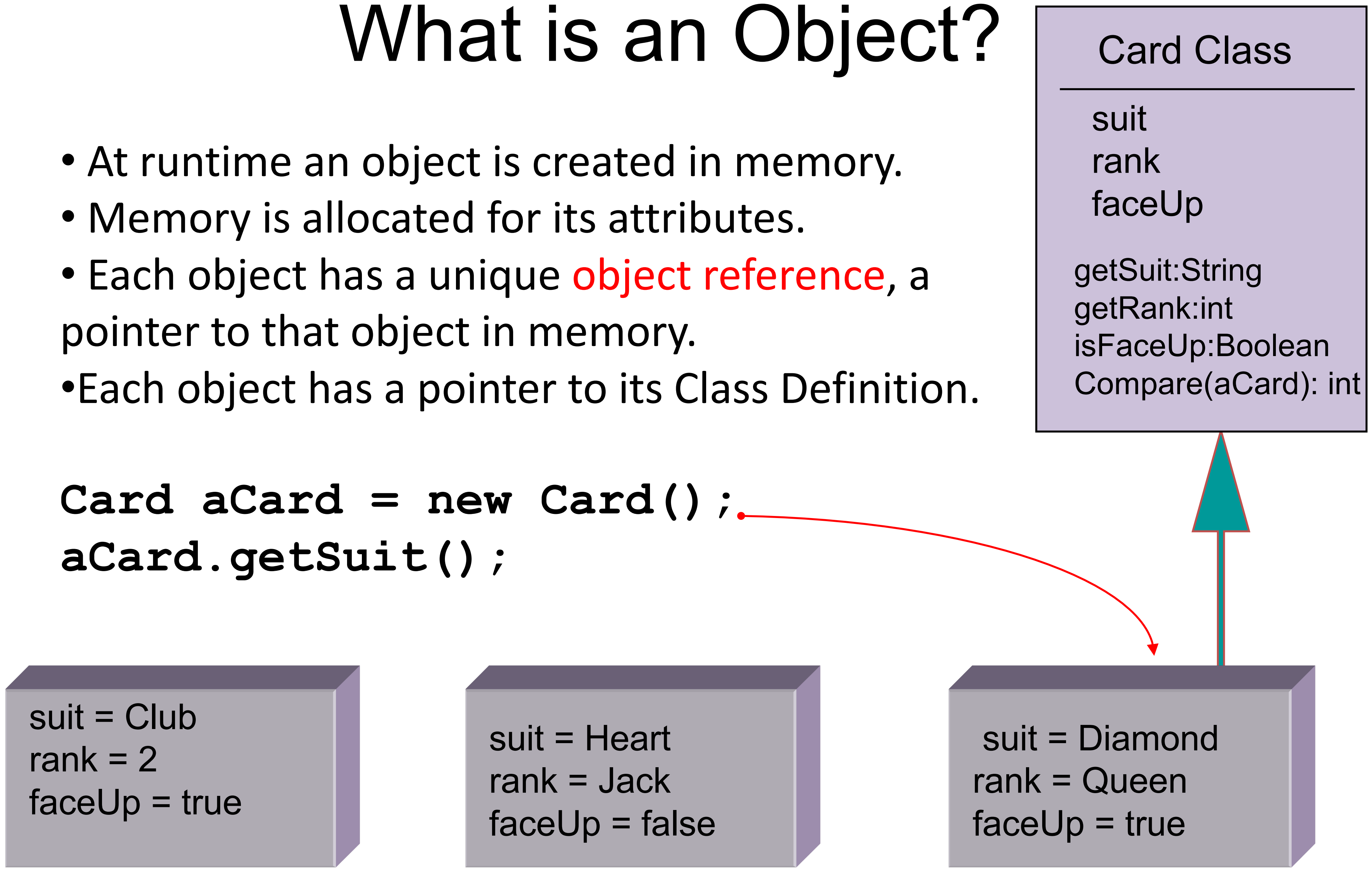
- the name of the class
- the attributes and their types
- the methods, their arguments, return types and the code to implement them



What is an Object?

- At runtime an object is created in memory.
- Memory is allocated for its attributes.
- Each object has a unique object reference, a pointer to that object in memory.
- Each object has a pointer to its Class Definition.

```
Card aCard = new Card();
aCard.getSuit();
```



Encapsulation and Data Hiding

- When an object is asked to do something, the result may be a change in its state

  Ex: If the Card is told it has been turned over, its state has changed (it now the opposite of what it was: face up or down)

- The ***only way*** an object's state can be ***changed*** is by asking it to do something
- It could be something simple:

  ```
  anEmployee.setName("Joe")
  ```

- Or something complex:

  ```
  anEmployee.completeReview()
  ```

- This is also called *encapsulating* the data

“Everything is an Object”

In OO, all data is *wrapped* by the methods of a class.
There is literally no way to “see” the data:

Retrieve Salary value:

```
Salary mySalary = emp.getSalary()
```

Compare:

```
if(mySalary.greaterThan(anotherEmp.getSalary()))
```

See! No data! Just compare salary objects

Calculation:

```
mySalary().increase(pctInc)
```

See! No data! – it knows how to increase its salary

Display:

```
System.out.println( aSalary.toString() )
```

Returns a string object- **`println`** will display the characters of the string one character at a time on the output device

“Everything is an Object”

“But, what if I need to “get” my salary as a floating point number or string or array of chars or as XML or JSON?”

Salary Class Methods:

- asFloat
- asString
- asCharArray
- asXML
- asJSON

We Never See Implementation

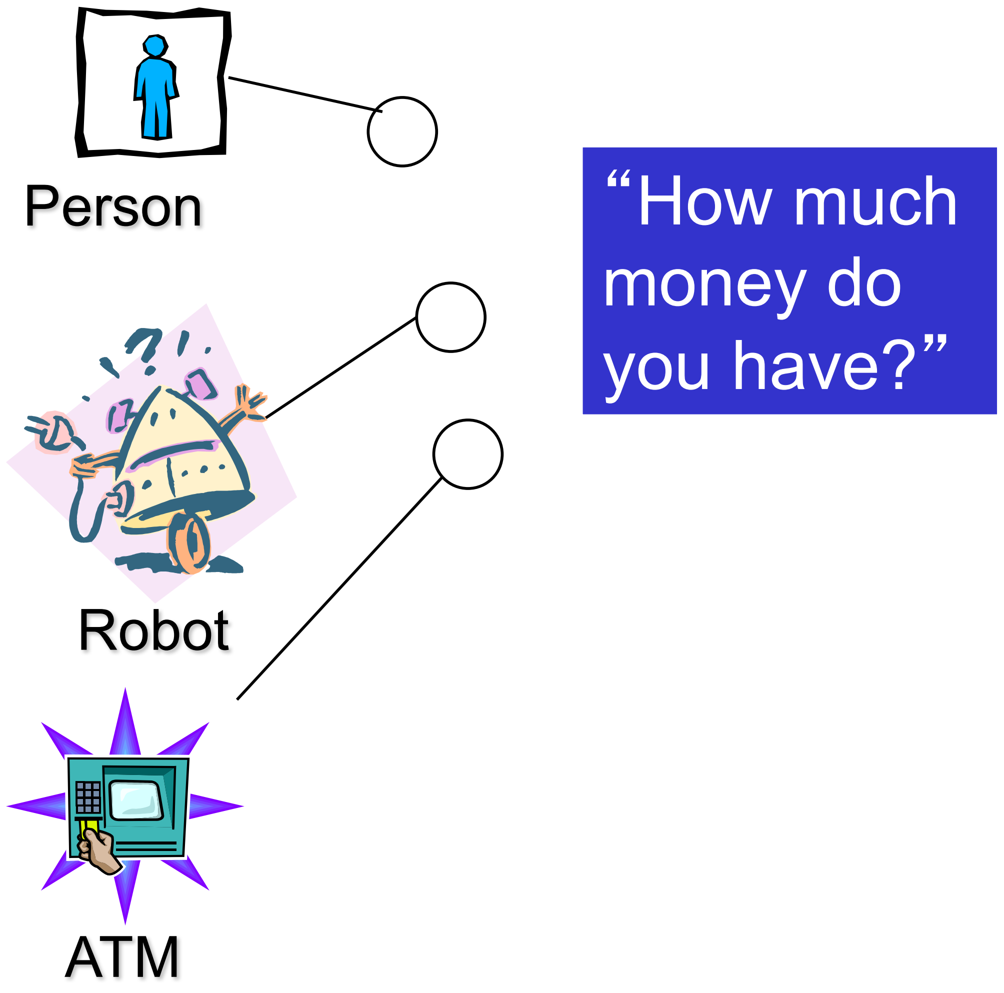
- In real life implementation is often hidden
- If you ask your neighbor

  “How much money do you have”
- You do not know how they come up with the right answer
- What other things work like this?

An Object’s Public “Face”

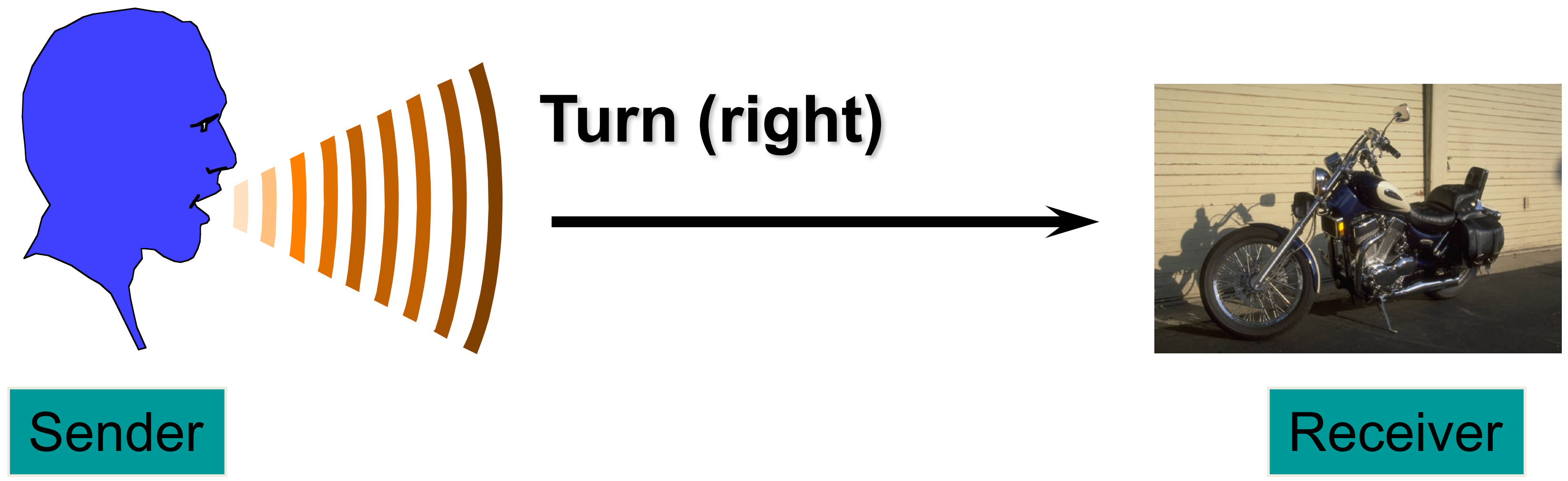
- The functions and subroutines defined for an object are its public face (“interface”)

- Many things can have the same interface, yet **implement it differently**



Messages

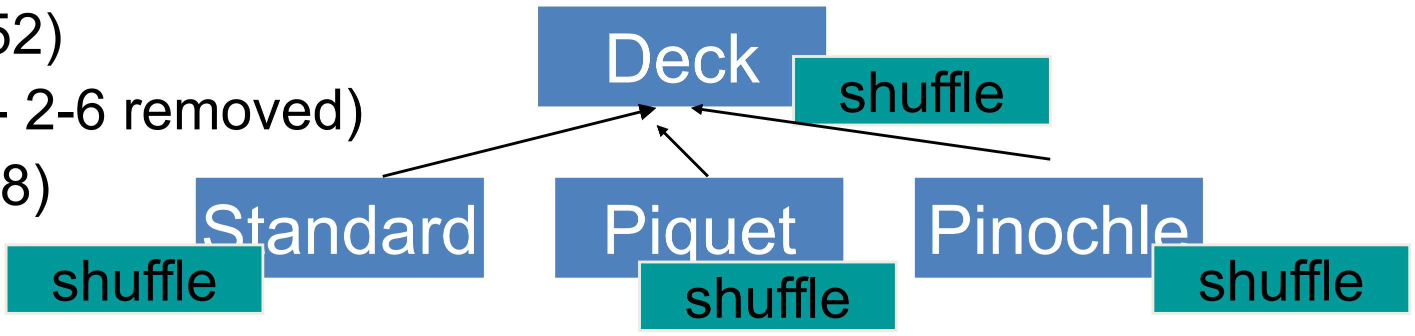
Method Invocation



- One object tells another object to do something by sending it a message
- The receiver determines which method implements that message
- The same message can result in different behavior on different objects

Inheritance

- What if you had several types of Decks
  - Standard (52)
  - Piquet (32 - 2-6 removed)
  - Pinochle (48)



- Imagine you could share behavior from the more general Deck
  - Shuffle
  - Deal
- Each type of Deck could implement shuffle differently
- Why might this be a cool thing?

With objects, you can give the same name to services that may work differently in different objects but perform the same basic function

How to Work with Different Types

- In Procedural languages, you would need an **`IF..ELSE`** statement to determine the type and call the correct function:

```
IF (is Standard) call shuffleStandard (Standard)
  ELSE (is Piquet) call shufflePiquet (Piquet)
     ELSE (is Pinochle) call shufflePinochle(Pinochle)
```

Polymorphism

Object systems move the **`IF`**...**`ELSE`** to the runtime Virtual Machine:

```
Deck d = DeckFactory.createNewDeck(type);
d.shuffle();
```

At runtime, the system asks **`d`** "What type of object are you"

Based on what **`d`** answers, the appropriate shuffle() function is called

This ability is called ***polymorpism***

Late binding, or **dynamic binding**, is a computer programming mechanism in which the method being called upon an object or the function being called with arguments is looked up by name at **runtime**. - Wikipedia

How Do We Use Classes in Java

```
public class DeckTester{

  public static void main(String[] args){

      StandardDeck theDeck = new StandardDeck();
      Card aCard = new Card(“Club”, “A”);

      theDeck.add(aCard);

      System.out.println(theDeck.dealACard().getSuit());

      if (theDeck.cardsRemaining() > 10)…
}
```

Quiz

What is a Method?

What is an Attribute?

What is an Object?

What is a Class?

What is Inheritance?

What is Abstraction?

What is Encapsulation?

What is Polymorphism?