Three Paradigms

Sacramento Web & Mobile Devs
3/28/2019

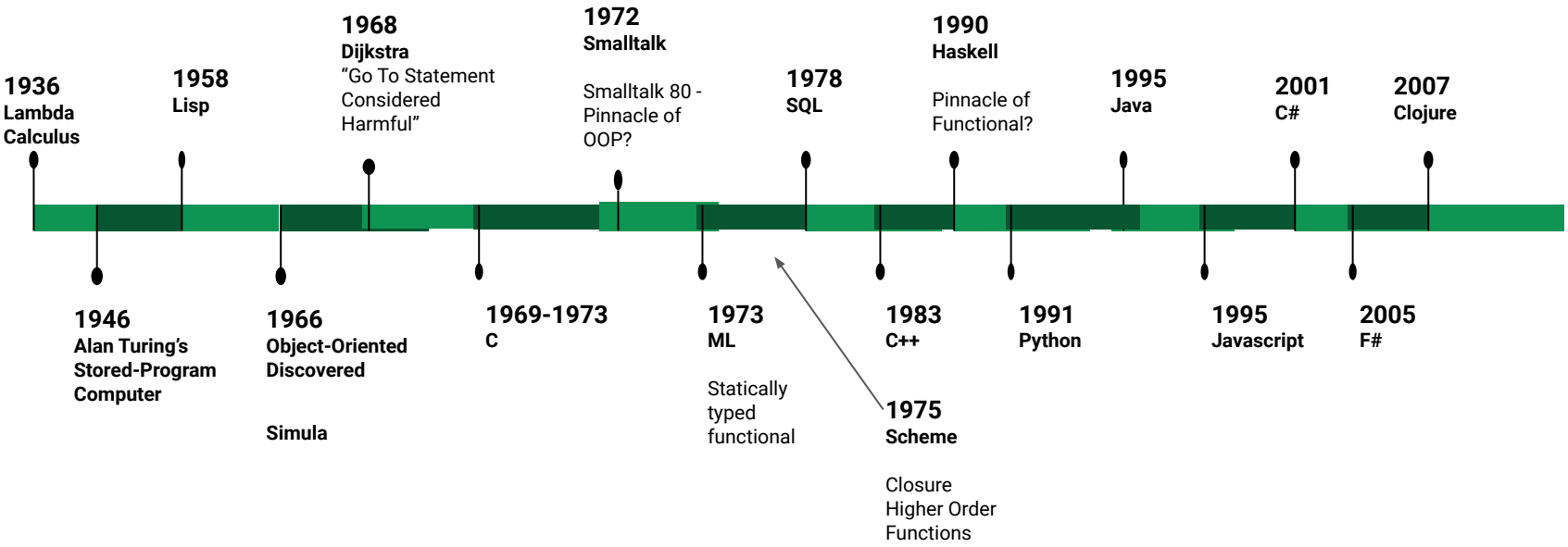
Goals

- Participation
- High level overview - Don’t get lost in the details
- Introduce new ideas to be explored later
- Create sense of possibility and inspiration
- What makes code “good” or “bad”?
- What are our responsibilities as programmers?

Three Paradigms (there are more)

1. Structural / Procedural Programming
2. Object Oriented Programming
3. Functional Programming

Programming Languages



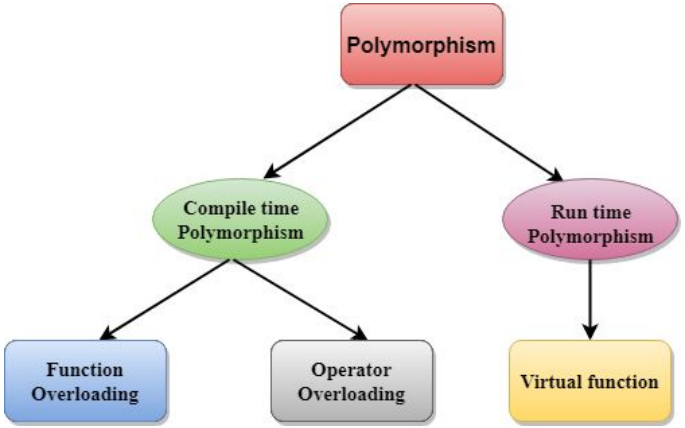
Structured Programming

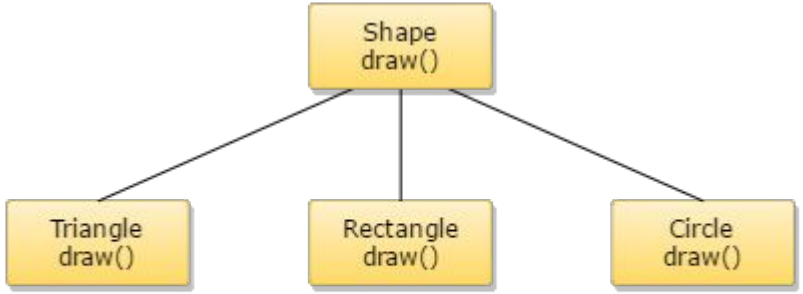
- Edsger Wybe Dijkstra
  - Netherlands' first "programmer"
  - 1955 - Chose programming as career because it provided a greater intellectual challenge than theoretical physics
- Proof
  - Attempted to apply formal mathematical proofs to programming
  - Simplified complex algorithms into recursively smaller units (Divide and Conquer)
  - Some uses of goto prevented decomposition
  - Other uses of goto lead to simple selection (if/then/else) and iteration (do/while/for)
  - 1968 - "Go To Considered Harmful"
- Scientific Method
  - Theories / Programs are falsifiable but not provable
  - Testing can establish presence of bugs, but not their absence

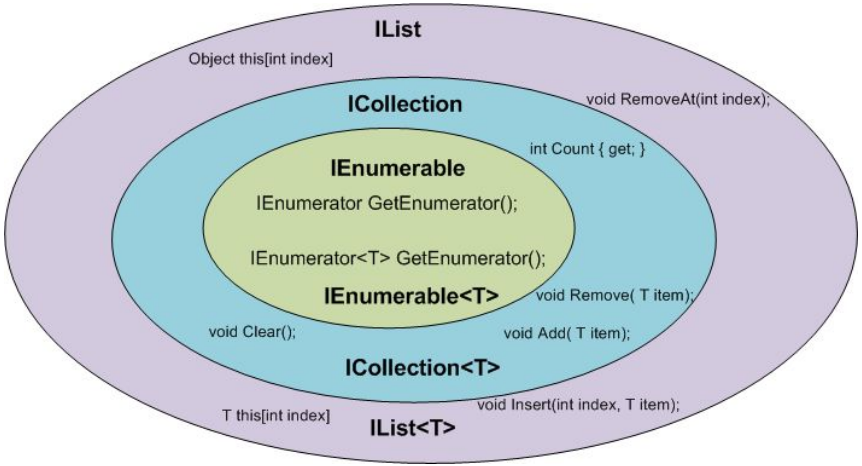
Object Oriented Programming

- What is it?
  - Combination of data and behavior
  - Way to model the real world
  - Three special words
    - Encapsulation
    - ~~Inheritance~~
    - Polymorphism
- What it should be
  - Passing messages between objects
  - Implicit “this” reference passed around
  - Key insight is moving function call stack frame to heap
  - Dynamic dispatch - selecting polymorphic implementation at run time

Polymorphism in Pictures

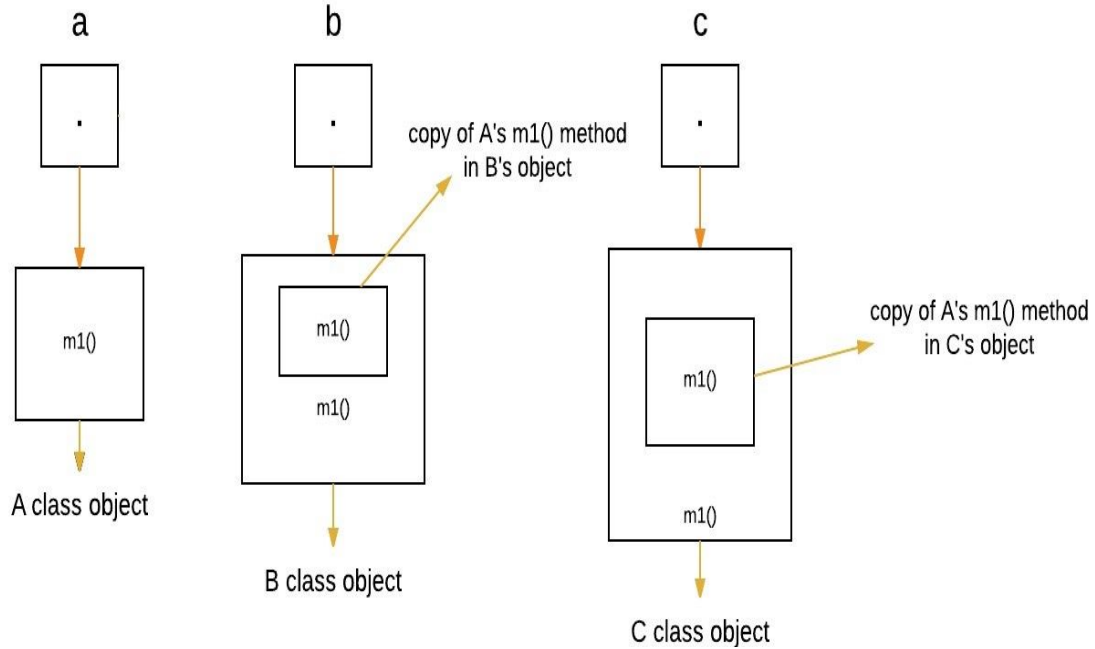






Polymorphism in Code

```
class A                { void m1() => log("Inside A's method"); }
class B extends A      { void m1() => log("Inside B's method"); }
class C extends A      { void m1() => log("Inside C's method"); }

main() {
      A a = new A();
      B b = new B();
      C c = new C();

      A ref;

      ref = a;
      ref.m1();

      ref = b;
      ref.m1();

      ref = c;
      ref.m1();
}
```



Break - Basic Procedural / Structured Approach

- Read from JSON object
  - Need Lastname, Firstname of all people in California
  - Exclude San Diego
  - Offer sale for people in Sacramento
  - Billing department wants FIRSTNAME LASTNAME
  - Need a count of all potential customers from states outside California

Functional Programming

- Pure Functions
- Immutability
- Higher Order Functions
- Partial Application
- Closure
  - “Most important discovery in history of programming” - Douglas Crockford
- Category Theory - developer vs academic vs engineer
- Examples
  - jQuery
  - LINQ in .NET
  - Promises in javascript
  - Observables and functional reactive programming

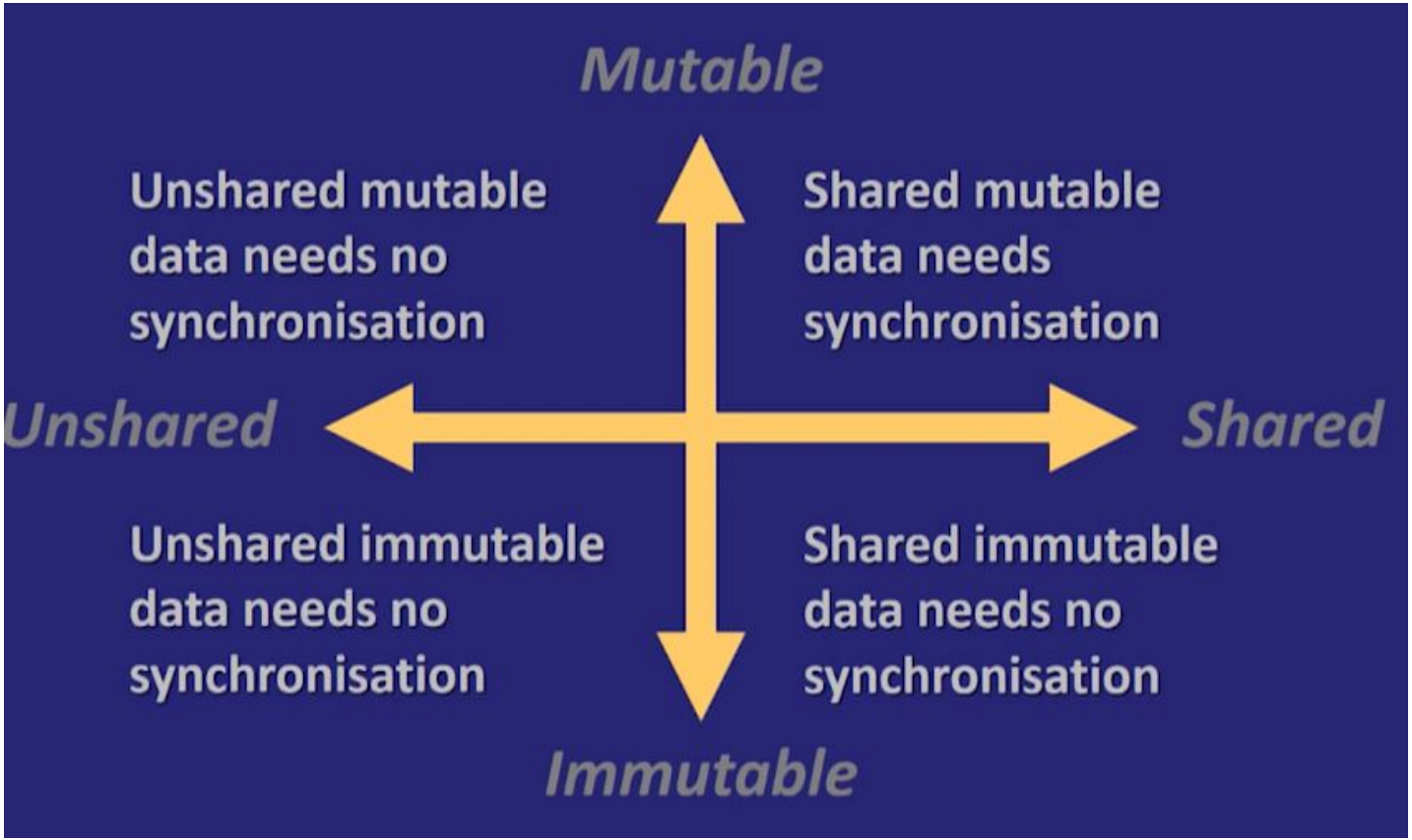
Imperative vs Declarative

- Imperative
  - Tell the computer “how” to do something
  - Loops
  - Counters
  - Extra variables to hold state
  - Working with details
  - Usually requires holding a great deal of the program inside a mental model
- Declarative
  - Ask the computer “what” to do
  - Rely on compiler / optimizers to build instructions
  - Work with abstractions
  - Goal is to make programs easier to think about with less mental model

Problem - Complexity

- Structured / Procedural
  - Decompose to smaller units
  - How to organize at higher levels?
- OO
  - Use Objects to model the problem
  - Encapsulation allows to work with abstractions
  - Higher level is organized with higher level abstractions
- FP
  - Decompose into small, pure functions
  - Compose pure functions into larger abstractions (higher order functions)
  - Higher level is typically modules

Problem - Shared Mutable State



Problem - null reference

- Structured / Procedural
  - Defensive programming
  - Check for null with if “guard” conditions
- OO
  - Null object
  - Example
    - Get person from database with an id
    - If (person == null) return Person.NullPerson;
- FP
  - Maybe type
  - Union of two types
    - Something
    - Nothing
  - getPerson(id): Maybe<Person>

Problem - Multiple Cores

- Structured / Procedural / OO
  - Threads
  - Locks
  - Resource contention
  - Race conditions
  - Deadlocks
- FP
  - Pure functions
  - Easy parallelization
  - Could be the future

Object Oriented vs Functional Programming

- Not a real conflict
  - They are solving problems with a different approach
  - Both paradigms can be used in the same codebase

- “OO makes code understandable by encapsulating moving parts. FP makes code understandable by minimizing moving parts.”
  - Michael Feathers

Break - Can we make it more functional?

- Read from JSON object
  - Need Lastname, Firstname of all people in California
  - Exclude San Diego
  - Offer sale for people in Sacramento
  - Billing department wants FIRSTNAME LASTNAME
  - Need a count of all potential customers from states outside California
- Now we need to read from CSV

Programmer’s Responsibility

- More time is spent reading code than writing it
- How do we make our code more readable?
- How can make our intentions more clear?
- How do we measure the quality of our code?
- What is our responsibility towards our fellow programmers?
- “If the first time your code is being debugged during a critical outage and the programmer concludes that it is easier to re-write it, you have failed.”