

## Recreation

Given that

$$\log(1+x) = x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \dots$$

why is it not the case that

$$\begin{aligned} \log 2 &= 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 - 1/8 + 1/9 - \dots \\ &= (1 + 1/3 + 1/5 + 1/7 + 1/9 + \dots) - (1/2 + 1/4 + 1/6 + 1/8 + \dots) \\ &= (1 + 1/3 + 1/5 + 1/7 + 1/9 + \dots) + (1/2 + 1/4 + 1/6 + 1/8 + \dots) \\ &\quad - 2(1/2 + 1/4 + 1/6 + 1/8 + \dots) \\ &= (1 + 1/2 + 1/3 + 1/4 + \dots) - (1 + 1/2 + 1/3 + 1/4 + \dots) \\ &= 0? \end{aligned}$$

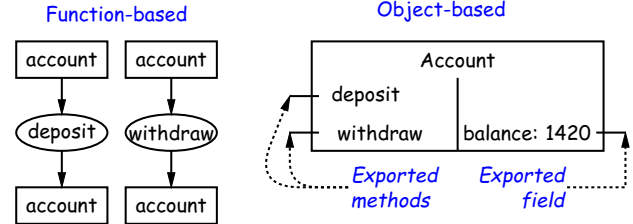
Last modified: Thu Sep 12 22:11:30 2019

CS61B: Lecture #7 1

## CS61B Lecture #7: Object-Based Programming

Basic Idea.

- **Function-based programs** are organized primarily around the functions (methods, etc.) that do things. Data structures (objects) are considered separate.
- **Object-based programs** are organized around the *types of objects* that are used to represent data; methods are grouped by type of object.
- Simple banking-system example:



Last modified: Thu Sep 12 22:11:30 2019

CS61B: Lecture #7 2

## Philosophy

- Idea (from 1970s and before): An **abstract data type** is
  - a set of possible values (a **domain**), plus
  - a set of **operations** on those values (or their containers).
- In IntList, for example, the domain was a **set of pairs**: (head, tail), where head is an int and tail is a pointer to an IntList.
- The IntList operations consisted only of assigning to and accessing the two fields (head and tail).
- In general, we prefer a purely **procedural interface**, where the functions (methods) do everything—no outside access to the internal representation (i.e., instance variables).
- That way, implementor of a class and its methods has complete control over behavior of instances.
- In Java, the preferred way to write the "operations of a type" is as **instance methods**.

Last modified: Thu Sep 12 22:11:30 2019

CS61B: Lecture #7 3

## You Saw It All (Maybe) in CS61A: The Account Class

```
class Account:
    balance = 0
    def __init__(self, balance0):
        self.balance = balance0

    def deposit(self, amount):
        self.balance += amount
        return self.balance

    def withdraw(self, amount):
        if self.balance < amount:
            raise ValueError \
                ("Insufficient funds")
        else:
            self.balance -= amount
            return self.balance

myAccount = Account(1000)
print(myAccount.balance)
myAccount.deposit(100)
myAccount.withdraw(500)
```

```
public class Account {
    public int balance;
    public Account(int balance0) {
        this.balance = balance0;
    }
    public int deposit(int amount) {
        balance += amount; return balance;
    }
    public int withdraw(int amount) {
        if (balance < amount)
            throw new IllegalStateException
                ("Insufficient funds");
        else balance -= amount;
        return balance;
    }
}

Account myAccount = new Account(1000);
print(myAccount.balance)
myAccount.deposit(100);
myAccount.withdraw(500);
```

Last modified: Thu Sep 12 22:11:30 2019

CS61B: Lecture #7 4

## You Also Saw It All in CS61AS

```
(define-class (account balance0)
  (instance-vars (balance 0))
  (initialize
    (set! balance balance0))

  (method (deposit amount)
    (set! balance (+ balance amount))
    balance)
  (method (withdraw amount)
    (if (< balance amount)
        (error "Insufficient funds")
        (begin
          (set! balance (- balance amount))
          balance))))

(define my-account
  (instantiate account 1000))
(ask my-account 'balance)
(ask my-account 'deposit 100)
(ask my-account 'withdraw 500)
```

```
public class Account {
    public int balance;
    public Account(int balance0) {
        balance = balance0;
    }
    public int deposit(int amount) {
        balance += amount; return balance;
    }
    public int withdraw(int amount) {
        if (balance < amount)
            throw new IllegalStateException
                ("Insufficient funds");
        else balance -= amount;
        return balance;
    }
}

Account myAccount = new Account(1000);
myAccount.balance
myAccount.deposit(100);
myAccount.withdraw(500);
```

Last modified: Thu Sep 12 22:11:30 2019

CS61B: Lecture #7 5

## The Pieces

- **Class declaration** defines a **new type of object**, i.e., new type of structured container.
- **Instance variables** such as balance are the simple containers within these objects (**fields** or **components**).
- **Instance methods**, such as deposit and withdraw are like ordinary (static) methods that take an invisible extra parameter (called **this**).
- The **new** operator creates (**instantiates**) new objects, and initializes them using constructors.
- **Constructors** such as the method-like declaration of Account are special methods that are used only to initialize new instances. They take their arguments from the **new** expression.
- **Method selection** picks methods to call. For example,

```
myAccount.deposit(100)
```

tells us to call the method named deposit that is defined for the object pointed to by myAccount.

Last modified: Thu Sep 12 22:11:30 2019

CS61B: Lecture #7 6

## Getter Methods

- Slight problem with Java version of Account: anyone can assign to the balance field
- This reduces the control that the implementor of Account has over possible values of the balance.
- Solution: allow public access only through methods:

```
public class Account {
    private int _balance;
    ...
    public int balance() { return _balance; }
    ...
}
```

- Now Account.\_balance = 1000000 is an error outside Account.
- (I use the convention of putting '\_' at the start of private instance variables to distinguish them from local variables and non-private variables. Could actually use balance for both the method and the variable, but please don't.)

Last modified: Thu Sep 12 22:11:30 2019

CS61B: Lecture #7 7

## Class Variables and Methods

- Suppose we want to keep track of the bank's total funds.
- This number is not associated with any particular Account, but is common to all—it is **class-wide**. In Java, "class-wide"  $\equiv$  static.

```
public class Account {
    ...
    private static int _funds = 0;
    public int deposit(int amount) {
        _balance += amount;
        _funds += amount; // or this._funds or Account._funds
        return _balance;
    }
    public static int funds() {
        return _funds; // or Account._funds
    }
    ... // Also change withdraw.
}
```

- From outside, can refer to either Account.funds() or to myAccount.funds() (same thing).

Last modified: Thu Sep 12 22:11:30 2019

CS61B: Lecture #7 8

## Instance Methods

- Instance method such as

```
int deposit(int amount) {
    _balance += amount;
    _funds += amount;
    return balance;
}
```

behaves sort of like a static method with hidden argument:

```
static int deposit(final Account this, int amount) {
    this._balance += amount;
    _funds += amount;
    return this._balance;
}
```

- NOTE: Just explanatory: Not real Java (not allowed to declare 'this'). (final is real Java; means "can't change once initialized.")

Last modified: Thu Sep 12 22:11:30 2019

CS61B: Lecture #7 9

## Calling Instance Method

```
/** (Fictional) equivalent of deposit instance method. */
static int deposit(final Account this, int amount) {
    this._balance += amount;
    _funds += amount;
    return this._balance;
}
```

- Likewise, the instance-method call myAccount.deposit(100) is like a call on this fictional static method:

```
Account.deposit(myAccount, 100);
```

- Inside a real instance method, as a convenient abbreviation, one can leave off the leading 'this.' on field access or method call if not ambiguous. (Unlike Python)

Last modified: Thu Sep 12 22:11:30 2019

CS61B: Lecture #7 10

## 'Instance' and 'Static' Don't Mix

- Since real static methods don't have the invisible this parameter, makes no sense to refer directly to instance variables in them:

```
public static int badBalance(Account A) {
    int x = A._balance; // This is OK
    // (A tells us whose balance)
    return _balance; // WRONG! NONSENSE!
}
```

- Reference to \_balance here equivalent to this.\_balance,
- But this is meaningless (whose balance?)
- However, it makes perfect sense to access a static (class-wide) field or method in an instance method or constructor, as happened with \_funds in the deposit method.
- There's only one of each static field, so don't need to have a 'this' to get it. Can just name the class (or use no qualification inside the class, as we've been doing).

Last modified: Thu Sep 12 22:11:30 2019

CS61B: Lecture #7 11

## Constructors

- To completely control objects of some class, you must be able to set their initial contents.
- A **constructor** is a kind of special instance method that is called by the new operator right after it creates a new object, as if

```
L = new IntList(1,null)  $\Rightarrow$   $\left\{ \begin{array}{l} \text{tmp} = \text{pointer to } \boxed{0} \\ \text{tmp.IntList}(1, \text{null}); \\ \text{L} = \text{tmp}; \end{array} \right.$ 
```

Last modified: Thu Sep 12 22:11:30 2019

CS61B: Lecture #7 12

## Multiple Constructors and Default Constructors

- **All** classes have constructors. In the absence of any explicit constructor, get **default constructor**, as if you had written:

```
public class Foo {  
    public Foo() { }  
}
```

- Multiple **overloaded** constructors possible, and they can use each other (although the syntax is odd):

```
public class IntList {  
    public IntList(int head, IntList tail) {  
        this.head = head; this.tail = tail;  
    }  
  
    public IntList(int head) {  
        this(head, null); // Calls first constructor.  
    }  
    ...  
}
```

Last modified: Thu Sep 12 22:11:30 2019

CS61B: Lecture #7 13

## Constructors and Instance Variables

- Instance variables initializations are moved inside constructors that don't start with `this(...)`.

<pre>class Foo {     int x = 5;      Foo(int y) {         DoStuff(y);     }      Foo() {         this(42);     } }</pre>	$\iff$	<pre>class Foo {     int x;      Foo(int y) {         x = 5;         DoStuff(y);     }      Foo() {         this(42); // Assigns to x     } }</pre>
--	--------	---

Last modified: Thu Sep 12 22:11:30 2019

CS61B: Lecture #7 14

## Summary: Java vs. Python

Java	Python
<pre>class Foo {     int x = ...;     Foo(...)     { ... }     int f(...)     { ... }     static int y = 21;     static void g(...)     { ... } }</pre>	<pre>class Foo: ...     x = ...     def __init__(self, ...):         ...     def f(self, ...):         ...     y = 21 # Referred to as Foo.y     @staticmethod     def g(...):         ...</pre>
<pre>aFoo.f(...) aFoo.x new Foo(...) this</pre>	<pre>aFoo.f(...) aFoo.x Foo(...) self # (typically)</pre>

Last modified: Thu Sep 12 22:11:30 2019

CS61B: Lecture #7 15