CSSE 220 Day 5

Implementing Classes in Java

Check out *ImplementingClasses* from SVN

Questions?

Today

- Encapsulation
- Java classes:
  - Implementation details
  - "How To" example

Encapsulation in Object-Oriented Software

▸ *Encapsulation*—separating implementation details from how an object is used
  ◦ Client code sees a *black box* with a known *interface*
  ◦ Implementation can change without changing client

	Functions	Objects
Black box exposes	Function signature	Constructor and method signatures
Encapsulated inside the box	Operation implementation	Data storage and operation implementation

Q1, 2

Bank Account Example

- Essentially based on *Big Java*
  - But using explicit `this` references
  - And putting fields at the top of the class
- Comparing and contrasting with Python
- Source code with Python examples in SVN for reference

Class Definitions

```java
/** javadoc… */
public class BankAccount {
    …
}
```

```python
class BankAccount:
    """docstring..."""
    …
```

Access specifier, one of:
public,
protected,
private, or
default (i.e., no specifier)

Java classes are usually declared public

Java

Python

Method Definitions

```java
/** javadoc… */
public void deposit(double
                    amount) {
   …
}
```

```python
def deposit(self, amount):
    """docstring..."""
    …
```

Access specifier

Return type

Parameters with types, do not list "self"

Java methods usually are a mix of public and private

Java

Python

Q4,5

Constructor Definitions

```java
/** javadoc... */
public BankAccount() {
    ...
}

/** javadoc... */
public BankAccount(double
                   initAmt) {
    ...
}
```

```python
def __init__(self,
             initAmt=0.0):
    """docstring..."""
    ...
```

Access specifier

Parameters with types, do not list "self"

No explicit return type

Constructor name is same as class name

Use *overloading* to handle default argument values

Java

Java constructors usually public

Python

Public Interface

- The *public interface* of an object is:
  - ◦ `public` constructors of its class, plus
  - ◦ `public` methods of its class
- The inputs and outputs of the black box
- Defines how we access the object as a user

BankAccount

`BankAccount()`
`BankAccount(double initAmt)`
`void deposit(double amount)`
`void withdraw(double amount)`
`double getBalance()`

Q9

Instance Field Definitions

```
/** javadoc… */
private double balance;
```

No instance field definitions in Python

Access specifier

Type

Name

Java instance fields usually private

An object is an *instance* of a class

Java

Python

Constructor Implementation

```java
/** javadoc… */
public BankAccount(double
                   initAmt) {
    this.balance = initAmt;
}
```

```python
def __init__(self,
                 initAmt=0.0):
    """docstring..."""
    self.balance = initAmt
```

Use `this` inside constructors and methods to refer to implicit argument

Java

Python

Method Implementation

```java
/** javadoc… */
public double getBalance()
{
    return this.balance;
}


/** javadoc… */
public void deposit(double
    amount) {
    double newBal =
        this.balance + amount;
    this.balance = newBal;
}
```
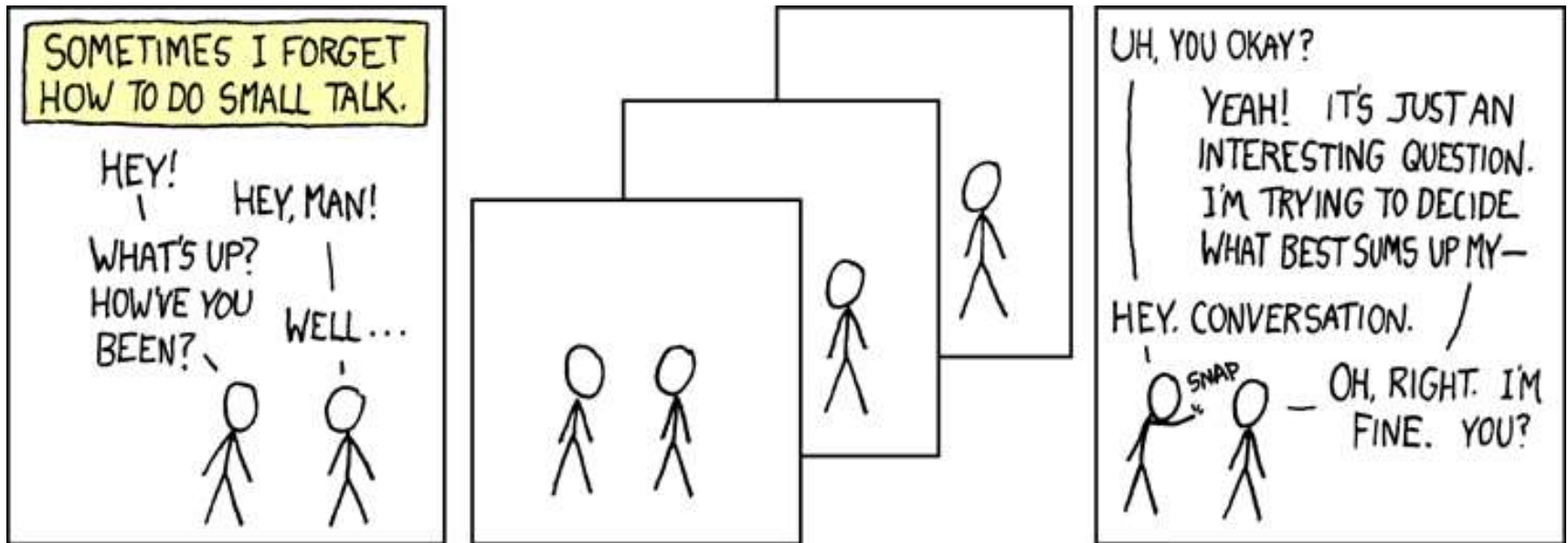
Java

```python
def getBalance(self):
    """docstring..."""
    return self.balance




def deposit(self, amount):
    """docstring..."""
    newBal =
        self.balance + amount
    self.balance = newBal
```

Python

Can omit return
for void methods

How To: Do Small Talk

But surely I owe you an accurate answer!

How To: Implement a Class

1. Find out which methods you are asked to supply
2. Specify the public interface
3. Document the public interface
4. Determine instance fields
5. Implement constructors and methods
6. Test

5. Test and implement each constructor and method

Live Coding

>> Implement a class that draws a face of a given size at a given location. You should also be able to mutate it and test it.

Lots of Faces

▸ Once you've got Face tested, implemented, and debugged…
  ◦ Change FacesComponent to draw lots of faces

▸ Add angle to Face
  ◦ See details in the homework problem