Terms

Class

A blueprint to create objects
Includes attributes and methods that the created objects all share

Object

Allocated region of storage
Both the data and the instructions that operate on that data
Ruby code. Yes I know most people don't know Ruby. There are lots of variation of syntax and semantics for classes in OO languages.
Alternative Definition

Object
First-class, dynamically dispatched behavior

Behavior
Collection of named operations
Operations can be invoked by clients
Operations may share additional hidden details

Dynamic dispatch
Different objects can implement the same operation name(s) in different ways

First class
Objects have the same capabilities as other kinds of values
Passed to operations
Returned as the result of an operation
I’m sorry that I long ago coined the term “objects” for this topic because it gets many people to focus on the lesser idea.

OOP to me means only
  messaging,
  local retention and protection and hiding of state-process, and
  extreme late-binding of all things.

The big idea is “messaging”

The key in making great and growable systems is much more to design how its modules communicate rather than what their internal properties and behaviors should be.
int x;

x = 10;

List y = new ArrayList<String>();

y.add("cat");
public void removeCat(List<String> animals) {
    for (int k = 0; k < animals.size(); k++) {
        String animal = animals.get(k);
        if (animal.toLowerCase().equals("cat"))
            animals.remove(k)
    }
}

ArrayList<String> a = new ArrayList<String>();

LinkedList<String> b = new LinkedList<String>();

something.removeCat(a)

something.removeCat(b)
Abstraction

“Extracting the essential details about an item or group of items, while ignoring the unessential details.”
   Edward Berard

“The process of identifying common patterns that have systematic variations; an abstraction represents the common pattern and provides a means for specifying which variation to use.”
   Richard Gabriel
Encapsulation

Enclosing all parts of an abstraction within a container
Information Hiding

Hiding of design decisions in a computer program

Hide decisions are most likely to change,
To protect other parts of the program
Class

Represents an abstraction

Encapsulates data and operations of the abstraction

Hide design decisions/details
Heuristics

2.1 All data should be hidden within its class

2.8 A class should capture one and only one key abstraction

2.9 Keep related data and behavior in one place
Non-OO items

Utility methods

Data classes
Utility method

Method in class that
  Does not access any field (data member, instance variables)
  Just uses parameters
Utility Method - Example

class CrosswordPuzzle {
    public void someMethodThatDoesStuff {
        bunch of stuff not shown
        count = vowelCount(aString);
        blah
    }
}

private int vowelCount(String word) {
    int vowelCount = 0;
    for (int k = 0; k< word.length(); k++) {
        char current = word.charAt(k);
        if ( (current == 'a') || (current == 'e') || (current == 'i') || (current == 'o') || (current == 'u') )
            vowelCount++;
    }
    return vowelCount;
}
class String {
  public int vowelCount {
    int count = 0;
    for (char current in this)
      if (current.isVowel()) count++;
    return count;
  }
}

class Character {
  public boolean isVowel() {
    return (this == 'a') || (this == 'e') || (this == 'i') || (this == "o") || (this == "u");
  }
}

class CrosswordPuzzle {
  public void someMethodThatDoesStuff {
    bunch of stuff not shown
    count = aString.vowelCount();
    blah
  }
}
Kent Beck's Properties of Good Style
Kent Beck's Properties of Good Code Style

Once and only once

Lots of little pieces

Replacing objects

Moving Objects

Rates of change
Once and Only Once

"In a program written with good style, everything is said once and only once"

If have
  several methods with same logic
  several objects with same methods
then rule is not satisfied
Lots of little pieces

"Good code invariably has small methods and small objects"

Small pieces allow you to satisfy "once and only once"
Data Class

```java
class Point {
    private int x;
    private int y;

    public void setX(int newX) {
        x = newX;
    }

    public int getX() {
        return x;
    }

    public void setY(int newY) {
        y = newY;
    }

    public int getY() {
        return y;
    }
}
```

Class with get/set methods
constructor
No or very few other methods
How is Bar Better than Foo

public class Foo {
    public int x;
    public int y;
}

gpublic class Bar {
    private int x;
    private int y;

    public int getX() {return x;}
    public int getY() {return y;}

    public void setX(int newX){
        x = newX
    }

    public void setY(int newY){
        y = newY
    }
}
Principles of OO Design, or Everything I Know About Programming, I Learned from Dilbert

Alan Knight
I'M ALREADY USELESS, BUT I'M THINKING ABOUT BECOMING TOXIC AS WELL.

THAT SEEMS AMBITIOUS FOR YOU.

THINK IT THROUGH.

AS A USELESS PERSON, I STILL GET INVITED TO MEETINGS BECAUSE I DON'T CAUSE MUCH TROUBLE.

BUT IF I GO FULL-TOXIC, NO ONE WILL INVITE ME TO MEETINGS IN THE FIRST PLACE.

I CAN AVOID A LOT OF WORK BY NIPPING IT IN THE BUD.

IS IT HARD TO BE TOXIC? HOW DO YOU DO IT?

IT'S EASY.

ALL YOU DO IS PROVIDE INCOMPLETE INFORMATION THAT MAKES PEOPLE ANXIOUS AND HATEFUL.

I CAN'T TELL YOU WHAT WAS SAID IN THAT LAST MEETING, BUT I DEFENDED YOU.
1. Never do any work that you can get someone else to do for you

Excuse me Smithers. I need to know the total bills that have been paid so far this quarter. No, don’t trouble yourself. If you’ll just lend me the key to your filing cabinet I’ll go through the records myself. I’m not that familiar with your filing system, but how complicated can it be? I’ll try not to make too much of a mess.

Verses

SMITHERS! I need the total bills that have been paid since the beginning of the quarter. No, I’m not interested in the petty details of your filing system. I want that total, and I’ll expect it on my desk within the next half millisecond.
Encapsulation & Responsibility

Encapsulation is about responsibility

Who does the work

Who should do the work
2. Avoid Responsibility

If you must accept a responsibility, keep it as vague as possible.

For any responsibility you accept, try to pass the real work off to somebody else.

class TernarySearchTree {
    public void insert(String word) {
        root.insert(new StringIterator(word));
    }
}
What Compsci textbooks don't tell you

What don't they tell you?

nearly every sample program in every textbook is a perfect and well-thought-out specimen, virtually no software out in the wild is, and this is rarely acknowledged
What are the causes of bad Software?

bad programmers, too good programmers, bad laziness, time
What is the simple fix?
What is a Big Ball of Mud?

How many have worked on a Big ball of mud?
What caused it?
What was the impact of the Big ball of mud?
What Forces Lead to Big Ball of Mud

Time, Cost, Experience, Skill, Visibility, Complexity, Change, Scale
Patterns

Big Ball of Mud
Throwaway Code
Piecemeal Growth
Keep it Working
Shearing Layers
Sweeping it Under the Rug
Reconstruction
Big Ball of Mud

You need to deliver quality software on time, and under budget.

Therefore, focus first on features and functionality, then focus on architecture and performance.
Enemy of Big Ball of Mud

Top down design

Hire good architects
Variable and function names
  uninformative

Functions themselves may make extensive use of
  global variables,
  long lists of poorly defined parameters.

The function themselves are
  lengthy and convoluted,
  perform several unrelated tasks.

The programmer’s intent is next to impossible to discern.
We built the most complicated system that can possibly work
Three ways to deal with BIG BALLS OF MUD

- Keep it healthy – expansion then refactoring
- Throw it away
- Live with it
Extreme Programming Practices

Pair programming
Planning game
Test driven development
Customer part of development team
Continuous integration
Refactoring or design improvement
Small releases
Coding standards
Collective code ownership
Simple design
System metaphor
Sustainable pace
Throwaway Code

You need an immediate fix for a small problem, or a quick prototype or proof of concept.

Therefore, produce, by any means available, simple, expedient, disposable code that adequately addresses just the problem at-hand.
Why do we need throwaway code?

What is the main problem with throwaway code?
Piecemeal Growth

Users’ needs change with time.

Therefore, incrementally address forces that encourage change and growth.

Allow opportunities for growth to be exploited locally, as they occur.

Refactor unrelentingly.
What is the main problem with Piecemeal Growth?
Keep it Working

Maintenance needs have accumulated, but an overhaul is unwise, since you might break the system.

Therefore, do what it takes to maintain the software and keep it going. Keep it working.
How do Piecemeal Growth and Keep it Working lead to a ball of mud?

How can we use Piecemeal Growth and Keep it Working and avoid the ball of mud?

Is it advisable to use Piecemeal Growth and Keep it Working?
Shearing Layers

Different artifacts change at different rates

Therefor

Factor your system so that artifacts that change at similar rates are together
Why?

Put things that change at different rates in different places?

Example?
Sweep it Under the Rug

Overgrown, tangled, haphazard spaghetti code is hard to comprehend, repair, or extend, and tends to grow even worse if it is not somehow brought under control.

Therefore, if you can’t easily make a mess go away, at least cordon it off.

This restricts the disorder to a fixed area, keeps it out of sight, and can set the stage for additional refactoring.
Reconstruction

Your code has declined to the point where it is beyond repair, or even comprehension.

Therefore, throw it away and start over.
"Plan to throw one away, you will anyway"

Fred Brooks
Problems with Starting Over

Cost

Time

Reintroduce bugs

Few features