Software engineering

Is more than programming

- Construction
- Durability
- Reuse
- Delivery

Is actually a process

- Design
- Development
- Maintenance

Software engineering

- Application of engineering practices to the construction of computer programs
- Engineers develop processes that are
  - Reproducible
  - Sound
  - Durable

Engineering versus hacking

- We've written code, but does that make us an engineer?
- (Typical) hacker: it's good enough for now. And good enough for me.
  - Enthusiasts
  - Graduate students
  - Researchers
- Engineering: getting results of a required quality within a specified time and budget

This semester

- You move from being a hacker to being an engineer
- Understand and appreciate the engineering aspects
- Integrate them into your current hackery
- Different techniques for different projects:
  - Games
  - Safety-critical control

Software engineering as a class

- Tough
  - Advanced coding
  - Software lifecycle management
- Difficult to appreciate advanced topics without years of experience
- Challenge to integrate
  - Deadlines with customers and budgets
  - Software written in large teams

Admin

- Office: A2-195
- Office hours:
  - Mon 5—6
  - Tues 4—6
  - Also by appointment/as needed
- Grading a combination of homeworks (~5) and final project

Schedule

- Focus on building Java knowledge
- Second half more project/special topic focused
- No tests

Week	Activity
2	Java: basics
3	Build systems, repositories
4	Design patterns
5	Testing, Java: special topics
6	Design patterns
7	Java: special topics
8	Projects: specifications
9	Software lifecycles
10	Guest speakers
11	Presentations: tools
12	TBA
13	TBA
14	Projects: presentations

Assumptions

- “Decent” programming ability
- Familiarity with object oriented programming
- Strong desire to figure things out

Procedural programming

- Primitive data types
- Functions defined that operate on those data types
- Functions are global

Object oriented programming

- Data types that are abstractions of primitives
- Combine attributes and behavior
- *Makes reasoning about interaction much more straightforward*

Classes

- Blueprint
- Define
  - Attributes
  - Methods
  - Access rules

Objects

- Instances of classes
- Instantiation of attributes makes objects of the same type unique

OOP: a review (hopefully)

- Object instances are distinguished from class definitions
- An object is a data structure containing:
  - Attributes
  - Methods
- Distinguishing characteristics of OOP
  - Encapsulation
  - Polymorphism
  - Inheritance

Encapsulation

- Mechanism to
  - bundle data and methods into a single structure
  - selectively restrict access to (those) fields

```
public class A {
    private int a;
    protected float b;

    private f() { … }
    public g() { … }
}
```

```
class A:
    def __init__(self):
        self.__a = 0
        self._b  = 1

    def __f(self): …
    def g(self): …
```

Polymorphism

- Operations specified on one type are valid on another
  - Typically: specified on type A, but used with type B, where B is a refinement, or subtype of A

Inheritance

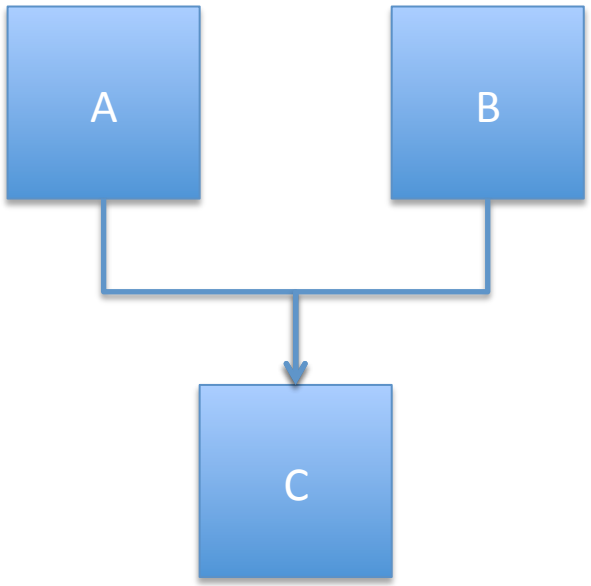
- Creating a new class that utilizes the implementation of another
- Creates hierarchy within the design
  - Child assumes the parents implementation by default
  - Can override the parent if necessary

```
public class B extends A { … }
```

```
class B(A):
```

Multiple inheritance

- In some languages it is possible to inherit from more than one class
  - Generally have well-defined rules about the precedent of parent implementation
- *Interfaces* are a means of getting around multiple-inheritance



Interface

- Specification of members
  - No implementation!
- Users of the interface must implement the specification
- Operate as types when building methods, but cannot be instantiated

```
public interface A { … }

public class B implements A
{ … }
```

Class/Interface distinction

Class is a noun

- Represents a class of persons, places, or (usually) things
- An instance is a member of the class
  - AbuDhabi is a City

Interface is an adjective

- Represents properties and capabilities
- Can be applied to classes
- A Comparable requires a compareTo

Abstract class: a tweener

- Like an interface: specifies behavior
- Like a class: can contain code; can be inherited
  - Cannot be instantiated!

```
abstract class A {
    private int a;
    protected float b;

    private f() { … }
    abstract g();
    abstract h();
}
```

```
class A:
    def g(self):
        raise NotImplementedError

    def h(self):
        pass
```

When to use

Interface

- A behavior or property likely to be used for possibly unrelated classes
- (Need multiple inheritance)

Abstract class

- Behavior or property that is specific to a conceptual class of nouns

- Language enforcement will vary!
- Understanding the concepts will allow you to use the constructs regardless

Composition

- A way to aggregate objects into something more complex
- An alternative to inheritance
  - inheritance: is-a (what they are)
  - composition: has-a (what they do)
- Inheritance is bad when have lots of functions that don't use
  - Multi-inheritance that doesn't make sense

The basics are important

- Object interaction allows software to be robust and malleable
- Good software engineers understand
  - The various relationships between objects
  - How to specify them conceptually
  - How to implement them physically
  - When, where, and why to use them
- The right answer depends on the situation!

CAREERS

CAREERS; New Job: Software Engineer

By Elizabeth M. Fowler
Published: October 15, 1985

COMPUTER software engineering is not a profession yet, but it will be if John H. Manley has his way. A former Air Force expert, he has just become the director of the Software Engineering Institute, financed by the Government.

The Defense Department has awarded a $103 million, five-year contract to Carnegie-Mellon University to manage the institute, which was established early this year.

What is a software engineer? Dr. Manley defines the job as "involving the planning, developing and evolution of software systems and products" for inclusion in the technical and management structure of companies. Such engineers are not to be confused with people whose work involves computer science, such as systems analysts. The software engineer needs a background not only in engineering - specifically in software systems - but also in accounting, economics, cost management and other business-type subjects.