Some Mini Wrapup

Hsuan-Tien Lin

Department of CSIE, NTU

OOP Class, April 22, 2013

- Professor, MIT
- 2004 IEEE John von Neumann Medal (who is von Neumann?)
- 2008 ACM A. M. Turing Award (who is Turing and what is Turing Award?)

> *For contributions to practical and theoretical foundations of programming language and system design, especially related to data abstraction, fault tolerance, and distributed computing.*

Barbara Liskov and OOP

- The CLU language

```
complex_number = cluster is add, subtract, multiply, ...
    rep = record [ real_part: real, imag_part: real ]
    add = proc ... end add;
    subtract = proc ... end subtract;
    multiply = proc ... end multiply;
    ...
end complex_number;
```

```
1       class complex_number{
2         double real_part; double imag_part;
3         ... add (...) { ... }
4         ... substract (...) { ... }
5         ... multiply (...) { ... }
6       }
```

a pioneering OOP language

Barbara Liskov and OOP

- The Liskov substitution principle

Let q(x) be a property provable about objects x of type T. Then q(y) should be true for objects y of type S where S is a subtype of T.

> Java: *S* extends *T* means
> (*y* of type *S*) **is an** (object of type *T*) [but more subtle than that]

http://en.wikipedia.org/wiki/Circle-ellipse_problem

- immutage ones?
- mutable ones? what happens after `stretchX`?
- solution? what if `Ellipse extends Circle`?

Inheritance in a Nutshell

- motivation: use subtyping to save repeated efforts in code writing and (to accelerate future code writing)
- top-down view: from general classes to specialized ones
- bottom-up view: gather similar code pieces to a higher level
- axiom: LSP
- (important) details: what gets inherited? which part gets accessed (called)?

Polymorphism in a Nutshell

- motivation: use parent type as an entry point for accessing (possibly future) subtypes
- object have their own characteristics (behavior, action) based on their run-time type, not their compile-time type
- mechanism: method overriding
- (important) details: what gets called?

S.O.L.I.D. Principles

- Single Responsibility: "a class should have only a single responsibility" (abstraction)
- Open/Closed: "software entities should be open for extension, but closed for modification" (polymorphism)
- Liskov Substitution: "objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program" (inheritane)
- Interface Segregation (will be discussed later)
- Dependency Inversion (will be discussed later)