Polymorphism

11/09/2015

Hsuan-Tien Lin (林軒田)
htlin@csie.ntu.edu.tw

Department of Computer Science
& Information Engineering

National Taiwan University
(國立台灣大學資訊工程系)

Summary on Polymorphism

- one thing, many shapes
- important in strongly-typed platforms with inheritance
- view from content: one advanced content with many compatible access
- view from reference: one compatible reference can point to many advanced contents
- view from method: one compatible method “contract”, many different method “realization”

Abstract Class (1/3)

```
public class Professor(){
  public void teach(){
    System.out.println("not␣sure␣of␣what␣to␣teach!");
  }
}
class CSIEProfessor extends Professor{
  private void teach_oop(){ /* lalala */ }
  public void teach(){ teach_oop(); }
}
class EEProfessor extends Professor{
  private void teach_elec(){ /* lululu */ }
  public void teach(){ teach_elec(); }
}
//in other places
Professor p = new Professor();
p.teach(); //not sure of what to teach!
```

- `teach` is a place-holder in Professor, expected to be overridden
- allows constructing a professor without any teaching ability! —absurd in some sense

Abstract Class (2/3)

```
public abstract class Professor(){
  public abstract void teach();
}
class CSIEProfessor extends Professor{
  private void teach_oop(){ /* lalala */ }
  public void teach(){ teach_oop(); }
}
class EEProfessor extends Professor{
  private void teach_elec(){ /* lululu */ }
  public void teach(){ teach_elec(); }
}
//in other places
Professor p = new Professor(); //hahaha!!
Professor p = new CSIEProfessor(); //okay
```

- `teach` is a place-holder in Professor, expected to be overridden
- but **cannot construct a pure Professor instance anymore**!

Abstract Class (3/3)

- abstract method [method not implemented]
  $\Rightarrow$ abstract class [cannot construct instance]?
- abstract class $\Rightarrow$ abstract method?
- public abstract method?
- proteced abstract method?
- private abstract method?
- keep being an abstract method in the subclass?
- concrete method(s) in an abstract class?
- instance variable(s) in an abstract class?
- static field(s) in an abstract class?
- constructor(s) in an abstract class?
- reference to an abstract class?

Key Point: Abstract Class

a **contract** for future extensions

Final Words

- `static final` variable: accessed through class, and assigned once (in declaration or static constructor)
- `final` instance variable: accessed through instance, and assigned once (in declaration or every instance constructor)
- `final` instance method: cannot be overriden ($\approx$ assigned once)
- `static final` method: cannot be hidden by inheritance ($\approx$ assigned once)
- `final` class: cannot be inherited (and hence all methods final)

Is `java.lang.Object` abstract?

Who Is She?

Barbara Liskov

- 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;
```

```
    class complex_number{
      double real_part; double imag_part;
      ... add(...){ ... }
      ... substract(...){ ... }
      ... multiply(...){ ... }
    }
```

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]

Is Circle an Ellipse?

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

- immutable 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)