Type Polymorphism

V22.0102 Data Structures

Feb 1, 2011

New York University
Chanseok Oh (chanseok@cs.nyu.edu)

- Contents
  - Overloading
  - Type Polymorphism
    - Main focus of this recitation
  - Subtype Polymorphism
    - Will be discussed briefly
    - Polymorphism in object-oriented programming

- Overloading
  - Has nothing to do with "overriding."
    - They are totally different concepts.
    - We do not discuss overriding today.

  - Defining several methods with the same name
    - In the same class (in OO-language)
    - Methods differ in the (input or output) type.

```
class Number {
        public int add(int a, int b);
        public Number add(Number a, Number b);
}
```

- Overloading (Cont'd)
  - More examples
    - Different number of arguments

      void increase()

      void increase(int inc)

    - Different types of input and output arguments

1 + 1	→	2	(integer)
2.4 + 3.6	→	6.0	(float)
3 + .141592	→	3.141592	(float)
1/2 + 1/3	→	5/6	(rational)
[1, 2] + [3, 4, 5]	→	[1, 2, 3, 4, 5]	(list)
"str" + "ing"	→	"string“	(string)

- Type Polymorphism (Cont’d)
  - “Having multiple forms”
  - A programming language feature
    - Allowing values of different data types to be handled using a uniform interface
  - A way to make a language more expressive
  - We will see examples of
    - polymorphic functions
    - polymorphic data types

- Type Polymorphism (Cont'd)
  - A polymorphic function

```
function identity (anything) {
        return anything
} // This language is not a statically-typed one.
identity ("string")     : evaluates to (or returns) "string"
identity (3.14)         : 3.14
identity (obj)          : the object obj itself
```

  - A polymorphic data type

```
class Array // Assume this is polymorphic.
Array arr_of_int        = [1, 2, 3, 4]
Array arr_of_char       = ['a', 'b', 'c', 'd']
Array arr_of_obj        = [obj1, obj2, obj3]
```

- Type Polymorphism (Cont’d)
  - In Java, you can achieve polymorphism, for instance,
    - By overloading

```
class PrettyPrinter {
        public void Print(Text t);
        public void Print(Image i);
        public void Print(List l);
}
```

    - By generics

```
List<String> ls = new ArrayList<String>();
List<Integer> li = new LinkedList<Integer>();
```

- Type Polymorphism (Cont'd)
  - In Java,
    - Of course, you can do it superficially,

```
public void Print(Object o) {
        if (o instanceof Text) { … }
        else if (o instanceof Image) { … }
} // Does not benefit from static type checking

// Prior to Java 1.5 (no generics),
List l = new LinkedList();
l.add(new String("add accepts any Object"));
String s = (String) l.getFirst(); // Downcasting
```

- Subtype Polymorphism
  - Another notion different from the type polymorphism we have seen so far.
  - This is related to (but not necessarily) subclassing (or inheritance).
  - However, this is almost universally called just polymorphism in the context of object-oriented language.

- Subtype Polymorphism (Cont’d)

```
interface Person {
    abstract public void work();
}
class Student implements Person {
    public void work() { doze(); }
    …
}
class Instructor implements Person {
    public void work() { teach(); }
    …
}
```

- Subtype Polymorphism (Cont’d)

```
Person p = someone;

// Will either doze or teach
// depending on its actual class
// determined at run-time.
p.work();
```

- Summary
  - Overloading
  - Type Polymorphism
    - Main focus of this recitation
  - Subtype Polymorphism