

Type Hierarchy

EE 564

Lecture 6

Daqing Hou, Winter 2007

Today:

- Type hierarchy
- Two uses of type hierarchy
- Kinds of supertypes
- The substitution principle

Type hierarchy

- Type hierarchy defines a group of types, making it easier to understand the group as a whole.
- Using code is written in terms of the supertype yet works for all subtypes.
- Type hierarchy allows for addition of new subtypes that still work with existing using code.
- Subtypes must obey the substitution principle to achieve these benefits.

Two uses of type hierarchy

- Provides multiple implementations for the same abstraction. E.g.:
 - Set: HashSet, TreeSet
- Defines new subtypes. E.g.:
 - IntSet: MaxIntSet, MinIntSet
 - Widget: TextField, Label, Panel

Kinds of supertypes

- **Complete supertypes:** provide entire data abstractions, with useful specifications for all methods.
- **Snippets:** provide a few methods that allows using code to be written in terms of these methods. Not data abstractions though.
- **Incomplete supertypes:** establish naming conventions for subtype methods but provide no useful specifications. No using code can be written in terms of incomplete supertypes.

Substitution principle

- Subtypes must satisfy a set of properties so that using code can be written and reasoned about using the supertype specification. I.e. for all subtypes, the following must always work:
Sup o;
o.e(...)
- These properties are summarized by the Liskov substitution principle.

Substitution principle

- **Signature rule:** the subtype must have all methods of the supertype, and the signatures of the subtype methods must be *compatible* with the corresponding supertype methods.
- **Method rule:** calls of subtype methods must *behave like* the corresponding supertype methods.
- **Property rule:** the subtype must preserve all properties that can be proved about supertype objects.

Signature rule

- Subtype must define all supertype methods.
- Signature of a subtype method must be *compatible* with the supertype method.
- Compatibility of two signatures is defined in terms of return type; method name; number, order, and types of parameters; and possible exceptions.
- Method names must be the same
- Enforced by compilers.

Signature rule

- **Return type** of a subtype method must be a subtype of the return type of the supertype method.
- **Number of parameter** must be the same.
- **Type of a parameter** of a supertype method must be a subtype of that of the subtype method (including for them to be the same).
- **Exceptions** a subtype method may throw must be a subset of that of the supertype.

Signature rule

- Early Java requires identical return types, as described in the book.
- Newer versions of Java implement the correct rule for return type.
- Parameter type rule is harder to implement. Now identical types are required.

Method rule

- **Method rule:** calls of subtype method must *behave like* the corresponding supertype method.
- **Precondition rule:**
 $\text{pre_sup} \Rightarrow \text{pre_sub}$
- **Postcondition rule:**
 $\text{pre_sup} \ \& \ \text{post_sub} \Rightarrow \text{post_sup}$
- $\text{pre_}m$: pre-condition of method m
 $\text{post_}m$: post-condition of method m

Method rule: Patterns of use

- Identical pre- and post-conditions
- “Weaker” pre-condition
- “Stronger” post-condition
- Exceptions

Property rule

- **Property rule:** subtype must preserve all properties that can be proved about supertype.
- Invariant properties
 - hold for each individual state.
 - e.g. size of a set is greater than or equal to 0.
- Evolution properties
 - involve two or more states.
 - e.g. size of a set after insertion either remains unchanged or increments by 1.