CS3500: Object-Oriented Design Spring 2014

Class 18

3.18.2014

Northeastern University
College *of* Computer and Information Science

CCIS Upper-Class Tutoring

- Effective Monday March 17, CCIS Upper-Class Tutoring is cancelled for Mondays & Thursdays. Tuesdays & Wednesday hours remain as posted.
  - Tuesday 6-9pm Eric Chin
  - Wednesday 7-10pm Matthew Manomivibul
- CCIS is working on the possibility of adding additional hours for Spring semester—stay tuned to http://www.ccs.neu.edu/undergraduate/tutoring/ for details.

Assignment 9

Two parts:

- Part 1 - Benchmarking: due Friday, March 21, 2014 at 9:50am paper copy in class
- Part 2 - Timing Testing: due Friday, March 21, 2014 at 11:59pm via Web-CAT

Assignment 10

- Will be posted on Friday
- Group assignment - groups of 3 or 4
- Friday in class you will sign up as groups

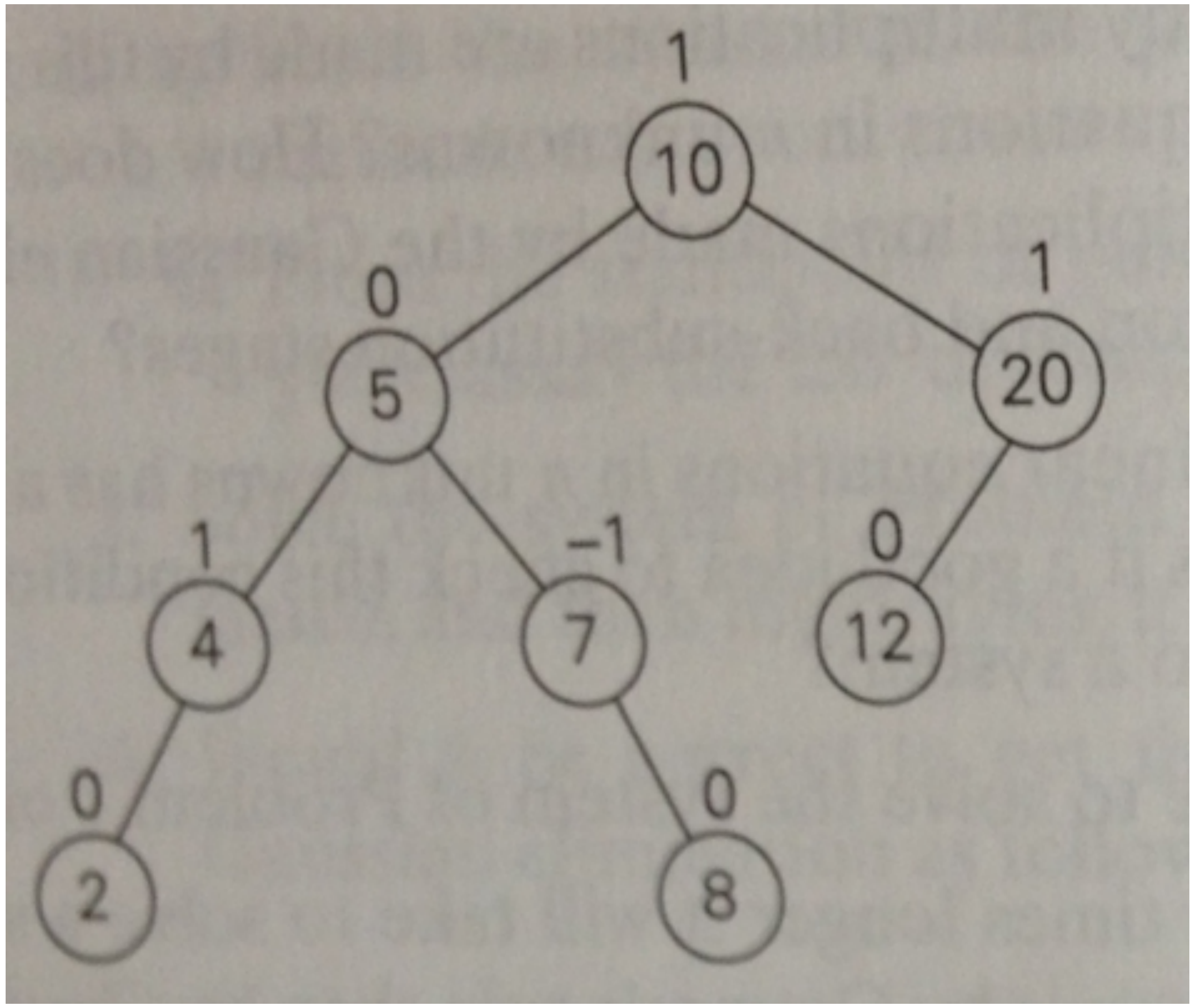
Student Code Review

Other Approaches to Balanced Trees

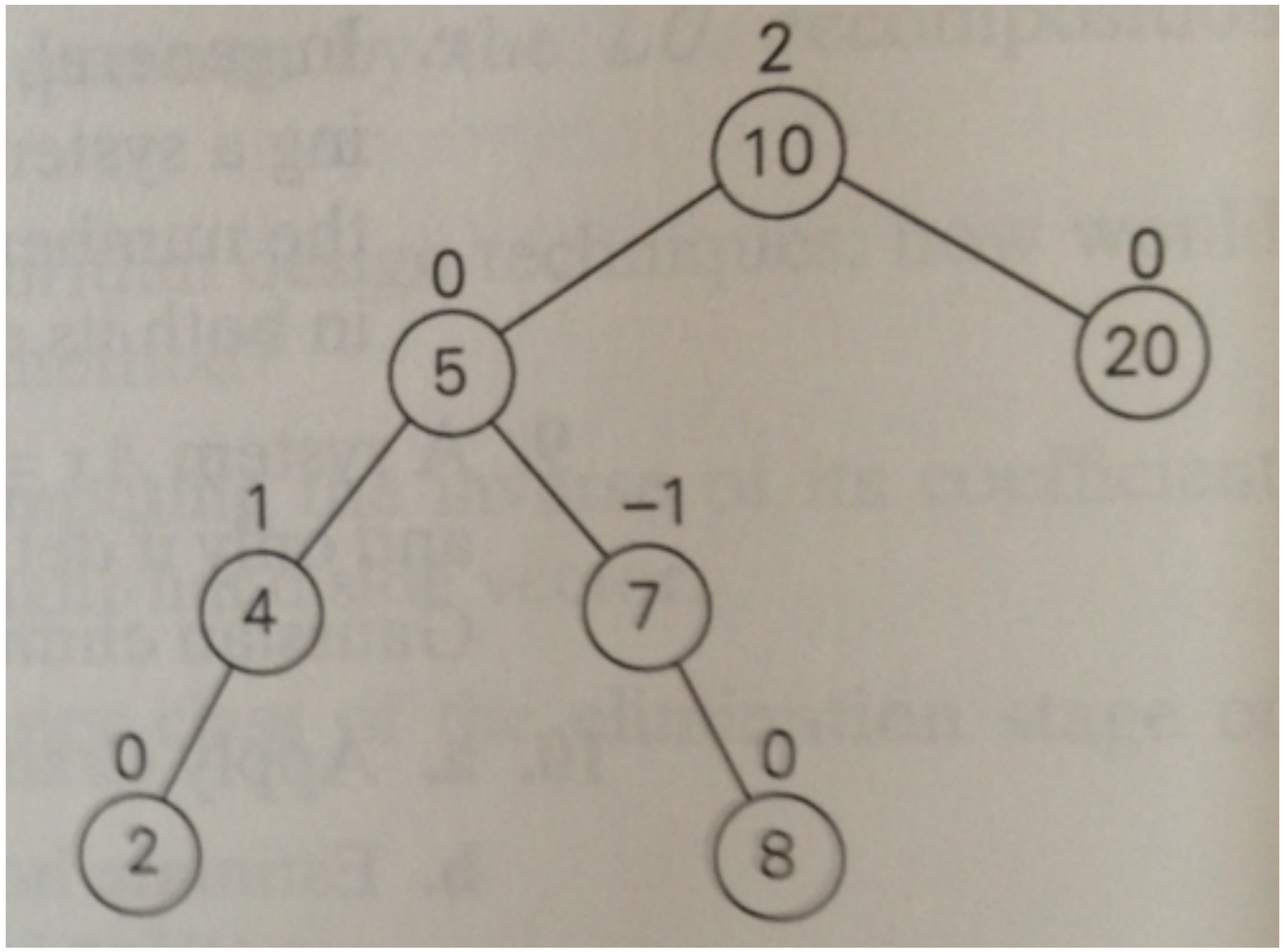
- AVL Trees
- 2-3 Trees

AVL Trees
[Levitin]

An *AVL tree* is a binary search tree in which the *balance factor* of every node, which is defined as the difference between the heights of the node's left and right subtrees, is either 0 or +1 or -1. (The height of the empty tree is defined as -1.)



AVL Tree



Not AVL Tree

Figures from Levitin

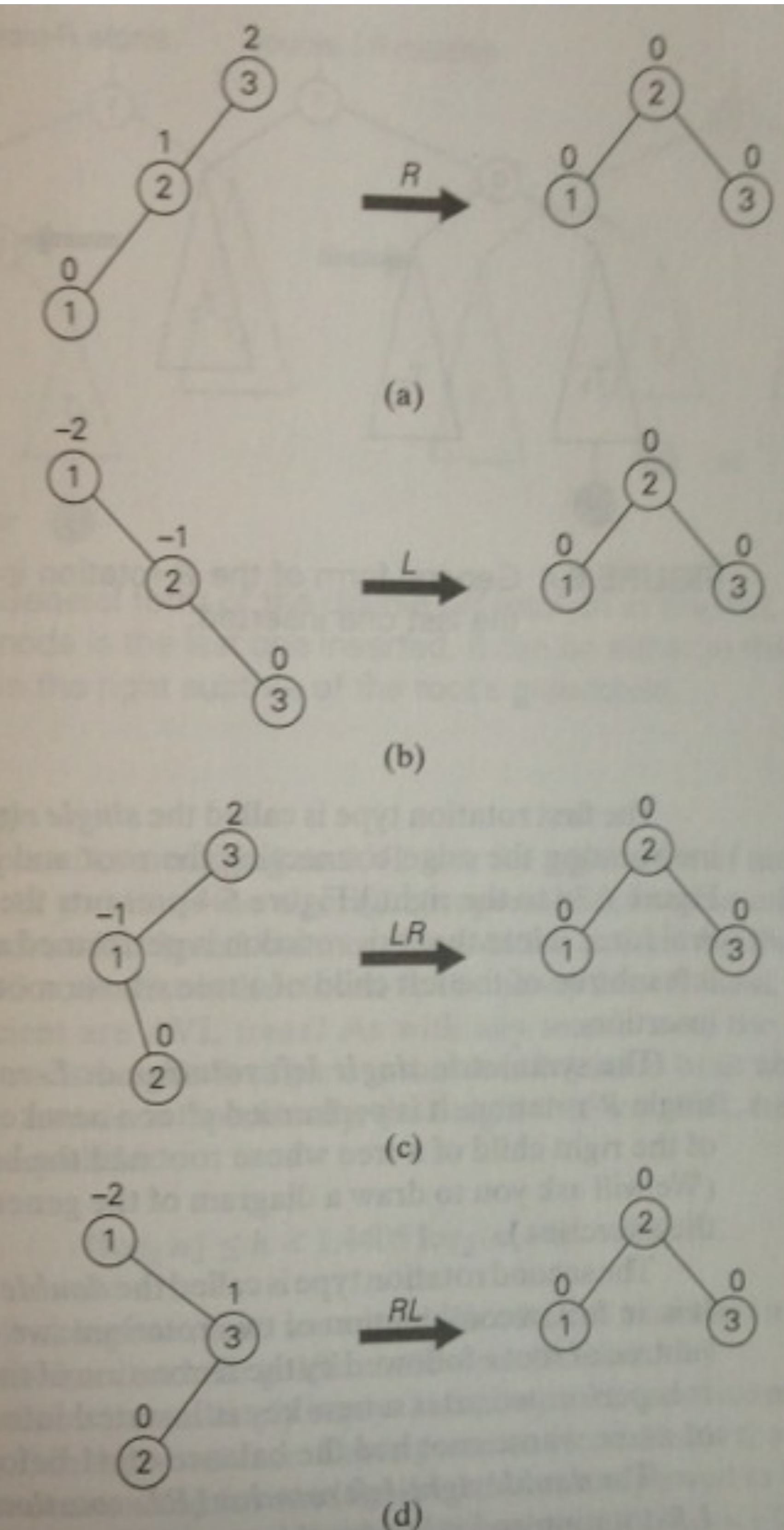


FIGURE 6.3 Four rotation types for AVL trees with three nodes. (a) Single *R*-rotation. (b) Single *L*-rotation. (c) Double *LR*-rotation. (d) Double *RL*-rotation.

Figure from Levitin

2-3 Trees
[Levitin]

- A *2-3 tree* is a tree that can have nodes of two kinds: 2-nodes and 3-nodes.
- A *2-node* contains a single key K and has two children.
- A *3-node* contains two ordered keys K1 and K2 (K1 < K2) and has three children.
- A 2-3 tree is always *height-balanced*

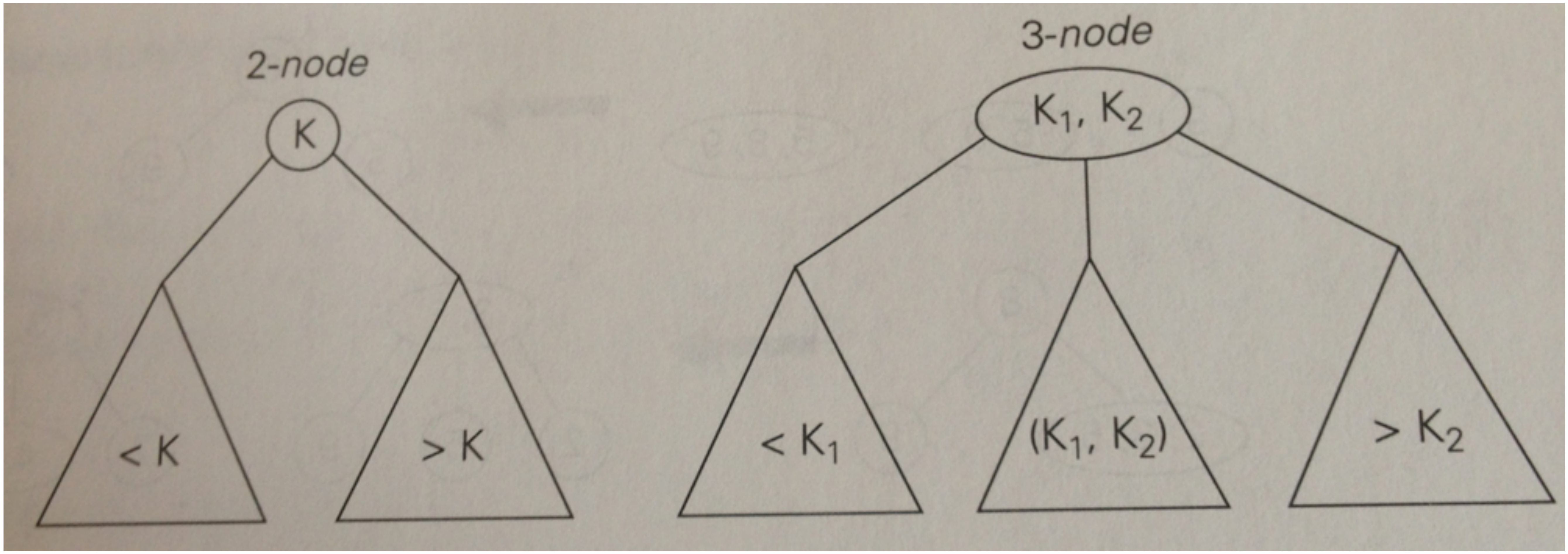


Figure from Levitin

Inserting into 2-3 Trees [Levitin]

- Always insert a new key K at a leaf except for the empty tree.
- The appropriate leaf is found by performing a search of K.
- If the leaf in question is a 2-node, we insert K there as with the first or the second key, depending on whether K is smaller or larger than the node's old key.
- If the leaf is a 3-node, we split the leaf in two: the smallest of the three keys (two old ones and the new key) is put in the first leaf, the largest is put in the second leaf, while the middle key is promoted to the old leaf's parent. (If the leaf happens to be the tree's root, a new root is created to accept the middle key.)
- Note that promotion of a middle key to its parent can cause the parent's overflow (if it was a 3-node) and hence can lead to several node splits along the chain of the leaf's ancestors.

Parameterized Types/ Generics

Sestoft, p.78

"**Generic types and methods** provide a way to strengthen type checking at compile-time while at the same time making programs more expressive, reusable and readable. The ability to have generic types and methods is also known as **parametric polymorphism**."

A little syntax...

- `class name<T1, T2, ..., Tn>`
- `method-modifiers <T1, ..., Tn> returntype m(formal-list)`
- Wildcards
  - `<?>`
  - `<? extends tb>`
  - `<? super tb>`

```
public class Box {
    private Object object;

    public void set(Object object) {
      this.object = object;
    }
    public Object get() {
      return object;
    }
}
```

Code from: http://docs.oracle.com/javase/tutorial/java/generics/types.html

```
/**
 * Generic version of the Box class.
 * @param <T> the type of the value
 *        being boxed
 */
public class Box<T> {
    // T stands for "Type"
    private T t;

    public void set(T t) { this.t = t; }
    public T get() { return t; }
}
```

Code from: http://docs.oracle.com/javase/tutorial/java/generics/types.html

Generic types that we have used this semester

- `ArrayList<E>`
- `Comparator<T>`
- `Iterable<T>`
- `Iterator<E>`

Generics
[Bloch]

- Item 23: Don’t use raw types in new code.
- Item 24: Eliminate unchecked warnings
- Item 25: Prefer lists to arrays
- Item 26: Favor generic types
- Item 27: Favor generic methods
- Item 28: Use bounded wildcards to increase API flexibility

Liskov Chapter 8: Polymorphic Abstraction

Polymorphic Abstraction

[Liskov]

"An abstraction that works for many types. A procedure or iterator can be polymorphic with respect to the types of one or more arguments. A data abstraction can be polymorphic with respect to the type of elements its objects contain."

Liskov Chapter 7: Type Hierarchy

Terminology
[Liskov]

- Type hierarchy
- Subtype
- Supertype
- Multiple implementations
- Apparent Type vs. Actual Type
- Inheritance
- Concrete Class vs. Abstract Class
- Template Pattern
- Singleton pattern

Reasoning about the Substitution Principle

[Liskov]

- The *signature rule* ensures that if a program is type-correct based on the supertype specification, it is also type-correct with respect to the subtype specification.
- The *methods rule* ensures that reasoning about calls of supertype methods is valid even though the calls actually go to code that implements a subtype.
- The *properties rule* ensures that reasoning about properties of objects based on the supertype specification is still valid when objects belong to a subtype. The properties must be stated in the overview section of the supertype specification.