CS 617 Object Oriented Systems
Lecture 7
Inheritance- Contracts, Extensions, Refinements
Single Inheritance
An Elaborate Example
3:30-5:00 pm Thu, Jan 24

Rushikesh K Joshi
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Outline

1. Single Inheritance
2. Is Implementation Available in Subclasses?
3. Examples: Interfaces, Abstract Classes, Concrete Classes
Outline

1. Single Inheritance

2. Is Implementation Available in Subclasses?

3. Examples: Interfaces, Abstract Classes, Concrete Classes
Inheritance for Conceptually Compatible Classes

- Contract Conformance (Conceptual Inheritance)
- Extension
- Refinement

**Is Kind Of** Relationship

Subtitle (Derived Class)

Superclass (Base Class)
Inheritance for Pure Extension

base = \{f1(), f2(), f3()\}

derived = base + \{f4(), f5()\}

Example:
basestream={read,write,close}
derivedstream={read,write,close,seek}
Inheritance for Refinement

base = \{f1(), f2(), f3()\}

derived = \{f1(), f2(), f3()\} with different behavior

Example:
baseStream={read, write, close}

derivedSafelySharableStream={read, write, close} automatically locks the stream during an operation
## Visibility in Derived Classes

<table>
<thead>
<tr>
<th>visibility in base</th>
<th>Accessibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>private</td>
<td>all in base</td>
</tr>
<tr>
<td>protected</td>
<td>all in base, all in derived</td>
</tr>
<tr>
<td>public</td>
<td>all</td>
</tr>
</tbody>
</table>
Outline

1. Single Inheritance

2. Is Implementation Available in Subclasses?

3. Examples: Interfaces, Abstract Classes, Concrete Classes
What Happens to Implementation?

- Inherited method bodies: available as they are, or replaceable through refinements
- Private Members: Not accessible, but available for the sake of method bodies that are ’inherited’
- Protected Members: Accessible. Communication between superclass’s member functions and subclass’s member functions can take place through these
Outline

1. Single Inheritance

2. Is Implementation Available in Subclasses?

3. Examples: Interfaces, Abstract Classes, Concrete Classes
Example: A Collection Hierarchy-The Collection Interface: from Java’s util library

```java
interface Collection {
    boolean add (Object o);
    boolean addAll (Collection c);
    boolean contains (Object o);
    boolean containsAll(Collection c);
    boolean equals(Object o);
    boolean isEmpty();
    boolean remove(Object o);
    void clear ();
    boolean removeAll(Collection c);
    boolean retainAll (Collection c);
    int size();
    Object[] toArray();
    Iterator iterator(); ...
}
```
The Iterator Interface

```java
interface Iterator {
    boolean hasNext(); // true if the iteration has more elements
    Object next(); // returns the next element
    void remove(); // remove last element returned
}
```
Can Some behavior be implemented in an Abstract Collection Class?
abstract class AbstractCollection {
  // concrete operations:
  remove: iterate over the collection and remove if you find it throws an UnsupportedOperationException if the iterator returned by iterator() does not implement remove.
  toArray: allocate a new array, iterate over the collection, insert objects in the array, return it
  contains: iterate over the collection to check whether it contains the given element
  isEmpty: check if size() == 0
  ...
  String toString(); // returns string representing the
collection—an added operation

// only two abstract operations:
  abstract int size();
  abstract Iterator iterator();

// what about add?
  add: always throws UnsupportedOperationException.
    Modifiable collections should implement add, and remove on iterator.
Abstract Set: Further Abstract Implementation

- extends Abstract Collection
- Skeletal implementation for Sets
- Also implements interface Set
- interface Set defines constrains on contracts of add
- Mainly no overriding of member functions of Abstract Collection
- Adds a new member function: boolean equals(Object o)
- *equals* checks for size, and then all memberships
TreeSet, HashSet

- They extend Abstract Set
- They use different data structures
- HashSet doesn’t provide any guarantees about iteration order
- TreeSet provides some guarantees about iteration order
TreeSet

- TreeSet also implements SortedSet interface
- SortedSet extends Set
- SortedSet interface adds constraints on iterator traversal
- The order used is ascending order based on a `compareTo()` operation
- Each element inserted must implement interface Comparable
Interface SortedSet

- extends Collection, Set
- Object first()
- Object last()
- Comparator comparator(): returns comparator associated with this set
- SortedSet headSet (Object toElement)
- SortedSet tailSet (Object fromElement)
- SortedSet subset (Object fromElement, Object toElement)
A Snapshot of the Inheritance Hierarchy