InterBit
Training & Consulting Ltd.

Bridge Pattern

Chapter Content

InterBit
Training & Consulting Ltd.

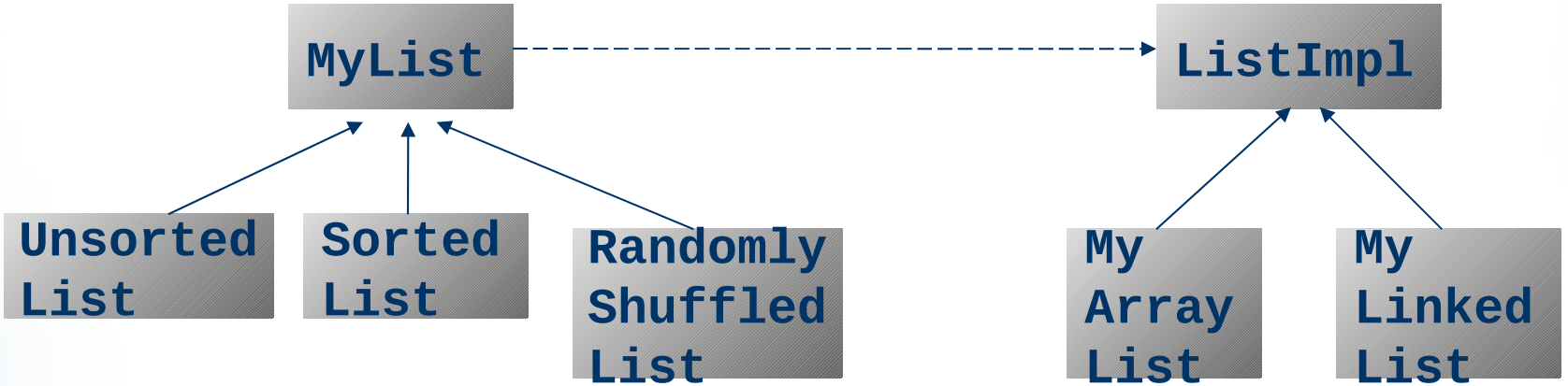
- Bridge Pattern Overview
- Bridge Pattern UML Diagram
- List Code
  - List the abstraction base-class
  - List - abstraction sub-class #1
  - List - abstraction sub-class #2
  - List - Impl classes
- Bridge - Windows example
- Bridge – discussion
- Exercise

Bridge Pattern Overview

InterBit Training & Consulting Ltd.

- Separates implementation from abstraction so that both may be varied (changed) independently.

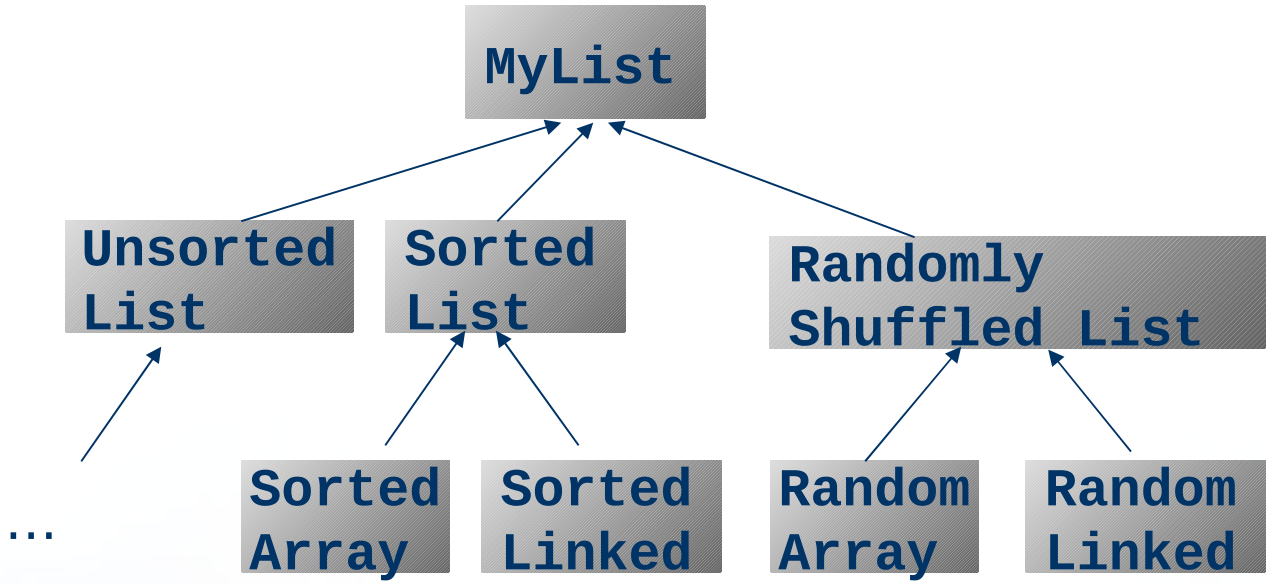
- Consider the following unrelated hierarchies:
  - Abstraction: Lists with different sorting capabilities.
  - Implementation: either a vector (array) or a linked list.



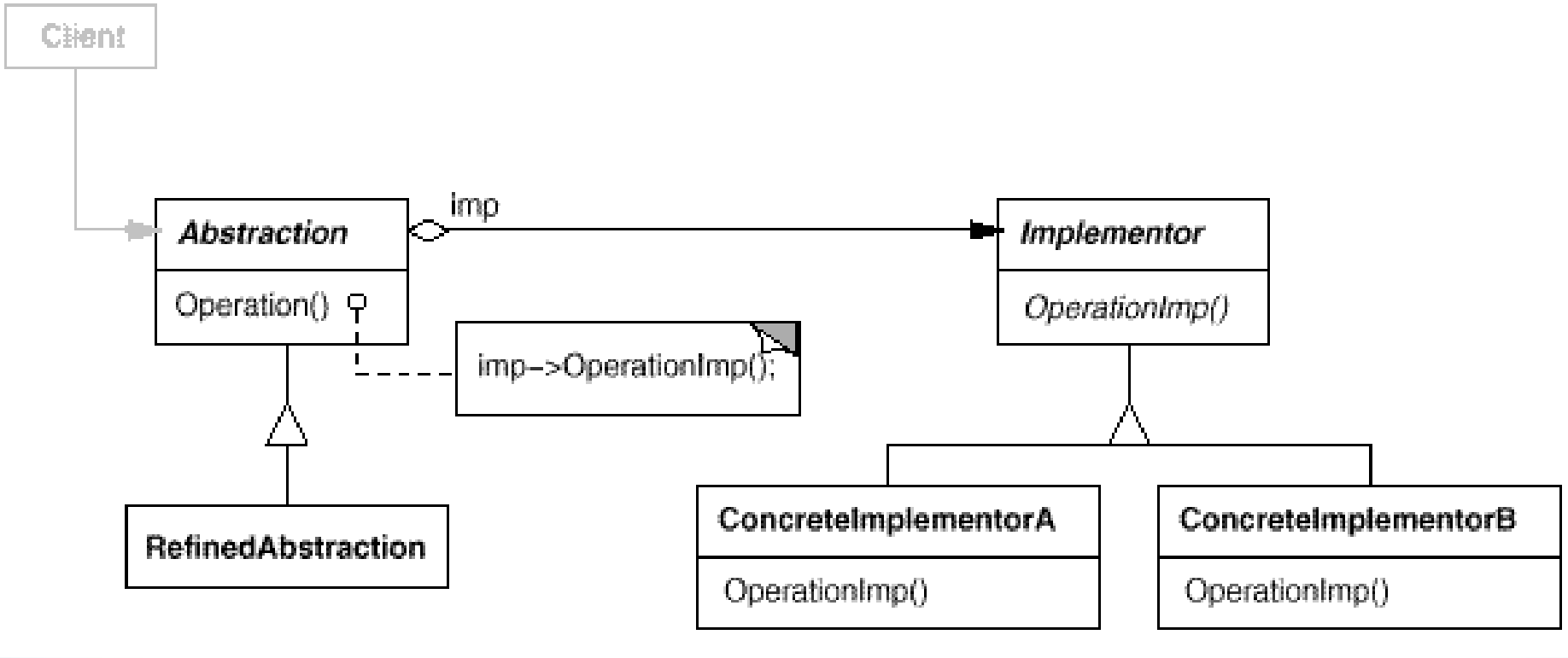
Bridge Pattern Overview

InterBit Training & Consulting Ltd.

- Bridge allows the evolution of 2 separate hierarchies (abstract / impl).
  - Use different combinations of them, while avoiding the following BAD design:



Bridge Pattern UML Diagram



Abstraction base class holds an instance of Impl. Thus, we allow the independent evolution of 2 hierarchy trees (one for abstraction, one for impl), and we may use different combinations of the two.

List Code

- **Note:**
  - Our example **doesn't** follow the existing java.util design.
    - our lists will sort themselves (while java collections are sorted through Collections.sort).
  - Some **simplification** have been made (this may be fixed once we discuss more patterns).
    - We use the *Comparable* interface instead of the fancier *Comparator*.
    - We don't use iterators.

List the abstraction base-class

InterBit Training & Consulting Ltd.

- Base class of the abstract hierarchy:

```
Class abstract MyList {
    private IListStructure impl;  // MyVector or MyLinkedList
    public MyList(IListStructure impl) {
        this.impl=impl;
        // Or select impl based on system properties, etc.
    }
    protected void insert(Object e, int i) { impl.insert(e, i);
    public int size()  {
        return impl.getSize();
    }
    public Object get(int I) {
        return impl.get(i);
    }
    public abstract void add(Object e);
}
```

MyList is the base of an hierarchy of lists with different sorting capabilities

Can be used with different kinds of impl (e.g. vector of linked list)

List- abstraction sub-class #1

InterBit Training & Consulting Ltd.

```
public class MySortedList extends MyList {
    public MySortedList(IListStructure impl) {
        this.impl=impl;
    }
    // Inserts into appropriate place, relying on base class methods:
    @Override
    public void add(Object comparable){
        if (!(comparable instanceof Comparable))
             throw new IllegalArgumentException("Not comparable");
        Comparable comp = (Comparable)comparable;
        int j=0;
        while (j<size()  &&  0 > comp.compareTo(get(j)) {
            j++;
        }
        super.insert(comp, j);
    }
}
```

Sub-class:

list that keeps itself sorted

List– abstraction sub-class #2

InterBit Training & Consulting Ltd.

```
// A randomly-shuffled list, where new elements
// are inserted into a random index:
public class MyRandomList extends MyList {
     public MyRandomList(..) {
         super(...);
     }

     // Inserts into random position:
    @Override
     public void add(Object obj){
         int j= (int) (Math.random() * size());
         insert(obj, j);
     }
}
```

Sub-class:

list that keeps itself randomly-shuffled

List – Impl classes

InterBit Training & Consulting Ltd.

- Interface for concrete implementations:

```
public interface IListStructure  {
     void insert(Object obj, int i);
     int getSize();
     Object get(int i);
}
```

- Concrete Implementation: vector

```
public class MyVector implements IListStructure {
     private Object[] data;
     ...
     // Or: we may simply wrap an adapter around the existing
     // java.util.ArrayList
}
```

Bridge – Windows example

InterBit Training & Consulting Ltd.

- Separate abstract window functionality from *platform-dependent* window implementation:
  - Abstract window hierarchy:
    - BlinkingWin.
    - CloseAfterTimeoutWin ...
  - Those may rely (compose a) on different OS-dependant windows:
    - LinuxWinImpl.
    - SolarisWinImpl.

Bridge: discussion

InterBit Training & Consulting Ltd.

- Bridges (like other designs that aim for flexibility and platform-independence) may encounter the problem of:

  lowest common denominator.

  - **Can you detect it with the list example** ?

    * hint: binary search