

§16 Предпочитайте компоновку классов – наследованию.

- В отличие от вызова метода, наследование нарушает инкапсуляцию.

```
// Broken - Inappropriate use of inheritance!
public class InstrumentedHashSet<E> extends HashSet<E> {
    // The number of attempted element insertions
    private int addCount = 0;

    public InstrumentedHashSet() {
    }

    public InstrumentedHashSet(int initCap, float loadFactor) {
        super(initCap, loadFactor);
    }

    @Override public boolean add(E e) {
        addCount++;
        return super.add(e);
    }

    @Override public boolean addAll(Collection<? extends E> c) {
        addCount += c.size();
        return super.addAll(c);
    }

    public int getAddCount() {
        return addCount;
    }
}
```

- Класс выглядит адекватно, но не работает.

```
InstrumentedHashSet<String> s =  
    new InstrumentedHashSet<String>();  
s.addAll(Arrays.asList("Snap", "Crackle", "Pop"));
```

- Иной подход:

```
// Wrapper class - uses composition in place of inheritance
public class InstrumentedSet<E> extends ForwardingSet<E> {
    private int addCount = 0;

    public InstrumentedSet(Set<E> s) {
        super(s);
    }

    @Override public boolean add(E e) {
        addCount++;
        return super.add(e);
    }
    @Override public boolean addAll(Collection<? extends E> c) {
        addCount += c.size();
        return super.addAll(c);
    }
    public int getAddCount() {
        return addCount;
    }
}
```

```

// Reusable forwarding class
public class ForwardingSet<E> implements Set<E> {
    private final Set<E> s;
    public ForwardingSet(Set<E> s) { this.s = s; }

    public void clear()                { s.clear();                }
    public boolean contains(Object o)   { return s.contains(o);   }
    public boolean isEmpty()            { return s.isEmpty();    }
    public int size()                   { return s.size();        }
    public Iterator<E> iterator()       { return s.iterator();   }
    public boolean add(E e)              { return s.add(e);        }
    public boolean remove(Object o)     { return s.remove(o);    }
    public boolean containsAll(Collection<?> c)
                                        { return s.containsAll(c); }
    public boolean addAll(Collection<? extends E> c)
                                        { return s.addAll(c);      }
    public boolean removeAll(Collection<?> c)
                                        { return s.removeAll(c);   }
    public boolean retainAll(Collection<?> c)
                                        { return s.retainAll(c);   }
    public Object[] toArray()           { return s.toArray();    }
    public <T> T[] toArray(T[] a)       { return s.toArray(a);  }
    @Override public boolean equals(Object o)
                                        { return s.equals(o);      }
    @Override public int hashCode()     { return s.hashCode();    }
    @Override public String toString()  { return s.toString();   }
}

```

- **Использование**

```
Set<Date> s = new InstrumentedSet<Date>(new TreeSet<Date>(cmp));  
Set<E> s2 = new InstrumentedSet<E>(new HashSet<E>(capacity));
```

- **Или**

```
static void walk(Set<Dog> dogs) {  
    InstrumentedSet<Dog> iDogs = new InstrumentedSet<Dog>(dogs);  
    ... // Within this method use iDogs instead of dogs  
}
```

- Подход известен как **wrapper class**. Или **decorator pattern**.
- Иногда комбинация компоновки и **forwarding** ошибочно называется **delegation**.

- Недостатки:
 - Взаимодействие с callback framework.
Проблема самоидентификации (SELF problem)
 - Производительность (на самом деле незначительно)
- При выборе наследования необходимо, чтобы B is a A.