Specification of Generic APIs, or: Why Algebraic May Be Better Than Pre/Post

Intro

- Focus on abstract APIs, rather than dealing directly with concrete data structures.
- Semantics are more easily specied by relating the operations to each other.
- The behavior of hashCode is specied entirely by its relationship to equals

```
public int hashCode();
public boolean equals(Object other);
```

- Signature
- Models
- Axioms

API specifications

```
public static <E>
void addEnsuresCollectionContainsElement(
    Collection<E> c, E t) {
  try {
  // Records some data on current status of c.
  int size = c.size();
  boolean contained = c.contains(t);

  // Attempt to add an element.
  if (c.add(t)) {
    // Element added to the collection.
    assertEquals(size + 1, c.size());
  } else {
    // Element already present.
    assertTrue(contained);
    assertEquals(size, c.size());
  }
  // Check that the element is present.
  assertTrue(c.contains(t));
  } catch (UnsupportedOperationException
    | ClassCastException
    | NullPointerException
    | IllegalArgumentException
    | IllegalStateException e) {
    // OK: precondition violation indicated.
  }
}
```

```
public static <E>
void add_unsupported(Collection<E> c, E t) {
  try {
    c.add(t);
    fail("add() did not throw.");
  } catch (UnsupportedOperationException e) {
    // OK: intended behavior.
  } catch (ClassCastException
    | NullPointerException
    | IllegalArgumentException
    | IllegalStateException e) {
      fail("add() throws wrong exception.");
  }
}
```

General requirement API

```
interface Comparable<T> {
  int compareTo(T o);
}
```

- x.compareTo(y) < 0 is $x < y$,
- x.compareTo(y) $<= 0$ is $x \leq y$,
- x.compareTo(y) $== 0$ is $x = y$,
- x.compareTo(y) $>= 0$ is $x \geq y$, and
- x.compareTo(y) > 0 is $x > y$.

```
1  public static <T extends Comparable<T>>
   void strongSymmetry(T x, T y) {
3    try {
       x.compareTo(y);
5      y.compareTo(x);
       // OK: neither call throws an exception.
7    } catch (RuntimeException e) {
       // at least one of the calls throws an exception
9      try {
         x.compareTo(y);
11        fail ("x.compareTo(y) does not throw!");
       } catch (RuntimeException e1) {
13       try {
           y.compareTo(x);
15          fail ("y.compareTo(x) does not throw!");
         } catch (RuntimeException e2) {
17         // OK! Both calls fail symmetrically.
         }
19     }
     }
21 }
```

API enrichment

```
/** Sorts the data in situ. */
abstract public void sort();
/** Counts the number of occurrences of t. */
abstract public int count(T t);
```

```
public static <T extends Comparable<T>>
void isSorted(MyArrayList<T> list, int i) {
  if (list.size() <= i) return;
  if (0 == i) return;
  list.sort();
  assertTrue(list.get(i-1).compareTo(list.get(i)) <= 0);
}

public <T extends Comparable<T>>
void isPermutation(MyArrayList<T> list, T t) {
  int precount = list.count(t);
  list.sort();
  int postcount = list.count(t);
  assertEquals(precount, postcount);
}
```

Pre/post specification

```
Specifications : pure
  public behavior
    requires o != null;
    ensures (* \result is negative
        if this is "less than" o *);
    ensures (* \result is 0 if this is "equal to" o *);
    ensures (* \result is positive
        if this is "greater than" o *);
    signals_only ClassCastException;
    signals (ClassCastException) (* ... *);
  also
  public behavior
    requires o != null &&
             o instanceof Comparable;
    ensures this.definedComparison((Comparable)o,this);
    ensures o == this ==> \result == 0;
    ensures this.sgn(\result) ==
           -this.sgn(((Comparable)o).compareTo(this));
    signals (ClassCastException)
        !this.definedComparison((Comparable)o,this);
int compareTo(non_null Object o);
```

API specifications subsume Pre/Post specifications

```
method m(...)
  requires Pre;
  ensures Post;
```

We can write this as an axiom in the following way.

```
axiom PrePost (...) {
  if ( ! Pre ) return;
  call m(...);
  assertTrue( Post );
}
```