Benjamin Fenster
CS-100 (Damon)
17 March 2003

Notes – Mutability

I. Concept
  a. Mutable objects can be changed.
  b. Immutable objects cannot be changed.
  c. The problem is to define what "changed" means
    i. It's changed if its behavior changes
    ii. It's changed if its abstract value changes
    iii. It's changed if the fields in its representation change.
    iv. It's changed if the value of that representation changes.
  d. Example
    i. Consider ListSet
    ii. Representation is a List called _elems.
    iii. Rep Invariant: No duplicates
    iv. Abstraction Function: Elements of the _elems are elements of the set.
    v. To add an element (stupidly)
      1. If it's already there, remove it.
      2. Always add it.
    vi. Does that change the object?
      1. Abstract value is unordered, so NO.
      2. Iterator yields a different order, so behavior changes, so YES.
      3. The same list is still held in _elems, so NO.
      4. Theabstract value changes (its order), so YES.
    vii. Mutability usually refers to the abstract value.
      1. Some fields may change (eg resizing an array)
      2. Some behavior may change (eg the result of asking the array's size)
      3. Neither affects the abstract value.
    viii. Some classes like Integer & String are immutable by every definition

II. Implementation
  a. Different languages will offer different levels of compiler support for this.
  b. C++
    i. `const` means fields cannot be assigned.
    ii. The values can still be changed!
    iii. There are also cases where you want to change the fields without changing the values.
  c. Java
    i. Java doesn't even offer that much support.
    ii. There just isn't any compiler support for mutability.
    iii. Always use MODIFIES comments to say where the abstract value changes.
  d. Finding where an object's value changes.
    i. Should be able to tell where the value is changing.
    ii. Look at the object, run some methods without MODIFIES, make sure it didn't change.
    iii. Should be easy to look at private data and find any modifications?
      1. Wrong!
      2. Somewhere outside may be able to change it if you've used the same object inside and outside.
      3. Called exposing your representation
      4. Only give *copies* of the data.
      5. Copy before importing and exporting anything.
    iv. If the representation isn't exposed, it's fairly easy to guarantee immutability
      1. Find every place the representation changes.
      2. Use abstraction function to see if each change really matters.

III. Reasoning about Abstraction
  a. Always want to study the class in terms of its abstraction function.

b. The abstraction function bridges the gap from real representation to the interpreted value.
c. Some methods are implemented in terms of the abstract behaviors of other methods.
    i. These are even easier to interpret.
    ii. The abstraction function isn't even needed, just the documentation for those methods.