Tutorial 6

01 - ADTs, Abstract Values, AFs & RIs

ADT is a behavioural specification of a set of operatoins that is independent of any representation

Example:

- IntegerToStringMap (set of operations)
  - IntegerToStringMap()
  - put(String key, Integer value)
  - get(String key) -> Integer
- IntegerToStringMap (behavioural specification)
  - If `put(k', v')` was the latest put operation on the map using the key k' then get(k') returns v'.
  - Let k' and k'' be two distinct Strings. If get(k') returns v', then after put(k'', v''), get(k') still returns v'.

Abstract Values

- Observationally distingiushable states of the ADT that are reachable by composing ADT operations

- If you use all observers and cannot distinguish differences of two ADT, they have the same abstract value.

- Sequence 1:
  - m1 = IntegerToStringMap()
  - m1.put("a", 1)
  - m1.put("b", 2)

- Sequence 2:
  - m2 = IntegerToStringMap()
  - m2.put("b", 2)
  - m2.put("a", 1)

- m1 and m2 have the same abstract value.

Concrete Values

- Variables inside implementation
- Cannot necessarily determine AV from CV

Abstraction Function

- For a choice of ADT and concrete representation, may exist multiple mappings from CV $\rightarrow$ AV
- AF specifies mapping for a given implementation. -"how to interpret" concrete representation to reach ADT
- ex. IntegerToStringMap:

$$\mathrm{AF}(\mathrm{keys}, \mathrm{values}) = \{\mathrm{keys}[\mathrm{i}] \rightarrow \mathrm{values}[\mathrm{i}] \mid i \in [0, \mathrm{keys.size}() - 1]\}$$

Rep Invariant:

- `this.keys.size() == this.values.size()`
- `Set.of(this.keys).size() == this.keys.size()`
- in other words, $\text{key exists} \Longleftrightarrow \text{value exists}$
- every key must be unique

02 - Lab 6 Review & RIs

- RI allows you to debug by calling a RI check at the end of each Function
- RI sometimes must be broken during a method, but should be fully formed by the time client observes AV

03 - Subtype Polymorphism

- Interface describes the way to interact with an objet; it does not provide the implementation.
- Multiple implementations of an interface can exist in the same program

```
interface Point {
    int getX();
    int getY();
}
class CartesianPoint implements Point {
    int x, y;
    CartesianPoint(int x, int y) { this.x=x; this.y=y; }
    int getX() { return this.x; }
    int getY() { return this.y; }
}
class PolarPoint implements Point {
    double len, angle;
    PolarPoint(double len, double angle) { this.len = len; this.angle = angle; }
    int getX() { return this.len * cos(this.angle) }
    int getY() { return this.len * sin(this.angle) }
    double getAngle() { ... }
}

// This is the utility of interface
class MiddlePoint implements Point {
    Point a, b;
    MiddlePoint(Point a, Point b) { this.a = a; this.b = b; }
    int getX() { return (this.a.getX() + this.b.getX()) / 2; }
    int getY() { return (this.a.getY() + this.b.getY()) / 2; }
}

...

Point pPolar = new PolarPoint(5, .245);
Point pCartesian = new CartesianPoint(1, 0);
```