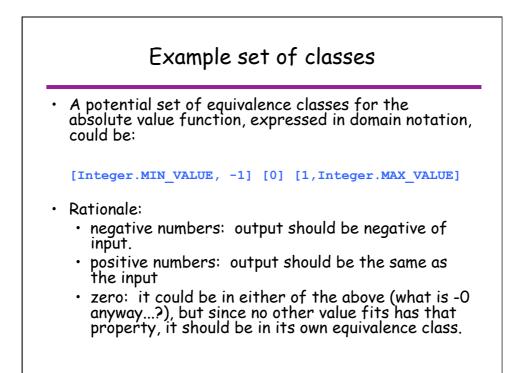
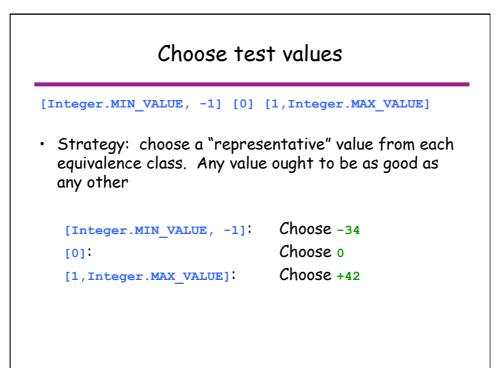
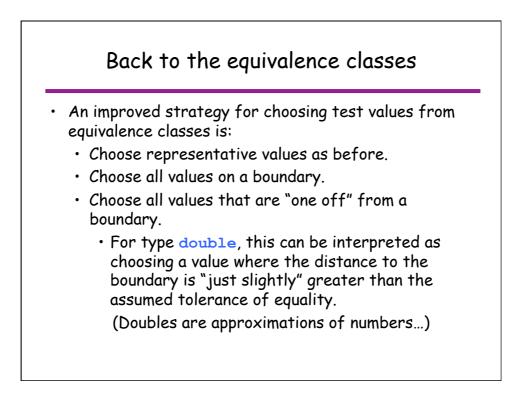


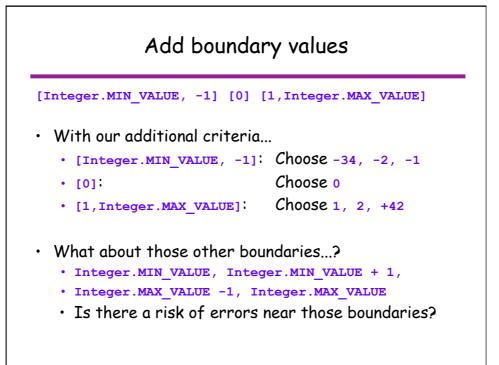
## Definition and Example

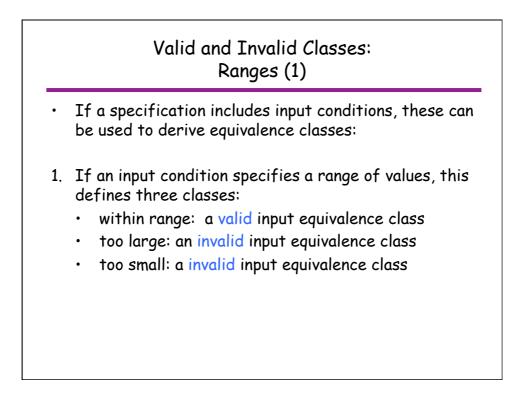
- A set or range of input domain values can be considered to be an equivalence class if they can reasonably be expected to cause "similar" responses from the implementation under test.
- Example: for the absolute value function
  - What would be different between -36 and -37 as input data?
    - Probably ... not much. The result is the negative of the input data. These two values are candidates to be in the same equivalence class.
  - On the other hand, -36 and +37 would react differently.
    - |-36 | = 36, while | 37 | = 37.
    - In one case, the absolute value is the negative of the input, while in the other case, the output is the same as the input. These two values should definitely be in different equivalence classes.











#### Valid and Invalid Classes : Ranges (2)

- 2. If an input condition specifies a range of values, and there is reason to believe the values would be handled differently, this leads to the following classes:
  - One valid equivalence class for each set of values that would be handled similarly
    - This may result in one equivalence class per value, if each value is distinctive.
      - e.g., Insurable age but also age groups within that range
    - Two invalid equivalence classes: too large, too small



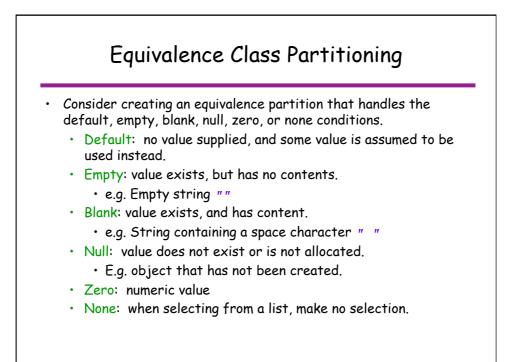
- 3. If an input condition specifies an enumerated set of values (e.g. "car", "truck", etc.):
  - One valid equivalence class for each value in the enumeration.
  - One invalid equivalence class: all values not in the enumerated set (i.e. everything else).
- Watch out for potential bugs related to implementation of enumerated types as integer code values, which has a larger domain (ie using an int that does NOT correspond to a value of the enum).
  - Example: public static final int CAR = 1; public static final int TRUCK = 2;

#### Valid and Invalid Classes: Presence / absence

- If an input condition specifies a "must be", situation (e.g. "first character of the identifier must be a letter"), this leads to:
  - One valid equivalence class (e.g. the first character is a letter).
  - One invalid equivalence class (e.g. the first character is not a letter).

#### Valid and Invalid Classes: When in doubt...

- 5. Finally, if there is any reason to believe that elements in an equivalence class are not handled in an identical manner by the implementation software, split the equivalence class into smaller classes.
  - e.g., 1-800 numbers are treated differently than other other 1-<3 digit area code> telephone numbers.



External condition	Valid equivalence classes	Invalid equivalence classes	
<ul> <li>integer value</li> <li>between 1 and 10</li> <li>one of X, Y, or Z</li> </ul>	V1: [1,10]	I1: [-∞,0] I2: [11,+∞] I3: [non-integer]	
	V2: [X] V3: [Y] V4: [Z]	I4: [not X, Y, or Z]	

# Dest Case Strategy Once the set of equivalence classes has been identified, here is how to derive test cases: Assign a unique identifier to each equivalence class. Until all valid equivalence classes have been covered by at least one test case, write a new test case covering as many of the valid equivalence classes as possible. Until all invalid equivalence classes have been covered, write a test case that covers one, and only one, of the uncovered invalid equivalence

classes.
 For each test case, annotate it with the equivalence class identifiers that it covers.

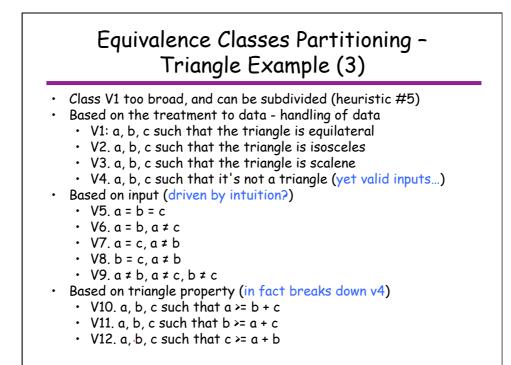


- Specification
  - Input is three integers (sides of a triangle: a, b, c)
  - Each side must be a positive number less or equal to 20.
  - Output type of the triangle:
    - Equilateral: if a = b = c
    - Isosceles: if 2 pairs of sides are equals
    - Scalene if no pair of sides is equal
    - Invalid: if  $a \ge b + c$ ,  $b \ge a + c$ , or  $c \ge a + b$

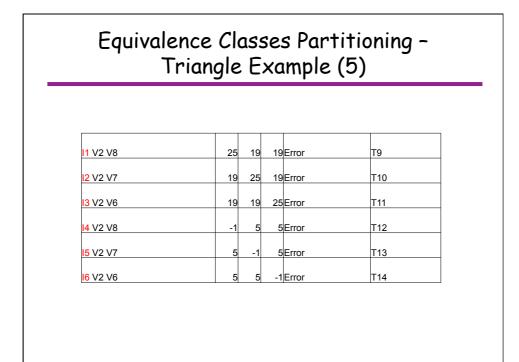
#### Equivalence Classes Partitioning -Triangle Example (2)

• According to heuristic #1

Input condition	Valid EC	Invalid EC
Sides (a,b,c)	V1: all are (0,20]	l1: a > 20
		l2: b > 20
		13: c > 20
		l4: a ≤ 0
		l5: b ≤ 0
		l6: c ≤ 0

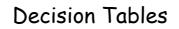


Equivalence Classes Partitioning - Triangle Example (4)					
Equivalence classes	A	В	С	Response	ID
V1 V5	3	3	3	Equilateral	T1
V2 V6	2	2	3	Isosceles	T2
V2 V7	2	3	2	Isosceles	Т3
V2 V8	3	2	2	Isosceles	T4
V3 V9	2	3	4	Scalene	T5
V4, V8, V10	20	2	2	Not a triangle	T6
V4, V7, V11	2	5	2	Not a triangle	T7
V4, V6, V12	2	2	5	Not a triangle	Т8



#### Equivalence Classes Partitioning -Problems

- Specification doesn't always define expected output for invalid test-cases.
- Strongly typed languages eliminate the need for the consideration of some invalid inputs.
- Brute-force approach of defining a test case for every combination of the inputs ECs
  - Provides good coverage, but...
  - ...is impractical when number of inputs and associated classes is large

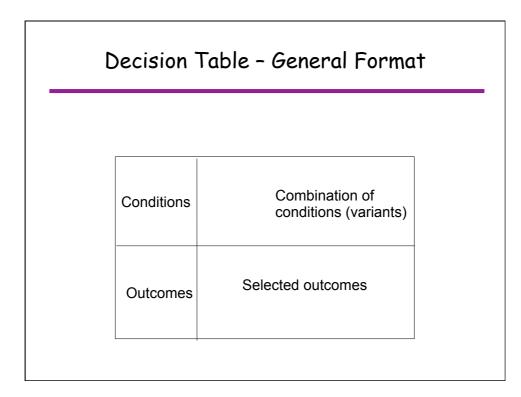


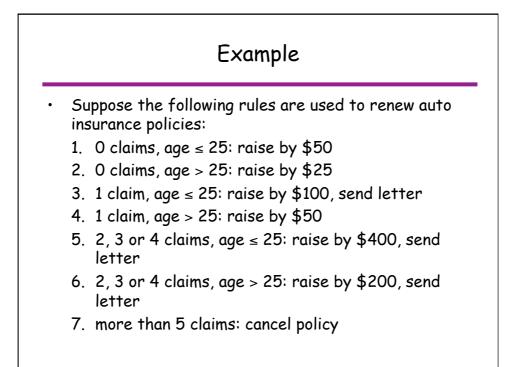
From S. Somé, A. Williams (insurance example for Binder)

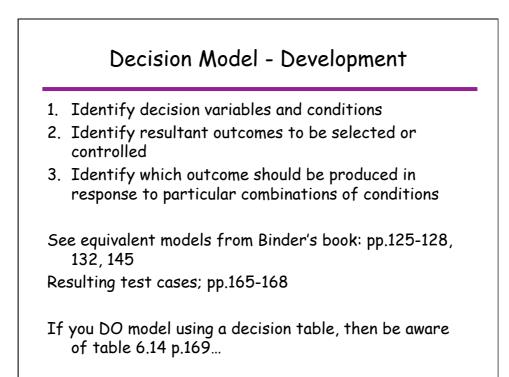
22

# **Decision Models**

- Ideal for situations where:
  - combinations of actions are taken under varying set of conditions
  - · conditions depends on input variables
  - response produced doesn't depend on the order in which input variables are set or evaluated, and
  - response produced doesn't depend on prior input or output







#### Details: Generating a Truth Table

- 1. Select an outcome to be present (1).
- 2. Find all combinations of causes subject to constraints that will set the effect to 1
  - see next slide
- 3. Create a column in the decision table for each combination of causes.
- 4. Having determined the causes for a selected outcome, determine the states of all other outcomes.
- 5. Repeat for each outcome set to absent (0).
- 6. Consolidate decision table columns when don't care values can overlap.

#### Details: Sensitization of outcomes in a Truth Table

- The goal is to set up the conditions such that changing a condition from 0 to 1 (or vice versa) will also change the desired outcome.
  - That is, a condition is not only sufficient to cause the outcome, but also necessary.
- Strategies:
  - If an outcome of 1 can be produced by several conditions (an OR constraint), only set one condition to be 1 at a time.
  - 2. If an outcome of 0 can be produced if one of any condition is absent (an AND constraint), set all conditions to 1 except the primary condition.
  - 3. Use the logical negation of these when trying to achieve an outcome of O.

## Don't Care condition

- Don't Care condition
  - May be true or false without changing the action
  - Simplifies the decision table
  - Corresponds to different implementation cases:
    - Inputs are necessary but have no effect for the variant
    - Inputs may be omitted but have no effect if supplied

# Can't Happen & Don't know conditions

- Can't Happen Condition reflects assumption that
  - · some inputs are mutually exclusive,
  - some inputs can't be produced by the environment, or
  - implementation is structured so as to prevent evaluation
- Don't Know Condition reflects an incomplete model
  - Usually indication of mis-specification
  - Tests needed to exercise these undefined cases
- Be careful not to confuse a Don't Care condition with either of the above.