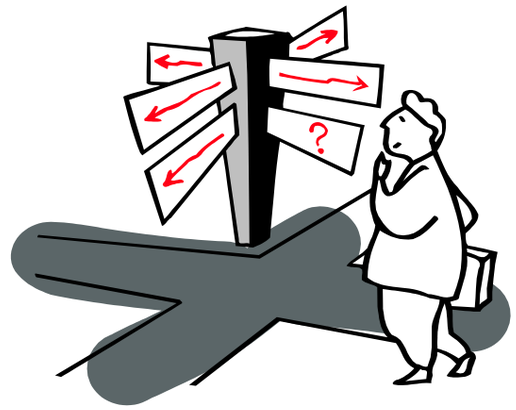
Software Testing

2. Test Design - part 1
Domain-Based Testing
(Equivalence partitioning & Boundary value analysis)
Associations

2. Test Design - part 1



(Loosely based on “Chapter 5: Domain Testing” of Practical Test Design
+“Chapter 6: Equivalence Class Testing” of Software Testing
+ “Class Association Test” pattern in Testing Object-Oriented Systems — Binder)

- State Space Explosion (the needle in the haystack)
- Coverage
  + Code Coverage vs. Test Coverage
  + Modified condition/decision coverage (MC/DC)
- RIPR criterion (reach - infect - propagate - reveal)
- Domain Analysis (Equivalence Partitioning)
- Boundary Value Analysis
  + in / on / off / out points
- Associations

Testing Approaches

(Smart) Poking Around

Brute Force

Systematic & Focused

Limits of Testing: State space explosion

Input space is surprisingly large

- Simplified case
  + Triangle example with points in coordinate system [1..10, 1..10]
  + 10^2 = 100 possible end-points;
    10^4 = 10.000 possible lines;
    10^4*3 = 10^{12} possible triangles

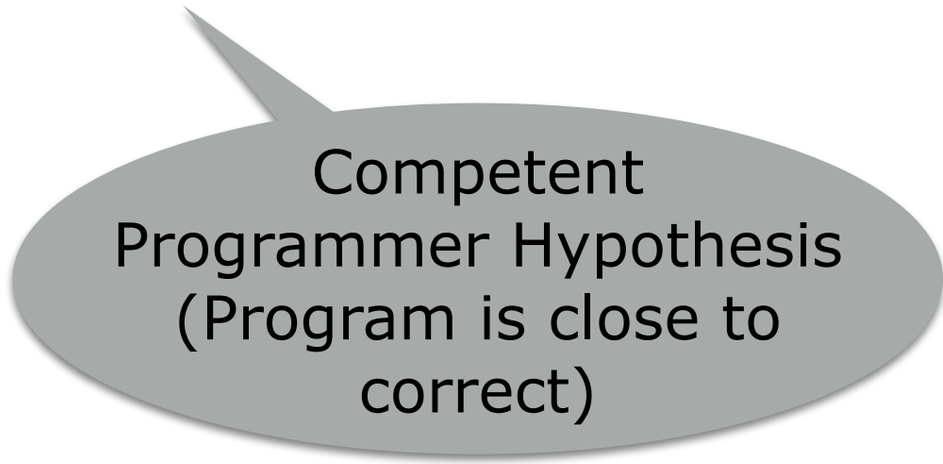
- Less simplified
  + display of 1024 x 768 pixels
    - possible lines = 786.4326
    - possible triangles = 2,36574 x 10^{35}

- Full integer coordinate system
  + 2^{16} x 4 possible lines
  + 2^{16} x 4 x 3 = 2^{192} = 6.277 x 10^{57} possible triangles
    (number of particles in the universe = +-10^{80})

Limits of Testing: Coincidental Correctness

- Coincidental correctness
  + buggy code may produce correct results under some circumstances

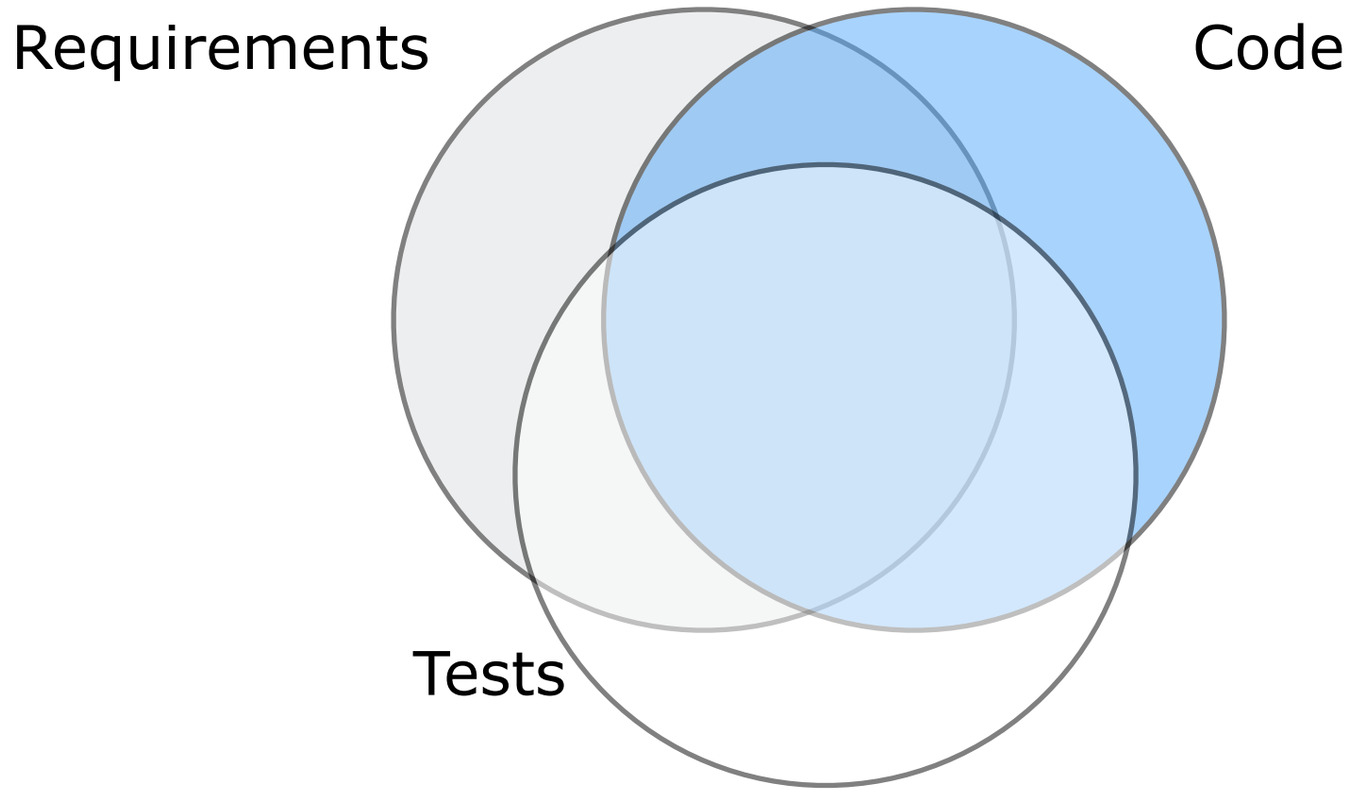
- example
  - write x + x instead of x * x
  - will produce correct result for x = 2



- example 2:
  - int scale(int j) {
    j = j -1 ; // should be j = j + 1;
    j = j / 30000;
    return j;
    }
  + out of 65.536 possible values for j, only six will reveal the fault !
    (-30001, - 30000, -1, 0, 29999 and 30000)

proverbial “Needle in a haystack”

Code Coverage vs. Test Coverage



- **Code Coverage** = Proportion of code covered by the test suite
  + statement coverage, branch coverage, decision coverage, …
- **Requirements Coverage** = Proportion of requirements covered by the test suite
  + features, uses cases, user stories, …
    + non functionals

Test Coverage	≈ Requirements coverage (incl. surprises)	≈ How much of planned tests have been executed

LCOV - code coverage report

Current view:	top level		
Test:	libbash t		
Date:	201		

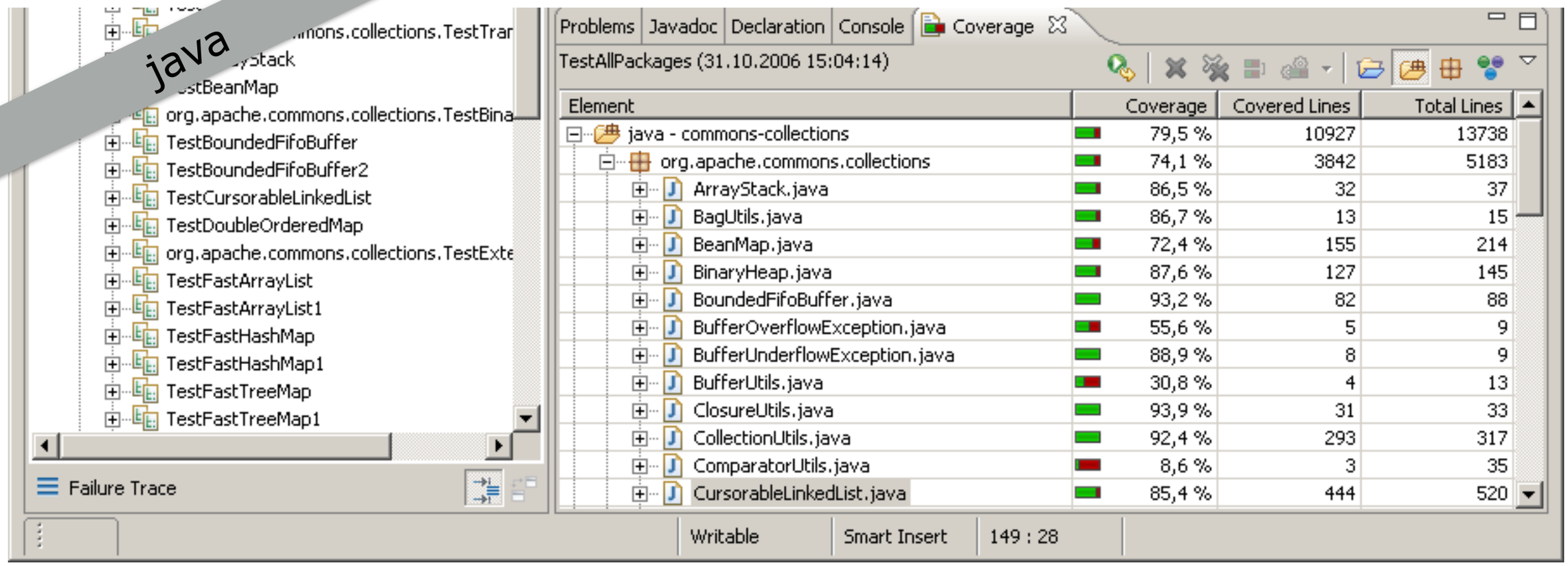
	H:	Total
Lines:	20640	34749
Functions:	1184	1287
Branches:	15689	37086

C++

Directory	Line Coverage			Functions	
src/core	95.7 %	314 / 328		98.2 %	55 / 56
test	97.0 %	98 / 101		100.0 %	72 / 72
src/builtins/tests	98.6 %	144 / 146		100.0 %	203 / 203
src/builtins	98.6 %	214 / 217		100.0 %	45 / 45
src/core/tests	98.9 %	351 / 355		99.3 %	133 / 134
./src/builtins	100.0 %	9 / 9		93.3 %	14 / 15
src	100.0 %	35 / 35		91.7 %	11 / 12
./src/core	100.0 %	190 / 190		98.0 %	99 / 101

Generated by: LCOV version 1.9

java



What is the Problem?

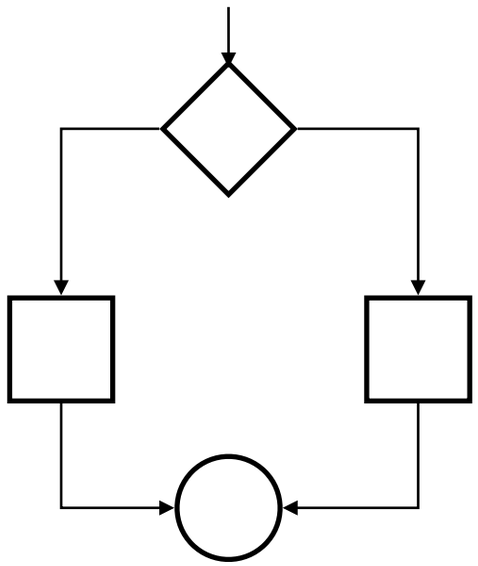
```java
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class TestEmployeeDetails {
    EmpBusinessLogic empBusinessLogic = new EmpBusinessLogic();
    EmployeeDetails employee = new EmployeeDetails();

    //happy day scenario for calculation of appraisal and salary
    @Test
    public void testCalculateAppriasal() {
        employee.setName("Rajeev");
        employee.setAge(25);
        employee.setMonthlySalary(8000);

        double appraisal = empBusinessLogic.calculateAppraisal(employee);
        double salary = empBusinessLogic.calculateYearlySalary(employee);
    }
}
```

```java
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class TestEmployeeDetails {
   EmpBusinessLogic empBusinessLogic = new EmpBusinessLogic();
   EmployeeDetails employee = new EmployeeDetails();

   //happy day scenario for calculation of
   @Test
   public void testCalculateAppriasal() {
      employee.setName("Rajeev");
      employee.setAge(25);
      employee.setMonthlySalary(8000);

      double appraisal = empBusinessLogic.calculateAppraisal(employee);
      double salary = empBusinessLogic.calculateYearlySalary(employee);
   }
}
```

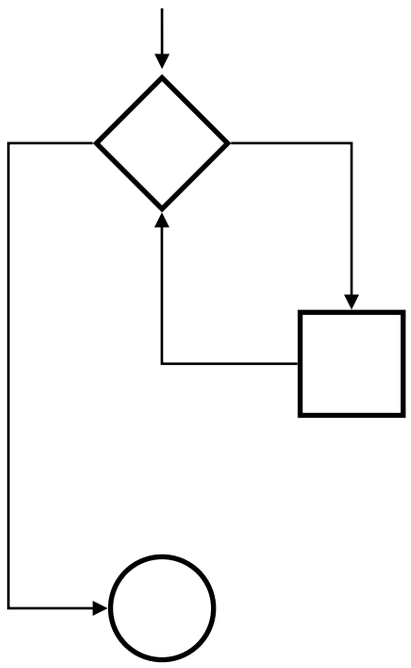
assertionless test

Control Flow Graph

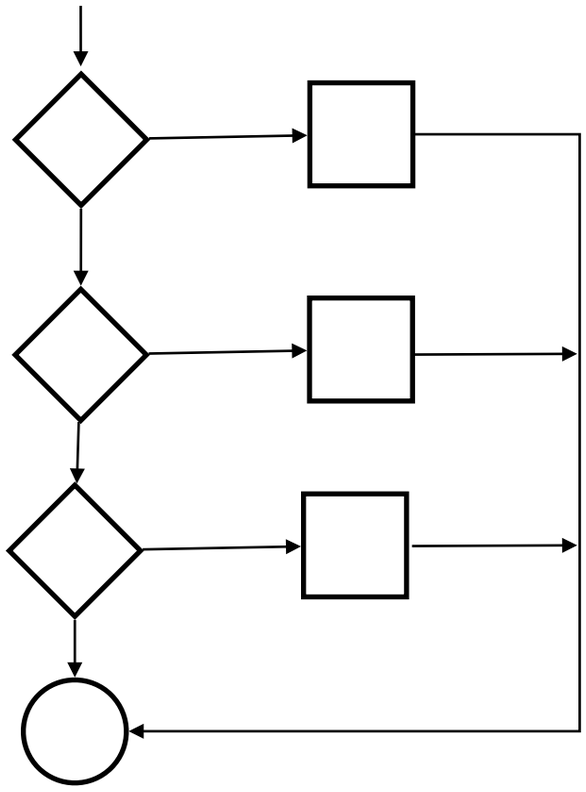
if-then-else



while



case



- Condition Coverage
  - Possible outcomes for each condition (“true” or “false”) at least once
- Branch coverage (a.k.a. decision coverage)
  - Every arrow leaving a decision is executed at least once

- Path coverage
  - Cover entry-exit paths

Modified Condition/Decision (MC/DC)

Every possible outcome of a condition is the determinant of the outcome of the decision at least once.

Decision with N conditions ⇒ N+1 test case

- x *AND* y
  - x is true ⇒ result of expression is determined by value of y
  - Neutral value for AND is true
- x *OR* y
  - x is false ⇒ result of expression is determined by value of y
  - Neutral value for OR is false

For every condition within the decision
- once true, once false
- remaining conditions: neutral value

Multiple Condition Coverage

All possible combinations of conditions in a decision at least once.
⇒ Complete decision table

Decision with N conditions ⇒ 2^N test case

- x AND y
  + x is true
    - y is true
    - y is false
  + x is false
    - y is true
    - y is false

- x OR y
  + x is true
    - y is true
    - y is false
  + x is false
    - y is true
    - y is false

Example

If (number of books > 8) or (sum >= 250 eur) THEN extra discount

	Number of Books > 8	Sum >= 250 eur	expected outcome
Condition coverage	true false	false true	extra discount extra discount
Decision coverage	false false	true false	extra discount —
MC / DC	true false false	false true false	extra discount extra discount —
Multiple condition coverage	true true false false	true false true false	extra discount extra discount extra discount —

RIPR Criterion for Effective Tests

Reach … the fault

Infect … the program state

Propagate … to the output

Reveal … via an oracle

Where is the fault?

```
/**
 * Find last index of element
 * @param x array to search
 * @param y element to look for
 * @return last index of y in x, if absent -1
 * @throws NullPointerException if x is null
*/
```

```
public static int findLast(int [] x, int y)
{
for (int i=x.length-1; i>0; i--)
    if (x[i] == y)
        return i;
return -1;
}
```

i>0 ⟼ i >= 0

```
/**
 * Find last index of element
 * @param x array to search
 * @param y element to look for
 * @return last index of y in x, if absent -1
 * @throws NullPointerException if x is null
*/
```

```
public static int findLast(int [] x, int y)
{
for (int i=x.length-1; i>0; i--)
    if (x[i] == y)
        return i;
return -1;
}
```

Input that does not reach the fault

```
/**
 * Find last index of element
 * @param x array to search
 * @param y element to look for
 * @return last index of y in x, if absent -1
 * @throws NullPointerException if x is null
*/
```

```
public static int findLast(int [] x, int y)
{
for (int i=x.length-1; i>0; i--)
    if (x[i] == y)
        return i;
return -1;
}
```

x = null; y = 5

Coverage = 0

```
/**
 * Find last index of element
 * @param x array to search
 * @param y element to look for
 * @return last index of y in x, if absent -1
 * @throws NullPointerException if x is null
*/
```

```
public static int findLast(int [] x, int y)
{
for (int i=x.length-1; i>0; i--)
    if (x[i] == y)
        return i;
return -1;
}
```

Input = reaches the fault

```
/**
 * Find last index of element
 * @param x array to search
 * @param y element to look for
 * @return last index of y in x, if absent -1
 * @throws NullPointerException if x is null
*/
```

```
public static int findLast(int [] x, int y)
{
for (int i=x.length-1; i>0; i--)
    if (x[i] == y)
        return i;
return -1;
}
```

x = [2,3,5]; y = 3

Coverage = Complete for loop

```
/**
 * Find last index of element
 * @param x array to search
 * @param y element to look for
 * @return last index of y in x, if absent -1
 * @throws NullPointerException if x is null
*/
```

```
public static int findLast(int [] x, int y)
{
for (int i=x.length-1; i>0; i--)
    if (x[i] == y)
        return i;
return -1;
}
```

Input = infects the program state

```
/**
 * Find last index of element
 * @param x array to search
 * @param y element to look for
 * @return last index of y in x, if absent -1
 * @throws NullPointerException if x is null
*/
```

```
public static int findLast(int [] x, int y)
{
for (int i=x.length-1; i>0; i--)
    if (x[i] == y)
        return i;
return -1;
}
```

x = [2,3,5]; y = 25

Coverage = All except “return i;”

```
/**
 * Find last index of element
 * @param x array to search
 * @param y element to look for
 * @return last index of y in x, if absent -1
 * @throws NullPointerException if x is null
*/
```

```
public static int findLast(int [] x, int y)
{
for (int i=x.length-1; i>0; i--)
    if (x[i] == y)
        return i;
return -1;
}
```

Together we have 100% coverage

```
/**
 * Find last index of element
 * @param x array to search
 * @param y element to look for
 * @return last index of y in x, if absent -1
 * @throws NullPointerException if x is null
*/
```

```
public static int findLast(int [] x, int y)
{
for (int i=x.length-1; i>0; i--)
    if (x[i] == y)
        return i;
return -1;
}
```

Input: reach / infect / propagate ⇒ reveal?

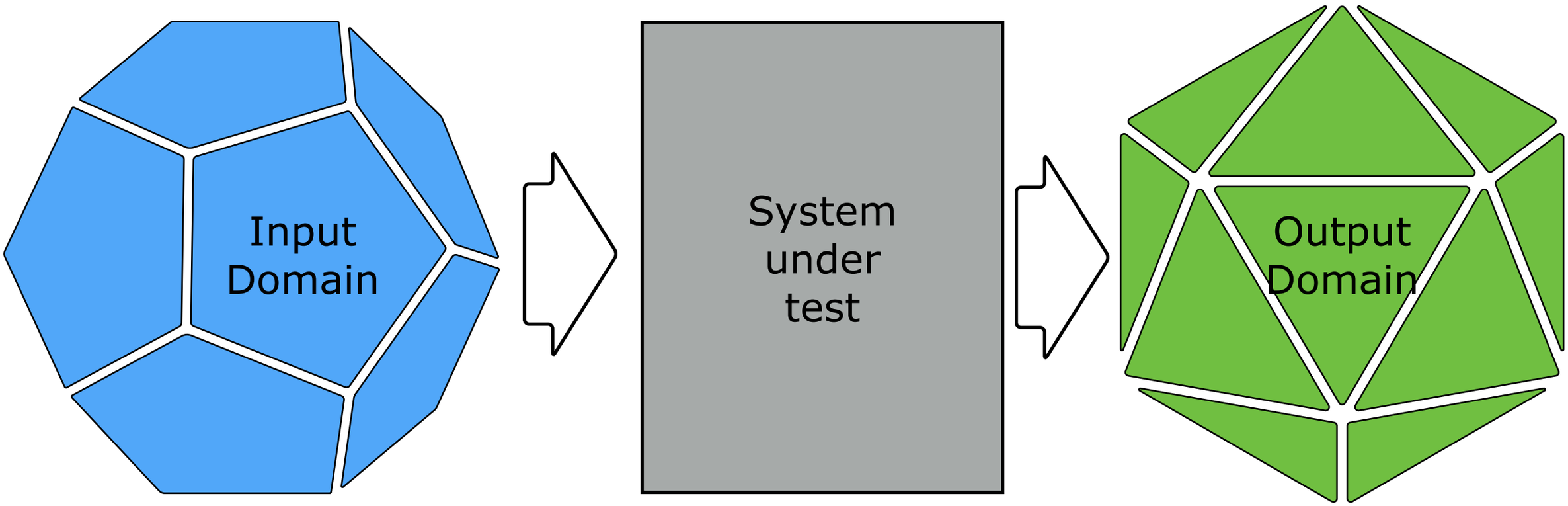
```
/**
 * Find last index of element
 * @param x array to search
 * @param y element to look for
 * @return last index of y in x, if absent -1
 * @throws NullPointerException if x is null
*/
```

```
public static int findLast(int [] x, int y)
{
for (int i=x.length-1; i>0; i--)
    if (x[i] == y)
        return i;
return -1;
}
```

x = [2,3,5]; y = 2

Domain Testing

a.k.a equivalence partitioning



Divide input/output domain in *partitions (equivalence classes)*.
= disjoint, non-empty, finite subsets
of values where the system behaviour is similar

Example: Valid password

- A valid password must contain at least 8 and at most 14 American Standard Code for Information Interchange (ASCII) characters.

- Among the characters there has to be
  + at least one lower case letter (a–z),
  + at least one upper case letter (A–Z),
  + at least one numeric character (0-9) and
  + at least one of the following special characters: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’ and ‘@’.

- In the case of less than 8 characters,
  + the error message ‘The number of characters is less than 8’ appears.

- In the case of more than 14 characters,
  + the error message ‘The number of characters is more than 14’ appears.

- In the case of a missing character type,
  + the error message ‘Missing character type’ appears, showing one of the four types (lower, upper, numerical, special).

Input Partitioning

1	Number of characters >= 8 and <= 14	At least one lower case character	At least one upper case character	At least one numeric character	At least one character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’
2	Number of characters < 8	At least one lower case character	At least one upper case character	At least one numeric character	At least one character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’
3	Number of characters > 14	At least one lower case character	At least one upper case character	At least one numeric character	At least one character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’
4	Number of characters >= 8 and <= 14	At least one lower case character	At least one upper case character	At least one numeric character	No special character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’
5	Number of characters >= 8 and <= 14	At least one lower case character	At least one upper case character	No numeric character	At least one character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’
6	Number of characters >= 8 and <= 14	At least one lower case character	No upper case character	At least one numeric character	At least one character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’
7	Number of characters >= 8 and <= 14	No lower case character	At least one upper case character	At least one numeric character	At least one character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’

Happy day scenario

Input Partitioning

1	Number of characters >= 8 and <= 14	At least one lower case character	At least one upper case character	At least one numeric character	At least one character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’
2	Number of characters < 8	At least one case charact	upper er	At least one numeric character	At least one character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’
3	Number of characters > 14	At least one case charact	upper er	At least one numeric character	At least one character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’
4	Number of characters >= 8 and <= 14	At least one case character	upper case character	At least one numeric character	No special character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’
5	Number of characters >= 8 and <= 14	At le case character	case character	ric character	At least one character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’
6	Number of characters >= 8 and <= 14	At least one lower case character	No upper case character	At least one numeric character	At least one character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’
7	Number of characters >= 8 and <= 14	No lower case character	At least one upper case character	At least one numeric character	At least one character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’

Happy day scenario

Devise a Test case for each row

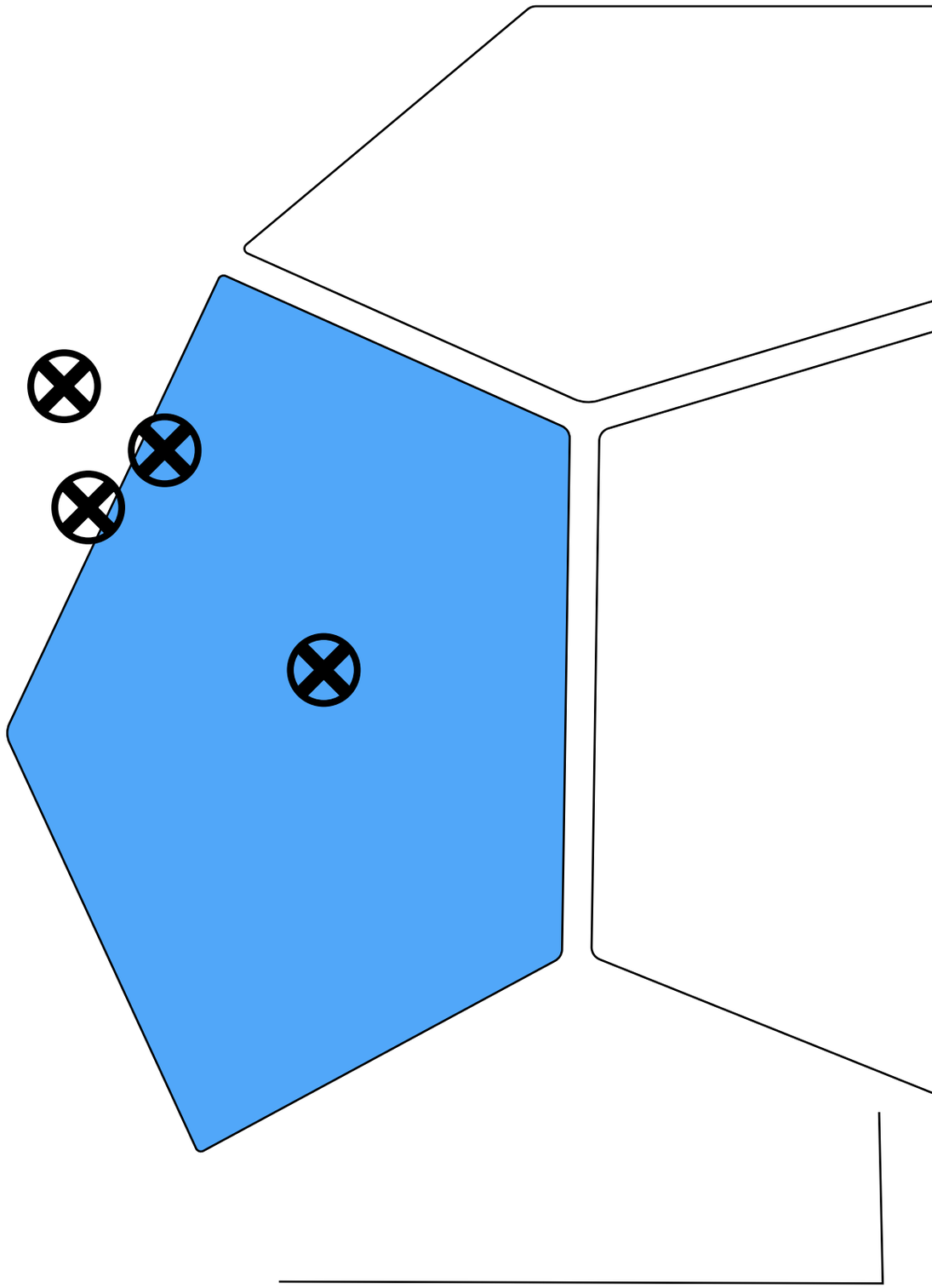
Input Partitioning

1	Number of characters >= 8 and <= 14	At least one lower case character	At least one upper case character	At least one numeric character	At least one character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’	6sG?B7u;j
2	Number of characters < 8	At least one lower case character	At least one upper case character	At least one numeric character	At least one character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’	a:B51
3	Number of characters > 14	At least one lower case character	At least one upper case character	At least one numeric character	At least one character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’	anm@@9A8B8 Cdfdff
4	Number of characters >= 8 and <= 14	At least one lower case character	At least one upper case character	At least one numeric character	No special character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’	avAQ9821
5	Number of characters >= 8 and <= 14	At least one lower case character	At least one upper case character	No numeric character	At least one character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’	SwDy:@JJ
6	Number of characters >= 8 and <= 14	At least one lower case character	No upper case character	At least one numeric character	At least one character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’	weo8712:
7	Number of characters >= 8 and <= 14	No lower case character	At least one upper case character	At least one numeric character	At least one character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’	:?OP34JK

Boundary Value Analysis

Choose test points close to the boundaries

- **In point** = Inside the domain
- **On point** = Closest to the boundary; inside the domain
- **Off point** = Closest to the boundary; outside the domain
- **Out Point** = Outside the domain



Examples

int Age		In point	On point	Off point	Out point
Age > 42	open boundary	50	43	42	20
Age >= 43	closed boundary	50	43	42	20
Age == 43	closed boundary	43	43	42 44	20 50
Age <> 43	open boundary	20 50	42 44	43	43

What happens if Age is a fix-point number with precisions 0,001?
What happens if Age is a floating point number?

Example Authorisation

3	Number of characters > 14	At least one lower case character	At least one upper case character	At least one numeric character	At least one character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’

- In Point = 20 characters
- On point = 15 characters
- Off point = 14 characters
- Out Point = 8 characters —> happy path

4	Number of characters >= 8 and <= 14	No special character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’

Boundaries are ASCII values 57 (‘9’) and 65 (‘A’)

- In point = `7’ and `b’
- On point = ‘9’ and ‘A’
- Off point = ‘:’ and ‘@’
- Out Point = `<’

Dec	Hex	Binary	HTML	Char
53	35	00110101	5	5
54	36	00110110	6	6
55	37	00110111	7	7
56	38	00111000	8	8
57	39	00111001	9	9
58	3A	00111010	:	:
59	3B	00111011	;	;
60	3C	00111100	<	<
61	3D	00111101	=	=
62	3E	00111110	>	>
63	3F	00111111	?	?
64	40	01000000	@	@
65	41	01000001	A	A
66	42	01000010	B	B

Example Authorisation (exercise)

5	Number of characters >= 8 and <= 14	At least one lower case character	At least one upper case character	No numeric character	At least one character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’

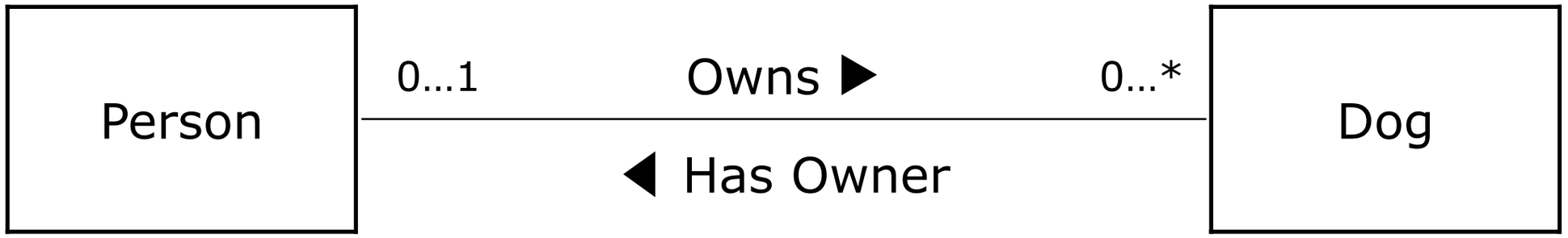
- In point = …
- On point = …
- Off point = …
- Out Point = …

6	Number of characters >= 8 and <= 14	At least one lower case character	No upper case character	At least one numeric character	At least one character: ‘:’, ‘;’, ‘<’, ‘=’, ‘>’, ‘?’, ‘@’

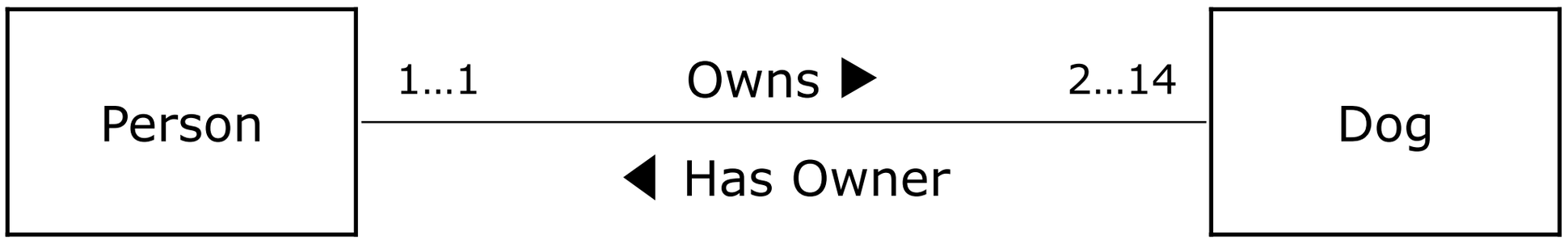
- In Point = …
- On point = …
- Off point = …
- Out Point = …

Choose the respective
In / on / off / out points

Associations



Any person may own an unlimited amount of dogs



Every person must own at least two dogs, but no more than 14

Consider multiplicity of associations as a special kind of boundary value analysis

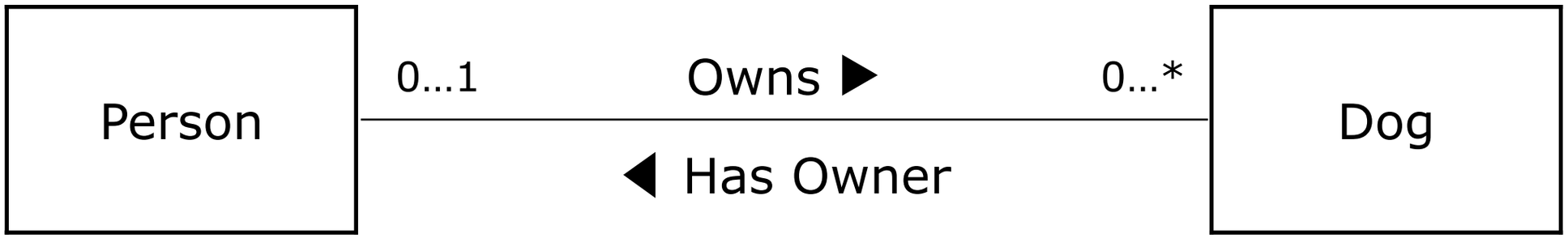
Boundary conditions for Legal Multiplicities

[A:n(B) <= 24]
For each instance of type A, the number of type B must be less than or equal to 24.

Multiplicity of A to B	Boundary Conditions	
*	A:n(B) >= 0	A:n(B) <= M (*)
0	A:n(B) = 0	
1	A:n(B) = 1	
42	A:n(B) = 42	
0 … 0	A:n(B) = 0	
0 … 1	A:n(B) >= 0	A:n(B) <= 1
0 … 24	A:n(B) >= 0	A:n(B) <= 24
0 … *	A:n(B) >= 0	A:n(B) <= M
1 … 1	A:n(B) = 1	
1 … 24	A:n(B) >= 1	A:n(B) <= 24
1 … *	A:n(B) >= 1	A:n(B) <= M
2 … 14	A:n(B) >= 2	A:n(B) <= 14
42 … 4096	A:n(B) >= 42	A:n(B) <= 4096

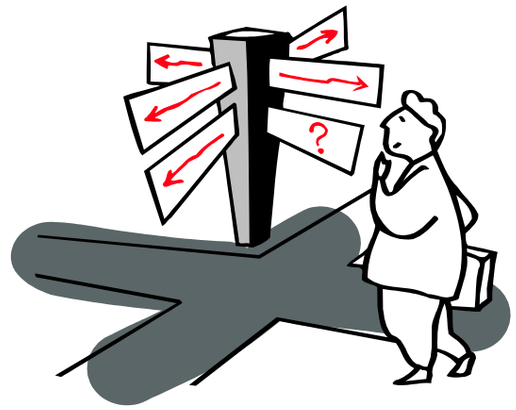
(*) Choose M arbitrary large. For instance, largest number of associations in a legacy database

Example: Person owns unlimited dogs



Variable / Condition / Type			Test Cases							
Number of dogs	Person:n(Dog) >= 0	On	0							
		Off		-1						
	Person:n(Dog) <= m	On			M					
		Off				M+1				
	Typical	In					2	42	5329	256
Number of persons	Dog:n(Person) >= 0	On					0			
		Off						-1		
	Dog:n(Person) <= 1	On							1	
		Off								2
	Typical	In	1	1	1	1				
Expected Result			OK	ERR	OK	ERR	OK	ERR	OK	ERR

2. Test Design - part 1



(Loosely based on “Chapter 5: Domain Testing” of Practical Test Design
+“Chapter 6: Equivalence Class Testing” of Software Testing
+ “Class Association Test” pattern in Testing Object-Oriented Systems — Binder)

- State Space Explosion (the needle in the haystack)
- Coverage
  + Code Coverage vs. Test Coverage
  + Modified condition/decision coverage (MC/DC)
- RIPR criterion (reach - infect - propagate - reveal)
- Domain Analysis (Equivalence Partitioning)
- Boundary Value Analysis
  + in / on / off / out points
- Associations