Equivalence classes

1. Make equivalences classes for the input variable for this method:

public boolean isEven(int n)

n % 2	True
n % 1	False

2. Make equivalences classes for an input variable that represents a mortgage applicant's salary. The valid range is $1000 pr. month to $75,000 pr. month

Salary < $1000	False
Salary > $75,000	False
$1000 <= Salary <= 75,500	True

3. Make equivalences classes for the input variables for this method:

public static int getNumDaysinMonth(int month, int year)

Month < 1	Invalid
Month > 12	Invalid
1 <= Month <= 12	Valid
Year < 1	Invalid
Year =< 1	Valid

Boundary Analysis

1. Do boundary value analysis for input values exercise 1

Invalid	Valid	Invalid
Min(int)	Min(int) < n < Max(int)	Max(int)

2. Do boundary value analysis for input values exercise 2

Invalid	Valid	Invalid
999	35000	75,501

3. Do boundary value analysis for input values exercise 3

Invalid	Valid	Invalid
Month = 0	Month = 7	Month = 13
Year = 0	Year = 2000	

Decision tables

1. Make a decision table for the following business case:

No charges are reimbursed (DK: refunderet) to a patient until the deductible (DK: selvrisiko) has been

met. After the deductible has been met, reimburse 50% for Doctor's Office visits or 80% for Hospital visits.

Condition				
Hos Doctor	True	True	False	false
Hos Hospital	True	False	True	False
Action				
50%		x		
80%			x	
Error	x			x

2. Make a decision table for leap years.

Leap year: Most years that are evenly divisible by 4 are leap years.

An exception to this rule is that years that are evenly divisible by 100 are not leap years, unless they are also

evenly divisible by 400, in which case they are leap years.

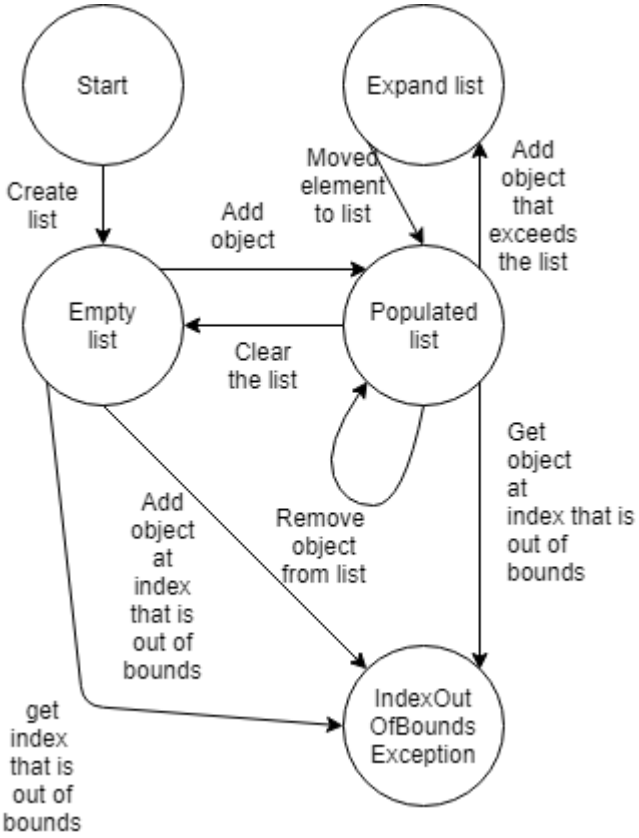
Conditions								
Year / 4	true	true	True	true	false	false	false	false
Year / 100	true	true	false	false	true	true	false	false
Year / 400	true	false	true	false	true	false	false	true
Action								
Leap year	x			x				
Not leap year		x					x	
Error			x		x	x		x

State transition

State transition testing is another black box test design technique where test cases are designed to execute valid and invalid transitions.

Use this technique to test the class MyArrayListWithBugs.java(find code on last page). It is a list class implementation (with defects) with the following methods:

1. Make a state diagram that depicts the states of MyArrayListWithBugs.java and shows the events that cause a change from one state to another (i.e. a transition).



2. Derive test cases from the state diagram.

Id	Scenario	Test Data	Result
1	Add object to empty list	Object	Valid
2	Remove object from empty list	Index	Error
3	Remove object from list	Index	Valid
4	Get number of object in list	Void	Valid
5	Remove last object form the list.	Index	Valid
6	Add object that exceeds the list size.	Object	Valid
7	Get object at index where there is no object	Index	Error

5. Consider whether a state table is more useful design technique. Comment on that.

When considering state table vs state diagram, it is useful to consider the purpose of the procedure. If the purpose is to get an overview of the program, then both are good since, it makes us go over every state step by step and consider what could go wrong and what could change. But If the purpose is to be more systematic then I would think state table is better. But state diagram is better for testing transitions between states.

6. Make a conclusion where you specify the level of test coverage and argue for your chosen level:

1. Percentage of states visited

I believe I have covered every state, since I have covered every state on my state diagram.

2. Percentage of transitions exercised

I also believe that I have covered every transition from one state to another. But I haven't tested every scenario in the program, so there are room for improvement.