Credit Card Validation

Design & Implementation

Summary

- Requirement
- Design Diagram
- Design Patterns
- Client API
- Luhn Validation implementation
- Method template pattern implementation
- Singleton pattern implementation
- Unit testing

Requirements

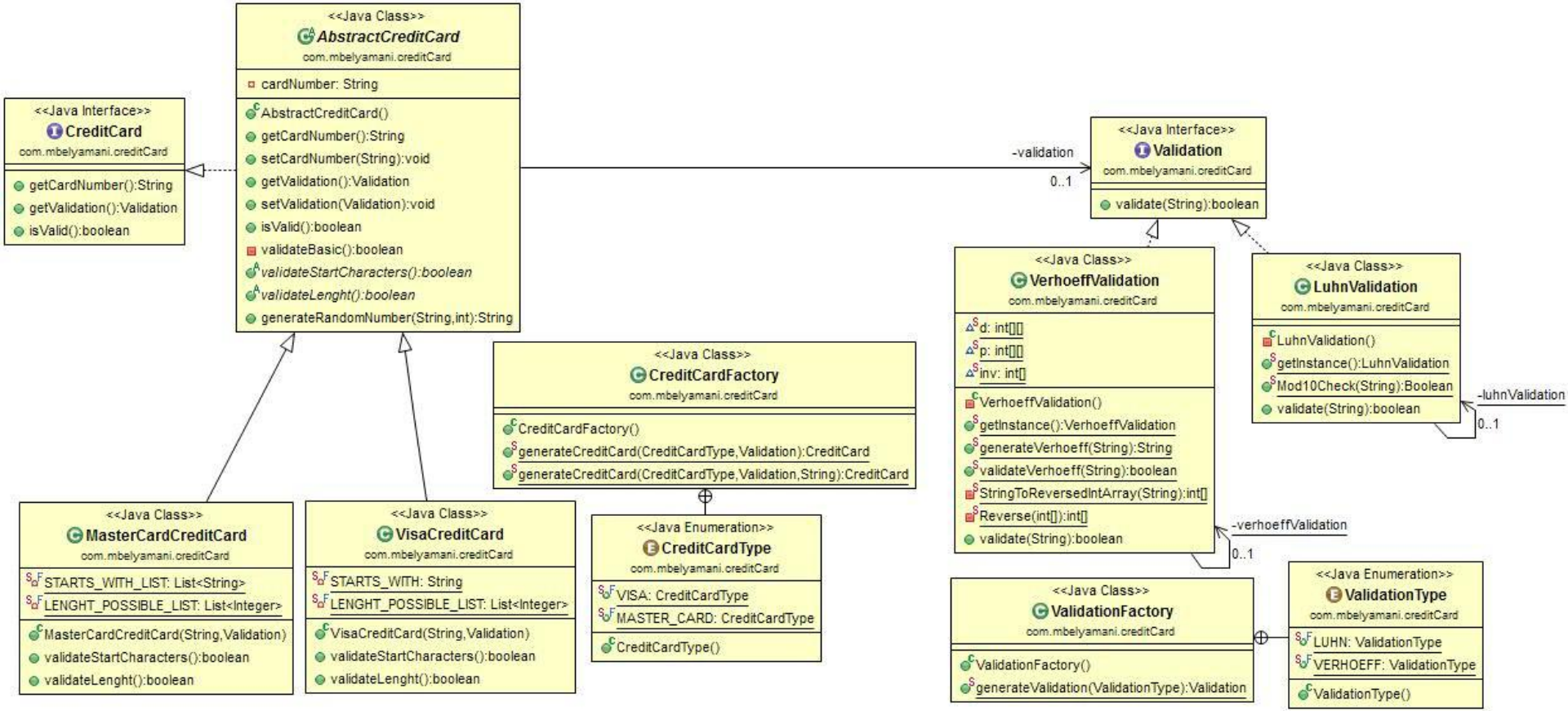
Functional requirements:

- Given a credit Card Number and Credit Card Type, user can create a credit card with the option to choose the validation algorithm
- Given a credit card type and a validation algorithm, user can can create a credit card with a valid credit card number (generate random valid card number).
- Given an existing credit card, user can check if the number is valid

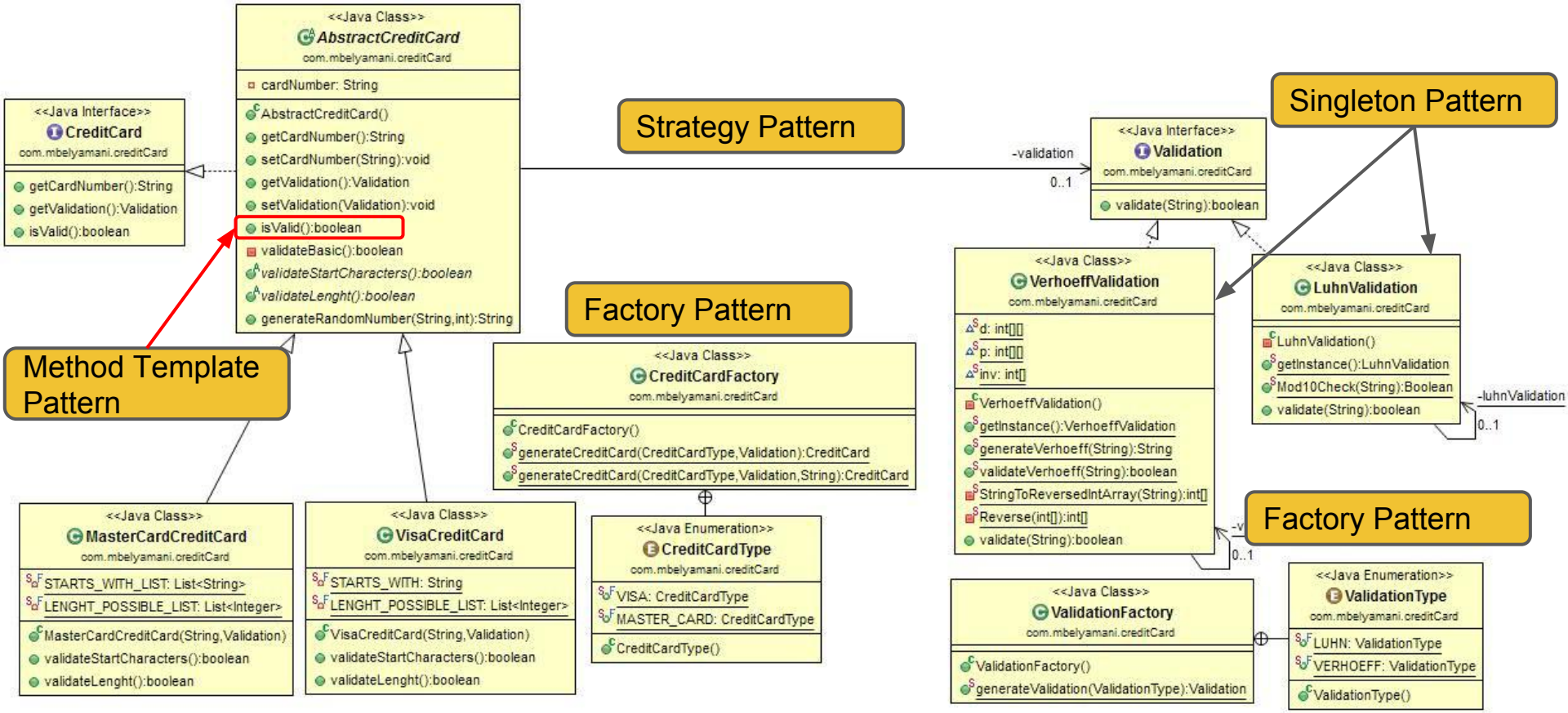
Architectural requirements

- Given a new credit card type, user can implement the code to handle it with minimal code without affecting the existing credit card supported
- Given a new algorithm validation, user can implement the code to handle it with minimal code change and without affecting the existing validation algorithm

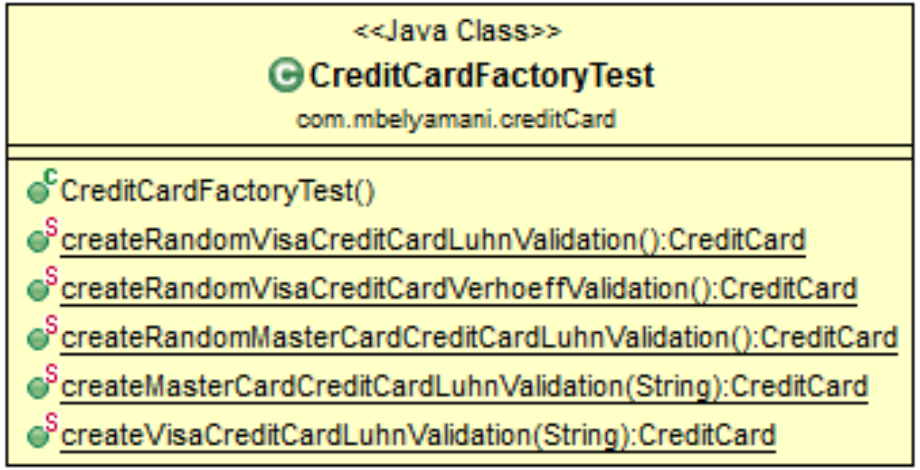
Credit Card validation diagram



Credit Card validation diagram - Design Patterns



Credit Card Validation - Client API



Luhn Validation Implementation - Java 8

```java
public static Boolean Mod10Check(String creditCardNumber) {
    // check whether input string is null or empty
    if (creditCardNumber == null || creditCardNumber.isEmpty())
        return false;

    // check whether input string contains non numerical characters
    String ccWithoutSpace = creditCardNumber.replace(" ", "");
    String regex = "\\d+";
    if (!ccWithoutSpace.matches(regex))
        return false;

    // 1.   Starting with the check digit double the value of every other digit
    // 2.   If doubling of a number results in a two digits number, add up
    //   the digits to get a single digit number. This will results in eight single digit numbers
    // 3. Get the sum of the digits

    Optional<Integer> result = IntStream.range(0, ccWithoutSpace.length()).mapToObj(e -> {
                                            if(e%2 == 0) {
                                                int val = Character.getNumericValue(ccWithoutSpace.charAt(e)) * 2;
                                                if(val > 9)
                                                    val = (int)(val / 10) + val % 10;
                                                return val;
                                            } else {
                                                return Character.getNumericValue(ccWithoutSpace.charAt(e));
                                            }
                                        }).reduce((a,b) -> a+b);


    System.out.println("Result :: " + result.get());
    return result.isPresent() && result.get().intValue()%10==0;
}
```

Method Template Pattern implementation

```
public abstract class AbstractCreditCard implements CreditCard{
...
        @Override
        public boolean isValid(){
            boolean basicValidation = validateBasic();
            if (!basicValidation)
                return false;

            boolean cardStatCharacterValidation = validateStartCharacters();
            if (!cardStatCharacterValidation)
                return false;

            boolean cardLenghtValidation = validateLenght();
            if (!cardLenghtValidation)
                return false;

            return getValidation().validate(getCardNumber());
        }

        private boolean validateBasic(){
            return getValidation()!=null && getCardNumber()!=null;
        }

        abstract public boolean validateStartCharacters();

        abstract public boolean validateLenght();
```

Singleton Pattern implementation

```java
public class LuhnValidation implements Validation{

    private volatile static LuhnValidation luhnValidation;

    private LuhnValidation() { }

    public static LuhnValidation getInstance() {
        if (luhnValidation == null) {
            synchronized(LuhnValidation.class) {
                if (luhnValidation == null) {
                    luhnValidation = new LuhnValidation();
                }
            }
        }

        return luhnValidation;
    }
```

Unit Testing

```java
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import org.junit.Test;

public class LuhnValidationTest {

    @Test
    public void testCreditCardValidatorEmptyString() {
        assertFalse(LuhnValidation.Mod10Check(""));
    }

    @Test
    public void testCreditCardValidatorValidNumber() {
        assertTrue(LuhnValidation.Mod10Check("4 0 1 2 8 8 8 8 8 8 8 8 1 8 8 1"));
    }

    @Test
    public void testCreditCardValidatorNonNumericalString() {
        assertFalse(LuhnValidation.Mod10Check("4 0 1 2 8 8 8 8 Toto 1 8 8 1"));
    }

    @Test
    public void testCreditCardValidatorNotValidNumber() {
        assertFalse(LuhnValidation.Mod10Check("4012 8888 8888 1882"));
    }


}
```

Unit Testing

```java
public class TestCreditCardFactoryTest {

    @Test
    public void testRandomVisaLuhn() {
        CreditCard cardNumber = CreditCardFactoryTest.createRandomVisaCreditCardLuhnValidation();
        assertEquals(cardNumber.isValid(), LuhnValidation.Mod10Check(cardNumber.getCardNumber()));
    }

    @Test
    public void testRandomVisaVerhoeff() {
        CreditCard cardNumber = CreditCardFactoryTest.createRandomVisaCreditCardVerhoeffValidation();
        assertEquals(cardNumber.isValid(), VerhoeffValidation.validateVerhoeff(cardNumber.getCardNumber()));
    }

    @Test
    public void testNotValidVisa() {
        CreditCard creditCard = CreditCardFactoryTest.createVisaCreditCardLuhnValidation("4650 0049 6922 0231");
        assertEquals("4650 0049 6922 0231",creditCard.getCardNumber());
        assertEquals(creditCard.isValid(), LuhnValidation.Mod10Check(creditCard.getCardNumber()));
    }

    @Test
    public void testValidVisaCreditCard() {
        CreditCard creditCard = CreditCardFactoryTest.createVisaCreditCardLuhnValidation("4 0 1 2 8 8 8 8 8 8 8 8 1 8 8 1");
        assertEquals("4 0 1 2 8 8 8 8 8 8 8 8 1 8 8 1",creditCard.getCardNumber());
        assertEquals(creditCard.isValid(), LuhnValidation.Mod10Check(creditCard.getCardNumber()));
    }

    @Test
    public void testValidNumberWithNonValidMasterCardStart() {
        CreditCard creditCard = CreditCardFactoryTest.createMasterCardCreditCardLuhnValidation("4 0 1 2 8 8 8 8 8 8 8 8 1 8 8 1");
        assertEquals("4 0 1 2 8 8 8 8 8 8 8 8 1 8 8 1",creditCard.getCardNumber());
```