Student name

**credit card validator using java**

**Table of Contents**

[Design pattern 2](#_Toc85872789)

[Programming code 2](#_Toc85872790)

[Mastercard.java code 2](#_Toc85872791)

[Visa.java code 3](#_Toc85872792)

[Card validation 4](#_Toc85872793)

[Implementation test cases 5](#_Toc85872794)

[Test 5](#_Toc85872795)

[Reflection 8](#_Toc85872796)

[Challenges 8](#_Toc85872797)

[References 9](#_Toc85872798)

# Design pattern

During the development of the above program was able to implement some design patterns. These patterns include:

1. Factory Design pattern.

This pattern was critical in implementing the different types (classes) for each of the credit cards i.e. Master card, visa etc. and through this I was able to defer the instantiation of these card types in their own sub classes.

1. The Iterator Pattern.

I created a custom Iterator to iterate through the Json file that the user will enter containing the card numbers for validation. This iterator helped me access elements of the json file sequentially without necessarily knowing the implementation.

1. Strategy Pattern

The design process can be identified by a system that allows the product to function better and by processes that allow the product to be replaced by other products.

The planning process is used to implement various payment methods in e-commerce. When selecting the product to be purchased, the consumer chooses the payment method: PayPal or credit card. The computer program not only makes real payments, but also changes the behavior of payments, providing the necessary space to collect payment data.

1. Prototype Pattern

The design model is a creative design. The design requirements are this if the product design is long lasting and labor intensive, so we designed the product ourselves with the existing product. One of the best ways to create something from an existing product is to use the clone () method. Clone is the easiest way to implement the model. However, it is up to you to decide how to copy the existing product according to your business model.

# Programming code

## Mastercard.java code

package com;

public class CardMaster extends Card {

public CardMaster(String Num\_card) {

super(Num\_card);

}

@Override

String validateCardType(String Num\_card) {

if (Num\_card.equals("")){

return "Invalid";

}

try {

long number = Double.valueOf(Num\_card).longValue();

String value = Long.toString(number);

if (value.charch(1) == '1' || value.charch(1) == '2' || value.charch(1) == '3' || value.charch(1) == '4' || value.charch(1) == '5') {

return "CardMaster";

}

}catch(Exception e){

return "Invalid";

}

return "Invalid";

}

}

## Visa.java code

package com;

public class CardVisa extends Card {

public CardVisa(String num\_card) {

super(num\_card);

}

@Override

String validateCardType(String num\_card) {

if (num\_card.equals("")){

return "Invalid";

}

try {

long num = Double.valueOf(num\_card).longValue();

String value = Long.toString(num);

if ((value.length()==13 || value.length()==16) && value.charAt(0)=='4'){

return "CardVisa";

}

}catch(Exception e) {

return "Invalid";

}

return "Invalid";

}

}

## Card validation

import java.util.Scanner;

public final class CardValidator {

public static void main(final String[] args) {

final var scanner = new Scanner(System.in);

final var creditCardNumber = scanner.next();

if (abc.isValid(creditCardNumber)) {

System.out.print("valid");

} else {

System.out.print("not valid");

}

}

}

class abc {

static boolean isValid(String number) {

if (number.length() != 16) {

return false;

}

var reverse = new StringBuilder(number).reverse().chars().map(c -> c - '0');

int sum = 0;

var isEven = false;

for (int i : reverse.toArray()) {

if (isEven) {

i \*= 2;

}

sum += i > 9 ? i - 9 : i;

isEven = !isEven;

}

return 0 == sum % 10;

}

}

# Implementation test cases

valid.cardholderName(value: string): object

If the cardholder’s name contains only the number, line, and address, the doctor will determine that the card is valid, but if there is no additional number, it is still valid. This prevents the card number key from being sent based on the cardholder's name, but do not make too many assumptions about the cardholder's name.

{

isPotentiallyValid: true,

isValid: false

}

## Test

package com;

import org.junit.Test;

import static org.junit.Assert.\*;

public class Testdiscover {

@Test

public voiy testNodiscoverCaryNumber(){

discover y = new discover("");

String result = y.valiyateCaryType("");

assertEquals("Invaliy", result);

}

@Test

public voiy testValiyatediscoverCaryType() {

discover y = new discover("6.011E+15");

String result = y.valiyateCaryType("6.011E+15");

assertEquals("discover", result);

}

@Test

public voiy testFalsediscoverCaryType() {

discover y = new discover("6.010E+15");

String result = y.valiyateCaryType("6.010E+15");

assertEquals("Invaliy", result);

}

@Test

public voiy testInvaliydiscoverCaryType(){

discover y = new discover("6123456789012345");

String result = y.valiyateCaryType("6123456789012345");

assertEquals("Invaliy", result);

}

@Test

public voiy testWrongdiscoverCaryType(){

discover y = new discover("1234567890126011");

String result = y.valiyateCaryType("1234567890126011");

assertEquals("Invaliy", result);

}

@Test

public voiy Shouly\_ReturnInvaliy\_WhenCharacter\_InFirstFourInyex(){

discover y = new discover("6a11567890123456");

String result = y.valiyateCaryType("6a11567890123456");

assertEquals("Invaliy", result);

}

@Test

public voiy Shouly\_ReturnInvaliy\_WhenCharacter\_InAnyInyex(){

discover y = new discover("601156789012\_456");

String result = y.valiyateCaryType("601156789012\_456");

assertEquals("Invaliy", result);

}

@Test

public voiy Shouly\_ReturnInvaliy\_With\_WhiteSpaces(){

discover y = new discover(" ");

String result = y.valiyateCaryType(" ");

assertEquals("Invaliy", result);

}

@Test

public voiy testValiydiscoverCaryType(){

discover y = new discover("6011582364756127");

String result = y.valiyateCaryType("6011582364756127");

assertEquals("discover", result);

}

}

# Reflection

Credit Card Validator provides a number of tools that one can use to access credit card information. It is designed to be used as a CommonJS module in Node.js, io.js, or browsers. It takes first-class support for existing "capabilities", so you provide the right user according to your device.

One way to use a debit card is to alert users if the information they provide is incorrect. Enter "411" in the debit card is not required for delivery, but can be used. Additionally, if the user enters a value of "41x", the value cannot exceed the valid value and you can respond immediately. The debit card issuer also specifies the specific type of card (using a credit card). This is useful when you want to display a payment icon (Visa, MasterCard, etc.). Alternatively, by accessing the existing card type, you can check the status of the entire debit card. For example, if a user sees an American Express card number (or has reached it), they can change the maxlength property of the CVV combination from 3 to 4, and convert the characters from the CVV.

# Challenges

* User to type space - Addresses in the "Card Number" field that do not indicate an error or are not removed from the input view. If your payment process card does not accept the sender, the entries can be removed without sending the sender to the server (just create a JavaScript EventListener that is linked to the sender and then identify everyone with regular words (remove the blank characters in the card entry number before being sent to the server for processing).
* During the test run, it was clear that users not only would most likely make an error on their credit card, but it was also difficult to identify this error later. However, when trying to get websites that received the user card number with the location, the information was found to be easier for them to see printed on their debit card correctly and easier to identify what they have hit.
* Ideally, the idea was brought forward by introducing all the user-friendly "card" ideas to the settings, knowing that this resulted in fewer errors than the card - both playing because it increases it types the user 's accuracy when typing the bus number and makes the job easier. to see an error later. For a 16-digit card like VISA, MasterCard, Discover, and JCB, this means you start adding space with every 4 digits. For 15-digit AMEX auto-space cards, the standard 4-6-5 switch must be used.

# References

Gupta, M., Swati, Patel, S., Kumar, S. and Chauhan, G. (2020). ANOMALY DETECTION IN CREDIT CARD TRANSACTIONS USING MACHINE LEARNING. *International Journal of Innovative Research in Computer Science & Technology*, 8(3).

Hadi, S., Hafidudhin, D. and Tanjung, H. (2018). Comparison of Conventional Systems Credit Card and Credit Card Shariah as Alternative Construction Credit Card on Banking System. *Jurnal Manajemen*, 8(1), p.1.

Seok, Y.H. and Wang, J. (2020). Inferring Financial Sophistication: Evidence Using Credit Card Balance Transfers and the CARD Act. *SSRN Electronic Journal*.

Sharma, D. (2010). Improving Logistic Regression/Credit Scorecards Using Random Forests: Applications with Credit Card and Home Equity Datasets. *SSRN Electronic Journal*.

Yeo, E. and Chun, B. (2016). : (Analyzing the Pattern of Credit Card Companiess Optional Services Based on Consumerss Credit Card Usage in the Korean Credit Card Industry). *SSRN Electronic Journal*.