Linnaeus University

1DV532 – Starting out with Java
Assignment 2 Report

Student: Evan Huynh
Student ID: eh223im@student.lnu.se

Table of Contents

Report

Exercise 1: Int

The exercise requires basic arithmetic for two integers in "Int" type. For the plus and div methods, we simply add or divide each of their respected field and return type "Int".

Exercise 2: SweID

This exercise contains lots of static classes for Swedish ID and checkers. However, we assume that users will supply ID the form of YYMMDD-XXXX, otherwise we will passthrough *inputHandlingAndConvertingIntoTheCorrectFormToFurtherProcessingIDCorrectly* method and convert it into the correct form. This was based on the previous source code with improvement. [1] Since data is shared between every other method, all of them are static in the SweID class.

- *getFirstPart* and *getSecondPart*: implying that user supply the correct form of the ID (i.e. 010203-0405), simply getting first 6 characters and last 4 characters of the ID, respectively.
- *isFemaleNumber* checks the third digit of 4 last digits is divisible by 2, otherwise it is a male number.
- *areEqual* is *toString* but worse in performance.
- *isCorrect* checks if the ID is valid and requires 3 other methods: *isValidMonth, isValidDate, isValidChecksum*
    - o *isValidMonth* checks if the month is between 01 to 12 (Gregorian calendar)
    - o *isValidDate* check if the date is within the range of the month, meaning that if the month does not exist, *isValidDate* always returns false. It is also check if the year is leap to decide whether that February has 29 days or not.
        - ▪ *isLeap* check if the year is leap. It is the year that divisible by 4, and if the year is divisible by 100, it should also be divisible by 400.
    - o *isValidChecksum* returns true if the checksum algorithm is similar as the result returned by *getChecksum*
        - ▪ *getChecksum* obtains first 9 digits of the ID then perform the Luhn's algorithm. The even-placed digits (count from 0) are multiplied by 2 then get the sum of each digit in the result, odd-placed digits are multiplied by 1. We then add each result in every step to the total sum. After that we get 10 minus last digit of the total sum to obtain the checksum digit.
    - o If all those 3 methods return true then the ID is valid.
- *IDVerificationWithInformationToString* returns the string to report whether the ID is correct (with genders) or invalid ID with reasons. All the outputs in the example are exported using this method.
- *inputHandlingAndConvertingIntoTheCorrectFormToFurtherProcessingIDCorrectly* returns the String in the form of YYMMDD-XXXX. It supports 4 types of strings:

[4Y/2Y]MMDD(-)XXXX.

Note: since processing 2-digit years was the problem of those developers before Y2K, we shall use the Windows XP default approach when dealing with 2-digit years, or only between 1930 and 2029.

Exercise 3: Pizza

This exercise requires another input for the pizza type, topping types and descriptions.

- Constructor: this constructor has 3 parameters, pizza size, topping type and number of toppings per pizza.
- *calcCost* returns the pizza costs base on the size and cost per toppings for each of the respected size.
- *getDescription* prints out order information.
- *getType* returns type of topping, if there is any topping that is larger than 0.
- *getTopping* returns the number of toppings per pizza.

In the class PizzaMain, it is simply an implementation of the Pizza with more texts.

Exercise 4: Money

Like exercise 1 and 3, this one requires input for money with number of dollars and cents.

- Constructors: there are three constructors:
    - With dollar and cent: set the number of dollar and cent according to the input
        - If the cent is larger than 100 or smaller than 0, the cent is then taken modulo of 100 and subtracted into the amount of dollar needed.
    - With dollar only: the cent is set to 0 and dollar to the user input.
    - Without dollar and cent: the default money $0.00 is constructed.
- *add* returns the sum of two money values by adding dollar to dollar, cent to cent and returns a new money with the dollar and cent values obtained, in the “Money” type.
- *minus* returns the different of two money values by subtracting first dollar to second dollar, first cent to second cent and returns a new money with the dollar and cent values obtained, in the “Money” type.
- *toString* returns the money description, with appropriate dollar sign, minus sign and delimiter (by default: comma).

Source code

1

Int

```java
/**
 * Int for exercise 1
 *
 * @version 2.1
 * @author Evan Huynh
 */
package eh223im_assign2;

public class Int {
    // Field
    private int number;

    // Standard getter and setters
    public int getNumber() {
        return number;
    }

    public void setNumber(int number) {
        this.number = number;
    }

    // Constructor
    public Int(int number) {
        this.number = number;
    }

    // Method

    /**
     * Add two int
     * @param a
     * @return
     */
    public Int plus(Int a) {
        return new Int(number+a.getNumber());
    }

    /**
     * Divide two int
     * @param a
     * @return
     */
    public Int div(Int a) {
        return new Int(number/a.getNumber());
    }

    /**
     * Compare to see if int > input
     * @param a
     * @return
     */
    public boolean isLargerThan(Int a) {
        return (number > a.getNumber());
    }
    /**
     * Return if a-b is 0.
     * @param a
     * @return
     */
    public boolean isEqualTo(Int a) {
        return (number == a.number);
    }

    /**
     * Returns the string
     * @return a string
     */
    public String toString() {
        return "Int("+ number +")";
    }

    /**
     * Main class does something
     * @param args
     */


    public static void main(String[] args) {
        Int i1 = new Int(5);
        Int i2 = new Int(2);
        Int sum = i1.plus(i2); // the plus method adds i1 and i2 and re-
turns
        // result of the addition, resulting sum = 7
        Int div = i1.div(i2); // the div method divides i1 by i2 and re-
turns
        // result of the division, resulting div = 2
        if ( sum.isLargerThan(i1) )
            System.out.println("Sum "+ sum.toString()+ " is larger than "
                    +i1.toString() );
        if ( div.isEqualTo(i2) )
            System.out.println("Both are equal");
        else
            System.out.println("They are not equal");
    }
}
```

2

SweID

```java
/**
 * Toolkit for Swedish ID
 *
 * @version 2.2
 * @author Evan Huynh
 */
package eh223im_assign2;

import java.util.Random;

public class SweID {
    // I have done this one before
    // Algorithm remains similar.
    // Here is the source, for citation, because of academic rules:
    // https://raw.githubusercontent.com/My2ndAngelic/1DV506/mas-
ter/src/main/java/eh223im_assign3/SweID.java
```

```java
    // Here is the proper citation for the code, just in case.
    /**
     * Author(s) name: My2ndAngelic
     * Date: 2017 Dec
     * Title of program/source code: SweID
     * Code version: N/A
     * Type: computer program
     * Web address: https://raw.githubusercontent.com/My2ndAn-
gelic/1DV506/master/src/main/java/eh223im_assign3/SweID.java
     */

    /**
     * Return the first part of the ID, requires full ID in string.
     * @param sweID
     * @return first part of the ID
     */
    private static String getFirstPart(String sweID) {
        return sweID.substring(0,6);
    }

    /**
     * Return the second part of the ID, requires full ID in string.
     * @param sweID
     * @return second part of the ID
     */
    private static String getSecondPart(String sweID) {
        return sweID.substring(7,11);
    }

    /**
     * Check if ID is female, requires full ID in string.
     * @param sweID
     * @return true if female
     */
    private static boolean isFemaleNumber(String sweID) {
        int a = Integer.parseInt(sweID.substring(9,10));
        return a%2==0; //first digit in second part is even
    }

    // Written in 2020. Return year in 2 digit or 4 digits.
    private static int getYear(String sweID, int n) {
        int a = Integer.parseInt(sweID.substring(0,2));
        if (n==2) { // Return only year in 2 digits
            return a;
        } else if (n==4) { // Return year in 4 digits
            if (a >= 30) { // No one is born in 2030, yet
                a += 1900;
            } else { // From 2000-2029
                a += 2000;
            }
            return a;
        } return 0;
    }

    /**
     * Self-explanatory
     * @param sweID
     * @return
     */
    private static int getMonth(String sweID) {
        return Integer.parseInt(sweID.substring(2,4));
    }

    /**
     * Self-explanatory
     * @param sweID
     * @return
     */
    private static int getDay(String sweID) {
        return Integer.parseInt(sweID.substring(4,6));
    }

    /**
     * Check for valid month
     * @param sweID
     * @return true if getMonth() returns 1 - 12
     */
    private static boolean isValidMonth(String sweID) {
        return getMonth(sweID) >= 1 && getMonth(sweID) <= 12;
    }

    /**
     * Check for leap year.
     * @param sweID
     * @return true if getYear return a leap year.
     */
    private static boolean isLeap(String sweID) {
        int year = getYear(sweID,4);
        return year % 4 == 0 && (year % 100 == 0 && year % 400 == 0); //
Standard leap year algorithm. Those year if divides 4, and if divides by
100 it should not divides 400.
    }

    /**
     * Check for valid date
     * This one has been adopted and fixed some bugs from the original
source.
     * @param sweID
     * @return true if it is valid
     */
    private static boolean isValidDate(String sweID) {
        int month = getMonth(sweID);
        int day = getDay(sweID);
        boolean leap = isLeap(sweID);
        int min = 1; int max;

        switch (month) { // Switching max and min for the day in the
month.
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                max = 31;
                break;
            case 2:
                if (leap) {
                    max = 29;
                } else {
                    max = 28;
                }
                break;
            case 4:
            case 6:
            case 9:
            case 11:
                max = 30;
                break;
            default:
                min = -1;
                max = -1;
                break;
        }

        // Check starts here
        boolean check = false;
        if (month >= 1 && month <= 12) { // Check for months
            if ((day >= min) && (day <= max)) { // Check for days in the
month
                check = true;
            }
        }
        return check;
    }

    /**
     * Get the checksum digit, only process first 9 digits YYMMDD-NNN of
the ID
     * @param sweID
     * @return checksum value
     */
    private static int getChecksum(String sweID) {
        int a, b = 0;
        String sweID2=sweID.substring(0,6) + sweID.substring(7,10);
        for (int i = 0; i < sweID2.length(); i++) { // multiplication
pattern: 21212-121
            int i1 = Integer.parseInt(sweID2.substring(i,i+1));
            if (i % 2 == 0) { // Even place multiply by 2 (java counts
from 0), then sum the digit. Max: 2*9=18, 1+8=9
                a = i1 * 2;
                a = a/10 + a%10;
            } else { // Just return the current digit
                a = i1;
            }
            b+=a; // Add them to the sum
        }
        b = (10 - (b % 10)) % 10;  // (10 - lastdigt) of the sum and get
the last digit of the result.

        return b;
    }

    /**
     * Check checksum validity
     * @param sweID
     * @return if checksum in getChecksum and in ID are the same.
     */
    private static boolean isValidChecksum(String sweID) {
        return getChecksum(sweID) == Integer.parseInt(sweID.sub-
string(10,11));
```

```
    }

    /**
     * Return if ID is valid
     * @param sweID
     * @return true if valid
     */
    static boolean isCorrect(String sweID) {
        return isValidMonth(sweID) && isValidDate(sweID) && isValidCheck-
sum(sweID);
    }


    /**
     * If those two string are equal
     * @param id1
     * @param id2
     * @return just like .equals, but worse
     */
    static boolean areEqual(String id1, String id2) {
        StringBuilder substr = new StringBuilder(), substr2 = new String-
Builder();
        for (int i = 0; i < id1.length(); i++) {
            if (Character.isDigit(id1.charAt(i))) {
                substr.append(id1.charAt(i));
            }
        }
        for (int i = 0; i < id2.length(); i++) {
            if (Character.isDigit(id2.charAt(i))) {
                substr2.append(id2.charAt(i));
            }
        }
        return substr.toString().equals(substr2.toString());
    }

    /**
     * ID generator without parameters, for testing purpose only.
     * @return a valid ID, YYMMDD-XXXX
     */
    private static String genID() {
        StringBuilder str;
        Random ran = new Random();
        do {
            str = new StringBuilder();
            int year = Integer.parseInt(String.valueOf(ran.nextInt(99) +
1));
            int month = Integer.parseInt(String.valueOf(ran.nextInt(12) +
1));
            int day = Integer.parseInt(String.valueOf(ran.nextInt(31) +
1));
            int gender = Integer.parseInt(String.valueOf(ran.nextInt(9) +
1));
            int person = Integer.parseInt(String.valueOf(ran.nextInt(99)
+ 1));
            str.append(String.format("%02d", year)).append(String.for-
mat("%02d", month)).append(String.format("%02d", day)).append("-").ap-
pend(gender).append(String.format("%02d", person));
            int checksum = getChecksum(str.toString());
            str.append(checksum);
        } while (!isCorrect(str.toString()));
        return str.toString();
    }

    /**
     * Verify and return ID check result
     * @param sweID
     * @return a string of ID check result
     */
    private static String IDVerificationWithInformationToString(String
sweID) {
        StringBuilder a = new StringBuilder();
        a.append(sweID);
        a.append(" is a");
        if (isCorrect(sweID)) { // if correct
            if (isFemaleNumber(sweID)) {
                a.append(" valid female number");
            } else {
                a.append(" valid male number");
            }
        } else { // if incorrect
            a.append("n invalid number "); // adding some grammar.
            a.append("(");
            if (!isValidMonth(sweID)) { // just month
                a.append("invalid month, ");
            }
            if (!isValidDate(sweID)) { // this will happen if invalid
month happens, or alone
                a.append("invalid date, ");
            }
            if (!isValidChecksum(sweID)) { // checksum
                a.append("invalid checksum, ");
            }
            a.delete(a.length()-2, a.length());
            a.append(")");
        }
        a.append(".");
        return a.toString();
    }

    /**
     * Convert inputs to correct form, supports [4Y/2Y]MMDD(-)XXXX
     * @param sweID
     * @return a string in the form of YYMMDD-XXXX
     */
    private static String inputHandlingAndConvertingIntoTheCorrectFormTo-
FurtherProcessingIDCorrectly(String sweID) {
        StringBuilder a = new StringBuilder();
        boolean b = sweID.length() >= 10;
        boolean c = sweID.contains("-");
        boolean d = sweID.indexOf("19") == 0 || sweID.indexOf("20") == 0;
        if (b) { // Check correct length, maybe???
            if (c) { // if there is a dash
                if (d) { // YYYYMMDD-XXXX
                    a.append(sweID, 2, sweID.length());
                } else { // YYMMDD-XXXX
                    a.append(sweID);
                }
            } else { // if there is no dash
                if (d) { // YYYYMMDDXXXX
                    a.append(sweID, 2, 8);
                    a.append("-");
                    a.append(sweID, 8, sweID.length());
                } else { // YYMMDDXXXX
                    a.append(sweID, 0, 6);
                    a.append("-");
                    a.append(sweID, 6, sweID.length());
                }
            }
        } // no outside else
        return a.toString(); // return everything if it can process, oth-
erwise nothing
    }

    // Finally
    public static void main(String[] args) {
        System.out.println(IDVerificationWithInformationToString(in-
putHandlingAndConvertingIntoTheCorrectFormToFurtherPro-
cessingIDCorrectly("19640123-8826"))); // 12 digits with dash
        System.out.println(IDVerificationWithInformationToString(in-
putHandlingAndConvertingIntoTheCorrectFormToFurtherPro-
cessingIDCorrectly("195504140913"))); // 12 digits without dash
        System.out.println(IDVerificationWithInformationToString(in-
putHandlingAndConvertingIntoTheCorrectFormToFurtherPro-
cessingIDCorrectly("551314-0913"))); // 10 digit with dash
        System.out.println(IDVerificationWithInformationToString(in-
putHandlingAndConvertingIntoTheCorrectFormToFurtherPro-
cessingIDCorrectly("5504140912"))); // 10 digit without dash
    }
}
```

3

Pizza

```
/**
 * Pizza calculator for exercise 3
 *
 * @version 2.3
 * @author Evan Huynh
 */

package eh223im_assign2;

public class Pizza {

    // Field
    private String size;
    private int cheese;
    private int pepperoni;
    private int ham;

    // Constructor
    public Pizza(String size, String type, int topping) {
        this.size = size;
        switch (type.toLowerCase()) {
            case "cheese":
                this.cheese = topping;
                break;
            case "pepperoni":
                this.pepperoni = topping;
                break;
            case "ham":
                this.ham = topping;
                break;
        }
    }

    // Methods

    /**
     * Pizza cost
     * @return how much it cost
     */
    public double calcCost() {
        double base = 0; double top = 0;
        switch (size.toLowerCase()) {
            case "small":
                base = 10;
                top = 3;
                break;
            case "medium":
                base = 15;
                top = 2.5;
                break;
            case "large":
                base = 20;
                top = 2;
                break;
        }
        return base+top*cheese+top*pepperoni+top*ham; // Assuming multi-
ple topping
    }

    /**
     * Pizza description
     * @return description
     */
    public String getDescription() {
        return "You ordered a "+ getSize().toLowerCase()+" pizza with "
+getTopping()+" "+ getType().toLowerCase()+ " toppings. Your bill is
"+String.format("%.02f",calcCost())+ " kr.";
    }

    /**
     * Return type, only one type allowed from cheese to ham.
     * @return whatever available first
     */
    public String getType() {
        String a = "";
        if (cheese > 0) {
            a += "cheese";
        } else if (pepperoni > 0) {
            a += "pepperoni";
        } else if (ham > 0) {
            a += "ham";
        }
        return a;
    }

    // Paperwork methods
    public String getSize() {
        return size;
    }
    public void setSize(String size) {
        if (size.toLowerCase().equals("small") || size.toLower-
Case().equals("medium") || size.toLowerCase().equals("large")) {
            this.size = size;
        }
    }

    /**
     * Return topping type
     * @return
     */
    public int getTopping() {
        // Since pizza only contains one topping.
        if (cheese > 0) {
            return cheese;
        } else if (pepperoni > 0) {
            return pepperoni;
        } else if (ham > 0) {
            return ham;
        }
        return 0;
    }

}
```

PizzaMain

```
/**
 * Pizza Main
 *
 * @version 2.3
 * @author Evan Huynh
 */
package eh223im_assign2;

import java.util.Scanner;

public class PizzaMain {
    public static void main(String[] args) {
        System.out.println(java.util.Calendar.getInstance().getTime());
// Idea: https://www.javatpoint.com/java-get-current-date

        Scanner s = new Scanner(System.in);
        String a = "";
        String b = "";
        int c = -1;

        System.out.println("\nWelcome to the Café LNU!\n");

        System.out.print("Please enter size of your pizza [small, medium,
or large]: ");
        while (!a.toLowerCase().equals("small") && !a.toLower-
Case().equals("medium") && !a.toLowerCase().equals("large")) {
            a = s.next();
        }

        System.out.print("Please enter type of topping [cheese, pepper-
oni, ham]: ");
        while (!b.toLowerCase().equals("cheese") && !b.toLower-
Case().equals("pepperoni") && !b.toLowerCase().equals("ham")) {
            b = s.next();
        }
        System.out.print("Please enter number of toppings you want: ");
        while (c<=0) {
            c = s.nextInt();
        }
        Pizza pizza = new Pizza(a,b,c);
        System.out.print("\nThank you. ");
        System.out.println(pizza.getDescription());
        System.out.print("\nEnjoy your food.");
    }
}
```

4

Money

```
/**
 * Money class for exercise 4
 *
 * @version 2.4
 * @author Evan Huynh
 */
package eh223im_assign2;

public class Money {
    // Fields
    private int dollar;
    private int cent;

    // Constructors
    // Idea for the cent counting: https://stackoverflow.com/ques-
tions/4412179/best-way-to-make-javas-modulus-behave-like-it-should-with-
negative-numbers
    /**
     * Basic constructor
     * @param dollar
     * @param cent
     */
    public Money(int dollar, int cent) {
        if (cent >= 0) { // for positive cent
            this.dollar = dollar + (cent / 100); // add them up, in case
someone enter something over 100 cents
            this.cent = cent % 100; // mod the cent down after adding
them to the dollar part
        } else { // for negative cent
            this.dollar = dollar - Math.abs(cent / 100);
            this.cent = (cent % 100 + 100) % 100;
        }
    }

    /**
     * Dollar only constructor
     * @param dollar
     */
    public Money(int dollar) {
        this.dollar=dollar;
        this.cent=0;
    }

    /**
     * Default constructor
     */
    public Money() {
        this.dollar=0;
        this.cent=0;
    }

    /**
     * Money constructor
     * @param money
     */
    public Money(Money money) {
        new Money(money.dollar,money.cent); // construct itself using the
first constructor.
    }

    // Methods

    /**
     * Add two moneys
     * @param money
     * @return
     */
    public Money add(Money money) {
        int a = money.getDollar();
        int b = money.getCent();
        a += this.dollar;
        b += this.cent;
        return new Money(a,b);
    }

    /**
     * Minus two moneys
     * @param money
     * @return
     */
    public Money minus(Money money) {
        int a = money.getDollar();
        int b = money.getCent();
        a = this.dollar - a; // subtract them by dollar
        b = this.cent - b; // subtract them by cent
        return new Money(a,b); // construct a new money using the result.
    }

    /**
     * Export dollar, with delimiter choice
     * @param delimiter
     * @return
     */
    public String toString(String delimiter) {
        if (dollar >= 0) {
            return "$" + dollar + delimiter + String.format("%02d",
cent);
        } else {
            return "-$" + Math.abs(dollar) + delimiter + String.for-
mat("%02d", cent);
        }
    }

    /**
     * Export dollar, with delimiter as ','
     * @return
     */
    public String toString() {
        String delimiter = ",";
        if (dollar >= 0) {
            return "$" + dollar + delimiter + String.format("%02d",
cent);
        } else {
            return "-$" + Math.abs(dollar) + delimiter + String.for-
mat("%02d", cent);
        }
    }

    // Standard getter and setters
    /**
     * Return dollar
     * @return dollar
     */
    public int getDollar() {
        return dollar;
    }

    /**
     * Set dollar
     * @param dollar
     */
    public void setDollar(int dollar) {
        this.dollar = dollar;
    }

    /**
     * Get cents
     * @return
     */
    public int getCent() {
        return cent;
    }

    /**
     * Generate new money when setting cent, only 0 - 99.
     * @param cent
     */
    public void setCent(int cent) {
        if(cent>=0 && cent<=99) {
            this.cent = cent;
        }
    }
}
```

MoneyMain

```
/**
 * MoneyMain class
 *
 * @version 2.4
 * @author Evan Huynh
 */
package eh223im_assign2;

public class MoneyMain {
    public static void main(String[] args) {

        System.out.print("Money 1 is: ");
        Money money1 = new Money (2,00);
        System.out.println(money1.toString());
        System.out.print("Money 2 is: ");
        Money money2 = new Money (5,05);
        System.out.println(money2.toString());

        System.out.print("Money1' dollar is: ");
        System.out.println(money1.getDollar());
        System.out.print("Money1' cent is: ");
        System.out.println(money1.getCent());
        System.out.print("Money2' dollar is: ");
        System.out.println(money2.getDollar());
        System.out.print("Money2' cent is: ");
        System.out.println(money2.getCent());

        System.out.print("Adding money1 to money2: ");
        Money c = money2.add(money1);
        System.out.println(c.toString());
        System.out.print("Subtracting money1 from money2: ");
        Money d = money2.minus(money1);
        System.out.println(d.toString());

        Money e = new Money(4,-101);
        System.out.print("Money 3 is: "+e.toString());
    }
}
```

Bibliography

[1] My2ndAngelic, "SweID," 30 Dec 2017. [Online]. Available: https://raw.githubusercontent.com/My2ndAngelic/1DV506/master/src/main/java/eh223im_assign3/SweID.java. [Accessed 1 July 2020].