Универзитет „Св. Кирил и Методиј“ - Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Напредно програмирање

Аудиториски вежби 3

Верзија 1.0, 23 Септември, 2016

Содржина

1. Датум

Имплементирајте едноставна класа за датум `Date`. Класата треба да овозможува:

- репрезентација на сите датуми од 1 Јануари 1800 до 31 Декември 2500
- одземање два датуми
- зголемување датум за даден број денови
- да споредува два датуми со помош на `equals` и `compareTo`.

Датумот внатрешно ќе се репрезентира како број на денови од почетното време, кое во овој случај е почетокот на 1800. Со ова сите методи освен методот `toString` се поедноставуваат. Да се запази правилото за престапни години (престапна година е секоја година која е делива со 4 и не е делива со 100 освен ако е делива со 400). Конструкторот треба да ја провери валидноста на датумот, а исто така и методот `toString`. Датумот може да стане неточен ако зголемувањето или намалувањето придонсе да излезе надвор од опсегот.

Потешкиот дел од задачата е конверзијата од интерната репрезентација во надворешна репрезентација на датум. Еден предлог алгоритам е следниот. Се иницијализираат две низи во статички членови. Првата низа е денови до први во месецот (daysTillFirstOfMonth) во не престапна година. Оваа низа содржи 0, 31, 59, 90, итн. Во втората низа, денови од почетокот на првата година (daysTillJan1) ќе содржи 0, 365, 730, 1095, 1460, 1826, итн. Со помош на овие низи ќе ја правиме конверзијата на различни репрезентации на датум.

Решение (`Date.java`)

```
package mk.ukim.finki.np.av3;

public class Date {

    private static final int FIRST_YEAR = 1800;
    private static final int LAST_YEAR = 2500;
    private static final int DAYS_IN_YEAR = 365;

    private static final int[] daysOfMonth = {
            31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31
    };
    private static final int[] daysTillFirstOfMonth;
    private static final int[] daysTillJan1;

    static {
        daysTillFirstOfMonth = new int[12];
        for (int i = 1; i < 12; i++) {
```

```
            daysTillFirstOfMonth[i] += daysTillFirstOfMonth[i - 1] + daysOfMonth[i - 1];
        }
        int totalYears = LAST_YEAR - FIRST_YEAR + 1;
        daysTillJan1 = new int[totalYears];
        int currentYear = FIRST_YEAR;
        for (int i = 1; i < totalYears; i++) {
            if (isLeapYear(currentYear)) {
                daysTillJan1[i] = daysTillJan1[i - 1] + DAYS_IN_YEAR + 1;
            } else {
                daysTillJan1[i] = daysTillJan1[i - 1] + DAYS_IN_YEAR;
            }
            currentYear++;
        }
    }

    private static boolean isLeapYear(int year) {
        return (year % 400 == 0 || (year % 4 == 0 && year % 100 != 0));
    }

    private final int days;

    public Date(int days) {
        this.days = days;
    }

    public Date(int date, int month, int year) {
        int days = 0;
        if (year < FIRST_YEAR || year > LAST_YEAR) {
            throw new RuntimeException();
        }
        days += daysTillJan1[year - FIRST_YEAR];
        days += daysTillFirstOfMonth[month - 1];
        if (month > 2 && isLeapYear(year)) {
            days++;
        }
        days += date;
        this.days = days;
    }

    public int subtract(Date date) {
        return this.days - date.days;
    }

    public Date increment(int days) {
        return new Date(this.days + days);
    }

    @Override
    public boolean equals(Object other) {
        Date date = (Date) other;
        return this.days == date.days;
    }

    public int compareTo(Date date) {
        return this.days - date.days;
    }

    @Override
    public String toString() {
        int d = days;
        int i;
        for (i = 0; i < daysTillJan1.length; i++) {
            if (daysTillJan1[i] >= days) {
                break;
            }
        }
        d -= daysTillJan1[i - 1];
        int year = FIRST_YEAR + i - 1;
        if (isLeapYear(year)) {
            d--;
        }
        for (i = 0; i < daysTillFirstOfMonth.length; i++) {
            if (daysTillFirstOfMonth[i] >= d) {
                break;
            }
        }
```

```
            int month = i;
            d -= daysTillFirstOfMonth[i - 1];
            return String.format("%02d.%02d.%4d", d, month, year);
        }

        public static void main(String[] args) {
            Date sample = new Date(1, 10, 2012);
            System.out.println(sample.subtract(new Date(1, 1, 2000)));
            System.out.println(sample);
            sample = new Date(1, 1, 1800);
            System.out.println(sample);
            sample = new Date(31, 12, 2500);
            System.out.println(daysTillJan1[daysTillJan1.length - 1]);
            System.out.println(sample.days);
            System.out.println(sample);
            sample = new Date(30, 11, 2012);
            System.out.println(sample);
            sample = sample.increment(100);
            System.out.println(sample);
        }
    }
```

2. Карти (PlayingCard)

`PlayingCard` (карта) е класа со која се репрезентира карта во игри со карти како покер и блек џек. Во неа се чува информација за бојата (херц, каро, пик, треф) и рангот (вредност од 2 до 10 или џандар, кралица, поп или ас). `Deck` (шпил од карти) е класа која репрезентира комплет од 52 карти. Додека `MultipleDeck` (повеќе шпилови) е класа која репрезентира повеќе шпилови со карти (точниот број се задава во конструкторот). Да се имплментираат 3 класи `PlayingCard`, `Deck`, и `MultipleDeck`, со стандардна функционалност за карта, за шпил и повеќе шпилови. Да се имплментираат методи за мешање `shuffle`, делење на карта `dealCard` и проверка дали има останато карти.

Решение (`PlayingCard.java`)

```java
package mk.ukim.finki.np.av3;

public class PlayingCard {
    public enum TYPE {
        HEARTS,
        DIAMONDS,
        SPADES,
        CLUBS
    }

    private TYPE type;
    private int rank;

    public PlayingCard(TYPE type, int rank) {
        this.type = type;
        this.rank = rank;
    }

    @Override
    public String toString() {
        return String.format("%d %s", rank, type.toString());
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + rank;
        result = prime * result + ((type == null) ? 0 : type.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        PlayingCard other = (PlayingCard) obj;
        if (rank != other.rank)
            return false;
        if (type != other.type)
            return false;
        return true;
    }


}
```

Решение (`Deck.java`)

```
package mk.ukim.finki.np.av3;

import java.util.Arrays;

public class Deck {
    private PlayingCard[] cards;
    private boolean[] picked;
    private int total;

    public Deck() {
        total = 0;
        cards = new PlayingCard[52];
        picked = new boolean[52];
        for (int i = 0; i < PlayingCard.TYPE.values().length; ++i) {
            for (int j = 0; j < 13; ++j) {
                cards[j + (13 * i)] =
                        new PlayingCard(PlayingCard.TYPE.values()[i], j + 1);
            }
        }
    }

    @Override
    public String toString() {
        StringBuilder result = new StringBuilder();
        for (PlayingCard playingCard : cards) {
            result.append(playingCard);
            result.append("\n");
        }
        return result.toString();
    }

    public static void main(String[] args) {
        Deck deck = new Deck();
        PlayingCard card;
        while ((card = deck.dealCard()) != null) {
            System.out.println(card);
        }
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + Arrays.hashCode(cards);
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Deck other = (Deck) obj;
        return Arrays.equals(cards, other.cards);
    }

    public PlayingCard dealCard() {
        if (total == 52) return null;
        int card = (int) (52 * Math.random());
        if (!picked[card]) {
            ++total;
            picked[card] = true;
            return cards[card];
        }
        return dealCard();
    }

}
```

Решение (`MultipleDeck.java`)

```
package mk.ukim.finki.np.av3;

public class MultipleDeck {
    private Deck[] decks;

    public MultipleDeck(int n) {
        decks = new Deck[n];
        for (int i = 0; i < n; ++i) {
            decks[i] = new Deck();
        }
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        for (Deck deck : decks) {
            sb.append(deck);
            sb.append("\n");
        }
        return sb.toString();
        // java 8
        /*
        return Arrays.stream(decks)
                .map(Deck::toString)
                .collect(Collectors.joining("\n"));
         */
    }

    public static void main(String[] args) {
        MultipleDeck md = new MultipleDeck(3);
        System.out.println(md);
    }


}
```

3. Изворен код од примери и задачи

https://github.com/finki-mk/NP/

Source Code ZIP