دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده مهندسی کامپیوتر

تمرین پنجم

آزمون نرم‌افزار

نگارش
محمدرضا اخگری زیری
محمدعلی کشاورز
علی نظری

استاد درس
دکتر معصومه طارمی‌راد

تابستان ۱۴۰۰

فهرست مطالب

صفحه

سوال اول

الف) عبارت منطقی‌ای که در این تابع تست می‌شود، عبارت زیر است:

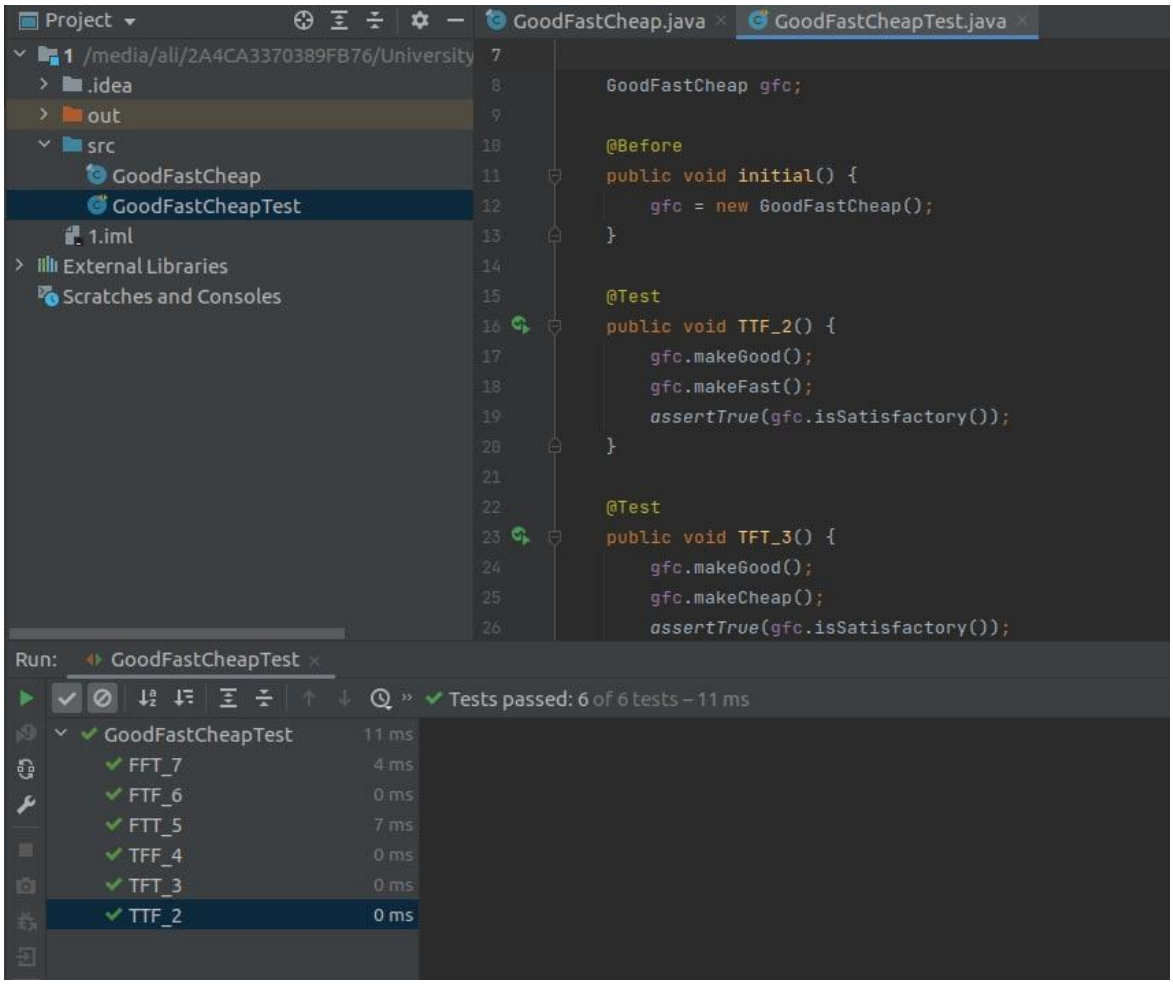
$$p = (f \wedge c) \vee (f \wedge g) \vee (c \wedge g)$$

برای این عبارت، جدول درستی را رسم کرده و حالاتی که RACC اتفاق می‌افتد را در جدول ۱ نمایش دادیم.

جدول ۱- نمایش جدول درستی و حالات RACC

	g	f	c	$p(g)$	$p(f)$	$p(c)$	p
1	1	1	1	0	0	0	1
2	1	1	0	1	1	0	1
3	1	0	1	1	0	1	1
4	1	0	0	0	1	1	0
5	0	1	1	0	1	1	1
6	0	1	0	1	0	1	0
7	0	0	1	1	1	0	0
8	0	0	0	0	0	0	0

با توجه به جدول بدست آمده، برای ردیف‌هایی که با رنگ مشخص شدند، آزمون نوشتیم که مجموعاً ۶ آزمون شد. نتایج اجرای این آزمون‌ها در شکل ۱ قابل مشاهده است.



شکل ۱- اجرای ۶ آزمون طراحی شده به همراه نتیجه

آزمون‌های طراحی شده در ادامه ذکر شده است:

```
import org.junit.*;

import static org.junit.Assert.*;


public class GoodFastCheapTest {

    GoodFastCheap gfc;

    @Before
    public void initial() {
        gfc = new GoodFastCheap();
    }

    @Test
    public void TTF_2() {
```

```
        gfc.makeGood();
        gfc.makeFast();
        assertTrue(gfc.isSatisfactory());
    }

    @Test
    public void TFT_3() {
        gfc.makeGood();
        gfc.makeCheap();
        assertTrue(gfc.isSatisfactory());
    }

    @Test
    public void TFF_4() {
        gfc.makeGood();
        assertFalse(gfc.isSatisfactory());
    }

    @Test
    public void FTT_5() {
        gfc.makeCheap();
        gfc.makeFast();
        assertTrue(gfc.isSatisfactory());
    }

    @Test
    public void FTF_6() {
        gfc.makeFast();
        assertFalse(gfc.isSatisfactory());
    }

    @Test
    public void FFT_7() {
        gfc.makeCheap();
        assertFalse(gfc.isSatisfactory());
    }
}
```

ب) با توجه به اینکه ما در قسمت قبل برای تمام حالات آزمون نوشتیم، از نظر منطقی آزمونی به مجموعه‌ی آزمون‌های ما افزوده نخواهد شد و تمامی آزمون‌های قبلی برای این حالت نیز قابل استفاده است.

سوال دوم

الف) پیش شرط به این صورت است که بعد از فراخوانی تابع next، نباید تابع remove صدا زده شود، زیرا این عمل باعث می‌شود تا طول شی، در هنگام پیمایش تغییر کند. اگر بخواهیم این عبارت را به صورت منطقی بنویسیم که بر روی آن بتوان ACC تعریف کرد، می‌توان حالت زیر را در نظر گرفت:

a: حالتی است که تابع next صدا زده شود.

b: حالتی است که تابع remove پس از next صدا زده نشود.

پس به صورت کلی می‌توان به شکل زیر تعریف کرد:

$$a \wedge b$$

ب) با توجه به اینکه iterator یک interface است، می‌توان به کلاس‌هایی که از Collection ارث‌بری کرده‌اند (مانند HashSet، HashMap و Arraylist) مراجعه کرد، ما برای این تمرین از پیاده‌سازی کلاس Arraylist استفاده کردیم.

```
import java.util.*;
import java.util.function.Consumer;

public class Itr implements Iterator<E> {
    int cursor;
    int lastRet = -1;
    int expectedModCount;

    Itr() {
        this.expectedModCount = ArrayList.this.modCount;
    }

    public boolean hasNext() {
        return this.cursor != ArrayList.this.size;
    }

    public E next() {
        this.checkForComodification();
        int i = this.cursor;
        if (i >= ArrayList.this.size) {
            throw new NoSuchElementException();
        } else {
            Object[] elementData = ArrayList.this.elementData;
            if (i >= elementData.length) {
```

```
                throw new ConcurrentModificationException();
            } else {
                this.cursor = i + 1;
                return elementData[this.lastRet = i];
            }
        }
    }

    public void remove() {
        if (this.lastRet < 0) {
            throw new IllegalStateException();
        } else {
            this.checkForComodification();

            try {
                ArrayList.this.remove(this.lastRet);
                this.cursor = this.lastRet;
                this.lastRet = -1;
                this.expectedModCount = ArrayList.this.modCount;
            } catch (IndexOutOfBoundsException var2) {
                throw new ConcurrentModificationException();
            }
        }
    }

    public void forEachRemaining(Consumer<? super E> action) {
        Objects.requireNonNull(action);
        int size = ArrayList.this.size;
        int i = this.cursor;
        if (i < size) {
            Object[] es = ArrayList.this.elementData;
            if (i >= es.length) {
                throw new ConcurrentModificationException();
            }

            while(i < size && ArrayList.this.modCount == this.expectedModCo
unt) {
                action.accept(ArrayList.elementAt(es, i));
                ++i;
            }

            this.cursor = i;
            this.lastRet = i - 1;
            this.checkForComodification();
        }
```

```
    }

    final void checkForComodification() {
        if (ArrayList.this.modCount != this.expectedModCount) {
            throw new ConcurrentModificationException();
        }
    }
}
```

ج) در ابتدا اقدام به رسم جدول درستی می‌کنیم، جدول ۲ نشانگر این موضوع است. تست‌های CACC به صورت جدول ۳ است.

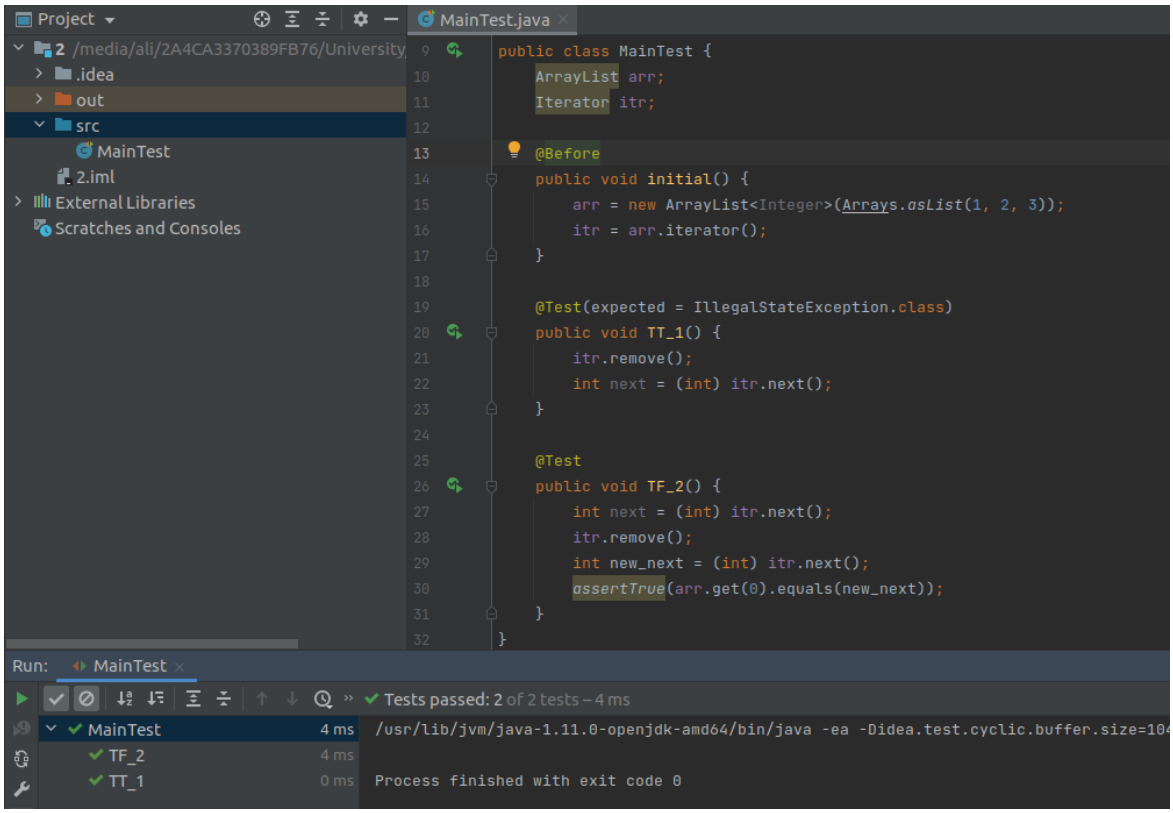
جدول ۲- جدول درستی

Row#	a	b	P	Pa	Pb
1	T	T	T	T	T
2	T				T
3		T		T	
4					

جدول ۳- مجموعه‌ی تست‌های CACC

Major Clause	Set of possible tests
a	(1,3)
b	(1,2)

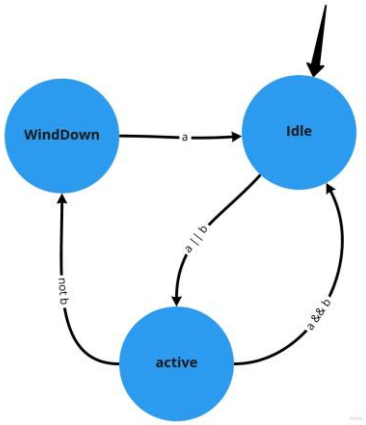
ردیف ۳ یا به عبارتی Fx که F نشان‌گر غلط بودن a است، به نظر قابل بیان شدن نیست. زیرا در صورت غلط بودن a (یعنی تابع next صدا زده نشده باشد)، نمی‌توان درباره‌ی جایگاه remove نسبت به next نظری داد. پس ما فقط برای حالات Tx آزمون می‌نویسیم. تست‌های نوشته شده در ۲ قابل مشاهده است.



شکل ۲- آزمون‌های نوشته شده

سوال سوم

الف) در شکل ۳ FSM خواسته شده رسم شده است:



شکل ۳- FSM

ب) به طور کلی باید برای پیدا کردن حالت‌هایی که باعث می‌شود تا به استیت دیگری منتقل نشود، باید نقیض یال‌ها را با هم and کنیم.

برای حالت idle:

$$(a \cup b)' \cap (a \cap b)' = (a' \cap b') \cap (a' \cup b') = (a' \cap b' \cap a') \cup (a' \cap b' \cap b') = a' \cap b'$$

برای حالت WindDown:

$$a'$$

برای حالت Active:

$$b \cap (a \cap b)' = b \cap (a' \cup b') = b \cap a'$$

به طور کلی جدول به صورت جدول ۴ می‌شود:

جدول ۴- نمایش تمامی حالت‌ها و انتقال‌ها

Current State	Edge	Next State
Idle	$a \wedge b$	Active
Idle	$\neg a \wedge \neg b$	Idle
WindDown	a	Idle
WindDown	$\neg a$	WindDown
Active	$\neg b$	WindDown
Active	$a \wedge b$	Idle
Active	$\neg a \wedge b$	Active

ج)

- برای حالت انتقال Active به WindDown: $\neg b$

$$(a, b) = (T, T), (F, T), (T, F), (F, F)$$

- برای حالت انتقال Active به Idle: $a \wedge b$

اگر a متغیر اصلی باشد، حالات:

$$(a, b) = (T, T), (F, T)$$

اگر b متغیر اصلی باشد، حالات:

$$(a, b) = (T, T), (T, F)$$

به طور کلی:

$$(a, b) = (T, T), (F, T), (T, F)$$

- برای حالت انتقال Active به Active: $\neg a \wedge b$

اگر a متغیر اصلی باشد، حالات:

$$(a, b) = (T, T), (F, T)$$

اگر b متغیر اصلی باشد، حالات:

$$(a, b) = (F, T), (F, F)$$

به طور کلی:

$$(a,b) = (T,T),(F,T),(F,F)$$

گزارش انجام تمرین به صورت گروهی:

برای انجام این آزمون دو جلســـه‌ی آنلاین برگذار کردیم. در این جلســـات اقدام به همفکری برای حل سوالات کردیم. در آخر تمامی مطالب تجمیع و به تایید افراد گروه رسید.