INFO233: Obligatorisk oppgave 2

Innleveringsfrist: 12. mars, kl. 14:00

Testing, lister, sortering, og funksjoner

På forelesning så har vi nå gått igjennom hvordan lister og iteratorer fungerer. Vi har også vært igjennom hvordan vi tester programvare. Dette gir oss en god anledning til å bevege oss ut av komfortsonen i objekt-orientert programmering og dyppe tåen i funksjonell programmering.

I denne obligatoriske oppgaven så skal dere først og fremst implementere en lenket liste. Vedlagt finner dere et grensesnitt dere skal implementere. Dere finner også et sett med tester som hører til noen av oppgavene. Det er meningen at implementasjonen deres skal bestå de testene for å få full pott på de oppgavene det gjelder. De er ikke de eneste testene dere skal bestå. Flere av oppgaven ber også om at dere skriver deres egne tester. Dere vil bli evaluert på nyttigheten og omfanget av av testene.

Dere kan ikke bruke `java.util.stream` API'et i implementasjonen av `IList`. Det er dog lov til å bruke det i utformingen av tester.

Det er ikke nødvendig å gjennomføre deloppgavene i den samme rekkefølge de er oppført, men denne gangen så er det lagt opp på en slik måte at det også er mulig å få litt erfaring med test-drevet utvikling (TDD).

Legg ved det du blir bedt om å forklare, gjøre rede for, eller analysere i en egen fil. Les gjennom alle oppgavene før du begynner.

For å bestå oppgaven må du oppnå minst 80/100 poeng.

Om du ikke skulle bestå så tilbyr vi også egenretting. Du kvalifiserer til egenretting ved å oppnå minst 40/100 poeng.

Deloppgaver

Hver deloppgave beskriver hva som skal gjøres og hvor mange poeng som kan oppnås. En delvis implementasjon gir ikke full pott, men kan gi en redusert poengmengde hvis implementasjonen er i riktig retning.

I oppgaven du blir bedt om å skrive tester så blir du vurdert utifra hva du velger å teste. Dette er uavhengig av om koden du skriver består testene.

Deloppgave 1.1) – 2 poeng

Skriv tester for hva som skjer hvis listen er tom og du kaller metodene `first`, `rest`, `add`, `put`, og `remove`. Kommenter testene med hva du tester for.

Deloppgave 1.2) – 2 poeng

Skriv tester for hva som skjer hvis listen inneholder kun 1 element og du kaller metodene `first`, `rest`, `add`, `put`, og `remove`. Kommenter testene med hva du tester for.

Deloppgave 1.3) – 2 poeng

Skriv tester for hva som skjer hvis listen inneholder flere enn 1 element og du kaller metodene `first`, `rest`, `add`, `put`, og `remove`. Kommenter testene med hva du tester for.

Deloppgave 1.4) – 2 poeng

Lag en 2 konstruktører for initialiseringen av en ny liste. En konstruktør som tar imot et nytt element som blir det første elementet i listen og en konstruktør som tar et element som blir det første elementet og en liste (som potensielt kan være `null`).

Deloppgave 1.5) – 2 poeng

Implementer metodene `first`, `rest`, `add`, `put`, og `remove`.

Deloppgave 2.1) – 5 poeng

Skriv tester for `remove(Object)` i ulike scenariorer. Kommenter testene med hva du tester for.

Deloppgave 2.2) – 3 poeng

Implementer metoden `remove(Object)`.

Deloppgave 3.1) – 4 poeng

Skriv tester for metodene `contains(Object)`, `isEmpty`, og `size` i ulike scenariorer. Kommenter testene med hva du tester for.

Deloppgave 3.2) – 4 poeng

Implementer metodene `contains(Object)`, `isEmpty`, og `size`. Husk at det ikke er lov til å bruke `java.util.stream`.

Deloppgave 4.1) – 4 poeng

Skriv tester for metodene `append` og `prepend` i ulike scenariorer. Kommenter testene med hva du tester for.

Deloppgave 4.2) – 4 poeng

Implementer metodene `append` og `prepend`.

Deloppgave 5.1) – 4 poeng

Skriv tester for `concat` i ulike scenariorer. Kommenter testene med hva du tester for.

Deloppgave 5.2) – 5 poeng

Implementer metoden `concat`.

Deloppgave 6) – 6 poeng

Gjør en kjøretidsanalyse av `append`, `prepend`, og `concat`. Hvis du ikke har implementert en eller flere av metodene så skriv ned pseudokoden for metodene og gjør en kjøretidsanalyse av den i stedet. Gjør rede for svaret ditt.

Deloppgave 7.1) – 5 poeng

Liste-grensesnittet spesifiserer at det utvider (`extends`) `Iterable`. Dette betyr at hvis du skal implementere listen vår små du også implementer `Iterable`-grensesnittet. Heldigvis så kan vi være smart i dette tilfelle og vi trenger bare å implementere `iterator`. Legg merke til at i dokumentasjonen for `Iterable` så har resten av metodene i `Iterable` en normalimplementasjon (default). `iterator`-metoden returnerer en `Iterator<E>`. Legg merke til at en `Iterator` kun krever at du implementerer `hasNext` og `next`. Det er ikke nødvendig å implementere `remove` (se ekstraoppgave 2).

Skriv tester for metoden `iterator` i ulike scenariorer. Kommenter testene med hva du tester for.

Deloppgave 7.2) – 3 poeng

Les deloppgave 7.1. Implementer metoden `iterator`.

Deloppgave 8.1) – 2 poeng

Skriv tester for metoden `sort(Comperator<E>)` i ulike scenariorer. Her prøver vi for første gang å ikke bare kalle metoder, men også å sende komputasjonelle enheter som verdier og bruker de for å påvirke programmet vårt. Kommenter testene med hva du tester for.

Deloppgave 8.2) – 6 poeng 

Implementer metoden `sort(Comperator<E>)`. Sjekk at testene for `sort` som ligger i vedlegget også fungerer. I denne oppgaven er det meningen at du skal implementere metoden uten bruk av eksterne API'er. Sorteringsfunksjoner fra andre deler av java biblioteket blir ikke godkjent.

Deloppgave 9.1) – 4 poeng

Skriv tester for metoden `filter(Predicate<? super E>)` i ulike scenariorer. Et predikat er en spesiell funksjon som alltid returnerer `true` eller `false` for alle elementer i domenet sitt. Kommenter testene med hva du tester for.

Deloppgave 9.2) – 5 poeng

Implementer metoden `filter(Predicate<? super E>)`. Sjekk at testene for `filter` som ligger i vedlegget også fungerer. I denne oppgaven er det meningen at du skal implementere metoden uten bruk av eksterne API'er utover de som

trengs for å svare på grensesnittet. Bruk av f.eks. `java.util.stream` blir ikke godkjent.

Deloppgave 10.1) – 4 poeng

Skriv tester for metoden `map(Function<? super E, ? extends U>)` i ulike scenariorer. `map` tar en funksjon og returnerer en ny liste med nye elementer. En funksjon tar et element fra domenet sitt og returnerer et nytt element som ikke (nødvendigvis) er en del av domenet. Dette er grunnen til at vi ikke kan endre på listen vi manipulerer, men må opprette en ny liste. Kommenter testene med hva du tester for.

Deloppgave 10.2) – 5 poeng

Implementer metoden `map(Function<? super E, ? extends U>)`. Sjekk at testene for `map` som ligger i vedlegget også fungerer. I denne oppgaven er det meningen at du skal implementere metoden uten bruk av eksterne API'er utover de som trengs for å svare på grensesnittet. Bruk av f.eks. `java.util.stream` blir ikke godkjent.

Deloppgave 11.1) – 4 poeng

Skriv tester for metoden `reduce(T, BiFunction<T, ? super E, T>)`. `reduce` tar et grunnelement som definerer det første elementet i reduksjonen og en funksjon som slår sammen elementene i listen med grunnelementet. Kommenter testene med hva du tester for.

Deloppgave 11.2) – 5 poeng

Implementer metoden `reduce(T, BiFunction<T, ? super E, T>)` slik den er beskrevet i grensesnittet. I denne oppgaven er det meningen at du skal implementere metoden uten bruk av eksterne API'er utover de som trengs for å svare på grensesnittet. Bruk av f.eks. `java.util.stream` blir ikke godkjent.

Deloppgave 12) – 8 poeng

Gjør en kjøretidsanalyse av `filter`, `map`, og `reduce`. Gjør rede for svaret ditt. Hvis du ikke har implementer en eller flere av metodene så skriv ned pseudokoden og gjør en kjøretidsanalyse av den i stedet.

Ekstraoppgaver

Ekstraoppgave 1) – 4 poeng

Få `sort`-metoden til å bestå `ex1FastSort` testen.

Ekstraoppgave 2) – 4 poeng

Implementer `remove` for iteratoren `iterator`-metoden returnerer.

Ekstraoppgave 3) – 6 poeng

Hvis du gjennomførte oppgaven ved hjelp av test-drevet utvikling, skriv en kort rapport på ca 300 ord om erfaringene dine.

Grensesnitt for lister

```
import java.util.Comparator;
import java.util.NoSuchElementException;
import java.util.function.BiFunction;
import java.util.function.BinaryOperator;
import java.util.function.Function;
import java.util.function.Predicate;

public interface IList<E> extends Iterable<E> {
    /**
     * ,* Gir det første elementet i listen.
     * ,*
     * ,* @return Det første elementet i listen.
     * ,* @throws NoSuchElementException Hvis listen er tom.
     * ,
     */
    E first() throws NoSuchElementException;

    /**
     * ,* Returnerer alle elementene i listen bortsett fra det
     * ,* første.
     * ,*
     * ,* @return Resten av listen.
     * ,
     */
    IList<E> rest();

    /**
     * ,* Legger til et element på slutten av listen.
     * ,
     */
    boolean add(E elem);

    /**
     * ,* Legger til et element på begynnelsen av listen.
     * ,
     */
    boolean put(E elem);
```

```
/**
 * ,* Fjerner det første elementet i listen.
 * ,*
 * ,* @return Det første elementet i listen.
 * ,* @throws NoSuchElementException Hvis listen er tom.
 * ,
 */
E remove() throws NoSuchElementException;

/**
 * ,* Fjerner det angitte objektet fra listen.
 * ,*
 * ,* @param o Objektet som skal fjernes.
 * ,* @return true hvis et element ble fjernet, false
 * ,* ellers.
 * ,
 */
boolean remove(Object o);

/**
 * ,* Sjekker om et element er i listen.
 * ,*
 * ,* @param o objektet vi sjekker om er i listen.
 * ,* @return true hvis objektet er i listen, false ellers.
 * ,
 */
boolean contains(Object o);

/**
 * ,* Sjekker om listen er tom.
 * ,*
 * ,* @return true hvis listen er tom, false ellers.
 * ,
 */
boolean isEmpty();

/**
 * ,* Legger til alle elementene i den angitte listen på
 * ,* slutten av listen.
 * ,*
 * ,* @param listen som blir lagt til.
```

```
     * ,
     */
    void append(IList<? super E> list);

    /**
     * ,* Legger til alle elementene i den angitte listen på
     * ,* begynnelsen av listen.
     * ,*
     * ,* @param list listen som blir lagt til
     * ,
     */
    void prepend(IList<? super E> list);

    /**
     * ,* Slår sammen flere lister
     * ,*
     * ,* @param lists listene som skal slås sammen
     * ,* @return Ny liste med alle elementene fra listene som
     * ,* skal slås sammen.
     * ,
     */
    IList<E> concat(IList<? super E>... lists);

    /**
     * ,* Sorterer listen ved hjelp av en
     * ,* sammenligningsfunksjon
     * ,* @param c sammenligningsfunksjon som angir rekkefølgen
     * ,* til elementene i listen
     * ,
     */
    void sort(Comparator<? super E> c);

    /**
     * ,* Fjerner elementer fra listen som svarer til et
     * ,* predikat.
     * ,* @param filter predikat som beskriver hvilken
     * ,* elementer som skal fjernes.
     * ,
     */
    void filter(Predicate<? super E> p);
```

```
    /**
     * ,* Kjører en funksjon over hvert element i listen
     * ,*
     * ,* @param f en funksjon fra typen til elementene i
     * ,* listen til en annen type
     * ,* @return En liste over elementene som funksjonen
     * ,* returnerer
     * ,
     */
    <U> IList<U> map(Function<? super E, ? extends U> f);

    /**
     * ,* Slår sammen alle elementene i listen ved hjelp av en
     * ,* kombinasjonsfunksjon.
     * ,*
     * ,* @param t Det første elementet i sammenslåingen
     * ,* @param f Funksjonen som slår sammen elementene
     * ,* @return Den akkumulerte verdien av sammenslåingene
     * ,
     */
    <T> T reduce(T t, BiFunction<T, ? super E, T> f);

    /**
     * ,* Gir størrelsen på listen
     * ,*
     * ,* @return Størrelsen på listen
     * ,
     */
    int size();

    /**
     * ,* Fjerner alle elementene i listen.
     * ,
     */
    void clear();
}
```

Tester

```
import org.junit.jupiter.api.Test;

import java.time.Duration;
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
import java.util.Random;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTimeout;
import static org.junit.jupiter.api.Assertions.fail;

class ListTest {
    @Test
    void oppg8_sortIntegers() {
        // Se oppgave 8
        IList<Integer> list = new LinkedList<>();
        List<Integer> values = Arrays.asList(3, 8, 4, 7, 10, 6,
                                             1, 2, 9, 5);

        for (Integer value : values) {
            list.add(value);
        }
        list.sort(Comparator.comparingInt(x -> x));


        int n = list.remove();
        while (list.size() > 0) {
            int m = list.remove();
            if (n > m) {
                fail("Integer list is not sorted.");
            }
            n = m;
        }

    }
    @Test
    void oppg8_sortStrings() {
        // Se oppgave 8
```

```
    IList<String> list = new LinkedList<>();
    List<String> values = Arrays.asList(
            "g", "f", "a", "c", "b", "d", "e", "i", "j", "h"
    );
    for (String value : values) {
        list.add(value);
    }

    list.sort(Comparator.naturalOrder());

    String n = list.remove();
    while (list.size() > 0) {
        String m = list.remove();
        if (n.compareTo(m) > 0) {
            fail("String list is not sorted.");
        }

    }
}

@Test
void oppg9_filter() {
    // Se oppgave 9
    List<Integer> values = Arrays.asList(1, 2, 3, 4, 5, 6, 7,
                                         8, 9, 10);

    IList<Integer> list = new LinkedList<>();
    for (Integer value : values) {
        list.add(value);
    }

    list.filter(n -> n % 2 == 1);

    int n = list.remove();
    while(list.size() > 0) {
        if (n % 2 == 1) {
            fail("List contains filtered out elements.");
        }
        n = list.remove();

    }
```

```
}

@Test
void oppg10_map() {
    // Se oppgave 10
    List<String> values = Arrays.asList("1", "2", "3",
                                        "4", "5");

    IList<String> list = new LinkedList<>();
    for (String value : values) {
        list.add(value);
    }

    IList<Integer> result = list.map(Integer::parseInt);

    List<Integer> target = Arrays.asList(1, 2, 3, 4, 5);

    int n = result.remove();
    for (Integer t : target) {
        if (n != t) {
            fail("Result of map gives the wrong value.");
        }
        n = result.remove();
    }
}

@Test
void oppg11_reduceInts() {
    // Se oppgave 11
    List<Integer> values = Arrays.asList(1, 2, 3, 4, 5);

    IList<Integer> list = new LinkedList<>();
    for (Integer value : values) {
        list.add(value);
    }

    int result = list.reduce(0, Integer::sum);

    assertEquals(result, 5*((1 + 5)/2));
}
```

```
    @Test
    void oppg11_reduceStrings() {
        List<String> values = Arrays.asList("e", "s", "t");
        IList<String> list = new LinkedList<>();
        for (String s : values) {
            list.add(s);
        }

        String result = list.reduce("t", (acc, s) -> acc + s);

        assertEquals(result, "test");
    }

    @Test
    void ex1_FastSort() {
        // Se ekstraoppgave 1
        Random r = new Random();
        IList<Integer> list = new LinkedList<>();
        for (int n = 0; n < 1000000; n++) {
            list.add(r.nextInt());
        }

        assertTimeout(Duration.ofSeconds(2), () -> {
            list.sort(Integer::compare);
        });

        int n = list.remove();
        for(int m = list.remove(); !list.isEmpty(); n = m) {
            if (n > m) {
                fail("List is not sorted");
            }
        }


    }
}
```

Errata

Versjon 2

Fikset enda en feil i test til oppgave 10

Testen fjernet en for mange elementer fra listen fordi vi kaller remove før løkken og på slutten av løkken. Hvis vi bare kaller det der vi sammenligner så skal det være riktig. Takk til Sindre Sperrud Kjær som rapporterte feilen og foreslo endringen.

Versjon 1

Fikset feil i test til oppgave 10

Testen sjekket om det første elementet var alle elementene i `target`, vi må sjekke at hvert element er det korresponderende elementet. Takk til Vetle Håland Bergstad som rapporterte feilen.