Laboration 1 – ADT och generics

Avsikten med laborationen är att du ska träna på att skriva och använda några abstrakta datatyper . Skapa paketet ***laboration1*** innan du börjar lösa uppgifterna.

Uppgift 1 – DString

I uppgiften används filerna *DString.java* och *DynamicString.java*. Placera dem i paketet laboration1 (rödmarkeringen i DString försvinner snart).

Det går inte att ändra innehållet i ett String-objekt i java. I Uppgift 1 ska du komplettera klassen ***DString***, vilken representerar en sekvens av tecken. I objekt av typen *DString* ska det gå bra att lägga till och ta bort tecken.

DString ska implementera interfacet *DynamicString* vilket är givet:

```
public interface DynamicString {
    public int length();
    public char charAt(int index);
    public String toString();
    public void append(char chr);
    public void append(DString str);
    public void delete(int start, int end);
    public void delete(int index);
    public String substring(int start, int end);
    public String substring(int start);
    public int indexOf(char chr);
}
```

Delar av klassen är också given. En char-array (*text*) används för att för att lagra tecken. Instansvariabeln *length* håller reda på antalet tecken som strängen innehåller.

Det är *fyra konstruktorer* i klassen:

- En utan parametrer – en tom sträng med utrymme för 10 tecken skapas
- En med parametern *size* - en tom sträng med utrymme för *size* tecken skapas. Om size är mindre än 1 så skapas utrymme för 10 tecken.
- En med ett *DString*-objekt som argument. En sträng med samma teckensekvens som argumentet skapas. Arrayen *text* rymmer exakt teckensekvensen.
- En med ett String-objekt som argument. En sträng med samma teckensekvens som argumentet skapas. Arrayen *text* rymmer exakt teckensekvensen.

Den private metoden *grow* vilken dubblar antalet tecken som kan lagras i objektet.

Slutligen metoden append(char chr) vilken lägger till ett tecken i DString-objektet.

Din uppgift är att implementera resterande uppgifter i interfacet. Börja med att lägga till ett skal för vardera metod så försvinner rödmarkeringen i klassdeklarationen:
I menyraden välj ”Code” och sedan ”Implements Methods”. Man kan även använda kortkommandon, PC: ”Ctrl + I”, Mac: ”^I”, eller föra musen över klassdefinitionen och välja ”implement methodes” i fönstret som kommer upp. Välj de berörda metoderna.

Implementera och testa sedan en metod i taget.

Uppgift 2 – Generisk kö

Klassen ***ObjectQueue*** är given att studera. Vidare är klassen ***QueueException*** given. Din uppgift är att skriva interfacet ***Queue<T>*** vilket ska innehålla kö-metoderna i föreläsningen (add, remove, element, isEmpty, size).

Sedan ska du skriva klassen ***ArrayQueue<T>*** vilken ska implementera *Queue<T>*.
Ett *QueueException* ska kastas om:

- Användaren försöker lägga till ett element (add) i en full stack
- Användaren försöker ta bort (remove) / få referens till (element) ett element i en tom kö

Uppgift 3 – DString

Ändra interfacet *DynamicString* i Uppgift 1 till:

```
public interface DynamicString {
    public int length();
    public char charAt(int index);
    public String toString();
    public DynamicString append(char chr);
    public DynamicString append(DString str);
    public DynamicString delete(int start, int end);
    public DynamicString delete(int index);
    public String substring(int start, int end);
    public String substring(int start);
    public int indexOf(char chr);
}
```

Ändra sedan i metoderna *append* respektive *delete* i klassen *DString* så att *method chaining* går att använda. Det ska t.ex. gå bra att skriva:

```
DString str1 = new DString("Laboration");
str1.append(' ').append('1').append(new DString("\nUppgift3"));
System.out.println(str1);
str1.delete(2).delete(4,6).delete(8);
System.out.println(str1);
```

Köresultat

```
Laboration 1
Uppgift3
Laorion
Uppgift3
```

Uppgift 4 - DString

Lägg till ytterligare metoder i klassen *DString*, t.ex.

- **public boolean equals(Object obj)**
  Metoden ska returnera *false* om *obj* har fel typ eller om *obj* inte innehåller identiska tecken som aktuellt objekt (*this*).
- **public DString reverse()** // lägg till i interfacet
  vänder på teckensekvensen

Uppgift 5 - DString

Implementera interfacet ***Comparable<DString>*** i klassen *DString*. Om du anger typ så slipper du typkonvertera i *compareTo*-metoden:

```
public int compareTo(DString str) {…}
```

Om du däremot implementerar ***Comparable*** utan typ så måste du typkonvertera i compareTo-metoden:

```
public int compareTo(Object obj) {
    DString str = (DString)obj;
    :
}
```

Endast om samtliga tecken i två stränger är samma ska 0 returneras. Annars avgör den första skillnaden i strängarna om negativt värde eller positivt värde ska returneras.

Du kan jämföra tecken med ==, < och >. Detta innebär att tecknen å, ä, ö, Å, Ä, Ö ordnas konstigt men det spelar ingen roll.

Om en sträng innehåller samma tecken som en längre sträng börjar med ska den kortare strängen ordnas först. Nedan har du testkod och körresultat.

```
ArrayList<DString> strings = new ArrayList<DString>();
DString str1 = new DString("Laboration");
DString str2 = new DString("Lab");
DString str3 = new DString(str2.reverse());

strings.add(str1);
strings.add(str2);
strings.add(str3);

Collections.sort(strings);
System.out.println(strings.toString());

System.out.println(str1.equals(129));
System.out.println(str1.equals("Laboration"));
System.out.println(str1.equals(str1));
```

Körresultat:

```
[baL, Lab, Laboration]
False
True
True
```

Lösningar

Uppgift 1

```
public class DString implements DynamicString {
    private char[] text;
    private int length = 0;

    public DString() {
        this(10);
    }

    public DString(int size) {
        size = (size<=0) ? 10 : size;
        text = new char[size];
        length = 0;
    }

    public DString(String str) {
        text = str.toCharArray();
        length = text.length;
    }

    public DString(DString str) {
        text = new char[str.length()];
        for(int i=0; i<str.length(); i++) {
            text[i] = str.charAt(i);
        }
        length = text.length;
    }

    private void grow() {
        char[] newArr = new char[length*2];
        System.arraycopy(text, 0, newArr, 0, text.length);
        text = newArr;
    }

    public void append(char chr) {
        if(length==text.length) {
            grow();
        }
        text[length++] = chr;
    }

    public void append(DString str) {
        for(int i=0; i<str.length(); i++) {
            append(str.charAt(i));
        }
    }

    public int length() {
        return length;
    }

    public char charAt(int index) {
        return text[index];
    }

    public String toString() {
        return new String(text, 0, length);
    }

    public void delete(int start, int end) {
        System.arraycopy(text, end, text, start, length-end);
        length -= (end-start);
    }

    public void delete(int index) {
        delete(index,index+1);
    }
```

```
    public int indexOf(char chr) {
        for(int i=0; i<length; i++) {
            if(chr==text[i])
                return i;
        }
        return -1;
    }

    public String substring(int start, int end) {
        return new String(text,start,end-start);
    }

    public String substring(int index) {
        return substring(index, length-index);
    }
}
```

Uppgift 2

```
package laboration1;

public interface Queue<T> {

    /**
     * Inserts the specified element into this queue.
     * @param data the object to add
     * @throws QueueException if the element cannot be added at this
     *         time due to capacity restrictions
     */
    public void add(T data);

    /**
     * Retrieves and removes the head of this queue.
     * @return the head of this queue
     * @throws QueueException if this queue is empty
     */
    public T remove();

    /**
     * Retrieves, but does not remove, the head of this queue.
     * @return the head of this queue
     * @throws QueueException if this queue is empty
     */
    public T element();


    /**
     * Returns true if this stack contains no elements.
     * @return true if this stack contains no elements
     */
    public boolean isEmpty();

    /**
     * Returns the number of elements in this stack.
     * @return the number of elements in this stack
     */
    public int size();
}


public class ArrayQueue<T> implements Queue<T> {
    private T[] elements;
    private int size = 0;

    public ArrayQueue(int capacity) {
        elements = (T[])new Object[ capacity ];
    }
```

```
    public void add( T elem ) {
        if( size<elements.length ) {
            throw new QueueException("enqueue: Queue is full");
        }
        elements[ size++ ] = elem;
    }

    public T remove() {
        if(size==0) {
            throw new QueueException("dequeue: Queue is empty");
        }
        T value = elements[ 0 ];
        size--;
        for(int i=1; i<size; i++) {
            elements[i-1] = elements[i];
        }
        elements[size] = null;
        return value;
    }

    public T element() {
        if( size==0 ) {
            throw new QueueException("peek: Queue is empty");
        }
        return elements[ 0 ];
    }

    public boolean isEmpty() {
        return (size==0);
    }

    public int size() {
        return size;
    }
}
```

Uppgift 3

```
public DynamicString append(char chr) {
    if(length==text.length) {
        grow();
    }
    text[length++] = chr;
    return this;
}

public DynamicString append(DString str) {
    for(int i=0; i<str.length(); i++) {
        append(str.charAt(i));
    }
    return this;
}

public DynamicString delete(int start, int end) {
    System.arraycopy(text, end, text, start, length-end);
    length -= (end-start);
    return this;
}

public DynamicString delete(int index) {
    delete(index,index+1);
    return this;
}
```

Uppgift 4 och 5

```
public class DString implements DynamicString, Comparable<DString> {
    :
    public DString reverse() {
        char tmp;
        for(int i=0; i<length/2; i++) {
            tmp = text[i];
            text[i] = text[length-1-i];
            text[length-1-i] = tmp;
        }
        return this;
    }

    public boolean equals(Object o) {
        boolean res = (this==o);
        if(!res && (o instanceof DString)) {
            DString str = (DString)o;
            res = (length==str.length);
            for(int i=0; i<length && res==true; i++) {
                res = (text[i]==str.text[i]);
            }
        }
        return res;
    }

    public int compareTo(DString str) {
        int res = 0;
        int len = Math.min(length, str.length);
        for(int i=0; i<len && res==0; i++) {
            if(text[i]<str.text[i])  // if(text[i]<str.charAt(i))
                res = -1;
            else if(text[i]>str.text[i])  // if(text[i]>str.charAt(i))
                res = 1;
        }
        if(res==0) {
            res = length-str.length;
        }
        return res;
    }
    :
}
```