4CCS1DST – Data Structures

Lecture 2:

Solutions to exercises

Exercise 1

```
class TestProgression2 {
    public static void main(String[] args) {
        Progression prog;

        prog = new ArithProgression(5);
        prog.printProgression(5);
        prog.printProgression(7);

        prog = new GeomProgression(2);
        prog.printProgression(5);
        prog.printProgression(7);

        prog = new FibonacciProgression(3);
        prog.printProgression(5);
        prog.printProgression(7);
    }
}
```

```
>java TestProgression2
0 5 10 15 20
0 5 10 15 20 25 30
1 2 4 8 16
1 2 4 8 16 32 64
0 3 3 6 9
0 6 6 12 18 30 48
```

?

Explain why FibonacciProgression behaves differently than other subclasses.

Modify this class to achieve the expected behaviour.

Exercise 1 (cont.)

Answer:

Using the classes Progression and FibonacciProgression given in the lecture, this code:

```
prog = new FibonacciProgression(3);
prog.printProgression(5);
prog.printProgression(7);
```

prints:

```
0 3 3 6 9
0 6 6 12 18 30 48
```

but we would expect:

```
0 3 3 6 9
0 3 3 6 9 15 24
```

The resetting of the sequence to its first value in the beginning of the printProgression method is done by calling the method firstValue(). The problem is that class FibonacciProgression inherits this method from class Progression. Method firstValue(), which only resets "cur" to "first", is not sufficient for class FibonacciProgression. We also have to reset "prev" to its initial value. To fix this, add to class FibonacciProgression a field "second" to store the second element of the sequence, update the constructor to initialize that filed, and add an appropriate method firstValue().

Exercise 1 (cont.)

```
class FibonacciProgression3 extends Progression {
    // inherits fields:  first and cur and method printProgression(int)
    protected long  prev;    // the previous value
    protected long  second; // the second element of the sequence
    FibonacciProgression3(long  s) { // the second value given
        second = s;
        prev = second – first;                  // default (0) first value
    }
    // FibonacciProgression() and nextValue() as in class FibonacciProgression
    protected long firstValue() {
        cur = first;   prev = second - first;
        return cur;
    }
     public static void main(String[] args) {
        Progression prog = new FibonacciProgression2(3);
        prog.printProgression(5);       // prints "0 3 3 6 9"
        prog.printProgression(7);       // prints "0 3 3 6 9 15 24"
    }
}
```

Exercise 2

- In class SLinkedList<E>, show Java code for methods:

```
// return the first element, but don't remove it from the list
public  E  elementAtHead() { ... }

public  void  insertAtHead( E  element)  { … }
public  void  insertAtTail(E element) { ... }
public  E  removeAtHead() { ... }
```


Exercise 2 (cont.)

```
public E elementAtHead() {
    if ( head != null ) {
        return head.getElement();
    }
    else { return null; }
}

public E removeAtHead() {
    if ( head != null ) {
        E elem = head.getElement();
        head = head.getNext();
        size--;
        if ( head == null ) {  tail = null;  }
        return elem;
    }
    else {  return null;  }
}
```


Exercise 2 (cont.)

Answer:

```
public void insertAtHead(E element) {
    head = new Node<E>(element, head);
    size++;
    if ( size == 1 ) {
    tail = head;
    }
}

public void insertAtTail(E element) {
    Node<E> newNode = new Node<E>(element, null);
    if ( head != null ) {
        tail.setNext(newNode);
    }
    else {
        head = newNode;
    }
    tail = newNode;
    size++;
}
```


Exercise 3

Give code for method "contains" in this class:

```
public class SLinkedListExtended<E> extends SLinkedList<E> {
    // returns true if and only if, "element" is in the list
    public boolean contains(E element) { ... }

    public static void main(String[] args) {
    SLinkedListExtended<Integer> list =
                    new SLinkedListExtended<Integer>();
    list.insertAtHead(2);   list.insertAtHead(4);  list.insertAtHead(6);
    System.out.println( "the list contains 4: " + list.contains(4));
        // prints: "the list contains 4: true "
}
```

Exercise 3 (cont.)

```
// returns true if and only if, "element" is in the list
public boolean contains(E element) {
    Node<E> x = head;    // the current node of the list
    while (  (x != null)  &&
             (   // continue, if the current element isn't null and different from "element"
                 (  ( x.getElement() != null ) && !(x.getElement().equals(element))  )
                 ||
                  // or if the current element is null but "element" isn't null
                 (  ( x.getElement() == null )  &&  (element !=null) )
              )
          )
    {  // move to the next node of the list
       x = x.getNext();
    }

    if ( x != null ) {  // the "while" loop ended because we have found "element" in the list
          return true;
    }
    else {  return false;  }
}
```