Department of Informatics

Professor: R. Ingold
Teaching Assistants: M. Alberti, L. Vögtlin
Assistant: R. Geissbühler

S2019 Object Oriented Programming

Assignment 10

1 Theory

This exercise is taken from last year final exam. The estimated time for solving it was 20 minutes. Try to solve it first in a "exam-like" setting — no internet, no cheating — and only after fix it (if necessary) by using all resources at your disposal.

Fr Une boîte, de la classe `Box<F, S>`, peut stocker deux objets dans les champs `first` et `second`, respectivement de types `F` et `S`. Cette classe est définie ci-dessous.

De Eine Box vom generischen Typen `Box<F, S>` beherbergt zwei Objekte `first` und `second` der Typen `F` beziehungsweise `S`. Die Klasse ist wie folgt definiert:

```java
class Box<F, S> {
    F first;
    S second;
}
```

Fr Pour ce problème, vous devez définir les signatures de méthodes en respectant les spécifications indiquées ci-dessous. Remplacez chaque groupe de ??? par le type le plus générique possible. En aucune circonstance une exception de type `ClassCastException` ne doit pouvoir être levée, et toutes les vérifications de types doivent être faites au moment de la compilation.

De In dieser Aufgabe müssen Sie Methodensignaturen entsprechend der Spezifikationen erstellen. Gestalten Sie die Signaturen der Methoden, indem Sie alle ??? mit dem allgemeinsten generischen Typen ersetzen. Achten Sie darauf, dass unter keinen Umständen eine `ClassCastException` geworfen wird und dass alle Typenvalidierungschecks während der Kompilierung gemacht werden können.

Fr (a) (1 point) La méthode `void moveFirstTo(???  box)` transfère la valeur du champ `first` de l'objet courant vers le champ `first` de l'objet passé en paramètre.

De *Die Methode `void moveFirstTo(???  box)` transferiert das erste Element der gegenwärtigen Instanz in die erste Position von box.*

```java
void moveFirstTo( ........................................................ box);
```

Fr (b) (1 point) La méthode `void moveItemsFrom(???  box)` transfère les deux champs de l'objet passé en paramètre vers les champs correspondants de l'objet courant.

De *Die Methode `void moveItemsFrom(???  box)` transferiert beide Elemente des übergebenen Objektes box in die vorgesehenen Positionen der gegenwärtigen Instanz.*

```
void moveItemsFrom( ......................................................... box);
```

Fr (c) (1 point) La méthode `void swapItemsWith(???  box)` échange les valeurs des champs de l'objet courant avec celles des champs correspondants de l'objet passé en paramètre.

De *Die Methode `void swapItemsWith(???  box)` tauscht die Elemente des Parameters box mit denjenigen der gegenwärtigen Instanz aus.*

```
void swapItemsWith( ......................................................... box);
```

Fr (d) (2 points) La méthode `void moveFirstToSecond(???  box)` qui déplace la valeur du premier champ dans le second de la *même* instance de `Box`. Bien qu'il s'agisse d'une manipulation des données internes d'un seul objet, un paramètre est nécessaire. Expliquez pourquoi et quelles sont les conséquences pour la signature de la méthode.

De *Die Methode `void moveFirstToSecond(???  box)` verschiebt das erste Element auf die Position des zweiten. Auch wenn es sich dabei um eine Operation handelt, die nur ein Objekt betrifft, ist es nötig einen Parameter zu übergeben. Erklären Sie, wieso dies nötig ist und was die Konsequenzen für die Methodensignatur ist.*

....

....

....

....

....

```
...................... void moveFirstToSecond( ........................ box);
```

Fr (e) (2 points) La méthode `swapItems(...)` échange les valeurs des deux champs d'une instance de `Box`. Déterminez si un paramètre est nécessaire ou non et expliquez pourquoi.

De *Die Methode `void swapItems(...)` tauscht das erste mit dem zweiten Element in der gleichen Box. Entscheiden Sie, ob ein Parameter benötigt wird oder nicht, und erklären Sie wieso dies der Fall ist.*

```
...................... void swapItems( .................................. box);
```

2 Implementation

For this exercise you are required to submit two files: Box.java and Main.java, implemented as described below.

(a) Define a class `Box` implemented with everything described in the Theory exercise above. This includes all the methods described with their relative implementations.

(b) Implement a `Main` class with a `main()` method which creates some `Box` and verify that all the functionalities implements in `Box` works correctly, i.e. verify that with the correct setting everything works as expected and that compilation errors are raised when not.

3 Debugging

In this exercise we want to practice the use of the debugger and familiarize with the error messages of the compiler. Refer to the file `Main.java` attached to this series for this exercise.

(a) Run the `Main` file. Notice that despite it compiles, it does not work. Report the *exact* error message. Explain what is happening and why. Be as precise as possible.

(b) Use the debugger to inspect the state of the lists when the error is risen. Provide a screen-shot of your IDE which shows the content of the lists (similarly to the last assignment).

(c) Modify the code such that it works. Only a very small change is necessary!

For reference (you don't have to rewrite them yourself, we provide the java files attached to the series):

```java
package core;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;

public class Main {
    public static void main(String[] args) {

        ArrayList<Integer> intList = new ArrayList<>();
        for (int i = 4; i > 0; i--) {
            intList.add(i);
        }

        System.out.println("Integer list before");
        for(Integer i : intList){
            System.out.println("i = " + i);
        }

        shuffle(intList);
        Collections.sort(intList);

        System.out.println("Integer list sorted");
        for(Integer i : intList){
            System.out.println("i = " + i);
        }


        ArrayList<Double> doubleList = new ArrayList<>();
        for (double i = 4; i > 0; i--) {
            doubleList.add(i);
        }

        System.out.println("Double list before");
        for(Double d : doubleList){
            System.out.println("d = " + d);
        }

        shuffle(doubleList);
        Collections.sort(doubleList);

        System.out.println("Double list sorted");
        for(Double d : doubleList){
            System.out.println("d = " + d);
        }
    }

    private static void shuffle(ArrayList<? extends Number> list){
        ArrayList lst = list;
        Random rnd = new Random();
        for (int i=0; i<lst.size(); i++)  {
            int j = rnd.nextInt(lst.size());
            Number ii = ((Number)lst.get(i)).intValue();
            Number ij = ((Number)lst.get(j)).intValue();
            lst.set(i, ij);
            lst.set(j, ii);
        }
    }
}
```

4 Bonus exercise

Given the following code:

```java
public class ClassA<T> {
    public T t;
    public ClassA(T t) {
        this.t = t;
    }
    public void test(T o) {
        // Nothing done
    }
}
```

```java
public class ClassB<T extends Number> {
    T n;
    public ClassB(T n) {
        this.n = n;
    }
    public void test(Number t) {
        // Nothing done
    }
}
```

```java
public class ClassC<T extends Number> {
    T n;
    public ClassC(T n) {
        this.n = n;
    }
    public void test(Number t) {
        // Nothing done
    }
    public void test(ClassC<T> c) {
        // Nothing done
    }
}
```

Indicate whether the code snippets here below are working or not. If not, explain in the most precise way (but concise) why. Reminder: `Double` and `Integer` inherit from [1] `Number`.

(a)

```java
ClassA myA1 = new ClassA("Hello");
ClassA<String> myA2 = myA1;
System.out.println(myA2.t.toString());
```

(b)

```java
ClassA<String> myA1 = new ClassA("Hello");
ClassA<Integer> myA2 = myA1;
System.out.println(myA2.t.toString());
```

[1]`Double` does not inherit from `Integer`, and vice-versa.

(c)

```
ClassA<String> myA1 = new ClassA("Hello");
ClassA myA2 = myA1;
System.out.println(myA2.t.toString());
```

(d)

```
ClassA<String> myA1 = new ClassA("Hello");
ClassA myA2 = myA1;
myA2.test(myA1);
```

(e)

```
ClassA<String> myA1 = new ClassA("Hello");
ClassA myA2 = myA1;
myA1.test(myA2);
```

(f)

```
ClassA<String> myA1 = new ClassA("Hello");
ClassA myA2 = myA1;
myA1.test((String)myA2);
```

(g)

```
ClassB<Integer> myB1 = new ClassB(12);
myB1.test(myB1);
```

(h)

```
ClassC<Double> myC1 = new ClassC(3.14);
ClassC<Integer> myC2 = new ClassC(12);
myC1.test(myC2);
```

(i)

```
ClassC<Double> myC1 = new ClassC(3.14);
ClassC<Integer> myC2 = new ClassC(12);
myC1.test(myC2.n);
```

(j)

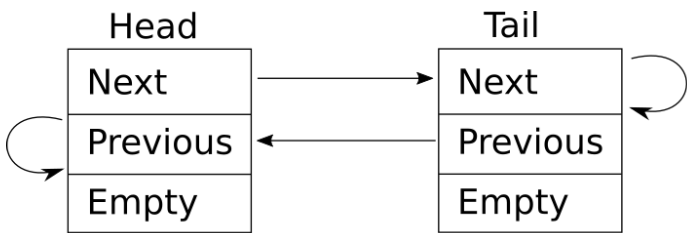
```
ClassC myC1 = new ClassC(12);
myC1.test(myC1);
```

5 Project

For this step of the project, you have to develop and test a generic linked list class to replace the previous created three lists `EnemyList`, `ShotList`, `IntegerList`. Also be aware of your class, method and variable scope (let IntelliJ help you).

5.1 Generic Linked List

Develop a generic linked list class which can contain an arbitrary number of elements. For this, you will need to create a class `Node` which can store both an element and a reference to the next node of the list[2]. The inner mechanisms of a linked list are detailed below.
The list must always contain at least two empty nodes, the head and the tail, as illustrated below:

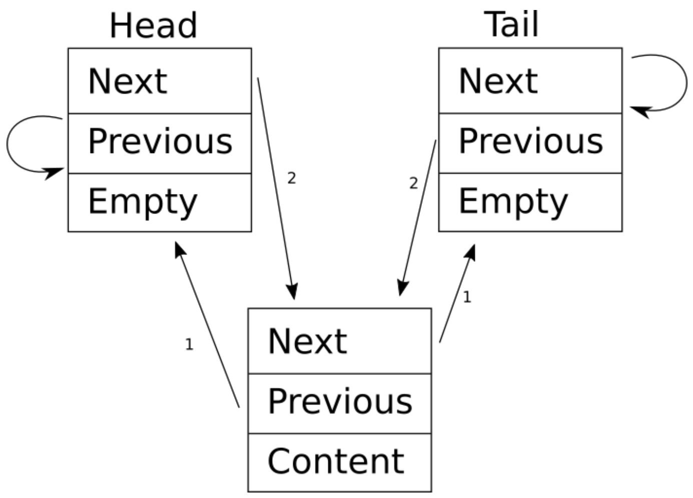


Inserting an element is done in the following way. First, create the new node, let us call it Bob, and make it have:

- The tail as next element
- The element before tail as previous element

Then, modify:

- The element coming after Bob's previous element should be Bob,
- The element coming before the tail should be Bob.



[2]You might want to search what a sentinel node is.

To remove Bob, make its previous element followed by Bob's next, and Bob's next preceded by Bob's previous.
Answering the following questions might help you for developing the list:

- In which class(es) should there be methods to add elements?
- In which class(es) should there be methods to remove elements?
- How can we know if a node is the head or the tail?
- We cannot remove the head and tail nodes. How can you make sure this is not possible?
- What is the best way to keep the count of elements in the list?

5.2 Implementation Node and MyLinkedList

Create the three generic classes called `MyLinkedList` which inherits from `java.util.AbstractSequentialList`, `Iterator` which implements `ListIterator` and `Node`.

Start by implementing the `Node` class. This class just contains three package-private variables:

- **previous:** Represents the previous node. It is of type `Node` (do not forget the generic type!)
- **next:** Represents the next node. It is of type `Node` (do not forget the generic type!)
- **content:** Represents the content of the node and has the generic type as type.

Also we need a constructor, which takes the content and assigns it.

`MyLinkedList` works as described in the section above. You need the following fields and methods:

- **head and trail:** These two nodes are the base of your list.
- **add and remove:** Override the methods from `AbstractSequentialList` as described.
- **get and size:** Override also these two methods.
- **listIterator:** You will later on also override this method. For testing purposes you can either raise an `UnsupportedOperationException` exception, or return null.

Thanks to the inheritance you dont have to implement other methods like removeAll() or `addAll()`.

5.3 Implementation Iterator

After you finished the `Node` and `MyLinkedList` class you can start with the `Iterator`. A list iterator allows you to access to the data of the list by moving through the nodes in both directions. The iterator has a field referencing the current node, and has the following methods:

- `add:` adds an element right after the current node, and update the reference of the current node to this new node
- `hasNext:` returns true if the next node is not the tail
- `hasPrevious:` returns true if the previous node is not the head
- `nextIndex:` returns the position in the list of the next node
- `next():` changes the current node to the next one and returns its content,
- `previous:` changes the current node to the previous one and return its content
- `previousIndex:` returns the position in the list of the previous node (hint: nextIndex() - 2)
- `remove:` removes the current node
- `set:` updates the content of the current node

For a more detailed description, please have a look at the official Javadoc.
If the list provides a correctly implemented iterator, you can iterate over it using a for-each loop:

```
MyListClass<String> lst = new MyListClass<>();
lst.add("Hello");
lst.add("World!");
for (String s : lst) {
  System.out.println(s);
}
```

You can also iterate backwards:

```
MyListClass<String> lst = new MyListClass<>();
lst.add("Hello");
lst.add("World!");
ListIterator<String> it = lst.listIterator(lst.size());
while (it.hasPrevious()) {
  System.out.println(it.previous());
}
```

5.4 Replace all the Old Lists

Now you have to replace all instances of `EnemyList`, `ShotList` and `Integer` with the repective generic `MyLinkedList`. To get the minimum and the maximum from your List with integers you can use the `Collection.max()/min()`. Also replace all of your for-loops with for-each-loops where it is possible.