Hinweise zur Klausur

- Bei Programmieraufgaben geht es **nicht nur** darum, eine Aufgabe korrekt zu lösen! Bewertet werden auch:
  - Sichtbarkeit von Variablen und Methoden; Kapselung
  - Qualität der Programmierung (z.B. kein überflüssiger Speicherverbrauch,...)
  - ausführliche und sinnvolle Kommentierung des Codes (besser zuviel als zuwenig)
- **Verwenden Sie nicht** die Java-Klassen `AbstractCollection`, `AbstractMap` oder `Collections`, und auch nicht deren Unterklassen wie z.B. `ArrayList, ArrayDequeue, LinkedList, HashMap, Set`, **es sei denn, dies ist ausdrücklich angegeben.**
- **Achten Sie auf Zugriffsrechte von Variablen und Methoden**. Klasseninterne Variablen sollten nach außen möglichst nicht sichtbar sein. D.h. Wenn Sie z.B. für einen Unittest den Zugriff auf eine interne Klassenvariable `var` benötigen, dann implementieren Sie in der Klasse eine lesende getter-Methode `getVar()`, die Ihnen den aktuellen Wert der Variable liefert.
- **Nur lesbare und eindeutige Lösungen werden bewertet!**
- Fehlende `import`-Anweisungen werden nicht als Fehler gewertet.

Falls eine Aufgabenstellung unklar sein sollte, ergänzen Sie die Aufgabenstellung sinnvoll. Notieren Sie Ihre Ergänzung bei Ihrer Lösung.

Ergebnis (bitte nichts eintragen!):

Frage:	1	2	3	4	5	6	7	8	9	10	Summe:
Punkte:	6	12	12	8	13	12	12	6	3	6	90
Erreicht:											

Falls die angegebene Punktzahl auf diesem Deckblatt von der Punktzahl bei der Aufgabenstellung abweichen sollte, gilt die Angabe bei der Aufgabe. Die maximal erreichbare Gesamtpunktzahl wird dann entsprechend angepasst.

1. (6 Punkte) In dieser Klausur sollen Sie den abstrakten Datentyp *Schlange* implementieren. Beschreiben Sie dafür mit eigenen Worten in maximal vier Sätzen die wesentlichen Eigenschaften dieses ADT.

Lösung:

Der DT Schlange kann als spezielle Liste aufgefasst werden, bei der die Elemente an einem Ende (hinten) eingefügt und am anderen Ende (vorne) entfernt werden.

Bewertungshinweis:

- 2 P: es handelt sich um eine (spezielle) Liste
- 2 P: es wird an einer Seite eingefügt und
- 2 P: es wird an der anderen Seite entnommen.
- 2 P: Die Listenelemente haben eine feste Reihenfolge
- Maximal 6 Punkte

Die Musterlösungen und Bewertungshinweise sind wirklich nur *Muster* bzw. *Hinweise*! Insbesondere sind oft auch andere Lösungswege möglich, und wenn jemand die eigentliche Aufgabenstellung nicht verstanden hat oder die Lösung unsinning ist, gibt es auch keine Punkte für z.B. einen korrekten Methodenkopf!

Die folgenden Aufgaben beziehen sich auf das Klassendiagramm in Abbildung 1. Es dürfen Elemente (Variablen, Methoden,..) hinzugefügt werden, *wenn dies für die Lösung der Aufgabenstellung sinnvoll ist.*

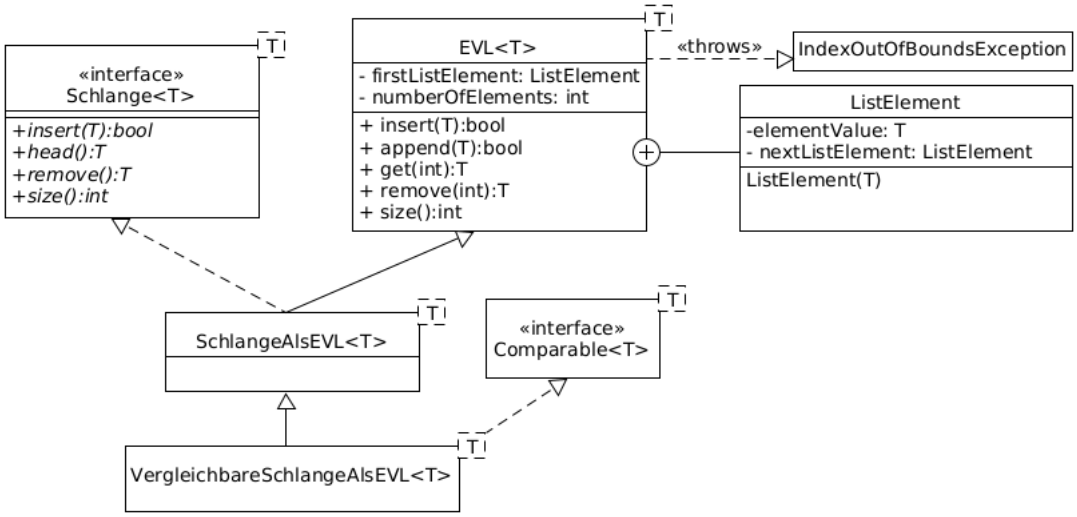

Abbildung 1: Klassendiagramm

Die Variablen und Methoden haben folgende Funktionalität: //

`Schnittstelle Schlange`:

insert(T) fügt das übergebene Element in die Schlange ein. Der Rückgabewert ist `True`, wenn dies gelingt, `False` sonst.

remove() entfernt ein Element aus der Schlange und gibt den Wert zurück; liefert `null`, falls diese leer ist.

head() wie `remove`, aber ohne das Element aus der Schlange zu entfernen.

size() liefert die Anzahl der Elemente in der Schlange.

`Klasse EVL`:

firstListElement Zeiger auf das erste Element der verketteten Liste; bei einer leeren Liste ist dieses Element `null`.

numberOfElements Anzahl der Listenelemente.

insert(T) fügt den übergebenen Wert in einem neuen Element am Anfang der Liste an. Der Rückgabewert ist `True`, wenn dies gelingt, `False` sonst.

append(T) hängt den übergebenen Wert in einem neuen Element am Ende an. Der Rückgabewert ist `True`, wenn dies gelingt, `False` sonst.

get(int pos) gibt den Wert des Elements an Position `pos` aus der Liste zurück ohne die Liste zu verändern. Das erste Listenelement hat die Position 0. Wirft eine `IndexOutOfBoundsException` wenn die Position nicht existiert (d.h. die Liste zu kurz ist).

remove(int pos) entfernt das Element an Position `pos` aus der Liste und gibt dessen Wert zurück. Wirft eine `IndexOutOfBoundsException` wenn die Position nicht existiert (d.h. die Liste zu kurz ist).

size() liefert die Anzahl der Listenelemente.

2. (12 Punkte) Der folgende Codeausschnitt skizziert, wie die Klasse `EVL` implementiert wurde.

```
public class EVL<T>{
  private class ListElement{
    T elementValue;          // der eigentliche Listenwert
    ListElement nextListElement;  // das nächste Element

    ListElement(T v){   // neues Element
      elementValue = v;
    }
  }

  private ListElement firstListElement = null;  // erstes
     Listenelement
  private int numberOfElements = 0;        // Länge der Liste

  public boolean insert(T value) {
    // Hierhin kommt Ihr Code!
  }

  // hier kommt mehr Code, den Sie NICHT implementieren sollen!
}
```

Implementieren Sie den fehlenden Code für die Methode `insert`.

Lösung:

```
  public boolean insert(T value) {    // insert 'value' at
     the head of the list
    ListElement newListElement = new ListElement(value);
    newListElement.nextListElement = firstListElement;
    firstListElement = newListElement;
    numberOfElements++;
    return true;
  }
```

Bewertungshinweis:

- 2 P: Korrekter generischer Methodenkopf (bei einigermaßen richtiger Lösung);
- 2 P: Einfügen **am Anfang** der Liste, also bei `firstListElement`;
- 2 P: Korrekte Behandlung der leeren Liste;
- 2 P: Korrekte Einordnung vorhandener Listenelemente;
- 2 P: Korrekte Erhöhung der Anzahl der Elemente `numberOfElements`;
- 2 P: Korrekter Rückgabewert `true`.
- Abzug für Programmierfehler oder umständliche / überflüssige Teile.

3. (12 Punkte) Die Methode `T get(int pos)` der Klasse `EVL` liefert den Wert an der Position `pos`. Dabei hat der erste Wert in der Liste die Position 0. Wenn der Wert von einer Position angefordert wird, die es nicht gibt (also `pos` über das Listenende hinaus positioniert), wird eine `IndexOutOfBoundsException` geworfen.

   Schreiben Sie einen Unittest als vollständige Klasse `class EVLTest`, der

   1. in einer Methode `void setUp()` vor jedem Test eine leere Liste erzeugt;
   2. in einer Testmethode `testGetBeyondEndOfList()` der Liste genau ein Element hinzufügt und dann prüft, ob beim Aufruf von `get` mit einer Position jenseits des Listenendes die richtige `Exception` geworfen wird.

Lösung:

```
class EVLTest {
  EVL<Integer> evl;

  @BeforeEach
  void setUp() {         // Initialisierung
    evl = new EVL<>();
  }

  @Test
  void testGetBeyondEndOfList() {   // Exception Test
    evl.insert(1);
    assertThrows(IndexOutOfBoundsException.class, () -> evl.
       get(1), "getBeyondEndOfList");
  }
}
```

Bewertungshinweis:

- 2 P: Korrekter Klassenkopf (bei einigermaßen richtiger Lösung);
- 1P: Korrektes `@BeforeEach`
- 2 P: Korrekte `setUp()`-Methode;
- 1P: Korrektes `@Test`
- 1P: Korrekter Methodenname (Testmethode)
- 1 P: Korrektes Einfügen genau eines Elements
- 4 P: Korrekte Prüfung der Exception mit Lambdaausdruck
- Abzug für Programmierfehler oder umständliche / überflüssige Teile.

4. (8 Punkte) Der ADT *Schlange* soll als generische Schnittstelle `interface Schlange` implementiert werden. Implementieren Sie die Schnittstelle entsprechend den Vorgaben im Klassendiagramm.

Lösung:

```
public interface Schlange<T> {
  boolean insert(T value);  // fügt 'value' am Ende der
     Schlange ein
  T head();        // liefert den Wert des ältesten Elements,
     ohne es zu entfernen
  T remove();        // entfernt ältestes Element aus der
     Schlange und gibt den Wert zurück
  int size();        // Länge der Schlange
}
```

Bewertungshinweis:

- 2 P: Korrekter generischer Klassenkopf (bei einigermaßen richtiger Lösung);
- 4 P: Korrekte Angabe der geforderten Methoden
- 2 P: Kommentar zu jeder Methode
- Abzug für Programmierfehler oder umständliche / überflüssige Teile.

5. (13 Punkte) Schreiben Sie für die Klasse `SchlangeAlsEVL` einen vollständigen *Unittest* `SchlangeAlsEVLTest` mit folgenden Testmethoden:

   testSizeOfEmptyList prüft, ob für eine leere Schlange die korrekte Länge zurückgegeben wird;

   testSizeAfterInsert prüft, ob nach dem Einfügen eines neuen Elements die korrekte Länge zurückgegeben wird;

   testSizeAfterRemove prüft, ob nach einer Entnahme die korrekte Länge zurückgegeben wird;

Lösung:

```
class SchlangeAlsEVLTest {
  SchlangeAlsEVL<Integer> schlange;

  @BeforeEach
  void setUp() {
    schlange = new SchlangeAlsEVL<Integer>();   // erzeuge
       neue (leere) Schlange vor jedem Test
  }

  @Test
  void testSizeOfEmptyList() {      // prüft, ob für eine
     leere Schlange die korrekte Länge zurückgegeben wird;
    assertEquals(0, schlange.size());
  }
  @Test
  void testSizeAfterInsert() {      // prüft, ob nach dem
     Einfügen die korrekte Länge zurückgegeben wird;
    schlange.insert(5);
    assertEquals(1, schlange.size());
  }
  @Test
  void testSizeAfterRemove() {      // prüft, ob nach dem
     Entfernen die korrekte Länge zurückgegeben wird;
    schlange.insert(5);
    schlange.remove();
    assertEquals(0, schlange.size());
  }
}
```

Bewertungshinweis:

- 2 P: Korrekter Klassenkopf (bei einigermaßen richtiger Lösung);
- 2 P: Korrekte Initialisierung, z.B. in einer `setUp()`-Methode
- 9 P: je 3 P für jeden Korrekten Test;
- Abzug für Programmierfehler oder umständliche / überflüssige Teile.

6. (12 Punkte) Implementieren Sie nun die generische Klasse `SchlangeAlsEVL` als Erweiterung der Klasse `EVL` und als Implementierung der Schnittstelle `Schlange`.

Lösung:

```
public class SchlangeAlsEVL<T> extends EVL<T> implements
   Schlange<T>{
  @Override
  public boolean insert(T value) {  // Einfügen
    return append(value);
  }

  @Override
  public T head() { // Schauen
    if(0 == size())
      return null;
    T value = get(0);
    return value;
  }

  @Override
  public T remove() { // Entfernen
    if(0 == size())
      return null;
    else
      return remove(0);
  }
}
```

Bewertungshinweis:

- 0 P **für die gesamte Aufgabe**, wenn statt der vererbten Liste eine neue interne Speicherstruktur (wie `T[]`) verwendet wurde
- 6 P: Korrekter generischer Klassenkopf (bei einigermaßen richtiger Lösung) mit `extends EVL<T>` und `implements Schlange<T>`;
- 6 P: Je 2 P für korrekte Implementierung der drei Methoden unter Verwendung der jeweiligen Methode aus `EVL`.
- Abzug für Programmierfehler oder umständliche / überflüssige Teile. Insbesondere auch für überflüssige Neuimplementierungen von EVL-Methoden.

7. (12 Punkte) Implementieren Sie nun die Klasse `VergleichbareSchlangeAlsEVL`. Objekte dieser Klasse implementieren die Java-Schnittstelle `Comparable`. Objekte der Klasse sind gleich, wenn die Anzahl der Elemente gleich ist und wenn die Werte der Elemente an jeder Position der Objekte gleich sind.

   - Bei ungleicher Länge ist die kürzere Schlange "kleiner". Der Vergleich einer *kurzen Schlange* mit einer *langen Schlange* liefert also einen negativen Wert.
   - Bei gleicher Länge aber ungleichen Elementen liefert der Vergleich das Resultat des Vergleichs der beiden ersten ungleichen Elemente.

   Welche Eigenschaft müssen die Werte der Listenelemente erfüllen, damit Sie die Klasse implementieren können? Schränken Sie diese entsprechend ein.

Lösung:

Die Werte der Listenelemente müssen vergleichbar sein, also vom Typ `T extends Comparable<T>`.

```
public class VergleichbareSchlangeAlsEVL<T extends Comparable
   <T>> extends SchlangeAlsEVL<T> implements Comparable<
   VergleichbareSchlangeAlsEVL<T>>{
  public int compareTo(VergleichbareSchlangeAlsEVL<T>
     vergleichsSchlange) {
    // Vergleiche Länge
    if(this.size() != vergleichsSchlange.size())
      return this.size() - vergleichsSchlange.size();

    // Vergleiche Elemente
    int returnValue;
    for(int pos=0; pos<size(); pos++) {
      returnValue = get(pos).compareTo(vergleichsSchlange.get
         (pos));
      if(0 != returnValue)
        return returnValue;
    }
    // Schlangen sind gleich
    return 0;
  }
}
```

Bewertungshinweis:

- 6 P: Korrekter generischer Klassenkopf (bei einigermaßen richtiger Lösung);
- 6 P: Korrekte Implementierung der Methode `compareTo`.
- Abzug für Programmierfehler oder umständliche / überflüssige Teile. Insbesondere auch für überflüssige Neuimplementierungen von EVL-Methoden.

8. (6 Punkte) Der generische `Comparator SchlangeComparatorLaenge` vergleicht zwei Objekte, welche die Schnittstelle `Schlange` implementieren, anhand ihrer Länge. Implementieren Sie diese Klasse.

Lösung:

```
public class SchlangeComparatorLaenge implements Comparator<
   Schlange<Object>>{
  // vergleiche zwei Schlangen anhand ihrer Länge
  @Override
  public int compare(Schlange s1, Schlange s2) {
    return s1.size() - s2.size();
  }
}
```

Bewertungshinweis:

- 3 P: Korrekter generischer Klassenkopf (bei einigermaßen richtiger Lösung);
- 3 P: Korrekte Implementierung der Methode `compare`.
- Abzug für Programmierfehler oder umständliche / überflüssige Teile. Insbesondere auch für überflüssige Neuimplementierungen von EVL-Methoden.

9. (3 Punkte) Die Klasse `EVL` verwendet die Methode `get(int pos)` um auf den Wert des Elements an der Position *pos* zuzugreifen. Überladen Sie die Methode, so dass sie auch ohne Parameter aufgerufen werden kann und ein sinnvolles Ergebnis liefert.

Lösung:

```
public T get() {  // Überladen der Methode
  return get(0);
}
```

Bewertungshinweis:

10. (6 Punkte) Bestimmen Sie für jede der folgenden Zuweisungen, ob diese korrekt ist:

```
class A{}
class B extends A{}
class C extends B{}
class D extends B{}

public class TypA {
  static <T> T f1(T p1, T p2) {return p2;}

  public static void main(String[] args) {
    A a1 = f1(new B(), new C());
    B b1 = f1(new B(), new C());
    C c1 = f1(new B(), new C());

    A a2 = (A) f1(new B(), new C());
    B b2 = (B) f1(new B(), new C());
    C c2 = (C) f1(new B(), new C());

    A a3 = (A) f1(new C(), new B());
    B b3 = (B) f1(new C(), new B());
    C c3 = (C) f1(new C(), new B());
  }
}
```

Beispiel	richtig
Beispiel2	falsch
a1	
b1	
c1	
a2	
b2	
c2	
a3	
b3	
c3	

Lösung:

Beispiel	richtig	
Beispiel2	falsch	
a1	richtig	0.5 P
b1	richtig	0.5 P
c1	falsch	1 P
a2	richtig	0.5 P
b2	richtig	0.5 P
c2	richtig	1 P
a3	richtig	0.5 P
b3	richtig	0.5 P
c3	falsch	1 P

Bewertungshinweis: siehe Tabelle; kein Punktabzug bei falscher Lösung.