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	3	12	6	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 *Stapel* (Stack) implementieren. Beschreiben Sie dafür mit eigenen Worten in maximal vier Sätzen die wesentlichen Eigenschaften dieses ADT.

Lösung:

Der DT Stapel kann als spezielle Liste aufgefasst werden, bei der die Elemente an einem Ende (oben) eingefügt und am gleichen Ende (oben) entfernt werden. Es wird also immer das zuletzt eingefügte Element als erstes wieder entfernt.

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 gleichen 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 (Klassen, Variablen, Methoden,..) hinzugefügt werden, *wenn dies für die Lösung der Aufgabenstellung sinnvoll ist.*

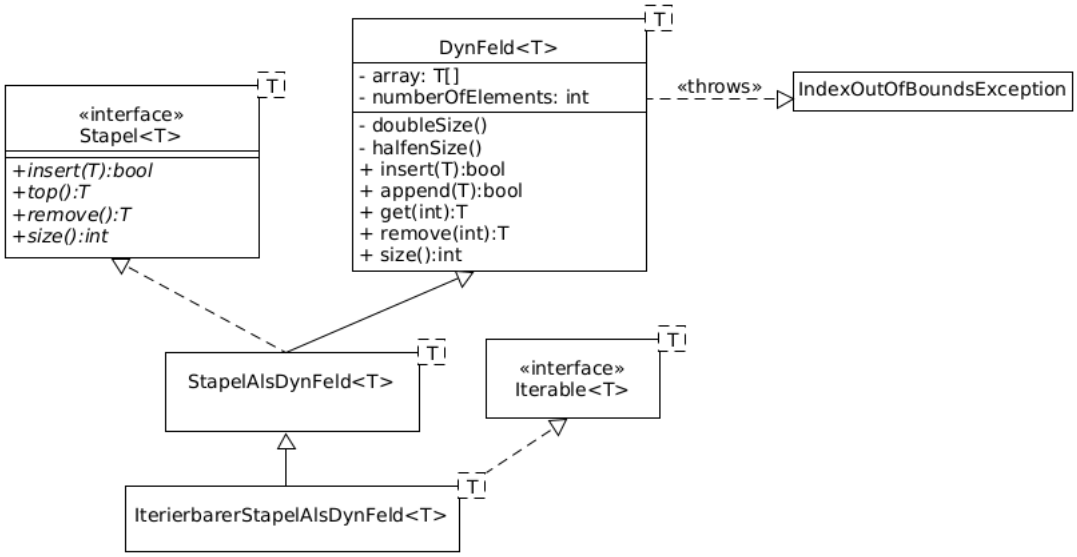

Abbildung 1: Klassendiagramm

Die Variablen und Methoden haben folgende Funktionalität:
`Schnittstelle Stapel`:

insert(T) fügt das übergebene Element dem Stapel hinzu. Der Rückgabewert ist `True`, wenn dies gelingt, `False` sonst.

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

top() wie `remove`, aber ohne das Element aus dem Stapel zu entfernen.

size() liefert die Anzahl der Elemente im Stapel.

`Klasse DynFeld` (dynamisches Feld):

array Das Feld, initial mit der Länge 1

numberOfElements Anzahl der gültigen (belegten) Listenelemente.

doubleSize() verdoppelt die Länge des Feldes

halfenSize() halbiert die Länge des Feldes

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 `DynFeld` implementiert wurde.

```
public class DynFeld<T> {
  T[] array;
  int numberOfElements = 0; // Anzahl der Listenelemente

  @SuppressWarnings("unchecked")
  public DynFeld() {
    array = (T[]) new Object[1];
  }

  private void doubleSize() {
    @SuppressWarnings("unchecked")
    T[] newArray = (T[]) new Object[2*array.length];
    for(int pos=0; pos<numberOfElements; pos++) {
      newArray[pos] = array[pos];
    }
    array = newArray;
  }

  public boolean insert(T value) {    // insert 'value' at the
     head of the list
    // hier fehlt Ihr Code
  }

  // hier kommt noch viel Code, den Sie aber nicht schreiben
     sollen!
```

Implementieren Sie den fehlenden Code für die Methode `insert`, die ein neues Element am Anfang der Liste, also auf Position 0, hinzufügt.

Lösung:

```
  public boolean insert(T value) {    // insert 'value' at
     the head of the list
    if(numberOfElements == array.length) {
      doubleSize();
    }
    for(int pos = numberOfElements; pos>0; pos--) {
      array[pos] = array[pos-1];
    }
    array[0] = value;
    numberOfElements++;
    return true;
  }
```

Bewertungshinweis:

- 2 P: Korrekter generischer Methodenkopf (bei einigermaßen richtiger Lösung);
- 2 P: Einfügen **am Anfang** der Liste, also bei `array[0]`;

- 2 P: Korrekte Erweiterung der vollen 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 `DynFeld` 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 DynFeldTest`, der

   1. in einer Methode `void setUp()` vor jedem Test ein leeres Feld erzeugt;
   2. in einer Testmethode `testGetBeyondEndOfList()` des Feldes 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 DynFeldTest {
  DynFeld<Integer> array;

  @BeforeEach
  void setUp() {      // Initialisierung
    array = new DynFeld<>();
  }
  @Test
  void testGetBeyondEndOfList() {   // Exception Test
    array.insert(1);
    assertThrows(IndexOutOfBoundsException.class, () -> array
       .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 *Stapel* soll als generische Schnittstelle `interface Stapel` implementiert werden. Implementieren Sie die Schnittstelle entsprechend den Vorgaben im Klassendiagramm.

Lösung:

```
public interface Stapel<T> {
  boolean insert(T value);  // fügt 'value' oben auf dem
     Stapel hinzu
  T top();         // liefert den Wert des obersten Elements,
     ohne es zu entfernen
  T remove();        // entfernt das oberste Element vom
     Stapel und gibt den Wert zurück
  int size();        // Höhe (Länge) des Stapels
}
```

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 `StapelAlsDynFeld` einen vollständigen *Unittest* `StapelAlsDynFeldTest` mit folgenden Testmethoden:

   testSizeOfEmptyList prüft, ob für einen leeren Stapel die korrekte Höhe (Länge) zurückgegeben wird;

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

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

Lösung:

```
class StapelAlsDynFeldTest {
  StapelAlsDynFeld<Integer> stack;
  int[] iFeld = {2,4,6,8};

  @BeforeEach
  void setUp() {
    stack = new StapelAlsDynFeld<>();
  }

  @Test
  void testSizeOfEmptyList() {      // prüft, ob für eine
     leeren Stapel die korrekte Länge zurückgegeben wird;
    assertEquals(0, stack.size());
  }
  @Test
  void testSizeAfterInsert() {      // prüft, ob nach dem
     Einfügen die korrekte Länge zurückgegeben wird;
    stack.insert(5);
    assertEquals(1, stack.size());
  }
  @Test
  void testSizeAfterRemove() {      // prüft, ob nach dem
     Entfernen die korrekte Länge zurückgegeben wird;
    stack.insert(5);
    stack.remove();
    assertEquals(0, stack.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 `StapelAlsDynFeld` als Erweiterung der Klasse `DynFeld` und als Implementierung der Schnittstelle `Stapel`.

Lösung:

```
public class StapelAlsDynFeld<T> extends DynFeld<T>
   implements Stapel<T>{
  public boolean insert(T value) {  // Einfügen
    return append(value);
  }

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

  public T remove() { // Entfernen
    if(0 == size())
      return null;
    else
      return remove(size()-1);
  }
}
```

Bewertungshinweis: Achtung: diese Aufgabe kann auf zwei verschiedene Arten korrekt gelöst werden! Alternativ zur Musterlösung kann `super.insert(T)` verwendet werden (und sollte dann hier nicht nochmal implementiert werden); dann müssen `top()` und `remove()` auf das **erste** Element zugreifen (`get(0)`).

- 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 DynFeld<T>` und `implements Stapel<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 DynFeld-Methoden.

7. (3 Punkte) Die Klasse `StapelAlsDynFeld` verwendet die Methode `get(int pos)` um auf den Wert des Elements an der Position *pos* zuzugreifen. Die `get`-Methode der Klasse `DynFeld` implementiert diese Methode so, dass die Position 0 (also der Aufruf `DynFeld.get(0)`) das Element auf Position 0 des Feldes `DynFeld.array` liefert.

   Überschreiben Sie die Methode in der Klasse `StapelAlsDynFeld`, so dass die Zählung am Ende des Feldes (und somit oben auf dem Stapel) beginnt. Der Aufruf `StapelAlsDynFeld.get(0)` liefert dann das oberste Element des Stapels, welches das letzte Element des dynamischen Feldes sein sollte (also `array[numberOfElements-1]`).

Lösung:

```
public T get(int stapelPos) {
  int dynFeldPos = size() -1 - stapelPos;
  return super.get(dynFeldPos);
}
```

Bewertungshinweis:

- 1 P korrekte Signatur (bei einigermaßen richtiger Lösung)
- 1 P korrekte Bestimmung des Ergebnisses
- 1 P korrekte Verwendung der Methode `get` der Oberklasse

8. (12 Punkte) Implementieren Sie nun die Klasse `IterierbarerStapelAlsDynFeld`. Objekte dieser Klasse implementieren die Java-Schnittstelle `Iterable`. Somit kann über die Elemente des Stapel iteriert werden. Dabei beginnt der Iterator mit dem obersten Element des Stapels (also dem zuletzt hinzugefügten Element).

Lösung:

```
public class IterierbarerStapelAlsDynFeld<T> extends
   StapelAlsDynFeld<T> implements Iterable<T> {
  private class StapelIterator implements Iterator<T>{
    int posOfNextElement = 0;

    public boolean hasNext() {
      return posOfNextElement < size();
    }

    public T next() {
      return get(posOfNextElement++);
    }

  }

  public Iterator<T> iterator() {
    return new StapelIterator();
  }
}
```

Bewertungshinweis:

- 6 P: Korrekter generischer Klassenkopf (bei einigermaßen richtiger Lösung);
- 6 P: je 2 P für korrekte Implementierung der Methoden `hasNext, next, und iterator`.
- Abzug für Programmierfehler oder umständliche / überflüssige Teile. Insbesondere auch für überflüssige Neuimplementierungen von DynFeld-Methoden.

9. (6 Punkte) Der generische `Comparator StapelComparatorLaenge` vergleicht zwei Objekte, welche die Schnittstelle `Stapel` implementieren, anhand ihrer Länge. Implementieren Sie diese Klasse.

Lösung:

```
@SuppressWarnings("rawtypes")
public class StapelComparatorLaenge implements Comparator<
   Stapel>{
    // vergleiche zwei Stapel anhand ihrer Länge
    @Override
    public int compare(Stapel s1, Stapel s2) {
      return s1.size() - s2.size();
    }
}
```

Bewertungshinweis:

- 3 P: Korrekter generischer Klassenkopf (bei einigermaßen richtiger Lösung); die Variante mit
  `implements Comparator<Stapel<Object>>` ist auch richtig. Lösungen mit konkreten Klassen wie z.B. `implements Comparator<Stapel<Integer>>` sind falsch.
- Varianten mit generischen Typen (`<T>`) können richtig sein.
- 3 P: Korrekte Implementierung der Methode `compare`.
- Abzug für Programmierfehler oder umständliche / überflüssige Teile.

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 TypB {
  static <T> T f1(T p1, T p2) {return p1;}

  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	falsch	1 P
a3	richtig	0.5 P
b3	richtig	0.5 P
c3	richtig	1 P

Bewertungshinweis: siehe Tabelle; kein Punktabzug bei falscher Lösung.