Klausur Nr.: 1 Fach: SD Klassen: IT2a/IT2b/IT2p Datum: 02.12.2024 Kürzel: KLG/MUE			Name: _____			
Punkte: _____ / 87 Prozent:			Note: Unterschrift:			
Note	**1**	**2**	**3**	**4**	**5**	**6**
Prozent	100-92	91-81	80-67	66-50	49-30	29-0
Punkte	87,0-80,0	79,5-70,5	70,0-58,0	57,5-43,5	43,0-26,0	25,5-0,0

Bearbeitungshinweise:

- Die Bearbeitungszeit beträgt 90 Minuten.
- Dokumentieren Sie die Programmieraufgaben 1 und 2 platzsparend in der bereitgestellten Word-Datei *SD_LF8_K1_NN.docx*. Sehen Sie auch Platz für Korrekturen vor!

Lernfeld 8: Daten systemübergreifend bereitstellen

Aufgabe 1 (Sortierverfahren in Java – 24 Punkte)

Verwenden Sie für diese Aufgabe das IntelliJ-Projekt „Sortieren“. Die zu programmierenden Methoden lassen sich in Kombination mit Hilfe des Hauptprogramms testen.

a) Programmieren Sie die Methode **swap**. (3 Punkte)

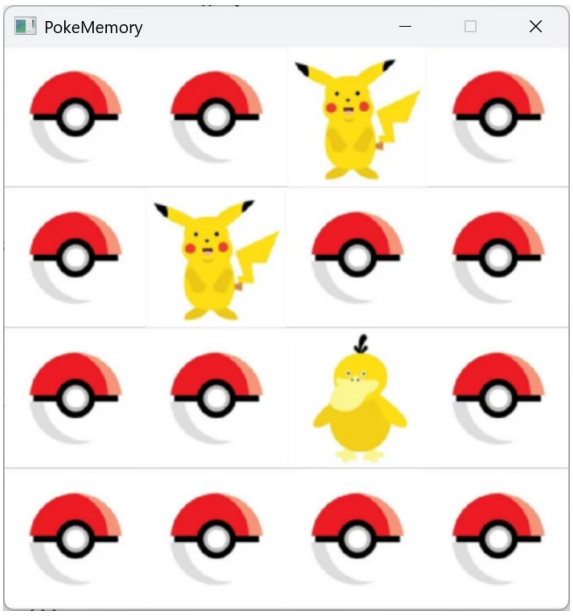
b) Programmieren Sie die Methode **isArraySorted**. (5 Punkte)

c) Programmieren Sie **genau zwei** der in **Anlage I** dargestellten Sortierverfahren. (16 Punkte)

Aufgabe 2 (PokeMemory mit Model-View-Controller – 33 Punkte)

Verwenden Sie für diese Aufgabe das IntelliJ-Projekt *PokeMemory* und bearbeiten Sie die Aufgabenteile a) bis g) auf Seite 2. Programmablauf:

- Zu Beginn werden die Zahlen von 1 bis 8 jeweils an zwei zufälligen Positionen eines zweidimensionalen Arrays hinterlegt, aber nicht angezeigt. Stattdessen werden zunächst alle Buttons mit einem Pokeball als Hintergrund dargestellt.
- Nach jedem Klick wird das zu der hinterlegten Zahl gehörige Pokemon-Bild als Hintergrund des geklickten Buttons angezeigt. Handelt es sich um das zweite „vorläufig aufgedeckte“ Bild, so wird geprüft, ob es sich um ein Paar handelt oder nicht. Falls es sich nicht um ein Paar handelt, werden die Bilder nach einer Wartezeit von einer Sekunde (bzw. einer Milliarde Nanosekunden) wieder zugedeckt.



Für das Projekt liegt bereits das folgende Klassendiagramm in **Anlage II** vor.

Die Klassen *Controller* und *View* sind weitgehend vollständig und die fertigen Methoden sollen nicht verändert werden. Die Klasse *Model* ist nach dem MVC-Prinzip vollkommen unabhängig von den Klassen *Controller* und *View*.

Erzeugen Sie die Klasse *Model* gemäß dem Klassendiagramm:

a) Die Klasse ist öffentlich und enthält drei öffentliche, statische und finale Attribute ANZAHLZEILEN, ANZAHLSPALTEN und ANZAHLBILDER, zwei private, zweidimensionale Arrays *zahlen* (mit dem Grundtyp *int*) und *aufgedeckt* (mit dem Grundtyp *boolean*). Außerdem speichert das *Model* in den Variablen *anzahl_vorlaeufig_aufgedeckt* und dem zweidimensionalen Array *vorlaeufig_aufgedeckt*, wie viele (0 bis 2) und ggf. welche Positionen aktuell vorläufig aufgedeckt sind. (6 Punkte)

b) Der Konstruktor der Klasse *Model* initialisiert den Array *zahlen* mit Zufallszahlen, die den Nummern 1 bis 8 aus den Bilddateien der Pokemon entsprechen, von denen jede aber genau zweimal vorkommt. (10 Punkte)

   Hinweis: Bei Übergabe einer *int*-Zahl *bound* liefert die Methode *nextInt* der Klasse *Random* aus *java.util* eine Zufallszahl zwischen 0 und *bound* - 1. Bei Übergabe von zwei *int*-Zahlen *origin* und *bound*, liefert *nextInt* eine Zahl ab *origin* (einschließlich) bis *bound* (ausschließlich).

   Bemühen Sie sich bitte (wie immer!) um einen Lösungsversuch, der die Aufgabe wenigstens teilweise erfüllt, ohne dass sie überproportional viel Zeit für diesen Aufgabenteil aufwenden.

c) Die Methode *getZahl(i : int, j : int)* liefert die an der entsprechenden Position hinterlegte Zahl. (2 Punkte)

d) Die Methode *istAufgedeckt(i : int, j : int)* liefert den entsprechenden Eintrag des Arrays *aufgedeckt*. (2 Punkte)

e) Die Methode *aufdecken(i : int, j : int)* überprüft zunächst, ob die Zahl mit den Indizes *i* und *j* noch nicht aufgedeckt und aktuell auch noch keine zwei Zahlen vorläufig aufgedeckt sind. Ist das der Fall, setzt sie den zu den Indizes *i* und *j* gehörigen Eintrag von *aufgedeckt* auf den Wert *true*, aktualisiert *vorlaeufig_aufgedeckt*, indem die Indizes *i* und *j* an den passenden Positionen gespeichert werden, und inkrementiert *anzahl_vorlaeufig_aufgedeckt*. (6 Punkte)

f) Die Methode *getAnzahlVorlaeufigAufgedeckt()* liefert den entsprechenden Wert. (1 Punkt)

g) Die Methode *aktuellesPaarPruefen()* liefert sofort *false*, falls nicht zwei Zahlen vorläufig aufgedeckt sind. Andernfalls wird überprüft, ob die Zahlen an den in *vorlaeufig_aufgedeckt* gespeicherten Positionen gleich sind. In beiden Fällen setzt die Funktion die Anzahl vorläufig aufgedeckter Karten zurück und liefert das Ergebnis der Überprüfung als Rückgabewert. Nur falls die Zahlen nicht übereinstimmen, werden sie vorher wieder „zugedeckt“. (6 Punkte)

Aufgabe 3 (Testen von Software - 30 Punkte)

a) Ein Benutzerportal hat sich für neue Passwortrichtlinien entschieden:

   - Es muss mindestens zehn Zeichen lang sein.
   - Es muss mindestens einen Buchstaben (*letter*), eine Ziffer (*digit*) und ein Sonderzeichen (*specialChar*) enthalten.

   Es liegt ein fehlerhaftes Struktogramm zur Überprüfung des Passworts vor (siehe **Anlage III**). Der Rückgabewert soll wahr oder falsch sein. Beschreiben Sie auf der Folgeseite vier Fehler und geben Sie jeweils eine Korrektur an. (12 Punkte)

   Hinweis: Es gibt streng genommen sogar fünf Fehler.

b) Beantworten Sie folgende Multiple-Choice-Fragen. (8 Punkte)

Tragen Sie bei richtigen Antworten eine 1 ein und bei falschen eine 0.

Frage 1: Welche Testmethode haben Sie bei der Bearbeitung von Aufgabenteil a) angewendet?

__ (A) Whitebox-Test

__ (B) Blackbox-Test

__ (C) Eine Kombination aus beiden

__ (D) Keine der genannten

Frage 2: Ein Tester überprüft, ob das Passwort den Anforderungen entspricht, ohne den Code selbst zu betrachten. Um was für eine Testmethode handelt es sich in diesem Fall?

__ (A) Whitebox-Test

__ (B) Blackbox-Test

__ (C) Unit-Test

__ (D) Integrationstest

Frage 3: Wie wird im Code der Passwortprüfung das stellenweise Überprüfen der Kriterien realisiert?

__ (A) Rekursiv

__ (B) Iterativ

__ (C) Parallel

__ (D) Dynamisch

Frage 4: Kreuzen Sie die richtigen Aussagen an:

__ (A) Ein rekursiver Algorithmus arbeitet meist schneller als ein iterativer.

__ (B) Bei rekursiven Algorithmen kommt es zu wiederholten Selbstaufrufen.

__ (C) Iterative Algorithmen eignen sich besonders für Probleme, die sich in Teilprobleme unterteilen lassen (*Divide and Conquer*).

__ (D) Iterative Algorithmen benötigen oft weniger Speicher als rekursive, da der Call-Stack kleiner bleibt.

c) Betrachten Sie den Algorithmus in Pseudocode in **Anlage III**, der eine Dezimalzahl in eine Binärzahl umrechnen soll.

ca) Führen Sie den Schreibtischtest aus und tragen Sie das Ergebnis in die rechte Spalte „Ergebnis Schreibtischtest“ ein. (4 Punkte)

Testfall	Dezimalzahl	Erwartetes Ergebnis	Ergebnis Schreibtischtest
1	7	111	
2	11	1011	

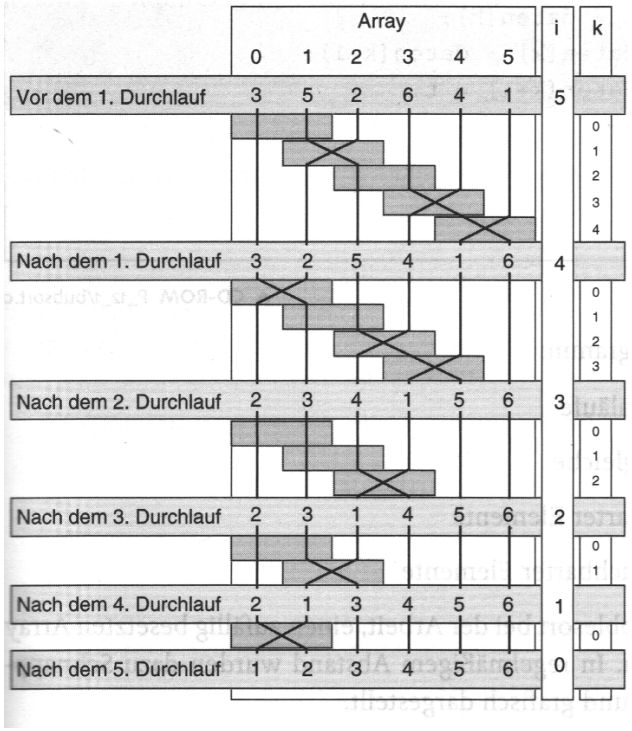
cb) Beschreiben Sie den Fehler und eine Korrekturmöglichkeit. (4 Punkte)

cc) Geben Sie eine möglichst kleine Menge von Testfällen an, um bei dem gegebenen Algorithmus Anweisungsüberdeckung zu erreichen. (2 Punkte)

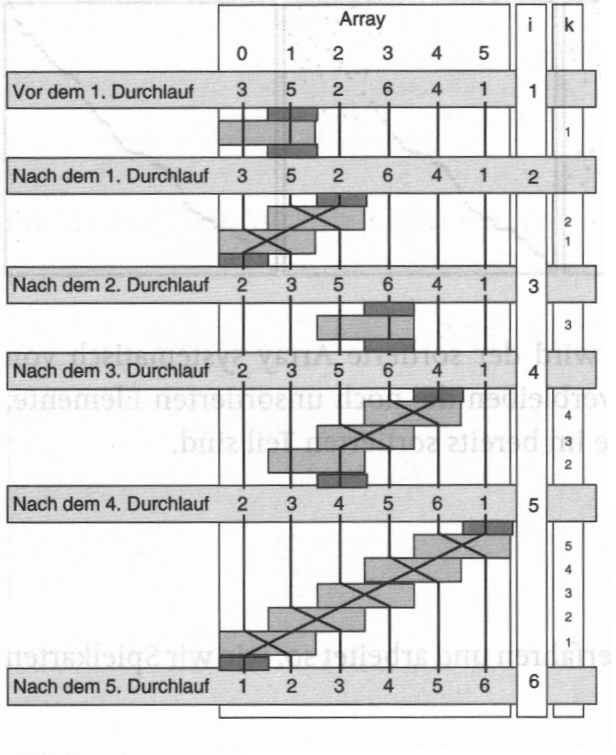
Anlage I

Grafische Darstellungen von BubbleSort, InsertionSort und SelectionSort für Aufgabe 1 c)

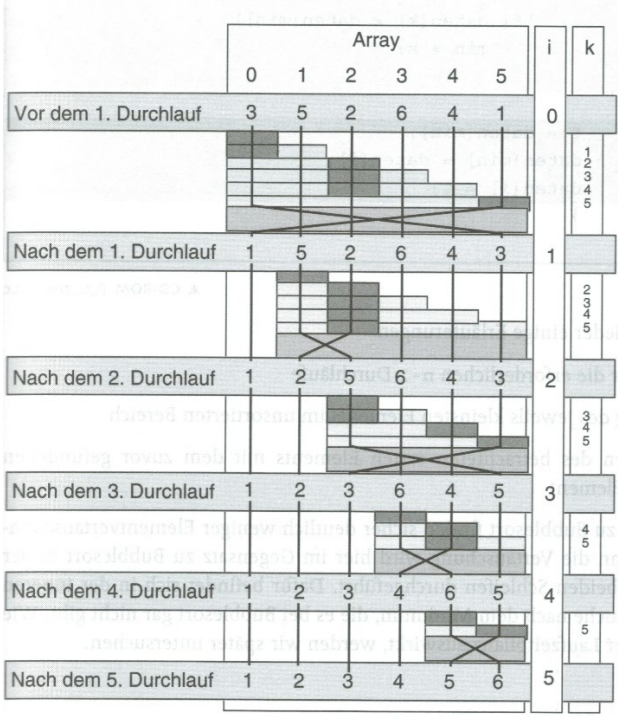
BubbleSort



InsertionSort

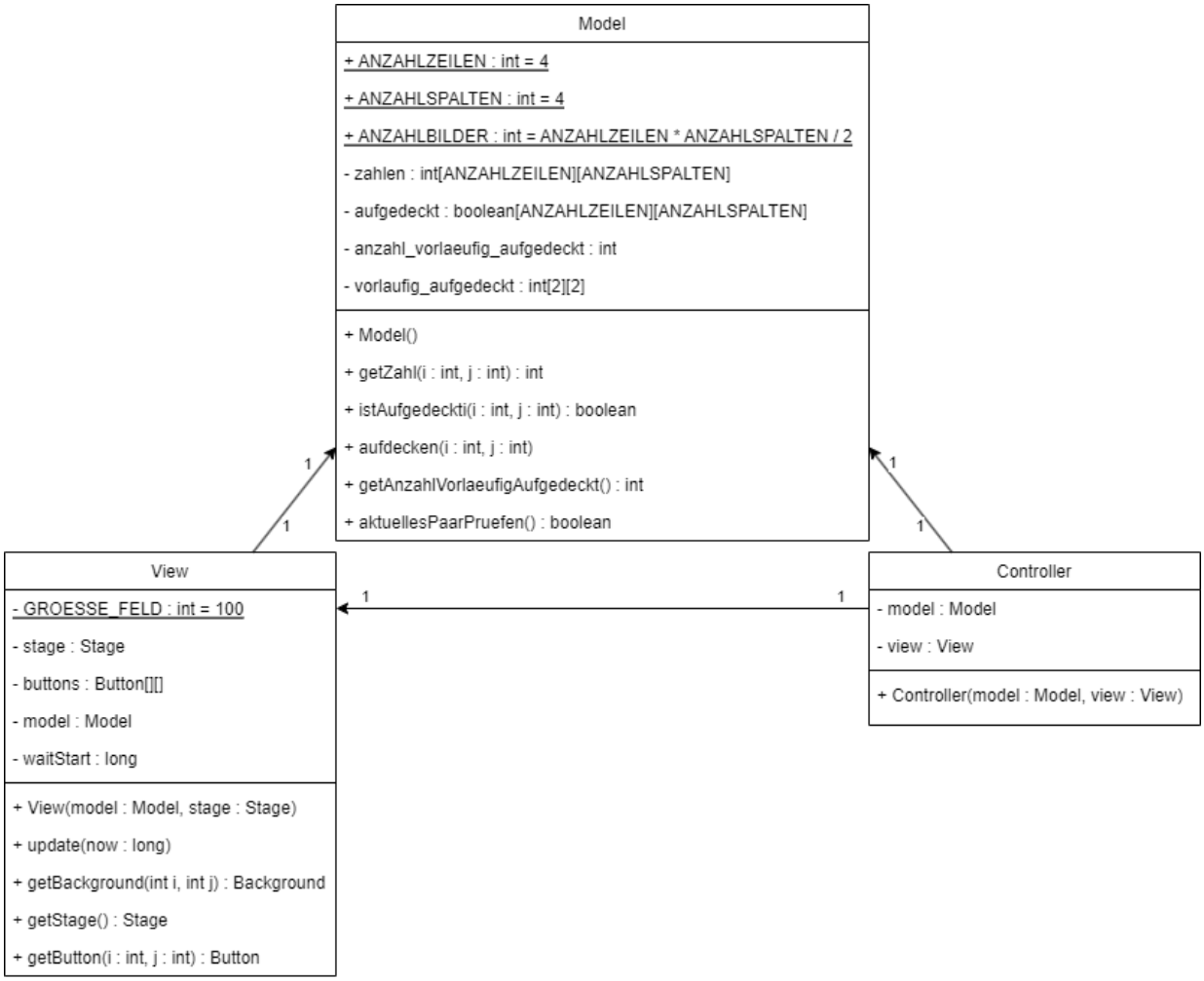


SelectionSort



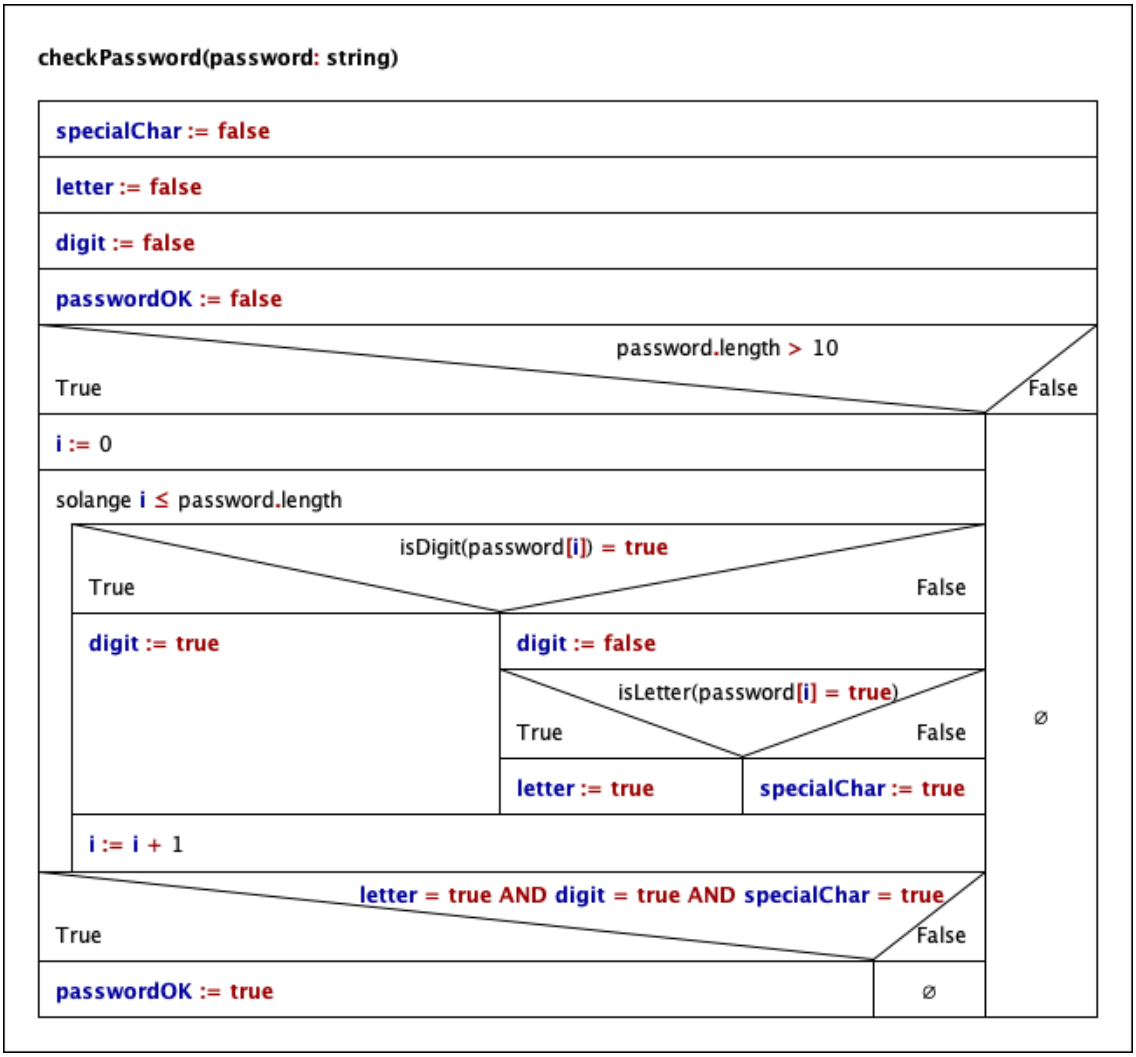
Anlage II

Klassendiagramm zu Aufgabe 2



Anlage III

Struktogramm zu Aufgabe 3 a)



Pseudocode zu Aufgabe 3 c)

```
void Umrechnung(dezimalzahl: int)
BEGINN
      rest: int;
      rest = dezimalzahl modulo 2;
      AUSGABE(rest);
      dezimalzahl = dezimalzahl div 2; //div ist ganzzahlige Division
      WENN dezimalzahl > 0 DANN
            Umrechnung(dezimalzahl);
      ENDE
ENDE
```