**Hinweis: Die folgenden Aufgaben sind Aufgaben, die in Klausuren gestellt wurden oder als Vorbereitung für die Klausur gedacht sind.   
Beachten Sie bitte,**

* **dass alle in der Vorlesung/Übungen behandelten Teilgebiete prüfungsrelevant sind, auch wenn vielleicht zu dem ein oder anderen Themengebiet hier keine Klausuraufgabe auftaucht.**
* **dass diese Musterklausur mehr Aufgaben enthält als üblicherweise gestellt werden.**

# Aufgabe

Erklären Sie für jede Aussage, ob sie richtig oder falsch ist:

1. „Durch den Modifier protected vor Instanzvariablen einer Klasse A wird die Datenkapselung für alle Unterklassen von A aufgehoben.“

**richtig**

1. „Eine innere Klasse hat immer direkt Zugriff auf die Instanzvariablen der äußeren Klasse“  
     
   **richtig**
2. “Ein interface wird nur durch genau eine Klasse implementiert “  
     
   **falsch**
3. „Ein Konstruktor einer Klasse dient der Initialisierung der Instanzvariablen einer Klasse“  
     
   **richtig**
4. „Component Objekte werden in einem Container ausschließlich durch Angabe der Koordinaten platziert. “  
     
   **falsch**
5. “Bei einem serialisierbaren Objekt werden die Werte derjenigen Instanzvariablen, die mit dem Schlüsselwort transient gekennzeichnet sind, beim Schreiben auf einen OutputStream (z.B. auf eine Datei) geschrieben. “  
     
   **falsch**

# Aufgabe

Erstellen Sie ein Programm Umwandle, das aus dem Inhalt einer Datei (Eingabe-Datei)

* **alle** **Großbuchstaben ’X’** (das Zeichen Gross-X) **zu Leerzeichen ’ ’** und
* **alle Leerzeichen ’ ’ zu Grossbuchstaben ’X’** (das Zeichen Gross-X) **wandelt** und

und **den konvertierten Inhalt dann in einer zweiten Datei** (Ausgabe-Datei) speichert. Alle anderen in der Eingabe-Datei enthaltenen Buchstaben werden unverändert in die Ausgabe-Datei übernommen. Die Namen beider Dateien werden beim Programmaufruf dem

Programm in der Kommandozeile übergeben.

Beispiel Eingabedatei:

eingabe.txt

Der X-te Satz mit X war wohl niX.

Mit dem Aufruf **java Umwandle eingabe.txt ausgabe.txt** wird im Inhalt der Datei eingabe.txt jedes Leerzeichen ’ ’ durch den Grossbuchstaben ’X’ und jeder

Grossbuchstabe ’X’ durch ein Leerzeichen ’ ’ ersetzt und in der Datei ausgabe.txt

gespeichert.  
  
  
 ausgabe.txt

DerX –teXSatzXmitX XwarXwohlXni .

# Aufgabe

Gegeben ist folgendes 8x8 Brett. Es gibt blaue und rote Brettfiguren. Alle Figuren können nur vertikal verschoben werden. Eine blaue Figur kann jeweils zwei Felder vorziehen, eine rote Figur nur ein Feld. Die Position eines Feldes wird durch die „Matrixposition“ definiert (z.B. (4,5) bedeutet 4. Zeile und 5. Spalte.)

(0,0)

(8,8)

Zugrichtung

Abb. 1: Brett mit entsprechenden Figuren

a. Implementieren Sie in Java Notation eine Klasse Brett, die das Brett als zweidimensionales Feld realisiert. Stellen Sie hierzu einen Konstruktor zur Verfügung, der das Brett mit 64 Feldern und den entsprechenden Brettfiguren gemäß Abbildung 1 erzeugt

b. Führen Sie eine interface Klasse Brettfigur ein . Ergänzen Sie Ihre Konstruktor Implementierung von a und initialisieren das Brett mit jeweils 4 roten und 4 blauen Figuren gemäß obiger Graphik. Außerdem führen Sie in der Klasse Brett eine Methode public void print() ein, die mit System.out die aktuelle Belegung des Bretts ausgibt und zwar so, dass für jedes Feld angegeben wird, ob es mit einer Figur belegt ist und wenn ja, welche Farbe die Figur hat.

c. Erweitern Sie Ihre Klassen so, dass blaue und rote Figuren gemäß der oben definierten Zugsemantik gezogen werden können. Hierzu führen Sie in der Klasse Brett eine Methode public void ziehen() ein, die ausgehend von der aktuellen Brett Belegung in einem Zug entweder alle roten oder alle blauen Steine gemäß ihrer Zugsemantik vorziehen lässt. Blau beginnt. Falls eine Figur das andere Ende erreicht ist, kann diese nicht mehr gezogen werden und verbleibt auf diesem Feld.

**Beachten Sie bitte, dass dieses Spiel zukünftig um weitere Figuren mit anderen Farben und Zugmöglichkeiten erweitert wird und versuchen Sie deshalb möglichst viel Code zu generalisieren und wiederzuverwenden**.

**Lösung siehe separates Verzeichnis (Die Java Lösung ist ausführlicher als in der Aufgabe gefordert.)**

# Aufgabe

Gegeben ist folgende Vererbungshierarchie:

abstract class Writer {

public abstract void write(String s);

}

interface Drucker {

public void druckenAbbrechen();

}

class DateiWriter extends Writer {

public void write(String s) { … };

}

class DruckerWriter extends Writer implements Drucker {

public void write(String s) { … };

public void druckenAbbrechen(){ … };

}

class PipedWriter extends Writer {

public void write(String s) { … };

}

Welche der folgenden Programmzeilen sind falsch? Begründen Sie Ihre Antwort. (Die erste Spalte zeigt die jeweilige Zeilennummer):

1. Writer w1;
2. Drucker d1;
3. PipedWriter p1;
4. Writer w2 = new Writer();  
   **falsch, da abstrakte Klasse**
5. DateiWriter da = new DateiWriter();  
   **richtig**
6. DruckerWriter d2 = new Drucker();  
   **falsch, da interface nicht instantiierbar**
7. PipedWriter p2 = new PipedWriter();  
   **richtig**
8. w2 = da;  
   **richtig**
9. w1 = (Writer) da;  
   **richtig (downcast überflüssig)**
10. p1 = p2;  
    **richtig**
11. p2 = (PipedWriter) w1;  
    **falsch, da nicht jeder writer ein PipedWriter**
12. d2 = new DruckerWriter();  
    **richtig**
13. d1 = d2;  
    **richtig, da jeder DruckerWriter das interface Drucker implementiert hat**
14. p2 = d2;  
    **falsch**
15. w1.write(“Testausgabe in Datei“);  
    **richtig**
16. d2.druckenAbbrechen();  
    **richtig, d2 ist in Zeile 12 initialisiert**

# Aufgabe

Entwickeln Sie ein System zur Verwaltung von Bankkonten.

Das System besteht aus einer Basisklasse Konto und Spezialisierungen für Girokonten (Klasse Girokonto), Sparkonten (Klasse Sparkonto) und Kreditkonten (Klasse KreditKonto).

**Basisklasse Konto:**

Jedes Konto hat eine eindeutige 7-stellige Kontonummer und einen aktuellen Kontostand. Deklarieren Sie entsprechende Felder und Zugriffsmethoden. Die Kontonummer muss eindeutig sein und sollte daher vom Programm generiert und unveränderlich sein. Überlegen Sie die Sichtbarkeitsattribute für die Felder und ob auch eine set-Methode definiert werden soll (d.h. ob das Feld von außen gesetzt werden kann).

**Hinweise:**

- Zur Vereinfachung können Sie für die Geldbeträge den Typ int verwenden.

- Verwenden Sie ein static int Instanzvariable, um 7-stellige Kontonummern zu verwalten und zu generieren.

Man kann von einem Konto Geld abheben und auf ein Konto Geld einzahlen. Definieren Sie dazu entsprechende Methoden.

Es soll möglich sein, die letzte Behebung oder Einzahlung rückgängig zu machen. Realisieren Sie dazu eine Methode void undoLastOperation().

Definieren Sie weiters eine Methode public String toString(), um eine String-Repräsentation des Kontos zu generieren.

**Spezialisierungen Girokonto, Kreditkonto, Sparkonto:**

Spezialisieren Sie diese Klassen als Unterklassen von Konto. Dabei sind folgende Zusätze und Einschränkungen zu beachten:

**Girokonten:**

- Haben eine maximale Überziehung. Über diese Grenze darf das Konto nicht belastet

werden.

\_ **Sparkonten**:

- Dürfen nur im Haben sein (Kontostand > 0).

\_ **Kreditkonten**:

- Kreditkonten werden anfänglich mit einem Soll (Kontostand < 0) angelegt. Man kann darauf nur einzahlen (Kredit zurückzahlen) und nicht weiter abheben.

- Auf das Kreditkonto kann nicht mehr eingezahlt werden als der ausstehende Kredit

ausmacht (Kontostand darf nicht > 0 werden).

Entwickeln Sie eine Klassenhierarchie für den oben beschriebenen Sachverhalt.  
Prüfen Sie dabei alle diese Bedingungen in den Methoden. Die Methoden sollen bei Verletzung von Bedingungen Exceptions auslösen. Definieren Sie sich dazu eine eigene Exception-Klasse.

**Lösung siehe separates Verzeichnis (Die Java Lösung ist ausführlicher als in der Aufgabe gefordert.)**