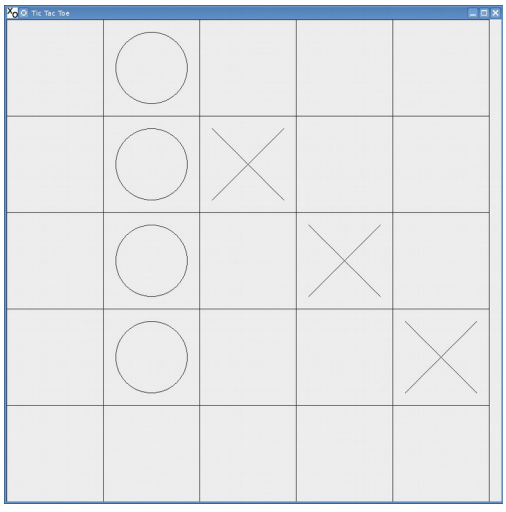
Examination
User Interaction (UI)

Prof. Dr.-Ing. Christian Heller <christian.heller@ba-leipzig.de>

Personal Data	
First and Last Name	Ihre Daten werden von der Klausuraufsichtsperson schriftlich auf der Anwesenheitsliste festgehalten.
Matriculation Number	
Subject and Year	CS 2014
Login	Das Login "klaus..." muss unbedingt schriftlich auf der Anwesenheitsliste festgehalten werden, da sonst keine Zuordnung des Logins zu Ihrem Namen und damit keine Korrektur der Klausur möglich ist!

Examination Data	
Date	2015-12-23
Duration [min]	100
Permitted Study Aids	Dokumentation im lokalen Netzwerkverzeichnis (Intranet); NICHT gestattet sind Kommunikationsmöglichkeiten (Internet) oder Anmeldung via SSH auf dem Rechner "fileserv", wo Ihr Homeverzeichnis liegt, oder eine Anmeldung mit Ihrem Klausur-Login nach Ende der Prüfung. Dies kann leicht geprüft werden (last cs12*, Server-Log-Dateien). Bitte unterlassen Sie also Täuschungsversuche in Ihrem eigenen Interesse.
Remarks	Hinterlegen Sie alle Programme und Antworten in elektronischer Form! Es wird kein Papier angenommen. Möchten Sie Lösungen erläutern, so nutzen Sie Quelltext-Kommentare oder legen eine Text-Datei an. Speichern Sie sämtliche Daten im HOME-Verzeichnis des Nutzers, d.h. unter Windows auf Laufwerk H:\ (NICHT auf C:\ oder "Eigene Dateien")! Idealerweise legen Sie dort ein Unterverzeichnis namens "klausur" an. Lesen Sie die Aufgaben komplett durch, bevor Sie sie lösen! Die Reihenfolge der Lösung ist Ihnen überlassen. Probieren Sie immer, eine Aufgabe zu lösen, da auch auf richtige Teile nicht vollständiger Lösungen Punkte vergeben werden! Falls vom Prinzip her richtig, so werden auch alternative Lösungen akzeptiert. Sie dürfen beliebig viele Bildschirmausgaben von Werten in den Quelltext einbauen, um ein Programm besser nachvollziehen zu können. Diese Aufgabenstellung in Papierform können Sie nach dem Ende der Klausur behalten.

Evaluation						
Task	1	2	3	4	5	Summe
Points	10	30	20	20	20	100



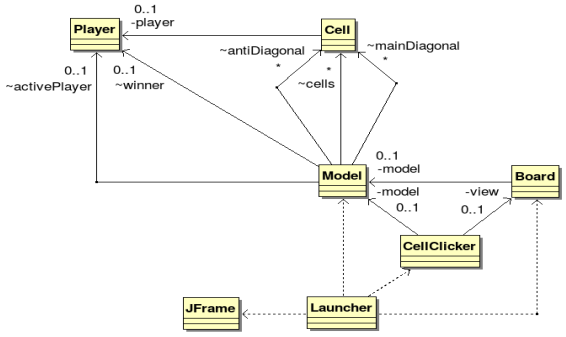
Im Verlauf der Klausur soll das Spiel "Tic-Tac-Toe", auch bekannt als "Drei gewinnt", "Kreis und Kreuz", "Dodelschach", erstellt werden. Nach [Wikipedia] ist es: "ein klassisches, einfaches Zweipersonen-Strategiespiel, dessen Geschichte sich bis ins 12. Jahrhundert v. Chr. zurückverfolgen lässt." Seine grafische Oberfläche soll dem obigen Bildschirmschnappschuss ähnlich sehen.

Es wird angenommen, dass die beiden Spieler abwechselnd die Maus betätigen. Der Rechner führt also keine automatischen Züge aus.

Hinweis: Wenn Sichtbarkeiten von Attributen oder Methoden in dieser und folgenden Aufgaben nicht explizit gefordert sind, ist die Verwendung von Kapselungsmethoden Ihnen überlassen.

Zum besseren Verständnis der Architektur der Anwendung sei das folgende UML-Klassendiagramm gegeben:

Task 1: Swing Frame [10]



Zweck: Es wird ein Rahmenfenster auf Basis der *Swing*-Klassenbibliothek erstellt.

a) Erstellen Sie innerhalb einer Startklasse mit statischer Einstiegsmethode ein Swing-Frame-Objekt! Geben Sie ihm den Titel "Tic Tac Toe"! [2]

b) Justieren Sie des Fensters Größe auf *800 x 800 Picture Elements*, indem Sie eine Kombination der Methoden *setPreferredSize* und *pack* verwenden! [2]

c) Fixieren Sie die Fenstergröße (unveränderlich) und platzieren Sie es auf dem Bildschirm an der Stelle *20 x 20 Picture Elements*! [2]

d) Verwenden Sie die gegebene Datei *icon.png* als Symbol (*Icon*) für das Fenster! [2]

e) Weisen Sie dem Fenster eine Schließoperation zu, die bewirkt, dass sich die Anwendung beendet! Schalten Sie es sichtbar! [2]

Ergebnis: Beim Start der Anwendung sollte nun ein Fenster zu sehen sein.

Task 2: Data Model [30]

Zweck: Gegeben sei eine Klasse namens *Model*. Damit diese funktioniert, werden hier zwei weitere Klassen zur Datenhaltung erzeugt.

a) Erstellen Sie eine Klasse namens *Player*, welche ein Instanzattribut *name* vom Typ *String* besitzt! [2]

b) Initialisieren Sie es mittels eines an den Konstruktor übergebenen Argumentes! [2]

c) Lassen Sie den Attributwert via überschriebener *toString*-Methode zurückgeben! [2]

d) Erzeugen Sie zwei Konstanten namens CROSS und NAUGHT des Types Player! [2]

e) Weisen Sie Ihnen direkt in der Deklaration (nicht im Konstruktor) je ein Objekt des Types *Player* zu! Übergeben Sie als Zeichenketten "*cross*" bzw. "*naught*" an die Konstruktoren! [2]

f) Erstellen Sie eine weitere Klasse namens *Cell*! Geben Sie ihr zwei Attribute *column* und *row*, jeweils vom Typ *int*, sowie ein Attribut *player* vom Typ *Player*! [2]

g) Setzen Sie die Sichtbarkeit für alle drei Attribute auf *private* und implementieren Sie entsprechende Zugriffsmethoden! [2]

h) Sorgen Sie dafür, dass in der Zuweisungsmethode *setPlayer* der Attributwert nur neu gesetzt wird, wenn noch kein Wert vorhanden ist! [2]

i) Initialisieren Sie die zwei Attribute vom Typ *int* mittels an den Konstruktor übergebener Argumente! Weisen Sie dem Attribut *player* den Wert *null* zu! [2]

j) Überschreiben Sie die geerbte *toString*-Methode, um der Zelle Zeile und Spalte sowie den ihr zugeordneten Spieler sinnvoll formatiert zurückzugeben! [2]

k) Erzeugen Sie nun in der *main*-Methode der Startklasse eine Instanz der Klasse Model und rufen Sie ihre *initialise*-Methode auf! [2]

l) Nehmen Sie ein Kommandozeilen-Argument entgegen und wandeln es in eine Ganzzahl um! Speichern Sie diese in einer lokalen Variablen namens *size*! [2]

m) Führen Sie eine weitere lokale Variable namens *starter* vom Typ *Player* ein und initialisieren Sie sie mit dem Wert der Konstanten *Player.CROSS*! [2]

n) Ermitteln Sie, ob es sich beim zweiten Kommandozeilen-Argument um den Wert "*naught*" handelt! Falls ja, weisen Sie der lokalen Variablen *starter* die passende Konstante zu! [2]

o) Übergeben Sie die Größe und den Startspieler als Argumente an die *initialise*-Methode des *Model*-Objektes! [2]

Ergebnis: Die Anwendung sollte nach wie vor fehlerfrei und startbar sein und ihr Fenster leer aussehen wie zuvor.

Task 3: Board Component [20]

Zweck: Die Hauptkachel wird justiert.

a) Erstellen Sie eine Klasse namens *Board*, welche von *JComponent* erbt! [2]

b) Geben Sie ihr ein Attribut *model* des Types *Model*, welches über ein an eine neu zu erstellende *initialise*-Methode übergebenes Argument zuzuweisen ist! [2]

c) Fügen Sie außerdem die zwei Attribute *sideLength* (Seitenlänge einer Zelle) des Types *int* und *upperLeftCorner* (Origo der Hauptkachel) des Types *java.awt.Point* hinzu! [2]

d) Ändern Sie den Mauszeiger des *Boards* innerhalb der *initialise*-Methode zu einer Hand! Verwenden Sie dazu eine vordefinierte Konstante der Klasse *Cursor*! [2]

Hinweis: Zusätzlicher Aufruf einer statischen Methode der Klasse *Cursor* nötig.

e) Erstellen Sie die Methode *adjust* ohne Argument und ohne Rückgabewert! Bestimmen Sie darin die Kachelgröße und speichern Sie sie in einer lokalen Variablen passenden Types! [2]

f) Bestimmen Sie unter Zuhilfenahme einer statischen Methode der *Math*-Klasse das Minimum von *width* und *height* der Kachelgröße und speichern Sie es in einer lokalen Variablen! [2]

g) Errechnen Sie die Seitenlänge einer Zelle durch Division des Minimums durch die im *model*-Attribut gespeicherte Größe und weisen Sie sie dem Attribut *sideLength* zu! [2]

h) Bestimmen Sie die Position der Kachel (relativ zur Eltern-Komponente) und weisen Sie sie dem Attribut *upperLeftCorner* zu! [2]

i) Erzeugen und initialisieren Sie ein *Board*-Objekt in der *main*-Methode der Startklasse! [2]

j) Fügen Sie es dem Rahmenfenster hinzu und rufen Sie seine adjust-Methode auf! [2]

Hinweis: Die adjust-Methode darf erst nach dem Setzen der Rahmenfenstergröße aufgerufen werden, da diese intern zur Berechnung verwendet wird.

Ergebnis: Es sind keine sichtbaren Veränderungen zu vorher festzustellen.

Task 4: Graphics Context [20]

Zweck: Das Gitternetz sowie Kreuze und Kreise werden in Routinen gezeichnet.

a) Überschreiben Sie die von *JComponent* geerbte *paintComponent*-Methode der *Board*-Klasse und rufen Sie zunächst jene der Super-Klasse auf! [2]

b) Schachteln Sie anschließend zwei Schleifen, die *this.model.size* Durchläufe haben! [2]

c) Bestimmen Sie die *x*- und *y*-Koordinaten der zu zeichnenden Zelle und speichern Sie sie in lokalen *Integer*-Variablen! [2] Hinweis: Addieren Sie dafür zum Attribut *this.upperLeftCorner.x* bzw. *this.upperLeftCorner.y* ein Vielfaches der Standard-Zell-Seitenlänge (Multiplikation mit Schleifenlaufvariable)!

d) Verwenden Sie den als Argument an die Methode übergebenen Grafikkontext zum Zeichnen eines passenden Rechteckes! [2]

e) Erzeugen Sie ein *Rectangle*-Objekt, welches mit den *x*- und *y*-Koordinaten der Zelle sowie ihrer *sideLength* initialisiert wird! [2]

f) Bestimmen Sie via *Model* die aktuelle Zelle und speichern Sie sie in einer lokalen Variablen des Types *Cell*! [2]

g) Rufen Sie eine nun zu erstellende Methode namens *drawSign* auf, welcher drei Argumente folgender Typen zu übergeben sind: *Graphics, Rectangle, Player*! [2]

Hinweis: Letzteres kann über die Zelle bestimmt werden.

h) Führen Sie in der *drawSign*-Methode eine Fallunterscheidung anhand des *Player*-Argumentes als Kriterium durch! Rufen Sie in passender Weise eine *drawCross*- oder *drawNaught*-Methode auf, wobei an beide der Grafikkontext und das Rechteck als Argumente mitzugeben sind! [2]

i) Implementieren Sie die *drawCross*-Methode zum Zeichnen eines Kreuzes (zwei Linien)! [2]

j) Implementieren Sie die *drawNaught*-Methode, so dass ein Kreis gezeichnet wird! [2]

Zusatzaufgabe) Bauen Sie einen kleinen Abstand zwischen Zellgrenze und Kreuz bzw. Kreis ein! [2]

Ergebnis: Ein als Spielfeld dienendes Gitternetz sollte zu sehen sein.

Task 5: Event Handling [20]

Zweck: Mausereignisse werden in passender Weise verarbeitet.

a) Erstellen Sie eine Klasse namens *CellClicker*, welche von einer geeigneten Super-Klasse erbt, um Mausereignisse abfangen zu können! [2]

b) Geben Sie ihr zwei Attribute: *model* vom Typ *Model* und *view* vom Typ *Board*! Initialisieren Sie beide mittels an eine neu zu erstellende initialise-Methode übergebener Argumente! [2]

c) Überschreiben Sie die Methode, welche auf Mausklicks reagiert und speichern Sie die Mauskoordinaten in einer lokalen Variablen des Types *Point*! [2]

d) Definieren Sie außerdem zwei lokale Integer-Variablen *i* und *j* und initialisieren Sie beide jeweils mit dem Wert *-1*! [2]

e) Prüfen Sie, ob die *y*-Mauskoordinate >= jener des Attributes *upperLeftCorner* der *View* ist! Falls ja, so weisen Sie der entsprechenden lokalen Variablen folgenden Wert zu:

```
i = (p.y - this.view.upperLeftCorner.y) / this.view.sideLength;
```

Wiederholen Sie Gleiches für die *x*-Mauskoordinate mit [2]:

```
j = (p.x - this.view.upperLeftCorner.x) / this.view.sideLength;
```

f) Prüfen Sie, ob die Zellenindizes *i* und *j* jeweils im gültigen Bereich zwischen *0* und *this.model.size* liegen! Falls ja, so markieren Sie die Zelle mittels der *take*-Methode des *Models* als gesetzt! [2]

g) Veranlassen Sie ein Neuzeichnen der *View*! Geben Sie die aktuelle Zelle auf der Konsole aus (wofür die implementierte *toString*-Methode zur Anwendung kommt)! [2]

h) Prüfen Sie schließlich über die *isFinished*-Methode der *Model*-Klasse, ob das Spiel beendet ist! Geben Sie in diesem Falle mittels *JOptionPane* eine Meldung aus und beenden Sie die Anwendung! [2]

i) Erzeugen und initialisieren Sie ein *CellClicker*-Objekt in der *main*-Methode der Startklasse! [2]

j) Passen Sie die *initialise*-Methode der *Board*-Klasse so an, dass sie das *CellClicker*-Objekt als zweites Argument entgegennimmt! Weisen Sie es dem *Board* als *MouseListener* hinzu! [2]

Ergebnis: Das Spiel sollte nun voll funktionstüchtig sein und auf Mausklicks reagieren.

Viel Erfolg!