Aufgabenblatt 6

Kompetenzstufe 1 & Kompetenzstufe 2

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Freitag, 11.01.2019 13:00 Uhr** in TUWEL hoch.
- Zusätzlich müssen Sie in TUWEL ankreuzen, welche Aufgaben Sie gelöst haben und während der Übung präsentieren können.
- Ihre Programme müssen kompilierbar und ausführbar sein.
- Ändern Sie bitte **nicht** die **Dateinamen** und die **vorhandene Ordnerstruktur**.
- Bei manchen Aufgaben finden Sie Zusatzfragen. Diese Zusatzfragen beziehen sich thematisch auf das erstellte Programm. Sie müssen diese Zusatzfragen für gekreuzte Aufgaben in der Übung beantworten können. Sie können die Antworten dazu als Java-Kommentare in die Dateien schreiben.
- Verwenden Sie, falls nicht anders angegeben, für alle Ausgaben `System.out.println()` bzw. `System.out.print()`.
- Verwenden Sie für die Lösung der Aufgaben keine Aufrufe (Klassen) aus der Java-API, außer diese sind ausdrücklich erlaubt, wie z.B. die Klassen `StdDraw`, `String`, `Scanner` und `Math` oder Klassen, die in den Hinweisen zu den einzelnen Aufgaben aufscheinen.

In diesem Aufgabenblatt werden folgende Themen behandelt:

- Ablaufsteuerung von Anwendungen
- Methoden
- Ein- und zweidimensionale Arrays
- Rekursion

Aufgabe 1

Implementieren Sie folgende Aufgabenstellung:

- Bei dieser Aufgabe geht es um die Implementierung eines kleinen Zeichenprogramms *Mini-Paint*. In *Mini-Paint* können mit Mausklicks Linien gezeichnet werden, wodurch auch geschlossene Flächen entstehen können. Diese geschlossenen Flächen sollen mit verschiedenen Farben gefüllt werden können. Ein Grundgerüst, das den Zeichenbereich mit der Farbpalette zeichnet, ist in `main` vorgegeben (Abbildung 1a). Erweitern Sie `main` so, dass Sie ein

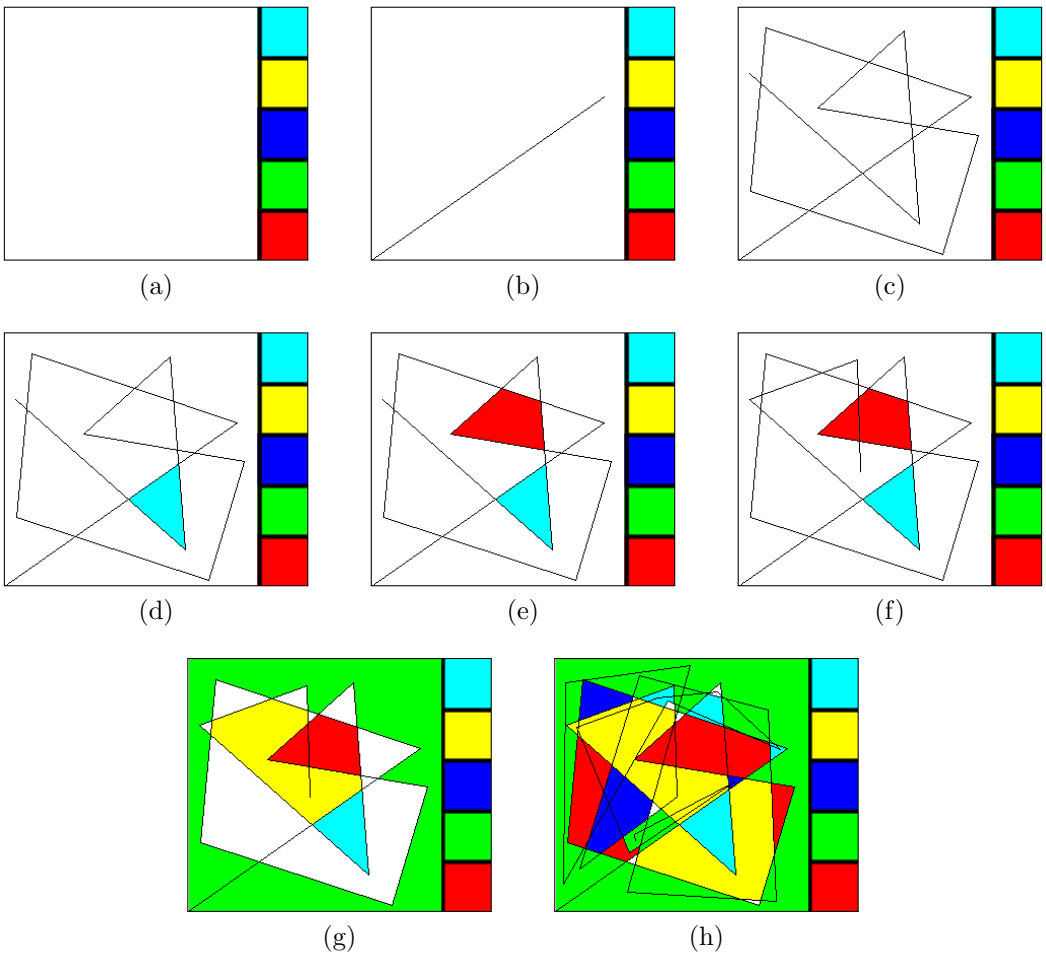


Abbildung 1: Mini-Paint Oberfläche in verschiedenen Zuständen.

funktionierendes Zeichenprogramm erhalten, das folgende Anforderungen erfüllt:

  - Der weiße Zeichenbereich soll eine Größe von 250×250 Pixel aufweisen.
  - Die Farbpalette am rechten Rand hat eine Größe von 50×250 Pixel, wobei jedes der Farbvierecke eine Größe von 50×50 Pixel hat.
  - Wird im Zeichenbereich mit der Maus geklickt, wird dieser Klick verarbeitet. Mit den Befehlen `StdDraw.mouseX()` und `StdDraw.mouseY()` können Sie die Koordinaten des Mausklicks auslesen. Es gibt zwei int-Arrays `xClick` und `yClick`, die die Koordinaten von zwei Mausklicks speichern. Zu Beginn sind alle Einträge 0 und dadurch wird

nach dem ersten Klick eine Linie vom Ursprung (0,0) bis zum Klickpunkt gezeichnet (Abbildung 1b). Um nach einem Mausklick eine Linie zu zeichnen, wird die Methode `paintLine(...)` aufgerufen. Danach kann erneut im Zeichenbereich geklickt werden, um eine weitere Linie zu zeichnen, die dann vom Ende der vorherigen Linie bis zum aktuellen Klickpunkt verläuft (Abbildung 1c). Bitte beachten Sie hier, dass in den Arrays (`xClick` und `yClick`) an Indexposition 0 die Koordinaten des vorangegangenen Mausklicks stehen und an Indexposition 1 die Koordinaten des aktuellen Mausklicks. Das Linienzeichnen kann beliebig oft wiederholt werden.

- Soll keine weitere Linie gezeichnet, sondern eine Fläche mit einer bestimmten Farbe gefüllt werden, dann muss auf eine der 5 Farbflächen geklickt werden. Dann wird keine Linie gezeichnet, sondern die Zeichenfarbe auf die entsprechende Farbe gesetzt und das Zeichenprogramm in den Zustand des *Flächenfüllens* versetzt. Sie müssen aufgrund der Klickkoordinaten unterscheiden, welche Farbe ausgewählt werden soll. Wenn nach Auswahl der Farbe auf den Zeichenbereich geklickt wird, dann wird nicht eine Linie gezeichnet, sondern vom entsprechenden Klickpunkt aus eine zusammenhängende Fläche mit der ausgewählten Farbe gefüllt (Abbildung 1d). Dazu wird die Methode `floodFill(...)` aufgerufen. Nach diesem Klick zum Füllen einer Fläche ist das Zeichenprogramm wieder im Zustand des *Linienzeichnens*. Dann können wieder im Zeichenbereich Linien gezeichnet werden und es wird vom Endpunkt der letzten Linie weiter gezeichnet. (Abbildung 1f). Soll erneut eine Fläche gefüllt werden, muss wieder zuerst eine Farbe ausgewählt werden. (Abbildung 1e).
- Eine bereits eingefärbte Fläche kann nicht neu gefüllt werden.
- Verwenden Sie den Befehl `StdDraw.pause(...)`, um eine Pause nach einem Klick zu machen, damit ein Mausklick nicht mehrfach gewertet wird.

(!) Es dürfen in `main` zusätzliche Variablen (auch außerhalb der `while`-Schleife) angelegt werden.

- Gegeben ist eine Methode `paintLine`:

```
void paintLine(int[][] picArray, int[] xClick, int[] yClick)
```

Diese bereits vorgegebene Methode nimmt ein zweidimensionales ganzzahliges Array `picArray` entgegen, das das aktuelle Bild mit allen gezeichneten Linien und gefüllten Farbflächen enthält. Angelehnt an das StdDraw-Fenster befindet sich der Ursprung (0,0) von `picArray` ebenfalls links unten. Die Parameter `xClick` und `yClick` beinhalten die Koordinaten von zwei Mausklicks, die für das Zeichnen einer Linie benötigt werden. Es wird eine Linie von (`xClick[0]`,`yClick[0]`) bis (`xClick[1]`,`yClick[1]`) gezeichnet und alle Punkte (Pixel) zwischen diesen beiden Punkten schwarz gezeichnet und an der entsprechenden Stelle des `picArray` mit dem Wert 1 eingetragen. Es können auch über gefüllte Farbflächen Linien gezeichnet werden.

- Implementieren Sie eine rekursive Methode `floodFill`:

```
void floodFill(int[][] picArray, int sx, int sy)
```

Diese Methode bekommt ein zweidimensionales ganzzahliges Array `picArray` übergeben, das das aktuelle Bild mit allen gezeichneten Linien und gefüllten Farbflächen enthält. Die

Parameter `sx` und `sy` entsprechen beim ersten Aufruf den Koordinaten des Mausklicks. Ausgehend von den Koordinaten des Mausklicks wird in alle 4 Richtungen (oben, unten, links, rechts) rekursiv weiter gesucht, ob noch ein weißer Wert vorhanden ist. Sollte das der Fall sein, dann wird dieses Pixel mit der gesetzten Farbe eingefärbt (StdDraw-Fenster) und die entsprechende Koordinate im `picArray` als *bemalt* (Wert wird auf 1 gesetzt) gekennzeichnet. Danach werden wieder die Nachbarn von diesem Pixel betrachtet. So arbeitet sich der Algorithmus in alle 4 Richtungen weiter, bis dieser auf Grenzen trifft. Diese Grenzen können gezeichnete Linien bzw. die Grenzen des Zeichenfensters sein (Abbildung 1g oder 1h).

Aufgabe 2

Implementieren Sie folgende Aufgabenstellung:

- Bei dieser Aufgabe geht es um die Implementierung des Spiels *Vier Gewinnt*. Dazu ist ein Teil der Spiellogik in `main` bereits vorgegeben. Erweitern Sie `main` so, dass Sie nach Implementierung und Verwendung der folgenden Methoden ein lauffähiges Spiel erhalten. Folgende Hinweise sollen Ihnen bei der Implementierung von `main` helfen:
  - Geben Sie Hinweise über Spielstatus und welcher Spieler an der Reihe ist immer mittels `System.out.println(...)` auf der Konsole aus.
  - Für Textausgaben auf dem StdDraw-Fenster wurde der Font `Font("Arial", Font.BOLD, 30)` gewählt.
  - Die in Abbildung 2 gezeigten Textfarben sind `StdDraw.GREEN` und `StdDraw.PRINCETON_ORANGE`.
  - Verwenden Sie den Befehl `StdDraw.pause(...)`, um eine Pause nach einem Spielzug zu machen, damit ein Mausklick nicht mehrfach gewertet wird.
  - Mit den Befehlen `StdDraw.mouseX()` und `StdDraw.mouseY()` können Sie die Koordinaten des Mausklicks auslesen.
- Implementieren Sie eine Methode `genGameBoard`:

  ```
  int[][] genGameBoard(int x, int y)
  ```

  Diese Methode erzeugt ein zweidimensionales int-Array mit `y` Zeilen und `x` Spalten.
- Implementieren Sie eine Methode `drawGameBoard`:

  ```
  void drawGameBoard(int[][] currentGameBoard, int oneSquareSize)
  ```

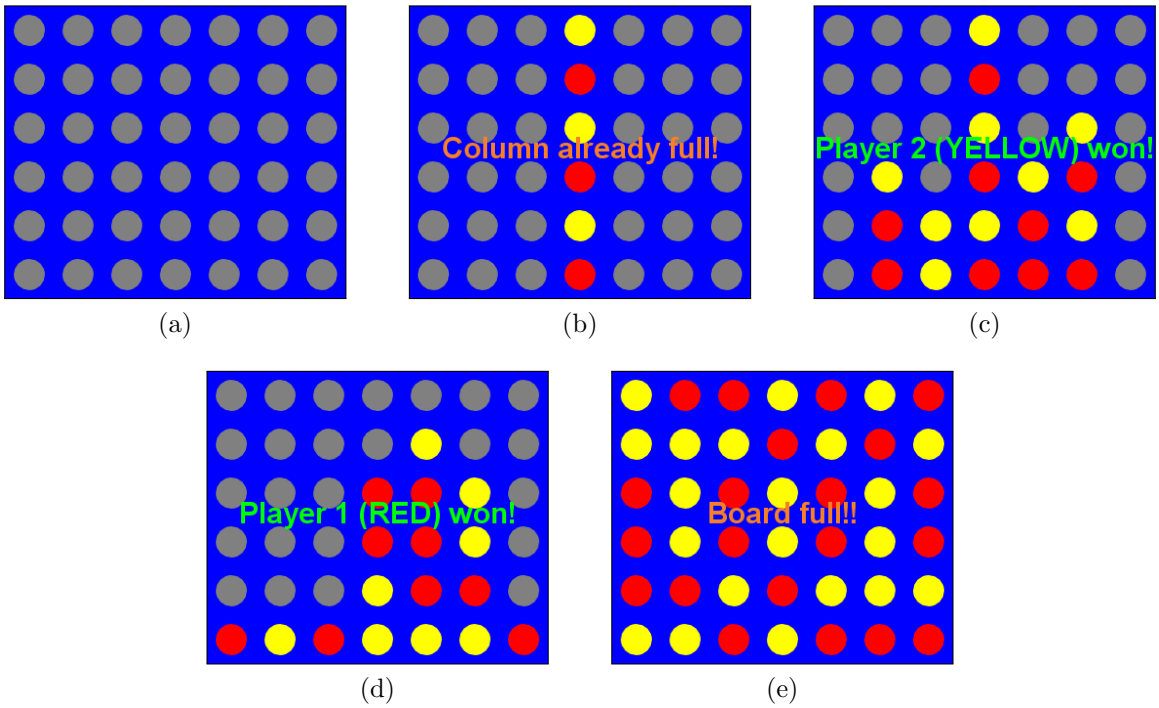
  Diese Methode zeichnet ein Spielbrett (Abbildung 2a) und nimmt dazu ein zweidimensionales int-Array entgegen. Der Parameter `oneSquareSize` gibt vor, wie viel Platz ein Spielstein einnimmt. In unserem Beispiel ist `oneSquareSize = 50;` eingestellt. Der Radius der Kreise ist ein Drittel von `oneSquareSize`.

(!) Player 1 wird im Array mit 1 gespeichert und wird mit der Farbe *rot* gezeichnet. Player 2 wird als 2 im Array abgelegt und durch die Farbe *gelb* repräsentiert.

- Implementieren Sie eine Methode `move`:

  ```
  boolean move(int[][] currentGameBoard, int player, int col)
  ```

  Diese Methode führt nach Klick eines Spielers auf die gewünschte Spalte im Spielfeld einen Spielzug durch. Dazu wird das aktuelle Spielbrett (`currentGameBoard`) übergeben. Zusätzlich wird mit dem Parameter `player` der Methode mitgeteilt, welcher Spieler den Zug durchführt. Ein weiterer Parameter `col` gibt die Spalte an, für die der Spielzug durchgeführt wird. Die Methode prüft, ob die entsprechende Spalte noch einen freien Platz hat. Wenn ja, dann wird für den Spieler `player` ein Eintrag im Array gemacht und `true` zurückgegeben. Falls nein, dann wird `false` zurückgegeben, in `main` eine entsprechende Meldung (Abbildung 2b) ausgegeben und derselbe Spieler darf nochmals einen Spielzug durchführen.



(a) (b) (c) (d) (e)

Abbildung 2: Spielbrett von *Vier Gewinnt* mit den verschiedenen Spielzuständen.

- Implementieren Sie eine Methode `checkGameStatus`:

```
boolean checkGameStatus(int[][] currentGameBoard, int player)
```

  Die Methode überprüft nach einem Spielzug mit `move`, ob der neu hinzugefügte Stein zu einem Sieg führt. Dazu bekommt die Methode das aktuelle Spielbrett (`currentGameBoard`) und den Spieler `player`, der den letzten Spielzug durchgeführt hat, übergeben. Die Methode überprüft nun, ob dieser Spieler vier Steine in einer Zeile, Spalte oder Diagonale hat. Wenn ja, dann gibt die Methode `true` zurück, ansonsten `false`. In `main` wird diese Information ausgewertet und falls es einen Gewinner gibt, angezeigt (Abbildung 2c und 2d). Falls es keinen Gewinner gibt und das Spielfeld noch nicht voll ist, dann darf der andere Spieler einen Spielzug durchführen. Sollte das Spielfeld voll sein und es keinen Gewinner geben, dann wird das Spiel beendet (Abbildung 2e). Für die Berechnung, ob das Spielfeld voll ist, können Sie die Variable `fieldsUsed` verwenden.