PTP - Aufgabe 1.2 und 1.3

Teilaufgabe 1.2

Erste Erweiterung und Refactoring: Einführung von **API-Methoden** und **automatischen Tests** für `Draw.java`:

1. Das Programm `Draw.java` und dazugehörige Klassen müssen ab sofort im `package mydraw` (ggf. mit Sub-packages) liegen.
2. Implementieren Sie (zumindest) folgende API-Methoden, um Zeichenbefehle aus dem Programm heraus (d.h. ohne Verwendung der Maus) aufrufen zu können:
   - Abfragen der Zeichenfarbe: `public String getFGColor()`
   - Setzen der Zeichenfarbe: `public void setFGColor(String new_color) throws ColorException`
     (Anm.: es gibt einen *festen Satz* von Farbwerten, die extern in Textform (*case-insensitive*) genutzt werden; `ColorException` ist eine selbstdefinierte Exception-Klasse → unbekannte Farbe)
   - Abfragen der Größe der *Zeichenfläche* (Pixel): `public int getWidth()` und `public int getHeight()`
   - Setzen der Größe der *Zeichenfläche* (Pixel): `public void setWidth(int width)` und `public void setHeight(int height)`
   - Setzen der Hintergrundfarbe: `public void setBGColor(String new_color) throws ColorException`
     (Anm.: für den Hintergrund sollte neben den 4 Vordergrundfarben zweckmäßiger Weise auch "White" zugelassen sein)
   - Abfragen der Hintergrundfarbe: `public String getBGColor()`
   - Zeichnen eines Rechtecks: `public void drawRectangle(Point upper_left, Point lower_right)`
   - Zeichnen eines Ovals: `public void drawOval(Point upper_left, Point lower_right)`
   - Zeichnen eines Polygonzugs: `public void drawPolyLine(java.util.List<Point> points)`
   - Abfragen des internen Images: `public Image getDrawing()`
   - Löschen der Zeichenfläche (aktuelle Hintergrundfarbe): `public void clear()`
   - Schreiben Sie eine weitere API-Methode `public void autoDraw()`, die mit Hilfe der obigen API-Methoden ein Bild zeichnet. Dabei müssen natürlich alle "malenden" API-Methoden und Farben *mindestens* jeweils ein Mal benutzt werden. (Anm.: Ein neuer *Button* "Auto" für den interaktiven Test ist hilfreich...; 'Speichern' ist nicht Aufgabe von *autoDraw()*)
   - Schreiben Sie eine Methode `public void writeImage(Image img, String filename)throws IOException`, um die aktuelle Zeichnung (die mit `Image getDrawing()` erfragt werden kann) abzuspeichern. Analog dazu ist eine Methode `public Image readImage(String filename) throws IOException` anzulegen, die ein extern gespeichertes Bild einliest und als `Image` anzeigt. Die Hilfsklasse `MyBMPFile.java` *kann* verwendet werden, um das Bild im BMP-Format (Pixel-Format) zu schreiben und wieder einzulesen; eigene IO-Hilfsimplementationen, die gleichartig mit dem BMP-Format umgehen, sind erlaubt.

Als Erleichterung sind die Signaturen als Rumpftext vorhanden und müssen nur in *Draw.java* eingebaut werden. Es soll möglichst wenig Implementation *direkt* in den API-Methoden vorgenommen werden.

3. Kontrollieren Sie mit einem externen Bildanzeigeprogramm, dass das mittels `autoDraw()` erzeugte (und dann gezielt abgespeicherte) Bild korrekt ist (korrekte Größe, Farben und Lage der Objekte). Dieses Bild wird nun als *Referenzbild* (read-only) für die weiteren Tests verwendet.
4. Benutzen Sie die API-Methoden und JUnit5, um automatische Unit Tests zu implementieren:
   Schreiben Sie eine Klasse `DrawTest` (im (Sub-) `package mydraw.test`), darin
   - testen Sie die `set...` und `get...`-Methoden
   - Schreiben Sie einen Test, der mittels `autoDraw(), getDrawing(), writeImage(...)` und `readImage(...)` das aktuell erzeugte Bild mit dem vorab geprüften, eingelesenen *Referenzbild* vergleicht und gegebenenfalls eine Fehlermeldung ausgibt.
   - Tests sollen positive und negative Fälle abdecken, ebenso Standardwerte und ggf. *exceptions*.
5. Verständnisfrage: Wodurch unterscheidet sich die Hilfsklasse `MyBMPFile.java` von der Hilfsklasse `BMPFile.java`, welche Auswirkung hat das?

Hinweise:

- Die o.g. API-Methoden sind *alle* in der Klasse `Draw` enthalten (kein *interface* o.ä.).
- `Draw` hat eine Doppelfunktion: es stellt die Schnittstelle zu Tests dar und ist gleichzeitig die Hauptklasse zum Starten der interaktiven Anwendung!
- Die interaktive Mal-Funktionalität muss erhalten bleiben!
- Die Signaturen der Methoden sind einzuhalten!
- Es geht in dieser Teilaufgabe noch nicht um eine "perfekte" Anwendung, sondern um Grundfunktionalität. Diverse Probleme werden später bereinigt!
- Eine mögliche Vorgehensweise "*refactored*" die Klasse *Draw* derart, dass die interne Verschachtelung aufgelöst, d.h. die inneren Klassen aus *Draw* herausgezogen werden. Das bedeutet zunächst aber auch erhöhten Aufwand und kann in einem späteren Schritt erfolgen. Empfehlung: vorerst Konzentration auf Funktionalität !
- Nicht-triviale Herausforderungen stellen u.a. das Aktualisieren bzw. Wiederherstellen der Zeichenfläche nach unterschiedlichen Störungen sowie das Abspeichern des Bildes dar, da der Bildspeicher nicht direkt für I/O zugreifbar ist. Informieren Sie sich über die Ideen/Techniken des Bildneuaufbaus und des *'double buffering'* (vgl. Partl und Painting in AWT and Swing), wobei letztere ein Flackern der Anzeige bei aufwändigen Grafikdarstellungen vermeiden soll. Die dahinter steckende Grundidee/dieser Ansatz, das Bild in einem Hintergrundpuffer aufzubauen und dann komplett in den Vordergrund zu kopieren, *kann* auch zur Behandlung von internen Images benutzt werden - hier aber nicht, um Flackern zu verhindern. Insbesondere die Kombination mit dem interaktiven Malen per Maus führt aber zu komplexen Abhängigkeiten zwischen Vorder- und Hintergrundbild. Die o.g. Herausforderungen werden später elegant adressiert, müssen also vorerst nur insoweit bearbeitet werden, wie sie für die Umsetzung der Vorgaben aus Aufg. 1.2 relevant sind.

- In den Teilaufgaben 1.4+1.5 wird dann durchgeführt:

    - weiteres **Refactoring** von `Draw.java`

        - **Erweiterung** von `Draw.java`

- Alle weiteren Schritte erfolgen nun unter Kontrolle der Tests!
  Details in der nächsten Woche!

Teilaufg. 1.3 - Vorbereitung für den Projektteil:

Bilden Sie 2er-Teams zur Bearbeitung des Projektthemas! Wählen Sie sich ein **Projektthema** (im 2er-Team) und erstellen Sie eine Grobbeschreibung und eine erste, kurze Anforderungsdefinition!

Abgabetermine:

Teilaufgabe 1.2	**Mittwoch, 28.4.21 / Donnerstag, 29.4.21 (am Ende des Tages)**
Teilaufgabe 1.3 / Projektbeschreibung	**Mittwoch, 28.4.21 / Donnerstag, 29.4.21 (am Ende des Tages)**

Die Termine sind als jeweils *letztmögliche* Termine zu sehen. Eine frühere Abgabe ist durchaus erwünscht, um für den Projektteil mehr Zeit zu haben!

Kontakt:

Andreas Heymann
Email: ptp@informatik.uni-hamburg.de