Praktikum zum Programmierkurs (WI), Wintersemester 2021/2022

Aufgabe 8 (Mehr IO mit Dateien, Datenströme)

Hinweis zu Bonuspunkten: *Zur Vergabe der Bonuspunkte werden wir in regelmäßigen Abständen den von Ihnen produzierten Quellcode begutachten. Informationen darüber, wann wir welche Praktikumsaufgaben begutachten, wie viele Bonuspunkte es gibt und wie Sie Ihren Quellcodestand einreichen, werden rechtzeitig bekanntgegeben.*

Hinweis zum entstehenden Code: *Verwenden Sie für den entstehenden Code das in Praktikum 1 in GitLab angelegte Projekt.*

1. Bilddateien mit Metadaten einlesen

Bisher haben wir unsere `Foto`-Objekte mit den zugehörigen `FotoMetadaten` nur "gemockt", d.h. wir haben die entsprechenden Daten manuell vorgegeben, ohne echte Bilddateien zu verwenden. Wir erweitern unsere "Foto-App" jetzt dahingehend, dass jedes `Foto`-Objekt mit einer Bilddatei verbunden ist, und dass die `FotoMetadaten` automatisch aus dieser Datei gelesen werden. Für das Einlesen der Metadaten verwenden wir die Open-Source-Bibliothek *metadata-extractor*, mit welcher sich Metadaten verschiedenster Formate aus Mediendateien extrahieren lassen (Beispiele von Metadatenformaten für Bilder sind Exif, IPTC und XMP).

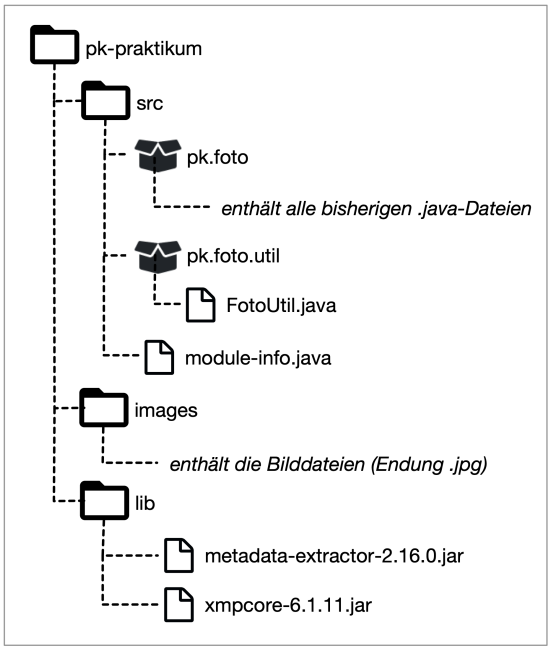
Gehen Sie wie folgt vor:

Schritt 1: Einbinden benötigter Dateien

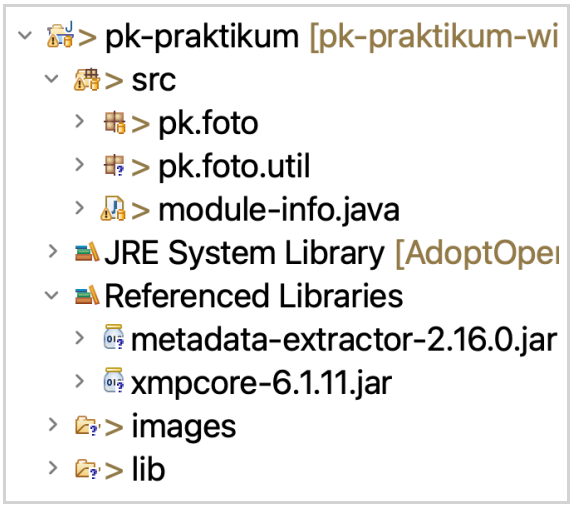
1. Laden Sie die Datei "praktikum08-dateien.zip" aus ILIAS herunter und entpacken Sie diese. Das ZIP-Archiv enthält folgende Verzeichnisse:
   - `images`: Enthält einige Bilddateien, mit welchen Sie im Folgenden experimentieren können.

     Hinweis: Sie können auch eigene JPG-Bilddateien verwenden, sofern diese saubere Exif-Metadaten besitzen. Für eigene Fotos, die Sie z.B. mit dem Smartphone erstellt haben, ist dies meistens der Fall.
   - `lib`: Enthält die benötigten Dateien für die Bibliothek *metadata-extractor*.
   - `src`: Enthält die Hilfsklasse `FotoUtil.java`, welche Ihnen die Benutzung der Bibliothek vereinfacht.
2. Kopieren Sie Inhalte des ZIP-Archives "praktikum08-dateien.zip" in Ihr Java-Projekt. Die Struktur Ihres

Projektes sollte danach folgendermaßen aussehen:



3. Damit die Bibliothek *metadata-extractor* nutzbar ist, müssen Sie diese in Ihrem Java-Projekt einbinden. In Eclipse erreichen Sie dies über einen Rechtsklick auf die beiden Dateien im `lib`-Verzeichnis (Dateiendung ".jar"). Wählen Sie dann im Menüpunkt *Build Path* die Aktion *Add to Build Path*. Die beiden JAR-Dateien sollten nun in Ihrem Projekt unter *Referenced Libraries* angezeigt werden:



4. Passen Sie die Datei `module-info.java` folgendermaßen an, damit Sie die Bibliothek *metadata-extractor* verwenden können:

```
module pk {
    requires java.desktop;

    requires metadata.extractor;
    requires xmpcore;
}
```

Hinweis: Wenn Sie die Punkte 1-4 durchgeführt haben, sollte die Klasse `FotoUtil` ohne Fehler

kompilieren. Ggf. müssen Sie noch den/einen Konstruktor der Klasse `FotoMetadaten` ergänzen bzw. anpassen, damit `FotoUtil` kompilierbar ist.

Schritt 2: Implementierung

1. Erweitern Sie die Methode `addFoto` der Klasse `Album`, so dass für das gegebene `Foto`-Objekt die `FotoMetadaten` eingelesen und gesetzt werden, bevor das `Foto`-Objekt zum Album hinzugefügt wird. Nutzen Sie zum Einlesen der Metadaten die Methode `readMetadata` der Hilfsklasse `FotoUtil`.

2. Jegliche Exceptions, die in der Methode `addFoto` der Klasse `Album` auftreten können, sollen in eine eigene Exception `FotoMetadatenException` verpackt und weitergeworfen werden. Implementieren Sie diese Exception selbst.

3. Wurde das Hinzufügen über das Menü ausgelöst, so soll den BenutzerInnen beim Auftreten einer `FotoMetadatenException` ein entsprechender Hinweis als `JOptionPane` (analog zu Praktikum 6) angezeigt werden. Nach Bestätigung dieses Hinweises wird wieder das Menü angezeigt.

4. Erweitern Sie das Menü, so dass beim Hinzufügen eines Fotos überprüft wird, ob die BenutzerIn den Namen einer existierenden Datei eingegeben hat. Existiert keine Datei mit dem entsprechenden Namen, so soll die BenutzerIn über eine `JOptionPane` aufgefordet werden, den Namen einer existierenden Datei einzugeben. Nach Bestätigung dieses Hinweises muss die Eingabe erneut vorgenommen werden.

5. Passen Sie ggf. die Beispieldaten in der `Tester`-Klasse an, so dass die `FotoMetadaten`-Objekte nicht mehr manuell erstellt werden.

2. Ausgaben auf Datenströme umbauen

Ändern Sie die Methode `drucke` in der Klasse `Fachobjekt`, so dass sie einen `OutputStream` als Parameter übergeben bekommt:

```
public abstract void drucke(OutputStream stream);
```

Passen Sie die Implementierungen in den abgeleiteten Klassen so an, dass die Ausgabe in den übergebenen Stream geschrieben wird.

Sichern Sie, dass alle Methoden Ihrer Anwendung, die `drucke` verwenden, nach der Änderung genauso funktionieren wie vorher. Insbesondere sollen die entsprechenden Ausgaben weiterhin auf die Konsole erfolgen.

3. CSV-Export auf Datenströme umbauen

Erstellen Sie von der Methode `exportiereEintraegeAlsCsv(File datei)` in der Klasse `FotoVerwaltung` eine Kopie `exportiereEintraegeAlsCsvNio(File datei)`, um den alten Stand zu sichern.

Bauen Sie die Methode `exportiereEintraegeAlsCsv(File datei)` dann so um, dass sie zum Schreiben der CSV-Datei nicht mehr `java.nio.file.Files`, sondern passende Datenströme verwendet. Verwenden Sie bei der Implementierung eine *try-with-resources*-Anweisung, um die entsprechenden Datenströme sauber zu schließen.

Alle anderen Bestandteile der Methode (Verwendung der Schnittstelle `CsvExportable`, Exception-Handling etc.) sollen beibehalten werden.

4. Commit und Push

1. Schreiben Sie den entstandenden Code per Commit in Ihrem lokalen Repository fest. Verwenden Sie als Commit-Message "Aufgabe 8: Mehr IO mit Dateien, Datenstroeme".
2. Bringen Sie die Änderungen dann per Push auf den GitLab-Server. Kontrollieren Sie in GitLab, dass Ihre Änderungen angekommen sind.