package utils;  
  
import model.Movie;  
import model.Person;  
import java.text.SimpleDateFormat;  
  
public class PrettyPrinter {  
 private static final SimpleDateFormat *dateFormat* = new SimpleDateFormat("yyyy-MM-dd");  
  
 public static String formatMovie(Movie movie) {  
 return String.*format*(  
 "Фильм #%d\nНазвание: %s\nКоординаты: %s\nДата создания: %s\n" +  
 "Оскары: %d\nКассовые сборы в США: %d\nДлина: %d мин\nЖанр: %s\nРежиссер:\n%s",  
 movie.getId(), movie.getName(), movie.getCoordinates(),  
 movie.getCreationDate(), movie.getOscarsCount(),  
 movie.getUsaBoxOffice(), movie.getLength(),  
 movie.getGenre(), *formatPerson*(movie.getDirector())  
 );  
 }  
  
 public static String formatPerson(Person person) {  
 return String.*format*(  
 " Имя: %s\n Дата рождения: %s\n Рост: %.2f м\n Вес: %s кг",  
 person.getName(),  
 person.getBirthday() != null ? *dateFormat*.format(person.getBirthday()) : "не указана",  
 person.getHeight(),  
 person.getWeight() != null ? person.getWeight().toString() : "не указан"  
 );  
 }  
}

package utils;  
  
import model.Movie;  
import java.util.Collection;  
  
public class CollectionUtils {  
 public static Movie findMinByOscars(Collection<Movie> movies) {  
 return movies.stream()  
 .min((m1, m2) -> Integer.*compare*(m1.getOscarsCount(), m2.getOscarsCount()))  
 .orElse(null);  
 }  
  
 public static long countByLengthLessThan(Collection<Movie> movies, int length) {  
 return movies.stream()  
 .filter(m -> m.getLength() < length)  
 .count();  
 }  
}

package validation;  
  
import model.Movie;  
  
public class MovieValidator {  
 public static void validate(Movie movie) {  
 if (movie == null) {  
 throw new IllegalArgumentException("Фильм не может быть null");  
 }  
 if (movie.getName() == null || movie.getName().isEmpty()) {  
 throw new IllegalArgumentException("Название фильма не может быть пустым");  
 }  
 CoordinatesValidator.*validate*(movie.getCoordinates());  
 PersonValidator.*validate*(movie.getDirector());  
 if (movie.getOscarsCount() <= 0) {  
 throw new IllegalArgumentException("Количество оскаров должно быть больше 0");  
 }  
 if (movie.getUsaBoxOffice() <= 0) {  
 throw new IllegalArgumentException("Кассовые сборы в США должны быть больше 0");  
 }  
 if (movie.getLength() == null || movie.getLength() <= 0) {  
 throw new IllegalArgumentException("Длина фильма должна быть больше 0");  
 }  
 if (movie.getGenre() == null) {  
 throw new IllegalArgumentException("Жанр не может быть null");  
 }  
 }  
}

package validation;  
  
import model.Coordinates;  
  
public class CoordinatesValidator {  
 public static void validate(Coordinates coords) {  
 if (coords == null) {  
 throw new IllegalArgumentException("Координаты не могут быть null");  
 }  
 if (coords.getX() == null) {  
 throw new IllegalArgumentException("Координата X не может быть null");  
 }  
 if (coords.getY() == null || coords.getY() <= -822) {  
 throw new IllegalArgumentException("Координата Y должна быть больше -822");  
 }  
 }  
}

package validation;  
  
import model.Person;  
  
public class PersonValidator {  
 public static void validate(Person person) {  
 if (person == null) {  
 throw new IllegalArgumentException("Person cannot be null");  
 }  
 if (person.getName() == null || person.getName().isEmpty()) {  
 throw new IllegalArgumentException("Person name cannot be empty");  
 }  
 if (person.getHeight() <= 0) {  
 throw new IllegalArgumentException("Height must be positive");  
 }  
 if (person.getWeight() != null && person.getWeight() <= 0) {  
 throw new IllegalArgumentException("Weight must be positive if not null");  
 }  
 }  
}

## Классы для считывания из файла

*/\*\*  
 \* Загружает коллекцию фильмов из файла.  
 \* При ошибке создаёт пустую коллекцию.  
 \*  
 \* @param fileName имя JSON-файла с данными  
 \*/*private void loadCollection(String fileName) {  
 try {  
 TreeSet<Movie> loaded = fileManager.loadFromFile(fileName);  
 loaded.forEach(collection::add);  
 System.*out*.println("Коллекция загружена. Элементов: " + loaded.size());  
 } catch (Exception e) {  
 System.*out*.println("Ошибка загрузки коллекции: " + e.getMessage());  
 System.*out*.println("Будет создана новая пустая коллекция.");  
 }  
}

*/\*\*  
 \* Загружает коллекцию фильмов из файла в формате JSON.  
 \*  
 \* @param fileName имя файла  
 \* @return коллекция фильмов в виде TreeSet  
 \* @throws IOException если произошла ошибка чтения  
 \*/*public TreeSet<Movie> loadFromFile(String fileName) throws IOException {  
 TreeSet<Movie> movies = new TreeSet<>();  
 File file = new File(fileName);  
  
 if (!file.exists()) {  
 return movies;  
 }  
  
 try (BufferedInputStream bis = new BufferedInputStream(new FileInputStream(file))) {  
 StringBuilder content = new StringBuilder();  
 int ch;  
 while ((ch = bis.read()) != -1) {  
 content.append((char) ch);  
 }  
  
 if (content.length() > 0) {  
 JSONArray jsonArray = new JSONArray(content.toString());  
 for (int i = 0; i < jsonArray.length(); i++) {  
 movies.add(parseMovieFromJson(jsonArray.getJSONObject(i)));  
 }  
 }  
 } catch (Exception e) {  
 throw new IOException("Ошибка при разборе JSON файла", e);  
 }  
 return movies;  
}

*/\*\*  
 \* Преобразует JSONObject в объект Movie при загрузке из файла.  
 \*  
 \* @param jsonMovie JSONObject, содержащий данные фильма  
 \* @return объект Movie  
 \* @throws Exception если произошла ошибка разбора данных  
 \*/* private Movie parseMovieFromJson(JSONObject jsonMovie) throws Exception {  
 Movie movie = new Movie();  
 movie.setName(jsonMovie.getString("name"));  
  
 JSONObject jsonCoords = jsonMovie.getJSONObject("coordinates");  
 Coordinates coords = new Coordinates();  
 coords.setX(jsonCoords.getDouble("x"));  
 coords.setY(jsonCoords.getInt("y"));  
 movie.setCoordinates(coords);  
  
 movie.setOscarsCount(jsonMovie.getInt("oscarsCount"));  
 movie.setUsaBoxOffice(jsonMovie.getInt("usaBoxOffice"));  
 movie.setLength(jsonMovie.getInt("length"));  
 movie.setGenre(MovieGenre.*valueOf*(jsonMovie.getString("genre")));  
  
 JSONObject jsonPerson = jsonMovie.getJSONObject("director");  
 Person director = new Person();  
 director.setName(jsonPerson.getString("name"));  
 if (jsonPerson.has("birthday") && !jsonPerson.isNull("birthday")) {  
 director.setBirthday(dateFormat.parse(jsonPerson.getString("birthday")));  
 }  
 director.setHeight((float) jsonPerson.getDouble("height"));  
 if (jsonPerson.has("weight") && !jsonPerson.isNull("weight")) {  
 director.setWeight(jsonPerson.getInt("weight"));  
 }  
 movie.setDirector(director);  
  
 return movie;  
 }  
}