# lab 7 submission

## source code

### Repository

This is the code to be tested.

```
package library.persistence;

import library.models.*;
import
org.eclipse.collections.api.block.function.primitive.BooleanFunctio
n;
import org.jetbrains.annotations.NotNull;
import org.mapdb.*;
import org.mapdb.serializer.SerializerArray;
import org.mapdb.serializer.SerializerArrayTuple;

import java.io.Closeable;
import java.util.function.Supplier;

public final class Repository implements Closeable {
  @NotNull
  final DB db;
  @NotNull
  final HTreeMap<User, User.Data> users;
  @NotNull
  final HTreeMap<Book, Book.Data> books;
  @NotNull
  final HTreeMap<User, String[]> userNotifications;
  @NotNull
  final BTreeMap<Object[], BookRequest.Data> userBookRequests; //
key: (User, BookRequest)
  @NotNull
  final BTreeMap<Object[], Borrow> borrows; // key: (User, Book)
  public final RepositoryUserOps userOps = new
RepositoryUserOps(this);
  public final RepositoryBookOps bookOps = new
RepositoryBookOps(this);
  public final RepositoryUserNotificationOps userNotificationOps =
```

```
new RepositoryUserNotificationOps(this);
  public final RepositoryBookRequestOps bookRequestOps = new
RepositoryBookRequestOps(this);
  public final RepositoryBorrowOps borrowOps = new
RepositoryBorrowOps(this);

  public Repository(@NotNull Supplier<DBMaker.Maker> dbMaker) {
    this.db = dbMaker.get().transactionEnable().make();

    final var userS = new User.S();
    final var userDataS = new User.Data.S();
    final var authorS = new Author.S(userS);

    final var bookRequestS = new BookRequest.S();
    final var bookRequestDataS = new BookRequest.Data.S();

    final var bookS = new Book.S(authorS);
    final var bookDataS = new Book.Data.S(bookS);
    final var borrowS = new Borrow.S();

    final var this2 = this;
    this.users = db.hashMap("users", userS,
userDataS).modificationListener(((key, _, newValue, _) -> {
      if (newValue == null) {
        final var key2 = new Object[]{key};
        this2.userNotifications.remove(key);
        this2.userBookRequests.prefixSubMap(key2).clear();
        this2.borrows.prefixSubMap(key2).clear();
      }
    })).createOrOpen();
    this.books = db.hashMap("books", bookS,
bookDataS).modificationListener((key, oldValue, newValue, _) -> {
      if (oldValue == null) {
        switch (newValue) {
          case Book.Data(_, _, _, final Book original, _) when
original ≠ null -> {
            switch (this2.books.get(original)) {
              case null -> throw new
IllegalStateException("Original book not found");
              case Book.Data(_, _, _, final Book
originalOrModified, _) when originalOrModified ≠ null -> throw new
IllegalStateException("Original book already linked");
              case Book.Data data -> this2.books.put(original,
data.withOriginalOrModified(key));
```

```
            }
          }
          case null, default -> {
          }
        }
      }
      if (newValue == null) {
        for (final var userBookBorrowKey : this2.borrows.getKeys())
{
          if (key.equals(userBookBorrowKey[1])) {
            this2.borrows.remove(userBookBorrowKey);
          }
        }
        switch (oldValue) {
          case Book.Data(_, _, _, final Book other, _) when other
≠ null -> {
            switch (this2.books.get(other)) {
              case null -> throw new IllegalStateException("Other
book not found");
              case Book.Data(_, _, _, final Book
originalOrModified, _) when !key.equals(originalOrModified) ->
throw new IllegalStateException("Other book wrongly linked");
              case Book.Data data -> this2.books.put(other,
data.withOriginalOrModified(null));
            }
          }
          case null, default -> {
          }
        }
      }
    }).createOrOpen();

    this.userNotifications = db.hashMap("userNotifications", userS,
new SerializerArray<>(Serializer.STRING)).valueLoader(_ -> new
String[0])
        .modificationListener((key, oldValue, newValue, _) -> {
          if (oldValue == null && newValue ≠ null) {
            if (!users.containsKey(key)) {
              throw new IllegalStateException("User not found");
            }
          }
        })
        .createOrOpen();
    this.userBookRequests = db.treeMap("userBookRequests", new
```

```
SerializerArrayTuple(userS, bookRequestS), bookRequestDataS)
        .modificationListener((key, oldValue, newValue, _) -> {
          if (oldValue == null && newValue != null) {
            @SuppressWarnings("SuspiciousMethodCalls") final var
containsUser = users.containsKey(key[0]);
            if (!containsUser) {
              throw new IllegalStateException("User not found");
            }
          }
        })
        .createOrOpen();
    this.borrows = db.treeMap("borrows", new
SerializerArrayTuple(userS, bookS), borrowS)
        .modificationListener((key, oldValue, newValue, _) -> {
          if (oldValue == null && newValue != null) {
            @SuppressWarnings("SuspiciousMethodCalls") final var
containsUser = users.containsKey(key[0]);
            if (!containsUser) {
              throw new IllegalStateException("User not found");
            }
            @SuppressWarnings("SuspiciousMethodCalls") final var
containsBook = books.containsKey(key[1]);
            if (!containsBook) {
              throw new IllegalStateException("Book not found");
            }
          }
        }).createOrOpen();
  }

  public void transact(@NotNull BooleanFunction<@NotNull
TransactData> action) throws TransactionException {
    try {
      var ok = false;
      try {
        ok = action.booleanValueOf(new TransactData(users, books,
userNotifications, userBookRequests, borrows));
      } finally {
        if (!ok) {
          db.rollback();
        }
      }
      if (ok) {
        try {
          db.commit();
```

```
            return;
          } catch (Exception exception) {
            db.rollback();
            throw exception;
          }
        }
      } catch (Exception exception) {
        throw new TransactionException(exception);
      }
      throw new TransactionException();
    }

    @Override
    public void close() {
      db.close();
    }

    public record TransactData(@NotNull HTreeMap<User, User.Data>
users,
                               @NotNull HTreeMap<Book, Book.Data>
books,
                               @NotNull HTreeMap<User, String[]>
userNotifications,
                               @NotNull BTreeMap<Object[],
BookRequest.Data> userBookRequests,
                               @NotNull BTreeMap<Object[], Borrow>
borrows
    ) {
    }
}
```

## RepositoryTest

This is the code to perform the testing.

```
package library.persistence;

import library.models.*;
import library.utils.ByteArray;
import org.jetbrains.annotations.NotNull;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mapdb.DBMaker;
```

```java
import java.time.Duration;
import java.util.Date;

import static org.junit.jupiter.api.Assertions.*;

class RepositoryTest {

  @BeforeEach
  void setUp() {

  }

  @AfterEach
  void tearDown() {
  }

  @Test
  void transact() {
    try (final var repository = new
Repository(DBMaker::memoryDirectDB)) {
      assertDoesNotThrow(() -> repository.transact(_ -> true));

      assertThrows(TransactionException.class, () ->
repository.transact(_ -> false));

      assertThrows(TransactionException.class, () ->
repository.transact(_ -> {
        throw new RuntimeException();
      }));

      assertThrows(Error.class, () -> repository.transact(_ -> {
        throw new Error();
      }));

      assertDoesNotThrow(() ->
repository.transact(RepositoryTest::populateRepository));

      repository.close();
      assertThrows(IllegalAccessError.class, () ->
repository.transact(RepositoryTest::populateRepository));
    }
  }
```

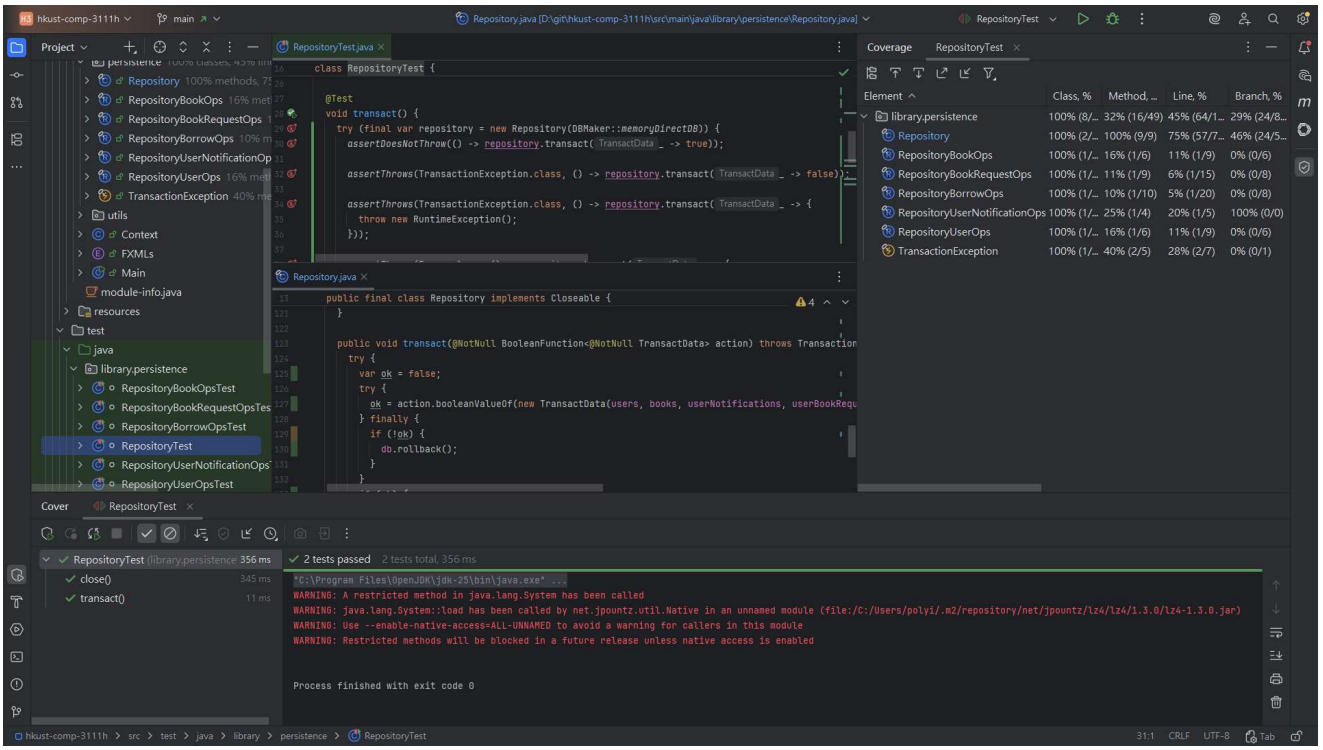
```
  @Test
  void close() {
    try (final var repository = new
Repository(DBMaker::memoryDirectDB)) {
      assertDoesNotThrow(() ->
repository.transact(RepositoryTest::populateRepository));

      repository.close();
      assertThrows(IllegalAccessError.class, () ->
repository.transact(RepositoryTest::populateRepository));
    }
  }

  public static boolean populateRepository(@NotNull
Repository.TransactData data) {
    final var reader = new User("reader");
    final var author = new User("author");
    final var librarian = new User("librarian");
    data.users().put(reader, new User.Data(User.Role.STUDENT_STAFF,
true, "reader", "reader"));
    data.users().put(author, new User.Data(User.Role.AUTHOR, true,
"author", "author"));
    data.users().put(librarian, new User.Data(User.Role.LIBRARIAN,
true, "librarian", "librarian"));

    final var book = new Book("book", new Author.ByRef(author));
    final var book2 = new Book("book", new
Author.ByName("author"));
    final var oldBook = new Book("book2", new
Author.ByRef(author));
    final var newBook = new Book("book2", new Author.ByRef(author),
true);
    data.books().put(book, new Book.Data("summary", "content",
Book.ApprovalStatus.APPROVED, null, 0));
    data.books().put(book2, new Book.Data("summary", "content",
Book.ApprovalStatus.REJECTED, null, 0));
    data.books().put(oldBook, new Book.Data("summary", "content",
Book.ApprovalStatus.APPROVED, null, 0)); // `originalOrModified`:
newBook
    data.books().put(newBook, new Book.Data("summary", "content",
Book.ApprovalStatus.PENDING, oldBook, 0));

    data.userNotifications().put(reader, new String[]
{"notification", "notification2"});
```

```
    data.userNotifications().put(author, new String[]
{"notification"});
    data.userNotifications().put(librarian, new String[]{});

    data.userBookRequests().put(new Object[]{reader, new
BookRequest("title", "author")}, new BookRequest.Data(new Date()));
    data.userBookRequests().put(new Object[]{author, new
BookRequest("title", "author")}, new BookRequest.Data(new Date()));
    data.userBookRequests().put(new Object[]{librarian, new
BookRequest("title", "author")}, new BookRequest.Data(new Date()));

    data.borrows().put(new Object[]{reader, book}, new Borrow(new
Date(), Duration.ofNanos(42), new ByteArray(new byte[42])));
    data.borrows().put(new Object[]{author, book2}, new Borrow(new
Date(), Duration.ofNanos(42), new ByteArray(new byte[42])));
    data.borrows().put(new Object[]{librarian, oldBook}, new
Borrow(new Date(), Duration.ofNanos(42), new ByteArray(new
byte[42])));

    return true;
  }
}
```

## test report and test coverage report

Both test report and test coverage report are shown.

Note that `library.persistence.RepositoryTest` is only intended to test `library.persistence.Repository` only. Please only care about the line coverage of that file and ignore the rest.


hkust-comp-3111h
main
Repository.java [D:\git\hkust-comp-3111h\src\main\java\library\persistence\Repository.java]
RepositoryTest
Project
RepositoryTest.java
class RepositoryTest {
@Test
void transact() {
try (final var repository = new Repository(DBMaker::memoryDirectDB)) {
assertDoesNotThrow(() -> repository.transact( TransactData _ -> true));
assertThrows(TransactionException.class, () -> repository.transact( TransactData _ -> false));
assertThrows(TransactionException.class, () -> repository.transact( TransactData _ -> {
throw new RuntimeException();
}));
Repository.java
public final class Repository implements Closeable {
}
public void transact(@NotNull BooleanFunction<@NotNull TransactData> action) throws Transaction
try {
var ok = false;
try {
ok = action.booleanValueOf(new TransactData(users, books, userNotifications, userBookRequ
} finally {
if (!ok) {
db.rollback();
}
}
Coverage
RepositoryTest
Element
Class, %
Method, ...
Line, %
Branch, %
library.persistence 100% (8/... 32% (16/49) 45% (64/1... 29% (24/8...
Repository 100% (2/... 100% (9/9) 75% (57/7... 46% (24/5...
RepositoryBookOps 100% (1/... 16% (1/6) 11% (1/9) 0% (0/6)
RepositoryBookRequestOps 100% (1/... 11% (1/9) 6% (1/15) 0% (0/8)
RepositoryBorrowOps 100% (1/... 10% (1/10) 5% (1/20) 0% (0/8)
RepositoryUserNotificationOps 100% (1/... 25% (1/4) 20% (1/5) 100% (0/0)
RepositoryUserOps 100% (1/... 16% (1/6) 11% (1/9) 0% (0/6)
TransactionException 100% (1/... 40% (2/5) 28% (2/7) 0% (0/1)
persistence
Repository 100% methods, 75
RepositoryBookOps 16% met
RepositoryBookRequestOps
RepositoryBorrowOps 10% m
RepositoryUserNotificationOp
RepositoryUserOps 16% met
TransactionException 40% me
utils
Context
FXMLs
Main
module-info.java
resources
test
java
library.persistence
RepositoryBookOpsTest
RepositoryBookRequestOpsTes
RepositoryBorrowOpsTest
RepositoryTest
RepositoryUserNotificationOps
RepositoryUserOpsTest
Cover
RepositoryTest
RepositoryTest (library.persistence 356 ms
close() 345 ms
transact() 11 ms
2 tests passed 2 tests total, 356 ms
"C:\Program Files\OpenJDK\jdk-25\bin\java.exe" ...
WARNING: A restricted method in java.lang.System has been called
WARNING: java.lang.System::load has been called by net.jpountz.util.Native in an unnamed module (file:/C:/Users/polyi/.m2/repository/net/jpountz/lz4/lz4/1.3.0/lz4-1.3.0.jar)
WARNING: Use --enable-native-access=ALL-UNNAMED to avoid a warning for callers in this module
WARNING: Restricted methods will be blocked in a future release unless native access is enabled
Process finished with exit code 0
hkust-comp-3111h > src > test > java > library > persistence > RepositoryTest
31:1 CRLF UTF-8 Tab