Software Engineering I (02161)
Persistency Layer of the Library Application

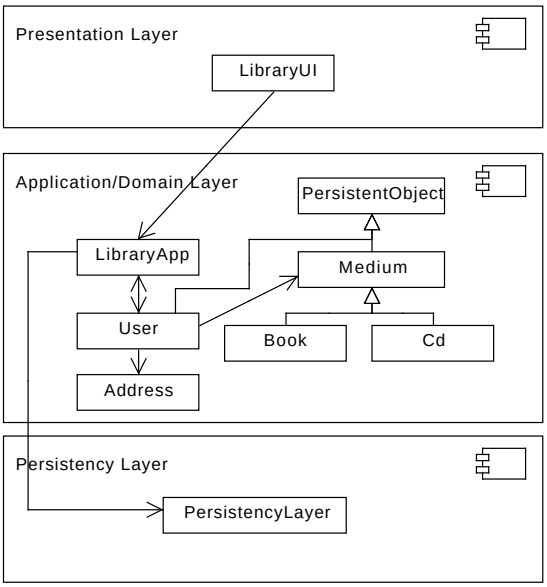
Assoc. Prof. Hubert Baumeister

Spring 2016

Layered Architecture: Persistency Layer for the library application

- The exercise presents a very simple persistency layer. Instead of using a database, the data is stored in a file. This is not a bad decision as such. If one has only relatively few data to store (e.g. several hundred users and media), then a file based database can be fast enough for most purposes.

We also use a very simple format to store the data in the files. These days, one would probably use XML to store the data; however, this would have required to understand XML and the Java class libraries that support XML reading and writing. The approach used in the exercise only needs knowledge of writing and reading files.



- Data is stored in two files `users.txt` & `media.txt`; address has no file of its own
- A book in media.txt

```
dtu.library.app.Book
b01
some book author
some book title
Mar 13, 2011
<empty line>
```

- A user in users.txt

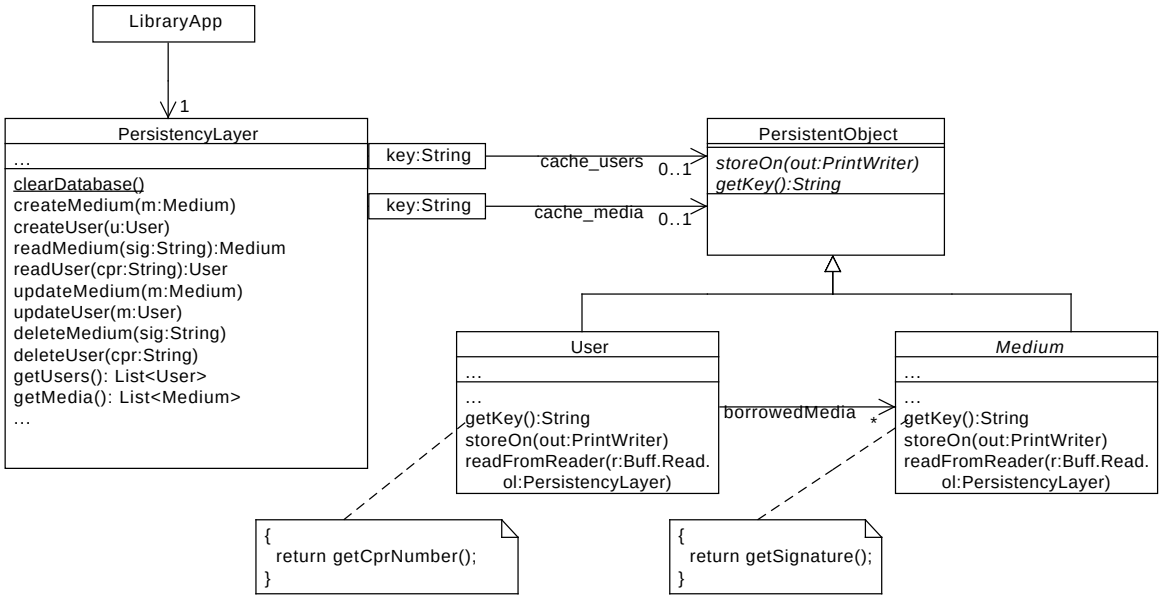
```
dtu.library.app.User
cpr-number
```

```
Some Name
a@b.dk
Kongevejen
2120
Hellerup
b01
c01
<empty line>
```

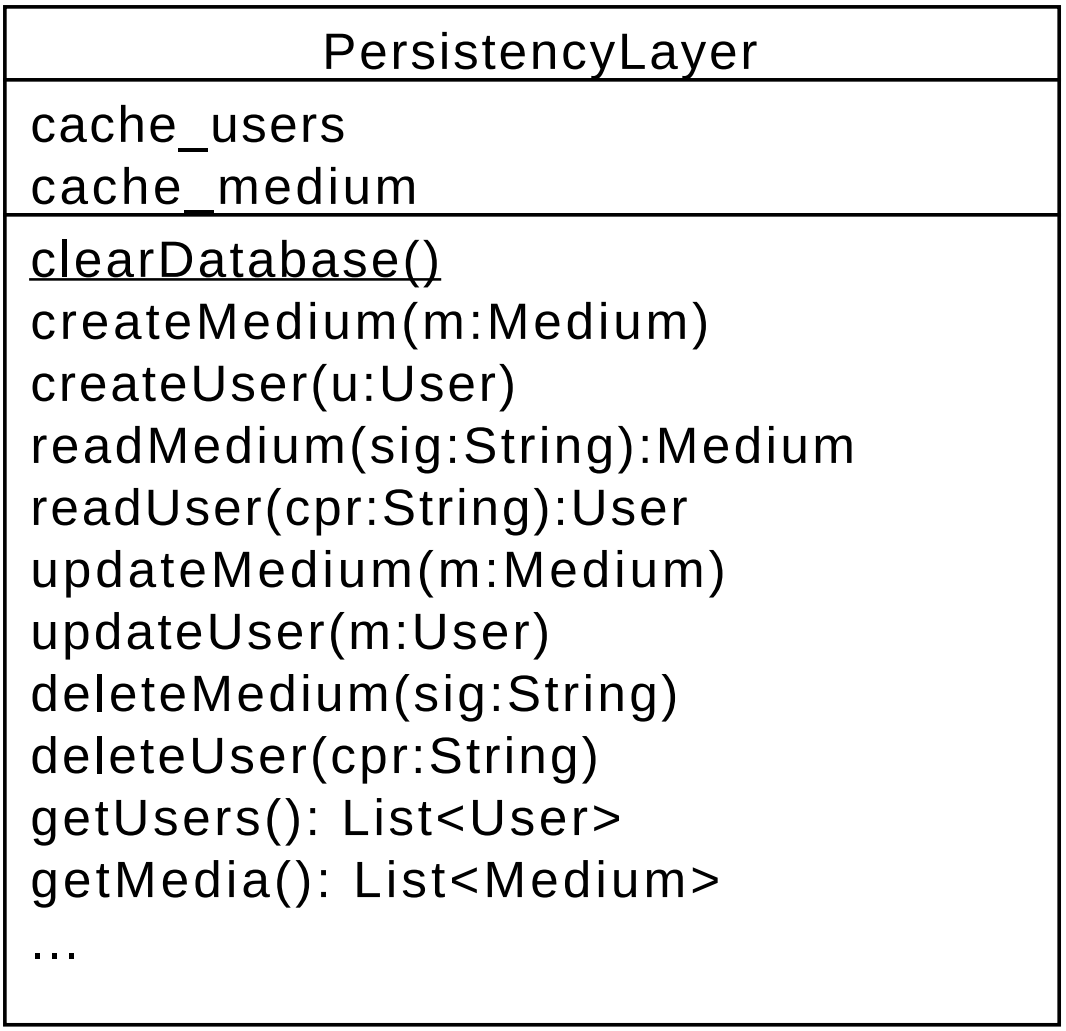
If the borrow date is not yet set, then the line should contain the string *null*; e.g.:

```
dtu.library.app.Cd
cd01
Phil Collins
some CD title
null
<empty line>
```

Persistency Layer



Layered Architecture: Persistency Layer for the library application

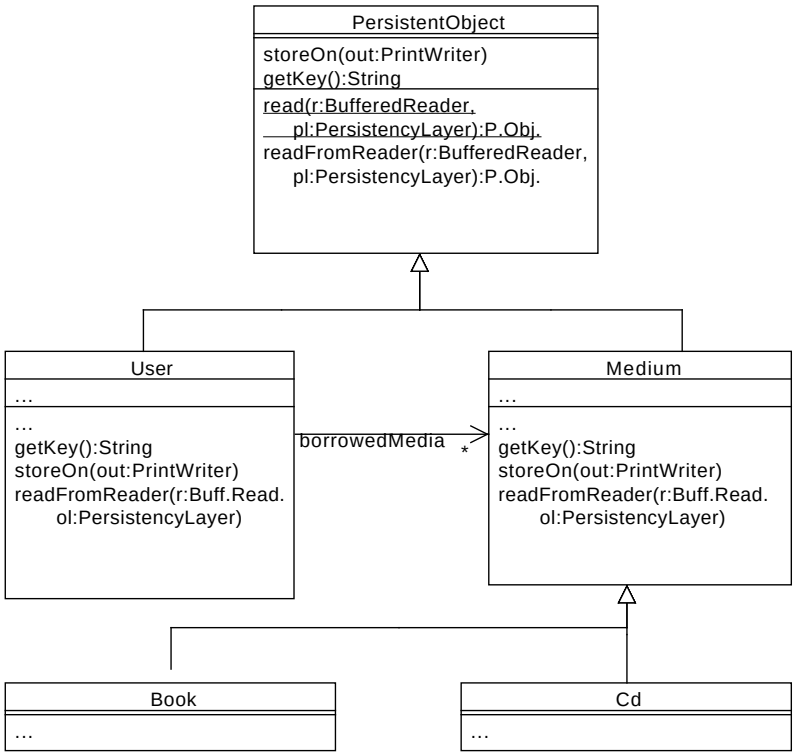


- CRUD (Create, Read, Update, Delete):
  - Typical database operations: Create, Read, Update, Delete a user or a medium
- clearDatabase:
  - removes the two files users.txt and media.txt
  - Gives a well defined start for all the tests that use the persistency layer
- createMedium/User:
  - appends a new record to the corresponding file
- readMedium/User:
  - go sequentially through the files, reads the object and returns it if the key matches
- updateMedium/User:
  - copy all entries in a new file; replace the old entry with the new entry on copying; rename the new file to the old file

- deleteMedium/User:
  - The same as updateMedium/User, the difference is that the object to delete is not copied
- getUsers/Media
  - Read all the users and media from the files into respective collections

The complexity of updateMedium/User and deleteMedium/User is approximately $O(n)$, where n is the number of users/media. In general, this complexity is not acceptable for and update and delete function (ideally, for databases, their performance is close to constant). However, the performance is okay for a few thousand entries, and again, we want to make a simple persistent layer, without using additional libraries.

Reading/Writing User and Media objects



- storeOn(PrintWriter writer) writes a representation of the object on a writer
- read(BufferedReader reader, PersistencyLayer pl) is a static method that creates a new object from a reader; it creates the object and delegates the initialisation to the object itself: i.e.
- readFromReader reads the state of an object from a reader
- Note that the user needs the PersistencyLayer to get from it the borrowed books based on their signatures

Reading User and Media objects

- Class PersistentObject

```
public class PersistentObject {
 ...
  public static PersistentObject read(BufferedReader in,
      PersistencyLayer pl) throws IOException {
    String type = in.readLine();
    PersistentObject po = null;
    if (type.equals("dtu.library.app.User")) {
      po = new User();
```

```
    } else if (type.equals("dtu.library.app.Book")) { ... }
    if (po != null) { po.readFromReader(pl, in);}
    return po.readFromRader(reader,pl);
  }
 ...
}
```

- Class User

```
public class User {
  ...
  public void readFromReader(PersistencyLayer pl, BufferedReader in)
      throws IOException {
    cprNumber = in.readLine();
    ...
    address = Address.readFrom(in);
    borrowedMedia = new ArrayList<Medium>();
    String signature = in.readLine();
    while (!signature.isEmpty()) {
      borrowedMedia.add(pl.readMedium(signature));
      signature = in.readLine();
    }
  }
}
```

Reading Objects

```
dtu.library.app.User
cpr-number
Some Name
a@b.dk
Kongevejen
2120
Hellerup
b01
c01
<empty line>
```

Use of Files

- Writing files

```
FileWriter fw = new FileWriter(filename, true);
                    // true = append; false = replace
PrintWriter out = new PrintWriter(fw);
out.println("Some line");
out.print("Some string without new lline");
```

- Reading files

```
FileReader fr = new FileReader(filename);
BufferedReader in = new BufferedReader(fr);
String line - in.readLine();
```

- Deleting and renaming files

```
File f = new File(filename);
f.delete();
f.renameTo(new File(new_filename));
```

Issues

- readMedium/User should return the *same object* if called twice with the same key

  → use a cache of media/users and return the object in the cache if it exists
  - The cache maps keys to persistent objects, i.e., `Map<String,PersistentObject> cache`
  - Note: the cache needs also to work together with the add and delete operations

- updateMedium/User needs to be called whenever the state of a user/media changes (e.g. through borrowing and returning media)

  → Don't forget to create tests for that

Issues: Object identity

```
PersistencyLayer pl = new PersistencyLayer();
User user1 = pl.readUser("12345");
User user2 = pl.readUser("12345");
assertNotSame(user1,user2) // Should be false
assertSame(user1,user2)    // Should be true
```

Solution: Qualified Associations / Maps

- Keep a map of cpr-number/medium signature to User/Medium

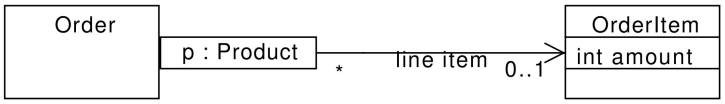
```
Map<String,PersistentObject> cacheUsers =
                        new HashMap()<String,PersistentObject>
Map<String,PersistentObject> cacheMedia =
                        new HashMap()<String,PersistentObject>
```

- First check if the object is already in the cache. If not, read it from the file and then add it to the cache. This ensures that only one User/Medium exists with the same cpr-number/signature.

```
public User readUser(String key) {
   if (cacheUsers.contains(key)) { return cacheUsers.get(key); }
   User user = readObjectFromFile(String key);
   if (user != null) { cacheUsers.put(key,user); }
   return user;
}
```

Qualified Assocations I

- A qualified association is an association, where an object is associated to another object via a qualifier (a third object)
- The interpretation is that to get to the order item, one has to provide an **order** *and* a qualifier, i.e. a **product**.
- UML Notation



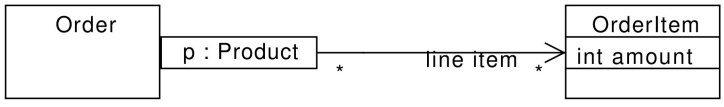
- An order is associated via a **product** to an order item called **list item**
  - This associations implies (with its multiplicity) that it is not possible to have two order items for the same product
- An order has for each product at most one list item

  → This is usually implemented by a **map** or **dictionary** mapping products to order items

```
public class Order {
   private Map<Product,OrderItem>
      listItem = new HashMap<Product,OrderItem>()
   ...
}
```

Qualified Associations II

- If the multiplicity is *, then several order items may be associated to a product



- Then the map has to return a collection for each product

```
public class Order {
   private Map<Product,Collection<OrderItem>>
      listItems = new HashMap<Product,Collection<OrderItem>>()
   ...
}
```

Map<K,V> Interface

- Map<K,V> is an interface describing maps (also called dictionaries) from objects of class K (*keys*) to objects of class V (*values*)
- A commonly used implementation of the map interface is *HashMap<K,V>* (implementation based on hash tables → Algorithm and Datastructures course)
- Most important operations
  - `m.containsKey(aK)` is true if there is a value to `aK` associated in `m`
  - `m.put(aK,aV)`: assigns the object `aV` to `aK`
  - `m.get(aK)`: retrieves the value stored under the key `aK`
  - Test 1 for maps: `m.put(aK,aV); assertTrue(m.containsKey(aK)); assertSame(aV,m.get(aK))`
  - Test 2 for maps: `assertFalse(m.containsKey(aK)); assertNull(m.get(aK));`
    - i.e. If the key is not in the map, `null` is returned
- The following are the basic properties of maps
  - It is possible to retrieve an object by the same key used to put he object in the map
    `m.put(aK,aV); assertTrue(m.containsKey(aK)); assertSame(aV,m.get(aK));`
  - If the object is not stored under a given key in the map, the get operation returns null
    `assertFalse(m.containsKey(aK)); assertNull(m.get(aK));`

Tasks:

1) Implement the persistency layer (tests provided)
2) Intergrate the persistency layer in the library application (tests have to be written)

Tests for the persistency layer

```
@Before
public void setUp() {
   PersistencyLayer.clearDatabase(); // Fresh database for each test
}

@Test
public void testAddUser() throws Exception {
   PersistencyLayer pl = new PersistencyLayer();
   Address address = new Address("Kongevejen", 2120, "Hellerup");
   User user = new User("cpr-number", "Some Name", "a@b.dk", address);
   pl.createUser(user);
   PersistencyLayer pl1 = new PersistencyLayer();
   User user1 = (User) pl1.readUser(user.getCprNumber());
   Utilities.compareUsers(user, user1);
}
```

Tests for the integration of application layer and persistency layer

```
@Before
public void setUp() throws Exception {
  libApp = new LibraryApp();
  PersistencyLayer.clearDatabase();
  libApp.adminLogin("adminadmin");
  Address address = new Address("Kongevejen", 2120, "Hellerupl");
  user = new User("cpr-number", "Some Name", "a@b.dk", address);
  libApp.register(user);
  b = new Book("b01", "some book title", "some book author");
  c = new Cd("c01", "some cd title", "some cd author");
  libApp.addMedium(b);
  libApp.addMedium(c);
}

@Test
public void testBorrowing() throws Exception {
  user.borrowMedium(b);
  user.borrowMedium(c);
  PersistencyLayer pl = new PersistencyLayer();
  User user1 = (User) pl.readUser(user.getCprNumber());
  assertEquals(2, user1.getBorrowedMedia().size());
  Utilities.compareUsers(user, user1);
}
```

Implementation in LibraryApp

```
public void borrowMedium(Medium medium) throws BorrowException {
  if (medium == null)
    return;
  if (borrowedMedia.size() >= 10) {
    throw new TooManyBooksException();
  }
  for (Medium mdm : borrowedMedia) {
    if (mdm.isOverdue()) {
      throw new HasOverdueMedia();
    }
  }
  medium.setBorrowDate(libApp.getDate());
  borrowedMedia.add(medium);
  try {
    libApp.getPersistencyLayer().updateUser(this);
  } catch (IOException e) {
    throw new Error(e);
  }
}
```