Waterfall Development

* Confirm requirements
* Design data flow
* Use case and Class UML diagrams
* Sketch gui
* Detailed design and development
* Organisation paradigm – directory structure – naming conventions pg20 – comment and file layout conentionspg 22-24
* What threads? Avoid multithreading in swing
* Document design choices as you go

Write code for an assessor or unknown co-worker, so that it is easier to use.

**Testing**

* unit testing: [www.junit.org](http://www.junit.org) pg16
* Test classes and methods with various kinds of input.
* Use people for application testing – user testing
* System testing – check that classes work together as a system.

**Documentation**

Pg17m pg 23

In code, beginning and javadoc comments pg23

Vava doc comments – p35 – tool parses your code looking for javadoc comments /\*\*….\*/

It then generates html documentation based on that.Makes use of tags using the @ symbol.

<http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html>

Javadoc comments go before the declaration of the item you are commenting.

Use the <code> tag for java keywords, class, package and interface names in javadoc comments <code>MyClass<code/>

Write documentation as you go

Document design choices as you go

Put design choices in comments near the actual code.

Include user documentation – manual

Programmer guide

**Packages – p44**

Collate classes of related functionality into packages i.e.

GUI

Networking

Database

**Logging – p51**

Keep log files small. Turn off logging to screen for production version

importjava.util.logging.\*;

import java.io.\*;

public class Main {

public static void main(String[] args) {

try{

Logger myLogger = Logger.getLogger("Test");

FileHandlermyFileHandler = new FileHandler("testLog.txt");

myFileHandler.setFormatter(new SimpleFormatter());

myLogger.addHandler(myFileHandler);

myLogger.severe("Oh shit");

myLogger.warning("Double shit");

}

catch(IOExceptionioe){}

}

}

It can be a good tool to pass a parameter to the logger when entering a method and then pass the variable to make sure it was assigned correctly.

**Synchronization**

Be wary of synchronized collections – their internal methods are synchronized so that those operations behave atomically but the object is not synchronized. In between calls to vector.lenght() and vector.get(i) another thread may have accessed the collection and changed the contents. If you make the method calling the vector methods get the vector lock then you are safe...but this only works if the other threads look for locks and obey thread safety.

Yeild occasionally during lengthy processes

Be frugal with synchronization - no need to synchronize methods that don’t use state information of a class or alter it – you must document your choice for synchronization. Immutable objects never need synchronization i.e Strings and Integers

Don’t synchronize a threads run method!

Use while loops to check conditions for calls to wait. – things may have changed since wait was last called.

**Patterns– design**

**NOTE – Have to impliment the interface but don’t have to use the methods!**

Use MVC

Controller – takes requests from view, packages and sends to model. Also receives data from model, packages it and presents it to the view. This is possibly a table class to help with assembling data ready for the view or receiving data to go to the model.

Use Façade to meet the interface requirements – use to link database and locking to a particular client. The Façade is actually our Model

Search will now send back an arrayList of Contractor objects representing the results of the search.

DBEngine class does all the work – parsing the file and returning the arrays and objects etc. The DBEngine Methods are called from the Database and ServerDatabaseclasses which are acting as a Façade – Database classes constructors must create a new DBEngine

ServerDatabase is serializable and is the object used in RMI – its methods are called from the gui controller class. The Controller class will hold variables to do with the locations of the databases and modes and make the choice as to which object it instantiates and uses. It will load last locations from config files.

Server launches and is only concerned with finding the database and connecting to the client. Client requests on the database must pass through the server and to the remote database class. The database class should be passed a file location (both server and direct) and have all the methods to access and manipulate the database. If the server is connected it will have a database object, if it is direct the controller class will have a database object. Only the server class will implement locking. It will have a locking manager that will query requests on records as they come through and lock or allow depending on the lock cookie registry that it has. Server will have a database object and a security object.

In database package will need access manager classes and db access classes to handle the locking.

JtableModel class holds an ArrayList of String Arrays representing the records of the database.

Have a GUI controller class that takes care of the searching retreiving and sending of dat and passing it between the database and the table. Table will have methods to add a class record object that the GUIcontrolerclas has received from the database into an array and into the table.

Have a launcher class that launches the application from its main method and takes in the command line parameters to determine which version is executed.

All data transfer between a view and a controller will be through a tablemodel object. Must keep a variable that refers to the current tables model so it can be updated. This is always updated to reflect changes in the databases state. Once the database is queried or updated by the controller the main window should use a method to replace the existing model with the updated version.

2 interfaces that implement the one given to us – a remote Database class and a local one

Use a value or transfer object to represent the client record.

Controller class will launch the server.

Use another interface to extend Both remote and the project interface, extend UnicastRemoteObject and then implement the other interfaces – pattern? Not façade the …adapter?

Saving settings - Use a cache populated at startup and written to at shutdown – can the .properties file be a serialized object? Use an object of type Config as a value/transfer object to save settings.

Lock manager will be passed a Contractor object which it will use to check the record number in its collection of reserved records.

ClientGui will generate a unique cookie on construction – this is passed into the lock manager which has a table of locked records ( it doesn’t need to know how many there are, just if they are locked or not) – if the record in the passed in contractor object is not locked, the executing code is allowed to access the file and update it. After updating code is executed notify all is called. If it is locked the code executes wait() WHICH OBJECTS ARE LOCKED FOR ACCESS – the DB file?

DatabaseWorker class will have a lock manager who controls the locking. The lock managers constructor will take a database RANDOMACCESS file which is the object to lock on.

Methods will accept a lock cookie which is passed in from up the call hierarchy from the client.

**GUI**

Modal Jdialog pops up when it runs. Which dialog is determined by the mode flag which is passed into the constructor for the main Window

Have main table and search and book options in different Jpanes

Have a launcher class

* Takes command line args
* Launches main window and passes args or
* Launches the server .server is like a stand alone program and must be reunning
* Prints a messafe to console asking for correct entry.

A main window class –

* maingui app
* jTable lives on a Jscrollpane.
* Has a connection type variable i.e. database or server
* Has a TableModel variable used to set and refresh the Jtable contents. The setup table method is used to refresh the jtable after searches etc
* Use a different popup to set the database location depending on mode! – pass this to controler who has overloaded methods who properly converts and instantiate a connector

A controller class

* Intermediary between view and database
* Methods return a new table model populated WITH SEARCH RESULTS ETC
* Actually makes the connection to the database – local or server. Has a method called by the MainWindow and receives the connection location data, then instantiates the required connection object and assigns it to the instance variable
* Organises data into format required by each side
* Holds an instance of the database class(concrete implementation of supplied interface) and calls the methods on it. This class calls the methods listed in the supplied interface and then packages them as required i.e. puts search results into a table model, before passing to the view.

for local mode Database selection window pops up first – is this a dialog? When the select button is pressed the dialog closes and the main window is launched. – config file is read into the dialog on construction displaying the last file used. For networked mode a network dialog is loaded(this is identical to the one loaded for the server – the server window extends this one and adds connections to the database.

Do I need a static variable somewhere that increments with each Contractor object created to keep track of record numbers and primary key? No – many objects may be generated for search results etc. no point.

**Networking**

– use RMI

No security manager and no dynamic class loading

Must generate own stubs and skeletons for both sides

How to loaunch Registry?

ServerDatabase class is Serializable and uses RMI – it has the same method calls as normal database but just adds the security manager and has a constructor that accepts some networking stuff.

**Config File**

* Database location
* Port number

Use a Config object and save the instance to the local directory and open again. If the file dodesn’texist the config is default, if it does then load it. Will need a config object for each version of the program – network and local.

**Database**

Object must be the same for both network and direct modes but treats security differently i.e. direct mode doesn’t lock records.

* Has a mode enum variable
* Will implement interface methods but also have more useful methods to do the same thing but pass value objects.
* Will have a LockManager object, which is only instantiated in network mode. LockManager implements all of the locking and checking on a record.

Have a DatabaseLocal and a DatabaseServer which extends DatabaseLocal and overides its methods to include the locking methods

Use a database connector class between the universal View Controller and either the server or the database object. Connectors will be a type that is switchable depending on the flag. There is a getDatabase method to return it to the controller so it can still call methods on the database object.

The server connector constructor takes a port. The object RMI is invoked on is shared between the server and the connector – it is a Database object.

The connector constructor gets a connection to the local db object (passing it a path), or to the server (passing path and port) the server version will have a db stub and getDB will return a reference to the stub.