Basic understanding after learning this project:

———————————————————————————————————

Class vs Interface

- Java is an Object-Oriented language.
- Class is a blueprint for an object.
- Interface is a 100% pure abstract class. So interface can not be instantiated. It has collection of method definitions(no implementation) and constant values.
- When a class implements an interface, it must provide a full definition for all the methods declared in the interface.
- A class can implement more than one interface by including several interface names in a comma-separated list of interface names.
- Interfaces provide a way of multiple inheritance in Java. If we use interfaces instead of concrete subclasses (or even abstract superclass types) as arguments and return types, we can pass anything that implements that interface.

Inner Classes

- They are good when they are not used outside the containing class.
- Non-static inner class can access all the private resources methods, properties of the top-level class
- Inner class will be used to perform some task over the outer class and returns the output. So it will ensure that the outer class object is created to access it.
- Inner class can be hidden from other classes in the same package.

User-defined Packages

- Packages name should begin with lowercase letters.
- Packages should be declared at the beginning of the statement in a Java source file.
- The steps for organizing code into the packages:
  - Create a directory under the source directory which store all the source files(.java files), and name it as the package name.
  - Store the listing as the classname.java files in the directory created.
  - Declare the package at the beginning of a file using the notation: *package packageName;*

- *Define the class that is to be put in the package and declare it public*

main() method

- When we need to test the real class, we use main( ) method.
- main( ) method is a static method, which means that we can launch or test another class, nearly always by instantiating a class in main and invoking a method on that new instance.
- We can do these things in the main( ) method: declaration, assignment, method calling, using loop structures(for/while) or other control structures(if/else).

StringBuilder vs StringBuffer vs String

- If we use String concatenation in a loop, we should use a StringBuilder(not StringBuffer) instead of a String, because it is much faster and consumes less memory.
- But if we have a single statement, we can use String, because the compiler will use StringBuilder automatically. e.g. String str = “Hello” + “World”.

Variable

- There is no "global" variables and methods in Java;
- If we modify some method "public" or "static", this method could be used everywhere, which sounds like "global";
- Something like constant "pi" or method "random()" are special cases, because they don't have multiple variables/objects.
- Package java.lang is sort of "pre-imported" for free.

Statics

- Static methods
  - When we invoking the static methods, we use the class name(Math.random( ), Math.min( )…)
  - They can not use non-static variables/methods
  - They can not use super/this
- Static variables
  - This kind of variables are same for all instances(one per class).
  - They should be instantiated before any object of that class can be created.

- If we do not instantiate them, there exists default value.(0/0.0 for numbers, false for booleans, null for references)

Public vs protected vs private

	scope	public	protected	default	private
1	**from within the object**	yes	yes	yes	yes
2	**from a class within the same package**	yes	yes	yes	no
3	**from a class outside of a package**	yes	no	**no**	no
4	**from a subclass located in the same package**	yes	yes	yes	no
5	**from a subclass located outside of the package.**	yes	yes	**no**	no
	For information hiding all variables should be declared private from here on forward.				
	final - final properties cannot be changed				
	final - final methods cannot be overridden.				
	static - properties/methods are global to objects of the same objecttypes.				

Return type

- Values passed in and out of methods can be implicitly promoted to a larger type or explicitly cast to a smaller type.

Encapsulation

- The cool thing about encapsulation is that we get to change our mind. And nobody gets hurt.
- REMEMBER: Make the instance variables private, and make the setter and getter public.
- Encapsulation is a safer way, through which the instance variables are not exposed to outside users, and we can do some change for our project later, without breaking anybody else's code.

Abstract method

- An abstract method has no body!
- If we declare an abstract method, we MUST mark the class abstract as well.
- The point of an abstract method is that even though we haven't put in any actual method code, we've still defined part of the protocol for a group of subtypes (subclasses).
- We MUST implement all abstract methods.

Object class

- Class Object is not abstract
- The Object class serves two main purposes:
  - to act as a polymorphic type for methods that need to work on any class that we or anyone else makes
  - to provide real method code that all objects in Java need at runtime
- Some of the most important methods in Object are related to threads.

Serializable

- If we want our class to be serializable, implement Serializable.
- Serialization is all or nothing.
- Mark an instance variable as transient if it can't (or shouldn't) be saved.
- Static variables are not serializable. Static means "one per class" not "one per object". Static variables are not saved, and when an object is deserialized, it will have whatever static variable its class currently has.
- The moral: don't make serializable objects dependent on a dynamically-changing static variable!

Exception Handling:

- There are two kinds of exceptions
  - **Checked:** are the exceptions that are checked at *compile time*. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using *throws* keyword (e.g. IOException, InterruptedException…).
  - **Unchecked** are the exceptions(*RuntimeException*) that are not checked at compiled time. It is up to the programmers to be civilized, and specify or catch the exceptions (*try/catch*). These are basically *programming errors* (e.g. NullPointerException, NumberFormatException, ClassCastException…).

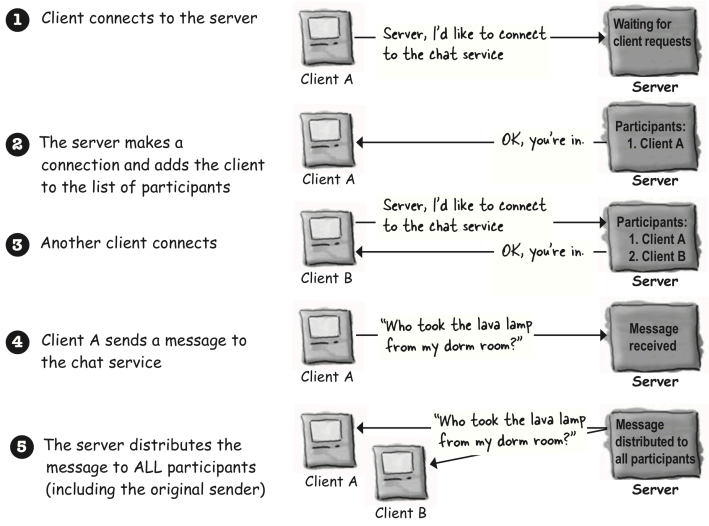
Extensibility:

- Extensibility is a software engineering and systems design principle where the implementation takes future growth into consideration. Extensions can be through the addition of new functionality or through modification of existing functionality.
- To achieve an extensible software, avoid traditional software development issues including **low cohesion** and **high coupling.**
- Extensibility imposes fewer and cleaner dependencies during development, as well as reduced coupling and more cohesive abstractions, plus **well defined interfaces.**
- **Interfaces are a way of decoupling separate software components.**

Cohesion & Coupling:

- **Cohesion** refers to what the class (or module) will do. Low cohesion would mean that the class does a great variety of actions and is not focused on what it should do.
- **coupling** refers to how related are two classes/modules and how dependent they are on each other. Being low coupling would mean that changing something major in one class should not affect the other.

Connection/Multithreading:



For the client

1.Establish the initial connection:

```
Make a Socket
Connect to IP address and TCP port
```

2.Send information to server:

```
PrintWriter(socket.getOutputStream( ) )
writer.println( )
```

3.Receive information from server:

```
InputStreamReader(socket.getInputStream( ) )
BufferedReader reader
reader.readLine( )
```

For the server

1.Make a socket and wait for the connection:

```
Make a ServerSocket:
    ServerSocket serverSocket
Make a new Socket:
    Socket sock = serverSocket.accept( )
```

Multithreading:

- Through multithreads we can do different things at the same time.
- To launch a new thread, we need to make a Runnable object, and give it to the new Thread object, then invoke run() method.
- Once the thread becomes runnable, it can move back and forth between runnable, running, and an additional state: temporarily not runnable(sleep( )).
- There would be unpredictable things happening and two threads run in disorder.

Synchronizing:

- Use the synchronized keyword to modify a method so that only one thread at a time can access it.
- Doing synchronizing can avoid data corruption.

Socket:

- Socket is used to connect the outside world, e.g. server socket and client socket.
- When establishing a socket, there are two things need to know: IP address and TCP port.
- The TCP port numbers from 0 to 1023 are reserved for well-known services. So we must choose a number between 1024 and 65535 for our own server programs.

HTTP:

- HTTP: Hyper Text Transfer Protocol
- HTTP is a stateless protocol which means it has no records of previous interactions. Each interaction is processed only with the information that comes with that particular interaction. Applications that use HTTP should contains the ability to remember information.
- HTTP is commonly used to exchange information for web application.

Local Jar & Remote Servlet

- Local Jar: executable, running as a stand-alone application
- Remote Servlet: running on a server system through web browser

JSP & Servlet

- With a servlet, we write a Java class that contains HTML in the output statements (if we're sending back an HTML page to the client).
- But with a JSP, it's the opposite—we write an HTML page that contains Java code!(for web developer)
- A servlet is a class with responds to a particular type of network request.
- Servlets run in a servlet container like Tomcat. The container will take care of things like wrapping the whole thing in a HTTP response object and send it over to the client(a browser).
