Exam 1 Study guide:

Inheritance:

1. know the difference between overriding and overloading and what each requires

**Overriding is when an abstract method in a super class is replaced or overridden with a method using the same naming and parameters in the subclass.**

**Overloading a method means that the method has the same name as another method but different parameters.**

1. Concrete Classes vs. Abstract Classes vs. Interfaces

**Abstract classes can contain data or abstract methods subtypes can override, while a concrete class only contains regular non-abstract methods and data. Unlike concrete and abstract classes, interfaces can not contain data, it simply indicates that any given class that implements the interface will contain the methods inside the interface class.**

3) Interfaces: common method signatures, no variables

4) Abstract Classes: common method implementations, common variables – may contain abstract methods.

b) Concrete Classes: common method implementations, common variables , no abstract methods– **AND we can build an object from one.**

Client-Server Model and Javadoc

1. What is the client-server model? What is the point?

**The client-server model is for developers to know what type of data to expect and provide to a given method. The model indicates a useful structure for programming, and it makes code more efficient by reducing redundant checks for invalid data.**

1. How do we specify the client-server model? What tags do we use?

**The server method documentation provides tags @require to define the pre-conditions of the client and @ensure for the post-conditions promised to the client – considering valid data is provided.**

1. When must the different conditions be true? Whose responsibilities are they?

**A pre-condition is the responsibility of the client to provide, these conditions provided must be true but can be more lenient when the method is being overridden by a sub-type. The server will indicate if the pre-condition is more lenient. It is never more restrictive.**

**A post-condition is the responsibility of the server, but it merely relies on the information provided by the client. Unlike the pre-conditions, the post-conditions can be more restrictive on the data promised when denoted in the documentation.**

1. What is the fundamental “thing” to which these conditions apply?

**The conditions indicate what type of data the underlying method is capable of handling and producing. To a developer using another person’s API, it applies to them in the sense that it guides implementation.**

1. What happens if the “contract” is broken?

**If data is passed in that does not suit the ‘contract’ the results will not be what is expected or promised by the ‘contract’**

Testing:

1. Unit testing vs. Functional Testing

**Functional Testing refers to the entire software system as a whole. The question is… Given input does it produce correct output? This is often referred to black box testing. This be because with functional testing the developer typically does not have direct access to the code.**

**Unit Testing refers to testing each of the independent components of the software system and making sure they behave properly. Each ‘unit’ is a method in the class.**

1. black, grey, and white (or clear) box

**Black Box testing is essentially functional testing. We only have the documentation of the source. We test the code using what we expect to happen based on the classes’ contracts.**

**Grey Box Testing is essentially test-driven implementation, we have the source, but it has not been implemented yet. We write the code to pass the test which it must pass.**

**White Box Testing is essentially implementation-driven testing, we have full access to the code we are testing. We can use the code to get an idea of how to test.**

1. test-driven-implementation vs. implementation-driven-testing

**Test-driven-implementation is when you write your test classes first, and then start developing your implementation. Use tests to gauge the correctness of your code.**

**Contrary to test-driven-implementation, implementation-driven testing is when the code already exists, and you test it to ensure functionality.**

1. JUnit, test fixture, JUnit labels

**@BeforeClass indicates that the method below is called once before calling any of other methods labeled @Test**

**@Before indicates that the method below is called before every method labeled with @Test**

**@Test indicates to Junit that the method below needs to be tested**

**@AfterClass indicates that the below method is called after everything is finished executing in the class**

**@After indicates that the below method is called after every method labelled with the @Test tag**

5) know how to call the different “assertion methods” in JUnit

**The junit Assert class contains methods to check output…**

**assertEquals(expected, actual);**

**Where the expected and actual arguments are the same datatype of types:**

* **Object**
* **Int**
* **Double**

**Additional methods include…**

**assertNotNull(Object object);**

**assertTrue(Boolean condition);**

**assertFalse(Boolean condition);**

Design Patterns:

1) Adaptor Pattern – **used as a go-between to allow two already written codebases to interact with one another.** Can be accomplished either with inheritance or composition – see example we did in class that is in examples repository

2) Strategy pattern – **allows for modification of the behavior of objects during runtime**

See code in examples repository.

3) Singleton pattern – **restricts the number of objects that can be instantiatied from a class.** The recipe for this is as follows:

**a) private constructor**

**b) private static reference to the single instance of the class – initialized to null**

**c) public static method that returns that instance if not null – otherwise calls**

**the constructor and assigns to the static reference**

4) Observer Pattern – **allows objects (the Observers) to be notified whenever another set of objects (the Observables) change state.**  It requires:

**a) Any Observer object must implement the Observer interface and provide an implementation of the method “update”. This is the “what to do” when an Observable changes state. An Observer must be registered with an Observable for this to work by calling the addObserver() method of Observable.**

**b) Any Observable object must extend the class Observable. At the end of any method that changes the object’s state it MUST call setChanged() followed by notifyObservers().**

Serialization, Files, Streams:

1. What is object serialization?

**It’s a way to store an object in memory even when software termination occurs.**

1. What’s the difference between serialization and writing a formatted text file?

**Serialization stores objects implementing serializable in hexadecimal. It is easier to load serialized files because the format is always the same. Formatted text is very specific in the way it is written and read, the format is interchangeable making it more difficult to read formatted text.**

1. What’s the difference between a relative path and an absolute path?

**Relative path is represented by ‘./’ which indicates the current working directory.**

**Absolute path is represented with ~ and brings the working directory to the home directory of the user currently interacting with the operating system.**