

Java Cheat Sheet

Object oriented programming concepts

- **Encapsulation** – this is concerned with hiding the implementation details and only exposing the methods. The main purpose of encapsulation is to; Reduce software development complexity – by hiding the implementation details and only exposing the operations, using a class becomes easy. Protect the internal state of an object – access to the class variables is via methods such as get and set, this makes the class flexible and easy to maintain. The internal implementation of the class can be changed without worrying about breaking the code that uses the class.
- **Inheritance** – this is concerned with the relationship between classes. The relationship takes the form of a parent and child. The child uses the methods defined in the parent class. The main purpose of inheritance is; Re-usability– a number of children, can inherit from the same parent. This is very useful when we have to provide common functionality such as adding, updating and deleting data from the database.
- **Polymorphism** – this is concerned with having a single form but many different implementation ways. The main purpose of polymorphism is; Simplify maintaining applications and making them more extendable.

Interface vs. Abstract Classes

- Interfaces are implicitly abstract and cannot have implementations outside the implementing class. Abstract classes can have instance methods that implement a default behavior
- Variables in an interface by default are final. An abstract class may contain non-final variables.
- Java interfaces are public by default meanwhile abstract class has can use the usual private, protected, public
- Interface gets implemented, abstract classes get extended
- Possible to have non abstract methods in an interface (you can have static and default)

Constructors

- No return type
- They are the skeleton of an object, they allow you to create an object
- When new instance of a class is called, the constructor is called to build the object
- May have input values

Serialization and File I/O

- Serialization is a way to transform objects into byte sequences and back (deserialization). It allows us to transmit serializable objects over the network and store them into files
- File I/O streams are for storing/reading any kind of data to/from files

List, Map, Set – Collections

- All are interfaces in java
- List
 - Provides ordered and indexed collection which may contain duplicates
 - Most popular would be ArrayList and LinkedList
 - Ordered Collection, maintains insertion order

- Nulls allowed (can have several)
- When to use
 - Frequent access via index. If you know the index ArrayList provides quick access
 - If you need to maintain an order on how they are inserted then go List again
- Set
 - Interface provides an unordered collection of unique objects (no duplicates)
 - Most popular would be LinkedHashSet, TreeSet and HashSet
 - Unordered collection, no guarantee on which order element will be stored (Some may maintain order, but it is not a requirement by a Set)
 - Only one null allowed
 - When to use
 - Collection of unique elements, use Set as no duplicates are allowed
 - Items stored on a TreeSet can be sorted easily when using a comparator so it is possible to maintain an order. However generally they do not maintain order
- Map
 - Map provides a data structure based on key-value pair and hashing
 - Most popular would be HashMap, Hashtable, LinkedHashMap
 - May contain duplicates but each value is sent in with a (key, and value) so each piece of data does have a unique key
 - Can have null values, and at most ONE null key
 - When to use
 - If you need to store data associated with a specific key then a Map is the way to go. For every value that goes in to a MAP a unique key also does which is used to access that value

Access Modifiers

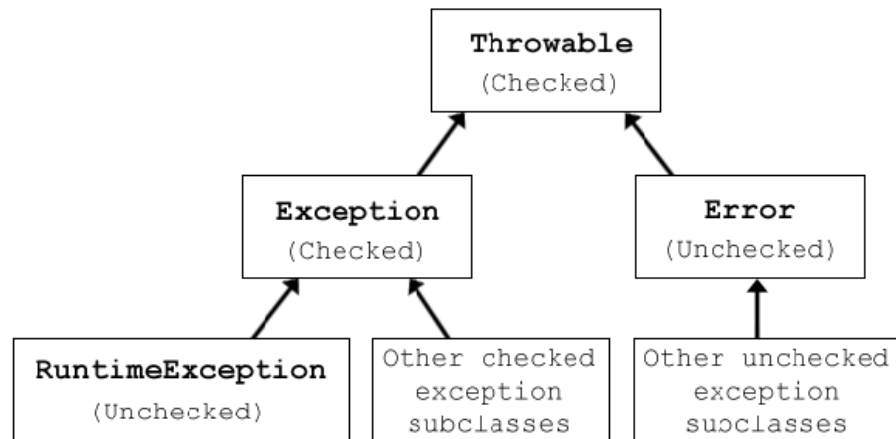
- Important for Encapsulation, force the need for setters and getters

Default	Private	Protected	Public
Same class	Same class	Same class	Same class
Same package subclass	Same package subclass	Same package subclass	Same package subclass
Same package non subclass	Same package non subclass	Same package non subclass	Same package non subclass
Different package subclass	Different package subclass	Different package subclass	Different package subclass
Different package non subclass	Different package non subclass	Different package non subclass	Different package non subclass

- Default – only the same package
- Private – only the same class
- Protected – only the same package and sub classes in a different package
- Public – Everywhere
 - Should be avoided, except for constants

Checked vs. Unchecked Exceptions

- Checked
 - Checked exceptions are checked at compile time
 - Checked exceptions must be thrown at method declaration, or placed in a try-catch
- Unchecked exceptions are not checked. Error and RuntimeException are unchecked exceptions, everything else under throwable is checked.
- Things should be checked if a client can reasonably be expected to recover from the exception. If a client cannot do anything, or it is unreasonable then make it an unchecked exception



Generics

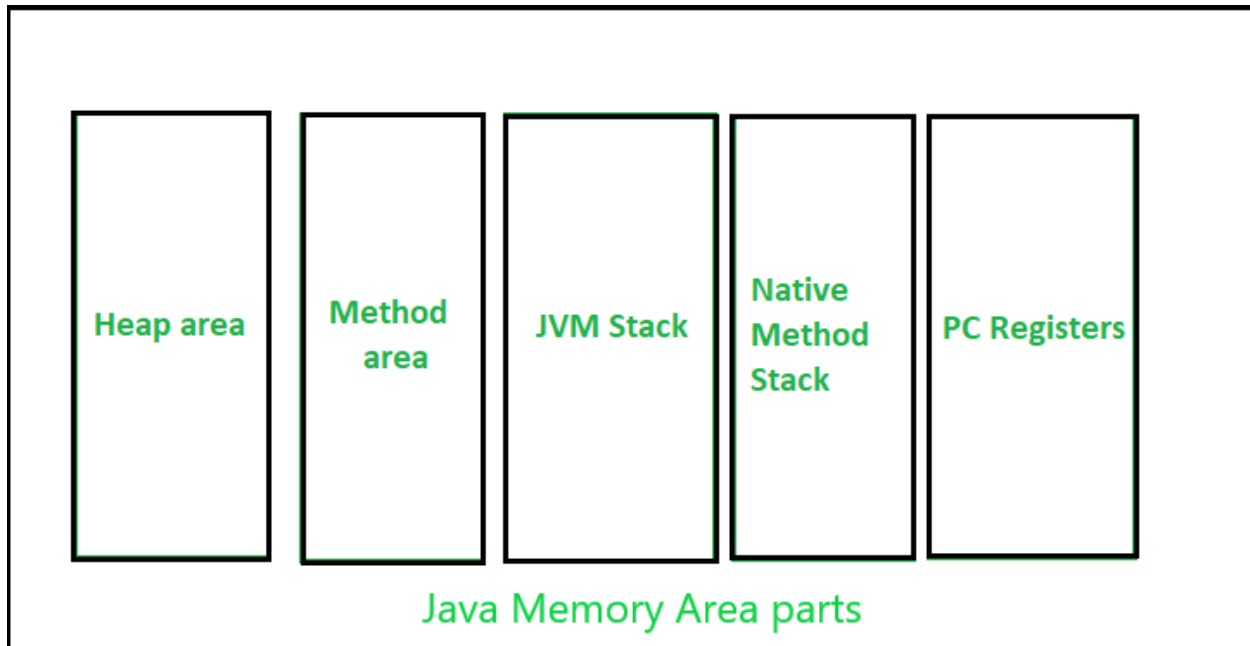
- Allows type to be a parameter to methods, classes and interfaces.
- ArrayList, HashSet, HashMap, etc do this very well
- Ex. `ArrayList<Integer> arrayList = new ArrayList<Integer>();`
 - The generic is `<Integer>`, a requirement for being in this list is that the value must be an integer
- Cannot use primitive types (int, double, char, etc) Must use the Object/Class version (Integer, Double, etc)

Java Keywords

- Final – variables are constants (cannot be changed), final methods cannot be overridden, final classes cannot be sub classed
- Static – Indicates that a variable or method, is a class method
- Volatile – indicates a variable may change asynchronously
- Synchronized – specifies critical sections or methods in multithreaded code
 - Synchronized is attached to some object which will only allow one thread at a time to access that object
- Transient – specifies that a variable is not part of an objects persistent state
 - Used in serialization
 - Used if we don't want to save the value of a variable
 - The JVM it will ignore the original value of the variable and save it in its default state (typically 0)
 - Great for password security
 - Cannot be static (volatile keyword can be static)

JVM and Memory Management

- The JVM has very intuitive memory management so it is not always needed by the programming
 - Ex. The garbage collector will come around and auto delete unused variables
- Some items are not handled automatically by the JVM and garbage collector forcing the programmer to get involved
 - Ex. Closing your scanner to ensure all resources are freed up



- Heap
 - Shared runtime data area, stores the actual object in memory. Instantiated during VM startup
 - Allocated for all class instances and array, can be fixed or dynamic size depending on system configuration
 - Only one heap for a running JVM process, never more
- Method Area
 - Logical part of the heap area, created on VM startup
 - Memory allocated for class structures, method data and constructor field data. Also for interfaces, can be fixed or dynamic size.
- JVM Stacks
 - Stack is created same time a thread is created. Used to store data and partial results which is needed while returning value for method and performing dynamic linking
 - Fixed or dynamic, stack size can be chosen independently when created
- Native Method Stacks
 - Not written in the Java language
 - Known as C stacks, memory allocated for each thread when its created, can be fixed or dynamic
- Program Counter (PC) Registers
 - Each JVM thread that has a task is associated with a PC Register.
 - It stores the address of the available JVM instruction

Multithreading and Synchronization

- Best to look at deadlock code to explain synchronization
- Multithreading
 - Using Threads allows you to concurrent execution of two or more parts of a program
 - Allows maximum utilization of the CPU
 - Threads are light-weight processes within a process
 - Achieved by extending thread class or implementing the runnable interface
 - Both require usage of the run method
 - If you extend thread class we cannot extend any other class b/c java does not support multiple inheritance
 - If you implement the runnable interface, our class can still extend other base classes
- Synchronized – specifies critical sections or methods in multithreaded code
 - Synchronized is attached to some object which will only allow one thread at a time to access that object
 - Be careful of DEADLOCKS, they can occur if two threads access synchronized methods in opposite order
 - Should always build in the same order of locks A->B always; never A->B AND B->A

Design Patterns

- Creational
 - All about class instantiation or object creation.
 - Great for Factory method, abstract factory, builder, singleton, object pool and prototype
- Structural
 - All about organizing different classes and objects to form larger structures and provide new functionality
 - Great for adapter, bridge, composite, decorator, façade, flyweight, private class data and proxy
- Behavioral
 - All about identifying common communication patterns between objects and realize these patterns
 - Great for chain of responsibility, command, interpreter, iterator, mediator, memento, null object, observer, state, strategy, template method, visitor

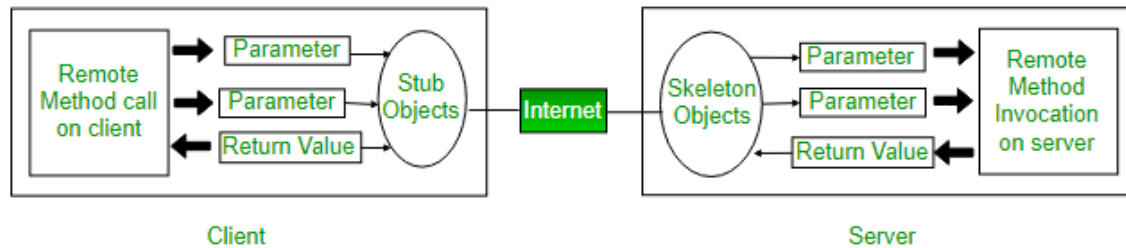
Sockets

- `Socket socket = new Socket(IP , Port);`
- Streams are used in both input and output of the data
- Sockets are needed to connect to another machine
- Think of the java chat program you built with Nicole

Remote Method Invocation RMI

- Is an API which allows an object to invoke a method on an object that exists in another address space on the same machine or on a remote machine.
- Through RMI an object on the client side can invoke methods on an object present in another JVM on the server side.
- How it works:
 - Stub object

- Client machine builds an information block and sends this info to the server
- The information contains: an identifier of the remote object, method name to be invoked, and parameters to the remote JVM
- Skeleton Object
 - This object passes the request from the stub object to the remote object
 - It performs tasks like calling the desired method on the real object (present on server), forwards parameters received from the stub object to the method



- Steps to implement interface
 - Define a remote interface
 - Implement the remote interface
 - Creating Stub and Skeleton objects from the implementation class using rmic (rmi compiler)
 - Start the rmiregistry
 - Create and execute the server application program
 - Create and execute the client application program

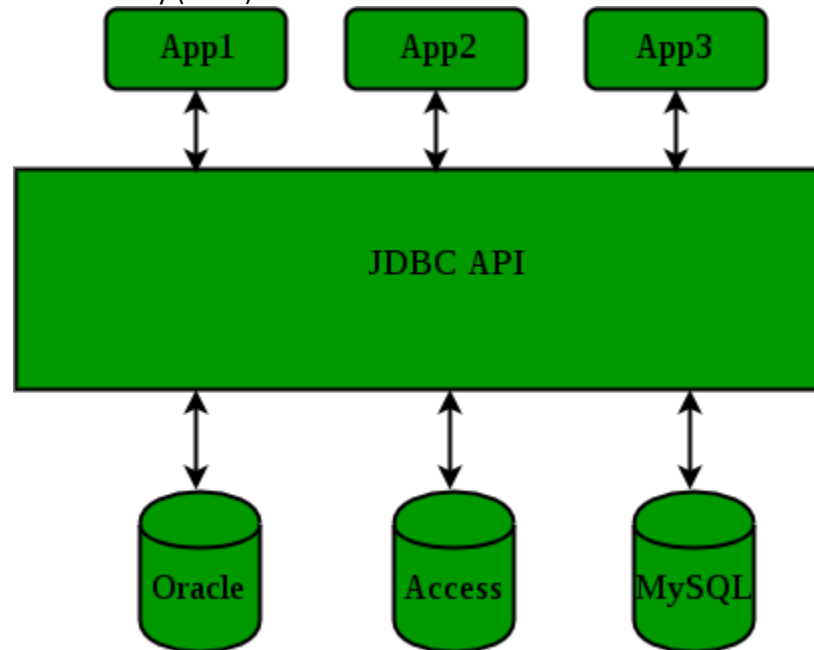
ORMs

- EJB (Session Beans, Entity Beans, MDBS)
 - Session bean stores data of a particular user for a single session.
 - Entity Beans represent persistent data storage. Data can be saved to database via entity beans and later on retrieved
 - Message Driven Bean are used in context of Java messaging service. Consumes JMS messages from external entities and acts based on those messages
- Hibernate
 - Used to overcome limitations of JDBC
 - It is open source and usable by anyone
 - Can modify the code to our need, very versatile
 - Most implementations are done for us, CRUD (create, read, update, delete) operations taken care of
 - First Level Cache vs. Second level Cache
 - 1st level came with Hibernate 1.0 where 2nd level came with Hibernate 3.0
 - 1st level is session specific (local) while 2nd level is shared by sessions (global)

Database Knowledge

Database Drivers

- Java Database Connectivity (JDBC)



- Type 1 driver
 - JDBC-ODBC bridge driver
 - Not written in java so its not portable
- Type 2 driver
 - Native API driver, converts JDBC method calls into native calls of the database API.
 - Allows interaction with different databases, data transfer is much more secure then type 1
- Type 3 Driver
 - Network protocol driver uses middle ware that converts JDBC calls directly or indirectly into the vendor-specific database protocol
 - Written fully in java, therefore portable
 - No client side library is required, b/c application server can preform tasks like auditing, load balancing and logging
 - Maintenance is costly because it requires database-specific coding to be done in middle tier
- Type 4 driver
 - Native Protocol driver interacts directly with database.
 - Known as Thin Driver b/c it does not require any native database library
 - Written fully in java, therefore portable
- Which to use?
 - Accessing one type of databse like Oracle, preferred driver is type 4
 - Multiple types of databases at the same time then use type 3
 - Type 2 drivers are useful in situations where T3/4 are not available yet for your database
 - Type 1 driver is not considered a deployment level driver, only really used for development and testing purposes

Normalization and Denormalization

- Normalization
 - Organizing the attributes of database to reduce or eliminate data redundancy
 - Data redundancy will increase the size of the database, inconsistency problems also arise during insert deletion and update queries
- Denormalization
 - Optimization technique where we add redundant data to one or more tables.
 - Helps avoid costly joins in a relational database
 - This does not mean not doing normalization, denormalization is an optimizing technique used after normalization the database
- Pros and Cons of denormalization
 - Denormalization helps us retrieve data quicker, we need to do fewer joins
 - Denormalization queries to retrieve can be more simple since we only need to look at few tables
 - Cons: updates and inserts are more expensive, denormalization will make update/insert code harder to write, data may be inconsistent, increases amount of storage

Stored Procedures

- It is like a function that contains a set of operations compiled together.
- Set of SQL statements with an assigned name, which are stored in a relational database management system as a group so it can be reused and shared by multiple programs
- It provides an important layer of security between the user interface and the database. Supports security through data access controls because end users may enter or change

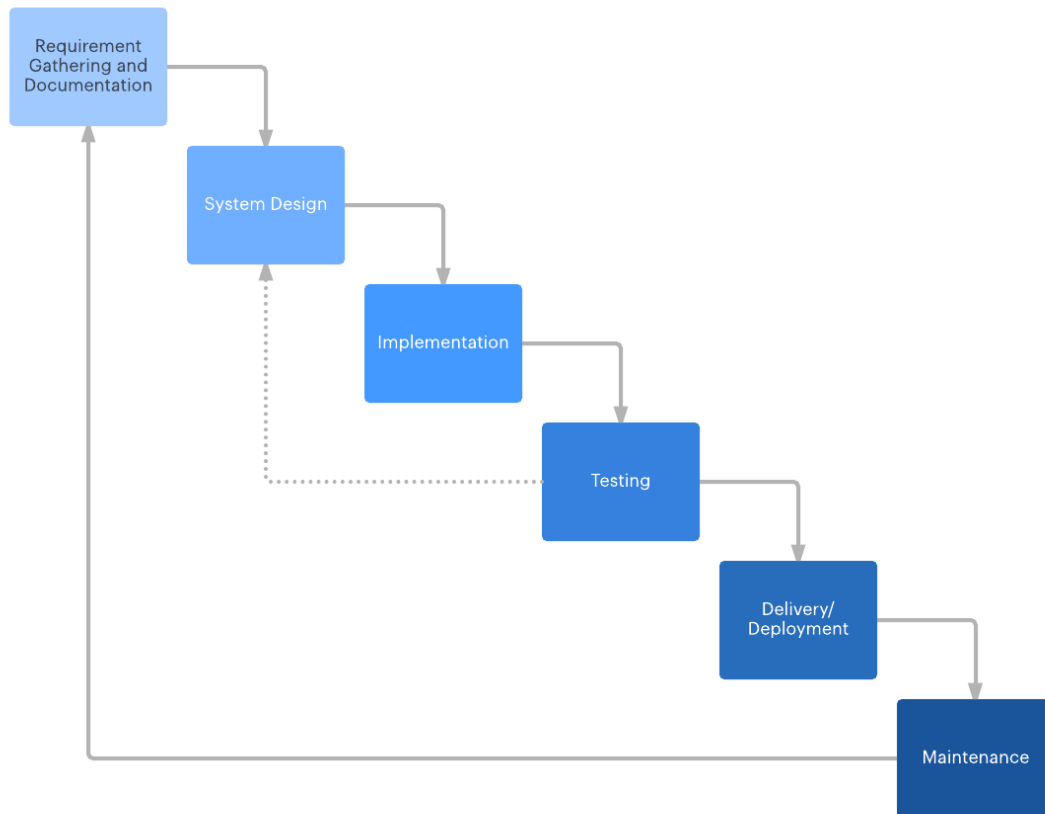
Triggers

- A database object that is associated with a table
- It will be activated when a defined action is executed for the table
 - It can be activated if you run an insert, update or delete and it can be invoked before or after the event
- Need SUPERUSER privileges to run MySQL triggers
- Ex. `CREATE TRIGGER agecheck BEFORE INSERT ON people FOR EACH ROW IF NEW.age < 0 THEN SET NEW.age = 0; END IF;`

Cursor

- A cursor is read-only (cannot write to it), non-scrollable (can only fetch rows determined by the select statement), and asensitive
- Asensitive
 - Asensitive cursor
 - Points to actual data
 - Faster than insensitive
 - Insensitive cursor
 - Uses a temporary copy of the data
 - Creates a copy and is safer to use an update command

Waterfall Process



Made in
 Lucidchart

Stages

- Requirement gathering and documentation. Phase 1
 - Gather comprehensive information about what the project requires.
 - Gather information through interviews, questionnaires, interactive brainstorming
 - By the end of this phase requirements should be clear, and a requirements document should be distributed to the entire team
- System Design. Phase 2
 - Using the requirements, the team designs the system.
 - No coding yet, specifications are proposed here, like programming language(s), software, hardware requirements
- Implementation. Phase 3
 - Coding takes place
 - Take information from previous steps and build a functional product
 - Implement in small pieces, which are integrated at the end of this phase and/or beginning of the next
- Testing. Phase 4
 - Test product in real (safe) environment
 - If problems arise team may need to return to phase 1 for reevaluation

- Delivery/deployment. Phase 5
 - Product is complete
 - Deploy product to client
- Maintenance. Phase 6
 - As issues arise you should be in contact with client to troubleshoot issues on 1st, 2nd, or 3rd, level.
 - Send patch updates
 - Major issues may require return to phase 1

Waterfall Facts

- Keeps training simple
 - Ensures success even if unanticipated changes in bandwidth occur
- Shows progress
 - You will see what stage you are at what is being completed, and what needs to be done still
 - Eliminates guesswork in project timeline because it does not allow to return to prior phase
- Makes project easy to manage
 - Know where project is, where it should be
 - Unexpected delays/changes in personnel allow team to quickly get back on track
- Saves time and money
 - Taking proper time to plan and design allows us to save time and money down the line in implementation and testing
- When to use
 - When requirements are clear and specific. They do not change for the most part
 - Use agile if project requirements could change drastically

Common Questions

Difference between == and .equals()

- X.equals(y) means the references x and y are holding objects that are equal
- X==y means that references x and y are referencing the same object

Common use of "this" keyword

- Refer to instance variable when local variable is the same name
- When passing itself to another method
- Calling another constructor in constructor chaining

Lambda Expression

- Useful to write shorthand code, saves effort of writing lengthy code
- Should promote developer productivity, which in turn creates better more reliable code

Different types of classes

- Access – public, protected, default, private
- Packaging – System, library, user defined
- Structure – Outer or Inner
- Object Derivation – abstract class or concrete class
- Object creation – normal, singleton, doubleton, immutable, or enum
- Functionality – String, Util, Stream etc.

Final vs. Finally vs. Finalize()

- Final
 - Final variables are constants, will never change.
 - Final Methods cannot be overridden
 - Final Classes cannot be sub classed
- Finally
 - Used after the catch portion of try catch block
 - Used to "clean up" no matter what happens in try block, it will always execute
 - Commonly used to close files, or close database connections
- Finalize()
 - Invoked before garbage collection, allowing it to clean up its state

ArrayList vs LinkedList

- If you need to get data in a sequential order then use LinkedList
- If you need to insert elements anywhere at anytime use linked list
- LinkedList does not support most optimized searches like binary search so use ArrayList
- If you know the size of your data ArrayList could be better, if you don't then use LinkedList because it allows for expansion

String vs StringBuffer vs StringBuilder

- String – Use if string will stay the same
 - Immutable, one created cannot be changed, if value is changed a new object is created
 - Strings are stored in the Constant String Pool
 - String cannot be used by two threads simultaneously
- StringBuffer – Use if it will change and be needed to run concurrently
 - Mutable which means we can change the value of the object
 - Stored in the heap
 - Same methods as StringBuilder, BUT StringBuffer methods are synchronized, therefore thread safe. However StringBuffer is slower than StringBuilder because of this
- StringBuilder – Use if it will change and will not be used concurrently
 - Basically, the same as StringBuffer except it is not thread safe, making it faster

Rules and differences of Overloading and overriding

- Overloading
 - Method signature must be different, different types,/amount of parameters
 - If you only change the return type it should return an error
 - You can overload a static method
- Overriding
 - Name and return type must be the same
 - Method signature should also be the same
 - Must be overridden in a subclass
 - Cannot increase security level, can only keep the same or reduce
 - Protected to private not allowed protected to protected allowed protected to public allowed
 - Cannot override: Private, static or final methods
- Differences
 - Overloading is during compile time, overriding is during runtime
 - Can overload in the same class, can only override a method in its subclass
 - Overloaded methods are fast compared to overridden methods in java

ClassLoader

- When you run a java program it gets converted in to a .class file which is basically byte code of the .java file. The ClassLoader is responsible to load that class file from the file system (or where ever its located).
 - Three types of class loaders: Bootstrap, Extension, and System class loader

What is double checked locking in Singleton

- Singleton means we can only create one instance of that class
- Double Checked Locking is how java ensures that that at any cost only one instance is created
 - If in a multi-threaded environment DCL may not be reliable so be sure to use synchronized keyword in this scenario