Java refresher notes

(Up To date with Java 16)

Latest Java version is Java 16 (non LTS - Long Term Support): Latest LTS is Java 11. Next LTS is java 17 set to be rolled out at Sep 2021. Java 11 is the widely used version as of today (Aug 2021).

Data types

- Local variables need to be initialized.
- Integer representation could be decimal or hexadecimal(0x..) or binary(0b..). Binary added in Java 7.
- Floating point literal should end with ‘f’ otherwise it is double. Double literal can end with ‘d’ optionally.
- ‘null’ is a special literal which could be assigned to a reference type.
- If an integer literal ends with “L” or small ‘l’, it is the long.
- Java 7. Underscores can be used within digits to improve the readability in the code. For example int value = 25_45. This is valid.
- Byte – 8 bit, Short – 16 bit, Int – 32 bit, Long – 64 bit, Float – 64bit (Single precision), Double – 64bit, Char – 16 bit.
- Java supports var type from java 10. Var keyword can be used for local variables with initialization. It cannot be used for class level variables or generics.
- String is valid in switch statements. Switch supports expression (java 14)
- Access specifier : use the most restrictive one.
- Parent’s parent’s protected instance variable accessible in the class. Meaning, protected instance variables are available throughout the hierarchy.
- Static initialization block gets executed once the glass gets loaded.
- Iterator Vs Enumerator – Iterator is the more recent version. Iterator allows modifying objects from the underlying collection.
- Java provides a container object Optional to deal with Null conditions. https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html
- Numbers – Java platform provides wrappers for primitive data type.
    - Compiler does the auto boxing and auto unboxing.
    - Use formatter for printing numbers or date to Sys.out.
        - System.out.printf
        - System.out.format
    - Java.text.DecimalFormat shall be used for formatting digits.
    - Java.lang.Math supports beyond basic arithmetic. All its methods are static.
        - If you use static import the Math class, you don’t have to say Math.cos() but just cos().
        - For a single random number, use Math.Random but for a series of random numbers use create java.util.Random
        - Character class supports character related operations.
- String
    - String literals – reused
    - Strings are immutable.
    - Do not generally use the toString() method of objects to convert the object into the string.
    - To convert numbers into the String use the Wrapper class methods.
    - For changes in the string, use StringBuilder which is mutable (internally variable length array). StringBuffer is the thread safe version of it.
    - Text blocks (Java 15) allow writing multi-line strings in a readable way. It’s still String type. Text should be placed within triple quotes.
- Class path – Where classes to be executed available
    - By default, the current directory & the java platform classes are already in classpath.
    - Multiple paths shall be given in classpath variable separated by semicolon
- Variable arguments
    - Should be used as the final argument in the method.
    - Reduce using variable arguments unless the benefit is compelling.
    - Do not overload methods with variable arguments as it confuses the API users.
    - Internally this is implemented as an array.
    - Reference http://docs.oracle.com/javase/1.5.0/docs/guide/language/varargs.html

Package

- Names shall be in all lowercase.
- The *static import* statement gives you a way to import the constants and static methods that you want to use so that you do not need to prefix the name of their class.
- No difference between import statement com.java.* and com.java.classname at run time but only at compile time.

Enum

- Enum in java is more powerful than its counterparts.
    - Enum can have methods and fields but enum cannot be inherited (compiler treats it as final class).
    - All enums implicitly extends java.lang.Enum.
    - The constructor of the enum should be of package-private or private access specifier.
    - Constants in enum should be defined first and before any methods/fields.

Record (java 14)

- Record is a special type of class which simplifies the creation of data transfer objects (DTO).
- Traditionally, we create POJOs to carry data between the layers with getter/setter, tostring, hashcode, etc.
- With record type, we don't have to code all those boilerplate (get/set); Compiler creates them for you.
- More details - https://howtodoinjava.com/java14/java-14-record-type/

Interfaces

- Defines contracts using which 2 components can be glued.
- Interfaces has
    - § Constants
    - § Method signatures
    - § Nested types
- A class can implement multiple interfaces.
- By default – These can be omitted.
    - § All methods in an interface are public.
    - § All constants are static, final and public
- Interface is a reference type.
- Interface supports static methods. It also provides default implementation - Link.

Inheritance

- All classes in java inherits java.lang.Object implicitly.
- Use keyword super to access the immediate parent.
- Use instanceOf operator to check if a particular object is of certain type. This is useful while casting and saves you from runtime exceptions.
- By overriding a parent class method, you have the option to increase the visibility (but not reduce) of the method through access specifier
- Polymorphic behavior in Java accomplished through virtual method invocation.
- A field in subclass hides the field of parent class with the same name even if the types are different. Hiding is not recommended. Parent field can be accessed using the keyword super.
- Constructor chaining – subclass constructor should call the super class constructor in its first line. If an explicit call is missing, the compiler inserts a call to the parent's default constructor.
- Sealed class / Interfaces (java 15)
    - With this feature, a class or an interface can now define which classes can implement or extend it. It enables fine-grained inheritance control in Java. https://www.baeldung.com/java-sealed-classes-interfaces

Nested classes

- Nested classes 3types https://www.geeksforgeeks.org/nested-classes-java/
    - Static nested class – Do not have access to members of the outer class.
        - § Unique behaviour
        - § No way similar to static class. Having a static in its header doesn't mean that all in it is static.
        - § Similar to top level class except controlling visibility of it through access specifier.
        - § Static inner class takes any scope, even the private.
        - § Static inner class can access the outer static variables.
    - Non-static nested class also called inner class – Inner class has access to even the private members of the outer class.
        - § Accessed through outer class. Outer.Inner = Outer.new Inner();
        - § 2 types
            - Local class – Defined within a code block, typically inside a method.
                - Local inner class is not a member of the enclosing class and hence cannot have any access specifier.
                - Local inner class will have access to all the members of the enclosing class and it'll have access to the local final variables in the scope it's defined.
            - Anonymous class.
                - Defined within a code block without a name and typically inside a method.
            - Anonymous class is the same as local inner class except without a name and hence cannot be used anywhere other than the point where it is defined.
    - Nested class shall be given any access specifier – even private.
    - Shadowing - If a declaration of a type (such as a member variable or a parameter name) in a particular scope (such as an inner class or a method definition) has the same name as another declaration in the enclosing scope, then the declaration *shadows* the declaration of the enclosing scope

Lambda expression

- Lambda expression allows you treat function as a method argument or code as data.
  - o Lambda expression replace anonymous class which deals with one interface function. An interface with only one method is called functional interface.
  - o Lambda expression takes the format (formal parameter1, parameter2)->{expression;}
    - § When one parameter exists, no need for ().
    - § When only one expression exists, no need for {}.
  - o Method references: Java supports referring methods within lambda expressions - Link

Generics

o Generics enable types (classes & interfaces) to be parameters while defining classes, interfaces, methods. Generics are called parameterized types.
o Benefits
  § Avoids casting.
  § Type safe – Strong compile time type check.
  § The parameter you supply is called the type parameter.
o Diamond operator <> used in 7 allows the compiler to infer the right type.
o Raw type – A parameterized type but called with no argument type. General types with no generics are NOT called raw types.
  § When raw type is used, class Object will substitute in place of the type argument.
  § Raw type object cannot be assigned to generic counterpart. Doing it will lead to type safety issues.
o Generics can work with methods as well. While method invocation, you could do the following or simply ignore the type arguments (type inference)
  § `boolean same = Util.`**`<Integer, String>`**`compare(p1, p2);`
o Bounded type parameters – Restricting the type arguments of certain type. For example, you want a generic type to accept only types of Number and its sub classes.
  § Use the keyword 'extends'. If multiple type are there, use '&'. Class type should always go first.
  § boolean same = Util.**<Integer, String>**compare(p1, p2);
  § Bounded type is applicable for Generic methods also.
o Wildcards - ? – unknown type
  § Wildcard can be used as type parameter, field variable, local variable
  § Cannot be used as type argument
  § Upper bounded wildcard - To write the method that works on lists of Number and the subtypes of Number, such as Integer, Double, and Float, you would specify List<? extends Number>.
  § Lowed bounded wildcard - write the method that works on lists of Integer and the supertypes of Integer, such as Integer, Number, and Object, you would specify List<? super Integer>
  § Unbounded wildcard. when the code is using methods in the generic class that don't depend on the type parameter. For example, List.size or List.clear. In fact, Class<?> is so often used because most of the methods in Class<T> do not depend on T.
o Type erasure – Compiler generates type erasure methods/code at compile time to support generic programming.

Exception

o "Catch or specify" requirement for handling exceptions. Subclasses of Exception only honors this.
o 3 kinds of exception
  § Errors – External to the application; Non-recoverable.
  § RuntimeExceptions – Internal to the application. Generally, non-recoverable. Example, IndexOutOfArray, NullPointerException.
  § Exceptions – Classes that inherit the Exception class. It's checked exception.
o Checked vs Unchecked exception - Checked exception means it's checked during compile time.
o News in JSE 7
  § Specifying multiple exception type in a single catch
    · Catch(IoException | IndexOutOfArray ex){}
  § Try with resources. Try(resource declaration){} – gets automatically closed. Classes that implement Closeable or AutoClosable interface shall be implemented in this try with resource block.

Streams

- Streams – I/O streams available in java.io package and File I/O streams available in java.nio package
  § Stream is a sequence of data. A program reads from a data source and hence the input stream. A program writes to the source and hence the output stream.
  § ByteStream classes inherit from Inputstream & OutputStream. Bytestreams should be used for most primitive I/O such as image data. If the data is character, use character stream classes. All other streams are built on byte streams.
  § Character stream – All character stream classes inherit from Reader and Writer. All character stream classes are wrappers of byte stream classes. FileReader uses FileInputStream and FileWriter uses FileOutputStream. If none of wrapper classes suit you, use InputStreamReader and OutputStreamReader to create your own writer and Reader.

    - § Use buffered streams for improving efficiency. 4 buffered classes available. BufferedInputStream and BufferedOutPutStream classes for byte streams. BufferedReader and BufferedWriter classes for character streams.
    - § Scanner – Shall be used to break input stream into tokens.
    - § Stream objects that implement formatting are PrintStream & PrintWriter. PrintStream for byte streams and PrinterWriter for character class. System.out & System.err are the 2 print streams you normally use. Rest of the time, you will be using PrintWriter only. Formatting can be used in these PrintStream objects using the format method.
    - § Input from console shall be taken via Console & Standard stream
        - · Standard Stream – System.in.
        - · Console.
    - § Datastream – handling binary I/O of primitive data type & string value.
        - · EOF is known through the EOFException.
        - · Data can be read in the same order as how it was written.
        - · Data stream classes are DataInputStream & DataOutputStream.
    - § ObjectStream – handling binary I/O for object
        - · Serialization handled via ObjectStream.
        - · Object stream classes are ObjectInputStream and ObjectOutputStream.
        - · Supports object streams for classes which implement a serializable interface.
        - · Wraps around the data stream and hence support the mixture of object & primitive values.
        - · When writing an object to a stream, objects which are referred to in the given object will also be written into the stream.
        - · Object is unique within a stream. If 2 objects refer to a single object. 2 copies will not be created but only one copy.
- o File I/O – NIO.2
    - § Path class is not system independent. Path class represents the path in the operating system. Paths is a helper class for path.
    - § Like path class Files class supports lots of file operations.
    - § Use glob expressions for pattern matching in strings esp. with file related matching.
    - § Checking if a file is present
        - · Check with both Files.exists() and Files.notExists(). If both of them return false, it means that JVM doesn't know about the file, which may be an access issue.
    - § Commonly used file I/O – java.nio.file package.
        - · For common use - readAllBytes, readAllLines, Write.
        - · Reading/Writing buffered characters from the file – newBufferedReader and newBufferedWriter.
        - · UnBuffered file streams – newOutputStream, newInputStream.
        - · Channels – Create a pipe between buffers.
        - · Random access in file possible
    - § To walk a directory tree, support available through FileVisitor interface.
    - § Possible to watch the changes in the file.
- o Java 9 introduced the concept of Flow API to support reactive streams. http://www.reactive-streams.org/
- o java.util.Stream support functional-style operations on streams of elements, such as map-reduce transformations on collections.
- o Java support parallel streams.
- o Java 16 doc on Streams - https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/stream/Stream.html

Concurrency

- o Package java.util.concurrency
- o Runnable vs Callable – Runnable cannot return a result and throw an exception but callable can.
- o 2 ways to use threads in a program – Creating thread using Thread class and using executor framework. Java 1 has ThreadGroup concept but it's been deprecated.
- o Creating a thread – 2 ways.
    - § Extend Thread class.
    - § Implement Runnable interface.
- o Interrupt is a signal given to the thread asking it to stop what it is currently doing and do something else. It is up to programmer to decide what the thread should do. Interrupts are handled via exceptions. Many methods in thread throws InterruptException
    - § If the thread has been running for long time and if it is not invoking any method that throws InterruptExpection, it is prudent to check if any interrupt has been occurred by Thread.interrupted() and throw an Interrupt exception if it occurs.
    - § Interrupt is implemented via an internal flag know as interrupt status. Invoking Thread.Interrupt sets this status and invoking static Thread.Interrupted() clears this flag. Non static Interrupted() method doesn't clear the status.
- o Calling join() on a thread makes the calling thread to wait till the called thread returns. It also throws InterruptException.
- o Thread synchronization – Tool to facilitate communication between threads and to avoid thread interference issues / memory consistency issues.
    - § Each object has its own lock. Even the class has its own class object for static methods.
    - § Synchronized methods – Strategy: If a variable of a method is accessed by multiple threads, make all the read/writes of the variables through synchronized methods. When a thread enters into one of the synchronized methods, all the synchronized methods will be blocked. Synchronized methods acquire object lock in objects and class level lock on static methods.
    - § Synchronized statements – Synchronized statements are fine grained. You would have to define our own locks.
    - § Concept of re-entrancy: Synchronized code invokes another method having synchronized code, both the code will use the same lock.
    - § Atomic actions
        - · Variables marked with the keyword 'volatile' are atomic.
    - § Potential situations which need to be handled with Synchronization

            - Deadlock – If Thread T1 on object A waits for object B method. Thread T2 on object B waits for object A method. Object A & Object B methods are synchronized. This results in a deadlock waiting for each other.
            - Starvation – If an object has a synchronized method and if Thread T1 calling the method frequently and taking long time time to retun, then T1 almost block other threads which lead to starvation for the other threads.
            - Livelock – This is similar to deadlock but less frequent. In this case, threads are not blocked but they simply keep responding to the actions of each other thread. Analogy - This is comparable to two people attempting to pass each other in a corridor.
    - Guarded block – Thread synchronization through setting flags.
        - Do not simply put a while loop which keep looking for change in the status flag. This wastes the processor time. Instead use wait-notify/notifyall mechanisms.
        - Producer-Consumer is a good example of this. See the exercise example - Link
    - Immutable objects avoid thread interference
        - Pointers for creating immuatble object
            - Declare the class final and hence cannot be inherited.
            - Declare all the fields final and private, hence can't be accessed outside.
            - No setters only getters
            - Reference objects within the class.
                - Do not store reference of an external mutable object. If required take a copy & store.
                - Do not return the reference object through a method. If required, create a copy & return.
            - (good practice) make the constructor private and have a factor method returning the object.
    - High level concurrency – Most of them new to Java from version 5.0 and available in java.util.concurrent package.
        - Locks – use lock.tryLock method provides a non blocking attempt to acquire a lock. A thread can attempt to acquire the lock of partipating objects before entering into critical section to ensure that deadlock is prevented.
        - Executor framework – Thread creation and management framework.
            - Executor interfaces – 3 interfaces
                - Executor – simple interface supporting launch of new tasks
                - ExecutorService – sub interface of Executor + methods to manage the lifecycle of individual tasks & life cycle.
                - ScheduledExecutorService – Sub interface of ExecutorService and allows period execution of tasks.
            - Executor interface implementations – thread pool / promoting re-use of the threads.
            - Fork / Join – Java 8 – Option to split the task and collect the task back once executed. This is being introduced to leverage the capability of multi processor/multi core machines.
        - A set of concurrent collection objects
            - BlockingQueue – Blocks when attempted to add in a full queue. Blocks when attempted to retrieve from the empty queue.
            - ConcurrentHashMap – Concurrent analog to hasmap.
        - Atomic variables – java.util.concurrent.atomic package defines a set of atomic classes. Example – AtomicInteger
    - Asynchronous computation using Java Future
        - java.util.concurrent.Future represents results of an asynchronous computation. Useful link - http://tutorials.jenkov.com/java-util-concurrent/java-future.html
    - ThreadLocal concept allows storing data that will be accessible by the specific thread.

Platform facilities

- Configuration
    - Properties file – java.util.Properties.
    - Command line parameters.
    - Environment variables – System.getEnv. Accessing environment variables directly affect the portability of the system.
- System utilities
    - A set of streams
        - Console – Useful for providing password inputs
        - System.in, Sys.out, Sys.err.
    - System properties – System itself maintains a set of properties.
        - System.getProperties.
    - Security manager – System.getSecurityManager()
        - Applet has the security manager by default. For other applications, default security manager won't be present. If required, you have to define one. For example, accessing a file from filesystem may require a security manager to check if the access is allowed.
        - Security violations are usually reported through exceptions that are sub classes of SecurityException class.
    - System.exit – Shutdown the JVM. Status code 0 represents normal exit while non-zero represents error code.
    - System.ArrayCopy – Copy one array to another.
- Path variables
    - PATH – Should point to JDK home directory
    - CLASSPATH – Should point to user defined classes. By default current directory(.) is included. Use –cp switch in command line to supply class path.

Regular Expression

- Package – java.util.regex
- 3 key classes
    - Pattern – Represents regex. No construction. Use the static method.
        - Pattern provides predefined character class shortcuts
            - Dot(.) represents any character
            - \d represents a digit [0-9]
            - \D represents no digit [^0-9]
            - \s represents white space character [\t\n\x0B\f\r]
            - \S represents non-white space characters [^\s]
            - \w represents word character [a-zA-Z_0-9]
            - \W represents non word character [^\w]
            -
    - Matcher – For matching. No constructor. Obtain it from pattern object.
    - PatternSyntaxException – Unchecked exception. You can choose to catch it.
    -
- Dot(.) means any character
- Meta characters which have special meaning - <([{\^-=$!|]})?*+.>
    - If this is a part of the string use
        - Either backslash preceded with it (or)
        - Use \Q (quote starting) and \E (quote ending)
- Character classes
    - [abc] matches a or b or c
    - [^abc] matches negations – anything other than abc
    - [1-5] matches range 1 through 5
    - [a-d[m-p]] represents union a-d or m-p/equivalent to [a-dm-p]
    - [a-d&&[d-f]] represents intersection
- Quantifiers
    - ?Once or not at all
    - * means 0 or more times
    - + means 1 or more times
    - {n} means n number of times
    - {n,} means atleast n times
    - {n,m} means minimum n and maximum m
- Groups – characters that go together represented by ()
- Backreference represented by \
    - Backreference means recalling
    - Example – (\d\d)\1 means 2 digits repeated once. 2121.
- Boundary matchers
    - ^ beginning of a line
    - $ end of a line

Date time API

- Java.time package. (Available from JDK 8)
- 2 representation of time
    - Human terms.
    - Machine terms.
- Java 7
    - Java.util.Date class
    - Java.util.SimpleDateFormat

JAVA RIA

- Applets – Can access html/javascript markup. Can access cookies.
- Webstart – Access through a browser first time. Subsequent times, it can be launched from a desktop shortcut. Can access file system. Can access cookies. Cannot access the html markups. Can access filesystem.

Internationalization (i18n) & localization (i10n)

- Items to be taken care – language, date , currency, UI labels, etc.
- 4 steps to internationalize
    - Define a properties files having a mapping of UI messages.
    - Define a locale providing language info & country info.

  - § Define a resource bundle.
  - § Fetch text from the bundle.
  - § Use formatters to get the right format displayed.
- o Locale object defines the context.
- o Classes like NumberFormat, DateFormat are LocaleSensitive. You just need to pass Locale object in to it.
- o Language tag example - "En-US"
- o JDK provides Service Provider Interface for plugging in 3rd party Locale.
- o Best practice – Use separate resource bundles for different items. This avoids loading huge file into the memory. Example – MenuBundle, LabelBundle,etc.
- o Another alternative to properties files is that define a class for very locale which inherits ListResourceBundle class. However the problem with this approach is that whenever we add a new locale, a new class has to be defined and compiled.
- o Formatting for display shall be done using NumberFormat, DecimalFormat, DateFormat (long date, medium date, short or full – Length of date). Use message formatter to deal with compound messages. MessageFormatter needs a pattern to be applied.
- o Handling text
  - § Use Character class to check if the input character is digit/letter/Whitespace.
  - § Use Collator class to compare text.
- o Internet domain names – Shall be written in ASCII or Unicode.
  - § Java.net.IDN provides methods to convert it back & forth.

JAR files

- o JAR files is a zip file supporting lossless compression.
- o JARs can be created using jar tool which is an executable available with jdk.
- o Get the list of content in jar file in command prompt using jar jt jar-file-name.
- o Content of the jar file can be extracted.
- o Files can be added.modified in a jar file.
- o Jar supports wide range of functionality – Versioning, electronic signing, package sealing, etc. Manifest files provide this facility to Jars.
- o Manifest file – one/archive
  - § Manifest file contains meta information about the jar.
  - § Always placed in META-INF/MANIFEST.MF
  - § Content of manifest takes 'header:value' pair
  - § Application entry point can be set Main-Class:classname
  - § Class path information Class-Path: jar-name
  - § Version information shall be provided at package level.
  - § Package sealing
    - · All classes in the same package must be archived in the same jar.
    - · Use the header "Sealed:true" to seal a package
    - · This forces all the classes of the package come from a single jar. No one can put another class into my package.
  - § Securing jar files
    - · Codebase attribute to restrict to specific domain which prevents re-deploying / copying.
- o Jars can be signed to make sure that the content of the jar is not changed since it's created.
  - § Signature happens with private key.
  - § Public key placed in the jar itself.
  - § A verifier such as browser shall verify it.

Modules

- Java module is a packaging mechanism introduced in Java 10. It allows to think of Java application in terms of modules. Unit of module is still a JAR file.
- First benefit of module concept is the option of having a slimmer java runtime suited esp for smaller devices. Java platform APIs also have been splitted into multiple modules. Now, you have the option to pick the JDK modules your application would want and distribute a smaller package.
- Module requires a descriptor file which allows to declare the dependent modules using *requires* statement. We could also pick the right version of the module the application would want from the available versions. Means, multiple versions of the module can co-exist.
- Module provides an option to hide certain packages while exposing the others through the *exports* statement in the module descriptor.

JavaBeans

- o Javabeans is a concept.
- o 3 key rules
  - § A public no argument constructor.
  - § Properties – Define variables private. Implement getters/setters.
    - · For Boolean – use IsRunning() for a variablr running.
    - · Event notification capability through PropertyChangeSupport.
    - · Events follow this pattern. add<Event>Listener(<Event>Listener a)
  - § Persistence – Serializable

Serialization

- § Implements java.io.serializable if automated serialization or java.io.externalizable customized serialization
- § Classes which are not serializable – Image, Thread, Socket, InputStrem. Attempting to serialize this throws NotSerializable exception.
- § Fields marked static or transient not serializable.

Networking

- o 2 connection types – TCP/UDP. TCP – bi directional/delivery guaranteed whereas UDP is not.
- o Ports connects the TCP/UDP to the application. Applications read/write to the physical wire through the port. 0-1023 port numbers are reserved for http, ftp,etc.
- o Java – Package for networking java.net
- o URL
  - § Java.net.URL
    - · Use java.net.URI to encode URLs. Example – to encode space characters.
  - § Use url.OpenStream to get the stream of data to get the data fetched. If stream blocked through proxy, set proxy details through command line java -Dhttp.proxyHost=proxyhost, -Dhttp.proxyPort=portNumber] URLReader
  - § Open connection through URL.OpenConnection which returns URLConnection. Invoke URLConnection.Connect to establish a connection
  - § Writing to URL.
    - · Create a URL.
    - · Retrieve the URLConnection object.
    - · Set output capability on the URLConnection.
    - · Open a connection to the resource.
    - · Get an output stream from the connection.
    - · Write to the output stream.
    - · Close the output stream.
- o Socket – Represents a point to point connection with the server. A socket is one end-point of a two-way communication link between two programs running on the network
  - § URLs uses socket underneath.
  - § 2 classes.
    - · Socket – Client socket
    - · ServerSocket – Server socket.
      - o Initialize with the port number
      - o Accept() methods waits for the client connection. This method returns a client socket which is used for communication.
  - § Steps in reading/writing to socket
    - · Open a socket.
    - · Open an input stream and output stream to the socket.
    - · Read from and write to the stream according to the server's protocol.
    - · Close the streams.
    - · Close the socket.
- o Datagram
  - § Order of arrival, Arrival itself, Content not guaranteed.
  - § UDP protocol sends self contained packet of information called datagram.
  - § Java.net.DatagramPacket, java,net.DatagramSocket, MultigramSocket are 2 classes which supports datagram in Java.
    - · Socket is for communication. Socket.Send will send a datagram packet to the destination. Address (IP address using INetAddress) and the port number need to be supplied.
      - o Socket.Receive wait for the socket
    - · DatagramPackage is for sending message.
    - · Multicasting of datagram possible. Use MultiCastSocket to receive the multicasted message in the client side.
    - · Java.net.NetworkInterface supports accessing the information on multiple network cards in the system and allows us to choose one of them for communication.
- o Cookies
  - § Key class – java.net.CookieHandler. Supporting classes – HttpCookie, CookieManager, CookiePolicy, CookieStore. CookieManager is a concrete implementation of CookieHandler.
  - § CookieHandler takes care of state management.

Extensions

- o Extensions through libraries.
  - § Optional classes/packages that extend the functionality of java core platform. It has to be packaged as JAR and placed in JAVA_HOME/JRE/lib/ext. Once placed in this directory, we don’t need to include it in CLASSPATH to access the class.
  - § Extension JAR can also be downloaded using the class-path key or Extension-list key in manifest file. Class-path key downloads temporarily and removes once the program is closed. Extension-list key permanently placed it in ext directory.
- o Class loading
  - § Bootstrap classes > Extension packages in lib directory > Classes in Class path.

- Extension through SPI (Service Provider Interface) which allows custom implementation to be supplied for standard APIs.
    - 3 components
        - SPI – Service Provider Interface. It is a simple interface.
        - Service Provider – Implementations of SPI. Can come from different vendors. Need to register it by having a file named the fully qualified name of the SPI in directory /META-INF/services/. This file shall have the fully qualified name of the provider.
        - Service – Class which connects the SPI and the Service Provider. Uses a class java.util.ServiceLoader. ServiceLoader loads service providers available in class path and java extension directory (lib). Service loader makes the providers accessible through iterators. Service class is usually singleton. The SPI need to be supplied to the class loader.
        - Client can use the service class to access the service providers.
    - Security for extension classes.
        - Security permissions for extension packages granted only if the code has been wrapped in PrivelegedAction instance. This happens only when SecurityManager is present.
            - Wrap the code within run() method of PrivilegedAction class.
            - Supply PrivilegedAction instance to java.security.AccessController.
        - Security policy for extensions can be altered/created via JRE/lib/security/java.policy file.

- Full screen mode APIs – Available to support rendering of visuals/images on the screen directly. This is required for gaming apps.

JDBC

- JDBC – for connecting, querying relational databases and manipulating results.
    - Resultset and cursors – Resultset is the set of rows returned for a query and the cursors is pointer which allows us to access the reultset one row at a time.
    - Transactions shall end in commit or rollback.
    - JDBC drivermanager connects the program to the database driver.
    - Prefer DataSource wherever possible because with DataSource underlying source is transparent. DataSource also provides connection pooling and distributed transactions. Connection pooling improves performance.
    - DataSource.
        - Every JDBC driver will have atleast one implementation of datasource – basic or which supports either or both of connection pooling and distributed transactions. Connection pooling is transparent to programmer.
        - System admin set up the data source through the deployment specific configurations via deploying the driver and setting up the JNDI details.
        - Programmer will do a lookup using JNDI InitialContext to bind to the data source.
        - Aith distributed global transaction XA, no need to worry about commit/rollback as the system takes care of it
    - 4 types of JDBC drivers
        - Type 1 – Written in native code. Map JDBC API to another API such as ODBC. Portability is an issue.
        - Type 2 – partial java, partial native. Portability is an issue.
        - Type 3 – Pure java. Connects to a middleware in a database independent manner. Middleware connects to the database in the database dependent manner.
        - Type 4 – Pure java and connects directly to the database.
    - Installing a JDBC driver is as simple as putting the JAR in a folder and referring in a class path.
    - Typical steps
        - Establishing connections
            - Supply connection string to the DriverManager. ConnectionString depends on the database jdbc driver.
        - Create a statement – 3 types
            - Simple Statement
            - PreparedStatement which uses precompiled statement
            - Callablestatements – stored procedures
        - Execute the query
            - Execute() returns the first object query returns . Get the resultSet by called Statement.getResultSet
            - executeQuery – returns resultSet object
            - executeUpdate – Returns integer representing the number of rows affected. Use it for insert, delete, update statements.
        - Process ResultSet
            - Call resultSet.Next() to get the next record.
            - Call rs.getString()/rs.getInteger() to get the field values.
        - Close connection
    - SqlExcetion
        - Provides error code, Sql state code (ANSI standard), error message, cause
        - Warning messages are available with statement object and the ResultSet object.
    - ResultSet
        - Types
            - TYPE_FORWARD_ONLY – Only forward movement of cursor. Default implementation.
            - TYPE_SCROLL_INSENSITIVE – Forward & backward movement. Insensitive to the changes while is the resultset is alive.
            - TYPE_SCROLL_SENSITIVE – Sensitive to the changes.
        - Resultset update support
        - Holding the cursor on commit

        - Getting column values through getters.
        - Cursor can be manipulated if it is scrollable via resultset method like next, before, first, last,etc.
        - Batch statement
            - Use statement's addBatch & executeBatch commands.
        - PreparedStatement shall be used for executing paramterized queries.
    - Transactions
        - When a connection is created, it is in auto commit mode. This means that each individual SQL statement is treated as a transaction and is automatically committed right after it is executed.
        - The way to allow two or more statements to be grouped into a transaction is to disable the auto-commit mode. Connection.setAutoCommit(false)
        - When auto-commit is disabled, no sql statement is committed until the next call to commit.
        - Dirty read - Accessing an updated value that has not been committed is considered a *dirty read* because it is possible for that value to be rolled back to its previous value.
        - Lock mechanism in DBMS blocks access by others to the data that is being accessed in transaction. Once a lock is set, it remains in force until the transaction is committed or rolled back.
        - How locks are set are determined by the transaction isolation levels.
            - TRANSACTION_READ_COMMITTED
            - TRANSACTION_READ_UNCOMMITTED
            - TRANSACTION_REPEATABLE_READ
            - TRANSACTION_SERIALIZABLE
        - Savepoint - The method Connection.setSavepoint, sets a Savepoint object within the current transaction. The Connection.rollback method is overloaded to take a Savepoint argument.
    - Rowset – Easier to use than resultset. Derives properties of resultset. Also, adds functionalities like updatability and java beans support.
        - 2 types
            - Connected rowset – keep the connection alive throughout its life span
            - Disconnected rowset – makes connection only to read the resultset.
    - JDBC also supports large data types – BLOB, CLOB, XML.
    - JDBC supports calling stored procedures
    - JDBC supports calling DISTINCT & Structured types.

JMX

- JMX – Java Management Extensions
    - Simple way of managing resources such as applications, devices and services. It can also be used to manage / monitor JVM.
    - The object which is used for instrumenting the given resource is called MBean. These MBeans are registered in a core managed object server called MBean server.
    - Instrumentation of a resource handled by mBeans which are registered to a MBeanServer which is a part of JMX agent which allows remote management.
    - JConsole is an app that is bundled with jdk that helps in instrumenting JVM. JConsole is JMX implementation.
    - MBean are standard java bean which represents a device, an app or a resource. Mban agent can be written and deployed with JConsole.
    - MX beans could generate notifications.

JNDI

- JNDI – API for accessing naming & directory services. Included in JavaSE.
    - Package – javax.naming
    - Architecture
        - Java ships SPI for LDAP, DNS, RMI & Corba
    - Javax.naming.Context is the core interface for lookup/binding, etc. JNDI provides the InitialContext which is the root for lookup.
    - Javax.naming.Directory provides APIs to access directories like LDAP. DirContext is the root for it.
    - LDAP specific interface available with javax.naming.ldap. LdapContext available.
    - JNDI supports Events like object added/removed, etc. Provides API for it.
    - JNDI – typical steps
        - Mention the service provider
        - Mention the URL & other parameters
        - Create an initial context.
        - Do lookup using initial context
    - JNDI can be used to store Java objects.

Annotations

- Reference https://docs.oracle.com/javase/tutorial/java/annotations/predefined.html
- Annotations used by language
    - Override
    - Deprecated
    - Suppress warning - unchecked, deprecation

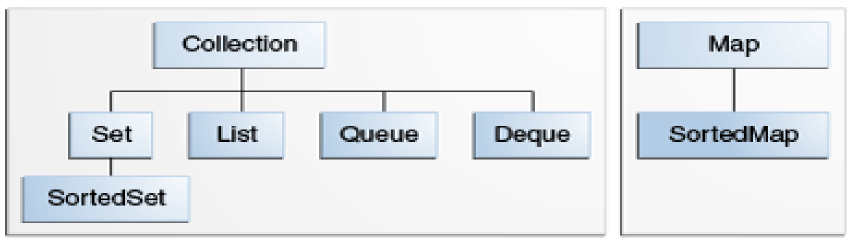
- Functional interface
- SafeVarArgs

- Annotations that apply to other annotations
  - Retention (source, class, run time) - annotation specifies how the marked annotation is stored.
  - Documented - Annotation indicates that whenever the specified annotation is used those elements should be documented using the Javadoc tool.
  - Target - Annotation marks another annotation to restrict what kind of Java elements the annotation can be applied to.
  - Repeatable
  - Inherited.
- Writing custom annotation - Start with @interface. @interface tells Java that it's a custom annotation.
  - Example http://www.javatpoint.com/custom-annotation
- Before the Java SE 8 release, annotations could only be applied to declarations. As of the Java SE 8 release, annotations can also be applied to any *type use*.

Collections

- Collection framework in java comprises of interfaces and implementation for data structures and algorithms such as sorting, searching.
- Key interfaces
  - Collection – Set of basic APIs for basic operations, traversal, bulk operations, array conversion, element access operations.
  - Set – Cannot have duplicates.
    - Key set implementations
      - HashSet – Popular/performance oriented. Not ordered. Constant time access O(1). Set in the initial capacity twice as much you expect it to grow. Capacity is usually the power of 2. There is also a load factor which is best to leave it at default.
      - TreeSet – Uses tree/Ordered(insertion order)/Slow. Accessing – O(log n).
      - LinkedHashSet-Uses linked list/Insertion ordered/Slightly costly than HashSet. https://www.cs.wcupa.edu/rkline/ds/hash-sets.html#linked-hash
      - Special purpose set implementations
        - CopyOnWriteArraySet – Take a copy of the array every time a write operation is done. So, thread safe. Use it only when the write operation is very rare.
        - EnumSet – High performance set operations on enum types.
  - List – Ordered collection. Precise control over the insertion / deletion order. Supports positional access, search, list specific iteration.
    - Key implementations
      - ArrayList – Non synchronized/ better performance. Access constant time. Optional parameter – initial capacity.
        - Allows null
      - LinkedList – better performance only in certain cases. Access linear time.
      - Vector - Synchronized / costly. Faster than Collections.SynchronizedList
      - Special purpose implementations
        - CopyOnWriteArrayList – Similar to CopyOnWriteArraySet.
    - Set() & get() are key methods for insertion and accessing.
  - Map – Key/Value pair. Each key can map to only one value. Key cannot be duplicated. Doesn't inherit collection interface.
    - Basic operations – Put & Get.
    - Map doesn't provide iterators but collection views.
    - Use Map.Entry to iterate over a map.
    - 3 map implementations
      - Hashmap
      - Treemap – Sorted map.
      - LinkedHashmap – Order of insertion preserved.
      - Hashtable – retrofitted. Synchronized.
    - MultiMap concept – Single key, multiple values. Simple use a List for the value.
    - Special purpose map
      - EnumMap – use it when key is a enum type
      - WeakHashMap – Stores weak references. It allows a key-value pair to be garbage collected when it is no longer referenced outside the map.
      - IdentityHashMap.
    - Concurrent maps
      - ConcurrentHashMap – high performance version of hash table. Implementation of java.util.concurrent.concurrentMap
  - Queue – FIFO
    - Offer() to insert, peek() to examine and poll() to remove.
    - Mostly stores in FIFO order except priority queue which orders based on the priority.
    - Element at the head will be removed first.
    - Doesn't allow null
    - Implementations
      - LinkedList
      - PriorityQueue – Uses heap data structure.
      - ConcurrentImplementations
        - BlockingQueue.
  - Deque – FIFO & LIFO. Pronounced as Deck.
    - Double ended queue – Supports insertion & removal at both ends.
    - Implementations
      - ArrayDeque
      - LinkedList

- Concurrent
  - LinkedBlockingDeque



- SortedSet – Ordered according to the natural ordering of element.
  - Possible to pass a Comparator while creating SortedSet.
- SortedMap – Ordered according to the natural ordering of key.

- Sorting
  - Collection.Sort() works only if the object to be sorted implements Comparator interface. String, Integer, Date implements comparator interface. If sorting is attempted for objects which doesn't implement Comparable interface ClassCastException will be thrown.
  - Use Comparator interface if you wanted to implement sorting for objects that haven't implemented Comparable. Collections.Sort(collection, Comparitor).
  - Both of the above return –ve,0 or +ve depending on the arugument less than, equal or more than the object compared with.
- General recommendations
  - Use general purpose collection – ArrayList, HashMap, HashSet.
  - Avoid using synchronized versions – Vector or HashTable. In case of multithreaded scenarios, use the newly introduced wrapper implementations in java.utils.concurrent package.
- Wrapper implementations – Decorator pattern.
  - Synchronization wrappers
    - Collections class provides static methods for synchronizing any general purpose collection.
  - Unmodifiable wrappers
    - These wrappers makes the underlying collection immutable
- Convenience implementations
  - Arrays.asList()
  - Empty Set
  - From Java 9, we have factory methods to create unmodifiable collection conveniently for the given small set of elements. Example - Set.of("One", "Two", "Three");
- Algorithms
  - Collections.Sort() – Merge sort
  - Collections.Shuffle – Just opposite to sort.
  - Collections.binarySearch
- For customizing collections, use the abstract collection classes.

Helpful links

- Java changes between 8 and 16. https://ondro.inginea.eu/index.php/new-features-in-java-versions-since-java-8/#language-features
- Java SE tutorials - https://docs.oracle.com/javase/tutorial/