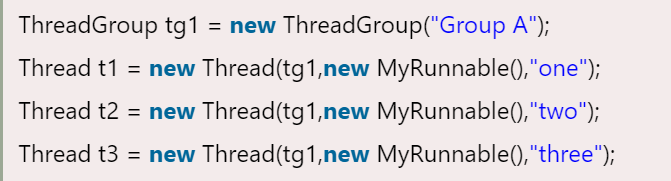
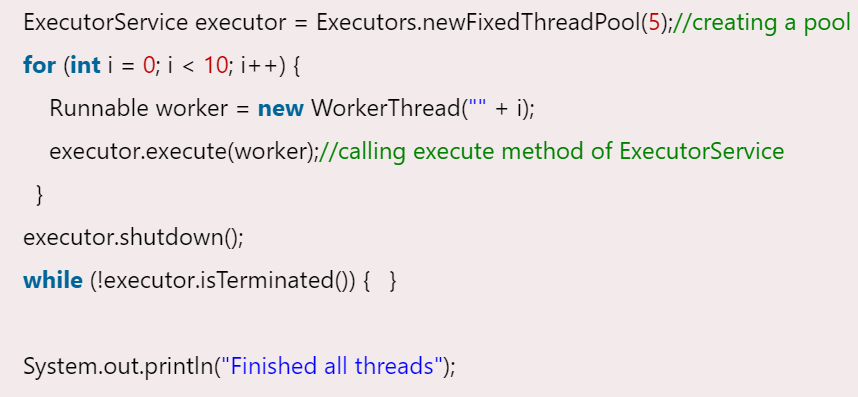
1. Class and Objects
   1. Principles of object - oriented programming
      1. Abstraction
         1. Hiding internal details and showing only required features.
      2. Encapsulation
         1. Encapsulate everything in single object
      3. Inheritance
         1. specialization
      4. Polymorphism
         1. Generalization
2. Instance initializer block
   1. Instance Initializer block is used to initialize the instance data member. It run each time when object of the class is created.
   2. The instance initializer block is created when instance of the class is created.
   3. The instance initializer block comes in the order in which they appear.
   4. Call order: Parent Class Initializer, Parent class constructor, Child Instance initializer and Child class constructor
3. Abstract Class
   1. We cannot create object of abstract classes but reference to abstract class is allow
   2. We can create constructor of abstract class and this is used to initialize data member of abstract class. It will be called when child class object is created.
   3. Abstract class can have 0 or any number of abstract method (method with no body and declare with abstract keyword). And have any number of non-abstract method.
   4. If a class contains at least 1 abstract method, then class should be declared as abstract as well.
   5. We cannot make abstract class as final because if we make final then we cannot extends them in child class.
   6. We cannot make abstract method as final.
   7. We cannot make abstract class and method as static.
   8. For Child class either override abstract method or declare as abstract if it extends abstract class.
   9. An abstract class can extend another abstract class.
   10. An abstract class can implements interface and no need to override abstract method, it will override directly in child class where child class extends abstract class.
   11. The static block and initializer is allowed in abstract class.
   12. A sub class which extends concrete class can be abstract.
   13. A sub class cannot only override concrete method, should override all abstract method.
   14. With abstract class we can only use public or protected modifier.
4. Interfaces
   1. In Interface all methods are abstract and public by default, so no need to add these keywords. And all data members are static and final by default.
   2. We can implement multiple interfaces, use to achieve multiple inheritance
   3. We cannot create object of interface but reference to interface is allow
   4. In interface, we can have only abstract method. And non-abstract method is not allowed.
   5. We can create non-abstract method using static and default keywords. Static methods can be access using Interface name and default method can be access using child class object.
   6. If interface have one default method and that also override in child class, then when we call that method using object then it will call child class method.
   7. In interface we can create private method which can be called from static and default methods. And these methods can only be access inside interface.
   8. In interface we can create private static method which can be called from static methods only.
   9. Conflict resolution rules for inherited default methods
      1. Rule 1 – Classes take higher precedence than interfaces
      2. Rule 2 – Derived interfaces or sub-interfaces take higher precedence than the interfaces higher-up in the inheritance hierarchy
      3. Rule 3 – In case Rule 1 and Rule 2 are not able to resolve the conflict then the implementing class has to specifically override and provide a method with the same method definition (<InterfaceName>.super.<methodName>())
5. Inner Classes
   1. Inner class can access members (data members and member functions) of outer class.
   2. Outer class can access members of inner class by creating object.
   3. We can create object of inner class outside outer class as well using Outer class object.
   4. Local Inner Class
      1. Inner class inside method is called Local Inner class
      2. This class can access inside that method only.
      3. We can make local inner class as abstract or final only.
   5. Anonymous Class or Anonymous Inner Class
      1. We can create Anonymous class of Interface and Abstract Class.
   6. Static Inner Class
      1. We can create object of static inner class with Outer class object using Outer class name.
      2. Static Inner class can access only static member of outer class.
6. Static and Final keyword
   1. The static keyword (static variable, static method, static inner class and static block)
      1. The static keyword used to define data related to a class instead of object.
      2. The static member can be accessed using Class name without object.
      3. The static method can only access static methods.
      4. We cannot use this and super keyword inside static method because these represent to object.
      5. We can create Inner class as static only, outer class static is not allowed.
      6. The static blocks
         1. It is used to initialize static data members.
         2. If we have multiple static blocks, then it run in order in which it is defined.
         3. It only accesses static data members; non-static data members are not allowed.
         4. It run while class loading.
         5. Call order: Parent class static block, Child class static block, Parent Class Initializer, Parent class constructor, Child Instance initializer and Child class constructor
   2. The final keyword (final variable, final method, and final class)
      1. The final variable is constant, means it cannot change once it is initialized.
      2. The final variable can be initialized inside constructor or initializer or during declaring.
      3. The final variable can be created without any value assigned, but if we try to access them then it will throw.
      4. The final method cannot be override in child class, but we can call these methods using child class.
      5. The final class cannot be extends in any child class.
   3. Singleton Class
      1. Class with one object.
      2. Step to create singleton class
         1. Make constructor as private
         2. Create static object of singleton class and assign null.
         3. Create getInstance() static method which create object and return object.
7. Exception Handling
   1. The java.lang.Throwable class is the root class of Java Exception hierarchy inherited by two subclasses: Exception and Error.
   2. Types of Java Exceptions
      1. Checked Exception: The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions.
      2. Unchecked Exception: The classes that inherit the RuntimeException are known as unchecked exceptions.
      3. Error: Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.
   3. Multiple Catch block: it checks from top to bottom and handle the exception run that catch block only.
   4. Finally block
      1. Java finally block is a block used to execute important code such as closing the connection, etc.
      2. Java finally block is always executed whether an exception is handled or not after catch block if catch block is here or it called after try.
   5. Throw an exception
      1. throw new AritheticException(“message”)
   6. Throws used to declare the exception that might be occur.
   7. Custom Exception
      1. Extends Exception or RuntimeException class to create custom exception and in constructor take message and pass into super().
   8. Try with resource (Auto close)
      1. If any resource which should be closed after using it (like file etc.) then in that cases we use, try with resource.
      2. Previously we close using finally block to close all resource after using it. Now we can use try with resource which will automatically close the resource after executing try block.
8. Multithreading
   1. A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.
   2. Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.
   3. Multithreading in Java is a process of executing multiple threads simultaneously.
   4. Create Thread
      1. By extending Thread class and override run() method
      2. By implementing Runnable interface and override run() method. It can be used when class already extends another class then we cannot extends Thread class.
   5. Lifecycle of a Thread
      1. New, Runnable (or ready), Running, Non-Runnable(Block) and Terminated
      2. The thread is in runnable state after calling of start() method, but the thread scheduler has not selected it to be the running thread.
      3. A thread is in terminated or dead state when its run() method exits.
   6. Thread Scheduler
      1. Thread scheduler in java is the part of the JVM that decides which thread should run.
      2. The thread scheduler mainly uses preemptive(high priority) or time slicing scheduling to schedule the threads.
   7. What if we call run() method directly instead start() method?
      1. Each thread starts in a separate call stack.
      2. Invoking the run() method from main thread, the run() method goes onto the current call stack rather than at the beginning of a new call stack.
   8. Priority of a Thread
      1. Default priority of a thread is 5 (NORM\_PRIORITY).
      2. The value of MIN\_PRIORITY is 1 and
      3. The value of MAX\_PRIORITY is 10.
   9. ThreadGroup
      1. Java provides a convenient way to group multiple threads in a single object. In such way, we can suspend, resume or interrupt group of threads by a single method call.



* 1. Daemon Thread
     1. Daemon thread in java is a service provider thread that provides services to the user thread. Its life depends on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.
     2. It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.
     3. It is a low priority thread.
     4. Example: garbage collector.
  2. Thread methods
     1. sleep() method
        1. The sleep() method of Thread class is used to sleep a thread for the specified amount of time.
        2. The sleep() method doesn't release the lock.
     2. join() method
        1. The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.
     3. interrupt() method
        1. This method of thread class is used to interrupt the thread.
     4. yield() method
        1. it can stop the currently executing thread and will give a chance to other waiting threads of the same priority.
        2. It’s a static method
  3. Thread Pool
     1. Thread pool represents a group of worker threads that are waiting for the job and reuse many times.
     2. a group of fixed size threads are created. A thread from the thread pool is pulled out and assigned a job by the service provider. After completion of the job, thread is contained in the thread pool again.

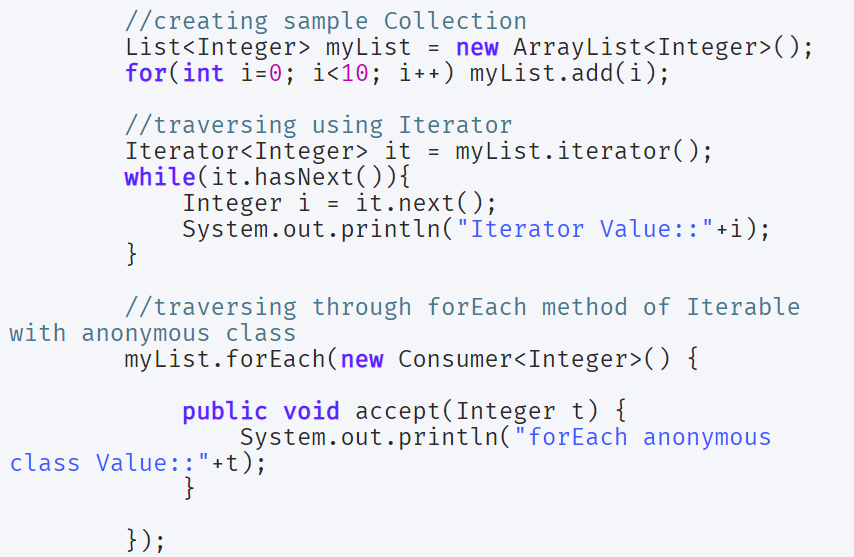


* 1. Garbage Collection
     1. In java, garbage means unreferenced objects.
     2. it is a way to destroy the unused objects.
     3. How can an object be unreferenced?
        1. By nulling the reference
        2. By assigning a reference to another
        3. By anonymous object
     4. finalize method
        1. finalize is the method in Java which is used to perform clean up processing just before object is garbage collected
     5. The System.gc() method is used to invoke the garbage collector to perform cleanup processing.
  2. Synchronization
     1. Synchronization in java is the capability to control the access of multiple threads to any shared resource.
     2. The main key points in synchronization
        1. Resource sharing – resource which shared to multiple thread.
        2. Critical Section – code of a thread which access shared resources.
        3. Mutual Exclusion – preventing thread to access shared resource by other thread. Only thread access shared resource
        4. We can achieve thread mutual exclusion two ways
           1. Locking/Mutex, Semaphore, and Monitor
           2. Race Condition, and Inter thread communication
        5. Locking/Mutex
           1. Using mutex as variable for shared resource and whenever any thread access shared resource it changes the value of mutex from 0 to 1 and after access change from 1 to 0.
           2. In case user is taking care of mutual exclusion
           3. Some time it creates problem multiple try to check mutex value at same time. Both enter in critical section. That why it should handle by operating system to manage mutual exclusion.
        6. Semaphone
           1. It has two method wait() and signal() with mutex and it these method taking care by operating system.
           2. First thread check value of mutex and call wait() which change mutex from 0 to 1
           3. And signal() change mutex from 1 to 0
           4. It put thread in block queue if check value of mutex and it is 1.
        7. Monitor (Used in java)
           1. Shared resource with locking mechanism. (Shared resource + mutex + read() + write() and block queue)
     3. Multithreading with Monitor
        1. Concept of Lock in Java
           1. Synchronization is built around an internal entity known as the lock or monitor. Every object has an lock associated with it.
           2. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.
        2. synchronized method
           1. Synchronized method is used to lock an object for any shared resource.
        3. Synchronized Block
           1. Synchronized block can be used to perform synchronization on any specific resource of the method.
           2. Java synchronized block is more efficient than Java synchronized method.
           3. It can define inside some function so no need to call
           4. Synchronized block on a class lock: synchronized (Table.class) {}
        4. Static Synchronization
           1. If you make any static method as synchronized, the lock will be on the class not on object.
     4. Deadlock
        1. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock.
        2. Avoid using join() or access shared resource in correct order.
     5. Inter-thread communication
        1. Producer and Consumer problem. Single producer write something and then single consumer read that.
        2. Race Condition: Single Producer and Multiple Consumer. One Producer write and then One Consumer read then it notify any one can access shared resource means race to access it. And they should access in order like First Producer write then only consumer read.
        3. This is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.
        4. It is implemented by following methods of Object class
           1. wait(): Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed. The wait() method releases the lock.
           2. notify() : Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened.
           3. notifyAll() : Wakes up all threads that are waiting on this object's monitor.

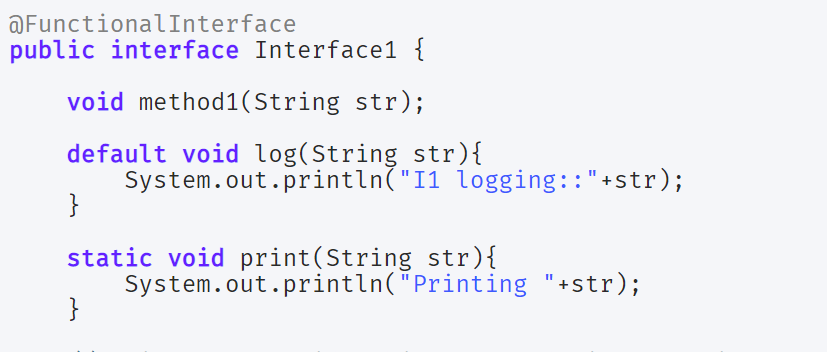
1. Serialization and Deserialization
   1. Serialization in Java is a mechanism of writing the state of an object into a byte-stream. It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies.
   2. The reverse operation of serialization is called deserialization where byte-stream is converted into an object.
   3. The serialization and deserialization process is platform-independent, it means you can serialize an object on one platform and deserialize it on a different platform.
   4. We must have to implement the Serializable interface for serializing the object.
   5. The Serializable interface must be implemented by the class whose object needs to be persisted.
   6. The String class and all the wrapper classes implement the java.io.Serializable interface by default.
   7. Externalizable in java
      1. The Externalizable interface provides the facility of writing the state of an object into a byte stream in compress format. It is not a marker interface.
      2. It provides two method: writeExternal and readExternal
   8. Java Transient Keyword
      1. If you don't want to serialize any data member of a class, you can mark it as transient.
   9. SerialVersionUID
      1. The serialization process at runtime associates an id with each Serializable class which is known as SerialVersionUID. It is used to verify the sender and receiver of the serialized object. The sender and receiver must be the same. To verify it, SerialVersionUID is used. The sender and receiver must have the same SerialVersionUID, otherwise, InvalidClassException will be thrown when you deserialize the object.
2. File System
   1. Streams
      1. A stream is a sequence of data (or flow of data) which send/receive to or from I/O devices. In Java, a stream is composed of bytes.
      2. If speed of sending data and receiving data is not same, then we use buffer to maintain speed both the side and hold data for some time.
      3. Byte Stream
         1. Stream flow in byte format
         2. Byte size 1 byte.
         3. InputStream Vs OutputStream
            1. InputStream/OutputStream class is an abstract class.
            2. Java application uses an input/output stream to read/write data from a I/O device
            3. FileInputStream/FileOutputStream – for reading from file.
            4. BufferedInputStream/BufferedOutputStream – uses internally buffer to store data
            5. SequenceInputStream – read from multiple streams. It reads data sequentially (one by one).
            6. ByteArrayInputStream/ByteArrayOutputStream – read data into a byte array.
            7. DataInputStream/DataOutputStream - It allows an application to read primitive Java data types from the input stream in a machine-independent way.
      4. Character Stream
         1. Stream flow in character format
         2. Character size 2 byte.
         3. Reader
            1. It is an abstract class for reading to character streams.
         4. Writer
            1. It is an abstract class for writing to character streams.
         5. FileWriter vs FileReader
         6. BufferedWriter vs BufferedReader
         7. CharArrayWriter vs CharArrayReader
         8. PrintWriter vs PrintReader: automatically call flush()
   2. File
      1. creatNewFile(): return true if file created else false
      2. mkdir(): create folder
      3. listFiles(): return array of files.
      4. isDirectory(): check it is file or folder
3. Java String
   1. String is immutable in nature. So once data set in string that can’t be changed later.
   2. Spring Pool
      1. Java String poll refers to collection of strings which are stored in heap memory
      2. Before creating a string literal first look for string with same value in the String pool. If found it returns the reference else, it creates a new string in the pool and returns the reference (hashcode).
   3. String objects are stored in a special memory area known as the "string constant pool".
   4. String is thread safe.
   5. Two ways to create string object
      1. By String Literal
         1. Java String literal is created by using double quotes.
      2. By new keyword
         1. String object created using new keyword it always creates a new object in heap memory (not in string constant pool)
   6. Create Immutable Class
      1. Instance Variable of the class is final i.e. After initializing in constructor, it cannot be changed.
      2. The class is final so we cannot create subclass.
      3. There is not setter method so no option to change instance variable.

**Java 8**

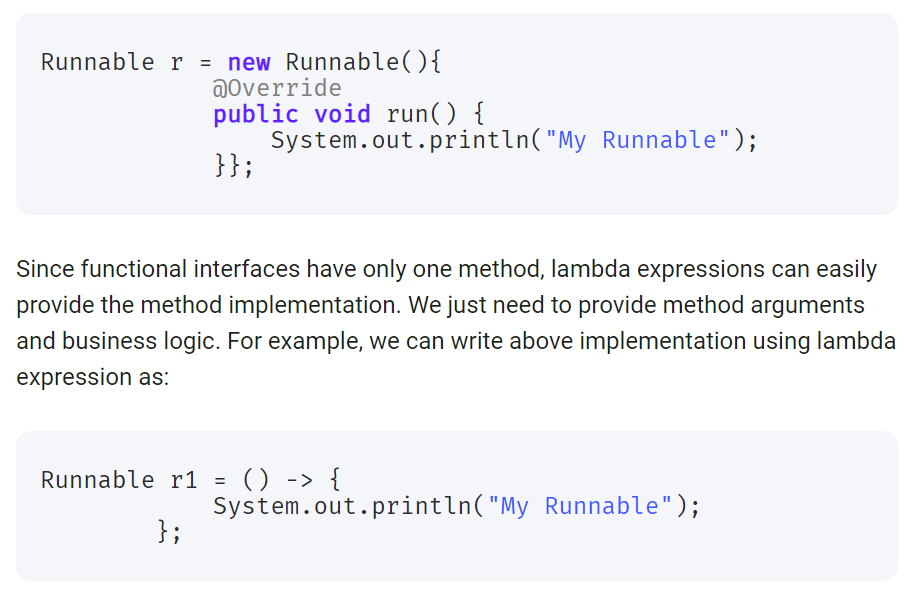
1. forEach() method in Iterable interface
   1. Whenever we need to traverse through a Collection, we need to create an Iterator whose whole purpose is to iterate over, and then we have business logic in a loop for each of the elements in the Collection.
   2. Java 8 has introduced forEach method in java.lang.Iterable interface so that while writing code we focus on business logic. The forEach method takes java.util.function.Consumer object as an argument, so it helps in having our business logic at a separate location that we can reuse.



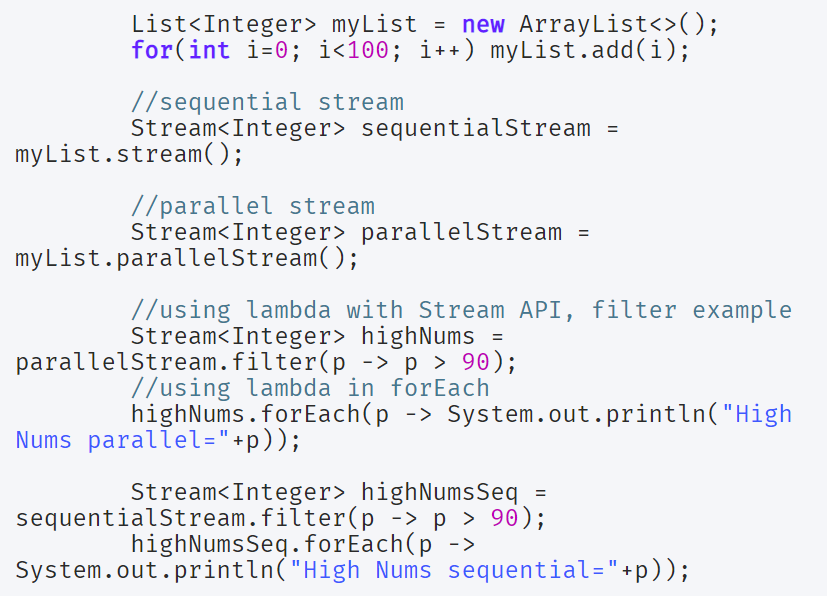
1. default and static methods in Interfaces
   1. Java 8, interfaces are enhanced to have a method with implementation. We can use default and static keyword to create interfaces with method implementation.



1. Functional Interfaces and Lambda Expressions
   1. Functional interfaces are a new concept introduced in Java 8. An interface with exactly one abstract method becomes a Functional Interface. We don’t need to use @FunctionalInterface annotation to mark an interface as a Functional Interface.
   2. java.lang.Runnable with a single abstract method run() is a great example of a functional interface.
   3. One of the major benefits of the functional interface is the possibility to use lambda expressions to instantiate them.
   4. A new package java.util.function has been added with bunch of functional interfaces to provide target types for lambda expressions and method references.
   5. Some of the useful java 8 functional interfaces are Consumer, Supplier, Function and Predicate.



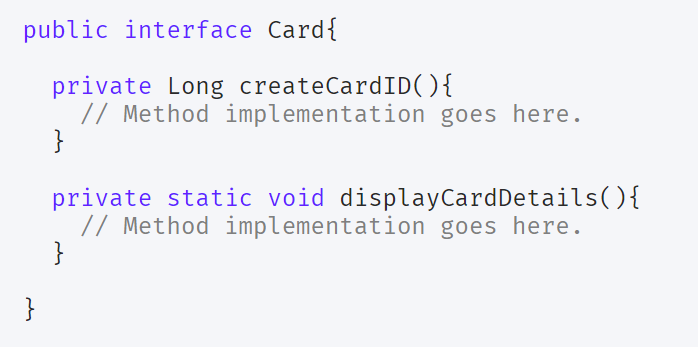
1. Java Stream API for Bulk Data Operations on Collections
   1. A new java.util.stream has been added in Java 8 to perform filter/map/reduce like operations with the collection. Stream API will allow sequential as well as parallel execution.
   2. Collection interface has been extended with stream() and parallelStream() default methods to get the Stream for sequential and parallel execution.



1. Collection API improvements
   1. We have already seen forEach() method and Stream API for collections.
   2. Some new methods added in Collection API are:
      1. Iterator default method forEachRemaining(Consumer action) to perform the given action for each remaining element until all elements have been processed or the action throws an exception.
      2. Collection default method removeIf(Predicate filter) to remove all of the elements of this collection that satisfy the given predicate.
      3. Collection spliterator() method returning Spliterator instance that can be used to traverse elements sequentially or parallel.
      4. Map replaceAll(), compute(), merge() methods.
2. Concurrency API improvements
3. Java IO improvements
4. Java Time API

**Java 9**

1. Private methods in Interfaces
   1. In Java 8, we can provide method implementation in Interfaces using Default and Static methods. However we cannot create private methods in Interfaces.
   2. From Java SE 9 onwards, we can write private and private static methods too in an interface using a ‘private’ keyword.



1. Factory Methods for Immutable List, Set, Map and Map.Entry
   1. introduced some convenient factory methods to create Immutable List, Set, Map and Map.Entry objects.
   2. These utility methods are used to create empty or non-empty Collection objects.

