```
why will go for collections?
**************************
Array will support primitive and does not support objects
size is fixed in Array
Array will support homogeneous data.

collections
***********
collections will support objects
size is growable in nature
collections will support homogeneous as well as heterogeneous data.

Stream API
**********
Stream API is used to process the data in sequnce manner
Stream API does not store any data.
Stream API will capture data from source
source may be arrays or collections
Stream API=intermediate opertaions + Terminal Operations

why stream API?
***************
Req:remove duplicates from ArrayList?
************************************
```

why stream API?
****
Req:remove duplicates from ArrayList without Stream API?
****

```
public class Client {

	public static void main(String[] args)
	{
		List<String> lst=new ArrayList<String>();
		List<String> uniqueList=new ArrayList<String>();
		lst.add("A");
		lst.add("B");
		lst.add("C");
		lst.add("A");
		lst.add("B");
		for(String s:lst)
		{
			if(!uniqueList.contains(s))
			{
				uniqueList.add(s);
			}
		}
		System.out.println(uniqueList);
	}
}
```

Req:remove duplicates from ArrayList without Stream API?
****

```
	}
}
Req:remove duplicates from ArrayList with Stream API?
**************************************************
public class Client {

	public static void main(String[] args)
	{
		List<String> lst=new ArrayList<String>();
		List<String> uniqueList=new ArrayList<String>();
		lst.add("A");
		lst.add("B");
		lst.add("C");
		lst.add("A");
		lst.add("B");
		lst.stream().distinct().forEach(
				(x)->
				{
					System.out.println(x);
				});
	}
}

we will resove complexity by using Stream API.
we will reduce application code by using Stream API.
We will improve application performance by using Stream

**********Real**********************************************
```

Functional Programming?
****
The way of promoting the programming with functions is called as Functional Programming
Functional Programming introduced from java8 onwards.
function
****
developer provide some i/p to function
Function will capture i/p and process that i/p,finally gies some response
Functional Programming is an fundamental approach in problem solving
We will reduce code by using Functional Programming.
we will achieve Functional Programming by using
	Functional interface
	Lambda Expressions
	Methods References
Structural Programming vs Functional Programming?
****
Structural Programming will follow top-down approach where as Functional Programming follow bottom-up approach
Structural Programming will focus on process where as Functional Programming will focus on data
Functional Programming provides more secure rather than Structural Programming

Stream API
****
Stream API is used to achieve Functional Programming.
Stream API=Intermediate operations + Terminal Operations
Stream API will not modify the data why because Stream API will not store any data.
Stream API will capture data from source
source may be Arrays or Collections

****
```
public class Client {

	public static void main(String[] args)
	{
		int[] a= {10,20,30};
		//convert Array into Stream
		long count = Arrays.stream(a).count();
		System.out.println(count);
	}
}
```

Array iteration?
****
```
public class Client {

	public static void main(String[] args)
	{
		int[] a= {10,20,30};
		Arrays.stream(a).forEach(
				(x)->
				{
					System.out.println(x);
				});
	}
}
```

Array |

****
```
public class Client {

	public static void main(String[] args)
	{
		int[] a= {10,20,30};
		//convert Array into Stream
		long count = Arrays.stream(a).count();
		System.out.println(count);
	}
}
```

Array iteration?
****
```
public class Client {

	public static void main(String[] args)
	{
		int[] a= {10,20,30};
		Arrays.stream(a).forEach(
				(x)->
				{
					System.out.println(x);
				});
	}
}
```

Array |

```
	public static void main(String[] args)
	{
		int[] a= {10,20,30};
		Arrays.stream(a).forEach(
				(x)->
				{
					System.out.println(x);
				});
	}
}
```

Array Sorting?

****

```
public class Client {

	public static void main(String[] args)
	{
		int[] a= {10,30,20};
		Arrays.stream(a).sorted().forEach(
				(x)->
				{
					System.out.println(x);
				});
	}
}
```

How to

```
Stream API id divided into 2 types
        sequential stream
        parallel stream
sequential stream
*****************
by using Arrays.stream() or CollectObj.stream()
        we will create sequential stream
isParallel()
        returns boolean value
        false->stream is sequential
        true-->stream is parallel
public class Client {

        public static void main(String[] args)
        {
                int[] a= {10,11,12,13};
                boolean flag = Arrays.stream(a).isParallel();
                System.out.println(flag);
        }
}

parallel stream
***************
```

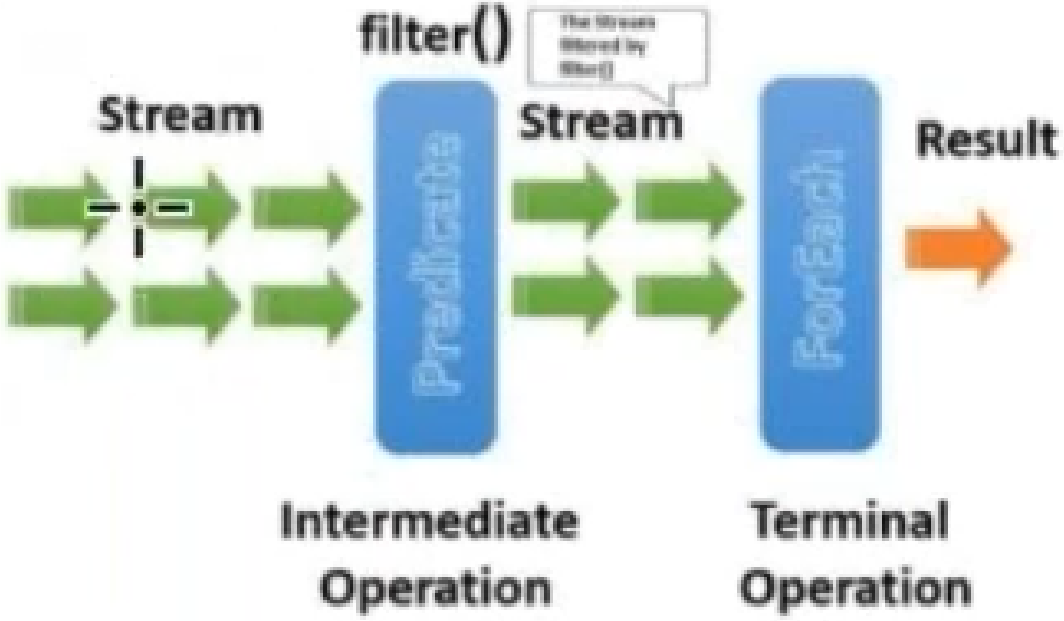
Intermediate operations
****
Intermediate operations will produce stream
filter(),map().sorted().distinct(),limit() and skip() are examples for Intermediate operations

Terminal Operations
****
Intermediate operations -->stream-->Terminal Operations
Terminal Operations will produce final result
forEach(),count(),min(),max(),toArray(),reduce(),fndFirst(),findAny(),allMatch(),noneMatch() comes under Terminal Operatio

Note
*****
we will apply
	any number of intermediate operations on stream
	only one Terminal Terminal operation

Collections-->Stream -->apply operations

Collections
	List
		ArrayList
		LinkedList
		Vector
		Stack
		CopyOnWriteArrayList


filter()
Stream
Stream
Result
Predicate
ForEach
Intermediate Operation
Terminal Operation

```
                                        System.out.println(x);
                                });
        }
}

Req:remove duplicates from ArrayList?
**************************************
public class Client {

        public static void main(String[] args)
        {
                List<String> lst=new ArrayList<String>();
                lst.add("c");
                lst.add("a");
                lst.add("b");
                lst.add("c");
                lst.add(null);
                lst.stream().distinct().filter(x->x!=null).forEach(
                                (x)->
                                {
                                        System.out.println(x);
                                });
        }
}
```

Terminal operations
****
example on reduce()
****
reduce() will reduce data into single element

```
public class Client {

	public static void main(String[] args)
	{
		List<Integer> lst=new ArrayList<Integer>();
		lst.add(1);
		lst.add(2);
		lst.add(3);
		lst.add(4);
		Optional<Integer> reduce = lst.stream().reduce((a,b)->a+b);
		System.out.println(reduce.get());
	}
}
```

count()
****

```
forEach()
*********
forEach() will take consumer as i/p,process one by one element and finally displayed to end-user

public class Client {

	public static void main(String[] args)
	{
		List<Integer> lst=new ArrayList<Integer>();
		lst.add(1);
		lst.add(2);
		lst.add(3);
		lst.add(4);
		lst.stream().forEach(
				(x)->
				{
					System.out.println(x);
				});

	}
}
```


****

toArray()
****
toArray() is used to convert stream to array

```
public class Client {

	public static void main(String[] args)
	{
		List<Integer> lst=new ArrayList<Integer>();
		lst.add(1);
		lst.add(2);
		lst.add(3);
		lst.add(4);
		Object[] obj = lst.stream().toArray();
		for(Object o:obj)
		{
			System.out.println(o);
		}
	}
}
```

minimum element from given ArrayList?

****

min()

```
public class Client {

	public static void main(String[] args)
	{
		List<Integer> lst=new ArrayList<Integer>();
		lst.add(1);
		lst.add(2);
		lst.add(3);
		lst.add(4);
		Optional<Integer> max = lst.stream().min((v1,v2)->v1.compareTo(v2));
		System.out.println(max.get());
	}
}
```

collect()
****

collect() is Terminal operator which is used to convert stream into List/Set/Map back

```
public class Client {

	public static void main(String[] args)
	{
		List<Integer> lst=new ArrayList<Integer>();
		lst.add(1);
		lst.add(2);
		lst.add(3);
		lst.add(4);
		lst.add(3);
		lst.add(4);
		List<Integer> list = lst.stream().distinct().collect(Collectors.toList());
		System.out.println(list);
	}
}
```

```
List-->stream()-->stream-->Collectors.toList()-->List

Set-->stream()-->stream-->Collectors.toSet()-->Set
Map-->stream()-->stream-->Collectors.toMap()-->Map

findFirst(),findAny(),anyMatch(),allMatch() & noneMatch()
**********************************************************
findFirst()
        findFirst() returns first element from given Stream
public class Client {

        public static void main(String[] args)
        {
                List<Integer> lst=new ArrayList<Integer>();
                lst.add(1);
                lst.add(2);
                lst.add(3);
                lst.add(4);
                Optional<Integer> findFirst = lst.stream().findFirst();
                System.out.println(findFirst.get());
        }
}
```

```
anyMatch(),allMatch() & noneMatch()
**********************************
        anyMatch()
                will take predicate an i/p
                return true if any element satisfy given condition else return false

public class Client {

        public static void main(String[] args)
        {
                List<String> lst=new ArrayList<String>();
                lst.add("We");
                lst.add("are");
                lst.add("going");
                lst.add("conduct");
                lst.add("workshop");
                lst.add("on stream api");
                lst.add("From");
                lst.add("Sreenu Tech");
                boolean f1 = lst.stream().anyMatch(x->x.startsWith("Suman"));
                System.out.println(f1);
        }
}
```

File Edit Format View Help

```
        nonMatch() return boolean value
        noneMatch() will return true if none of element satisfy given condition
        suppose any one element satisfy given condition then noneMatch() return false
public class Client {

        public static void main(String[] args)
        {
                List<String> lst=new ArrayList<String>();
                lst.add("We");
                lst.add("are");
                lst.add("going");
                lst.add("conduct");
                lst.add("workshop");
                lst.add("on stream api");
                lst.add("From");
                lst.add("Sreenu Tech");
                boolean f1 = lst.stream().noneMatch(x->x.startsWith("w"));
                System.out.println(f1);
        }
}
```

AllMatch()
****
allMatch()
	will take predicate as i/p
	allMatch() return boolean value
	allMatch() will return true if all elements should satisfy given condition
	suppose any one element doesn't satisfy given condition then allMatch() return false

```
public class Client {

	public static void main(String[] args)
	{
		List<String> lst=new ArrayList<String>();
		lst.add("We");
		lst.add("are");
		lst.add("going");
		lst.add("conduct");
		lst.add("workshop");
		lst.add("on stream api");
		lst.add("From");
		lst.add("Sreenu Tech");
		boolean f1 = lst.stream().allMatch(x->x.startsWith("w"));
		System.out.println(f1);
	}
}
```

Map
***
suppose we want to store data in the form of key&value pair then we will go for Map
key-->value==>entry
key should be unique
value should be duplicate

Gmail
        username -->key
        password-->value

keySet()
        is used to get only keys
values()
        is used to only values
entrySet()
        is used to get both key & values

mp.keySet().stream()-->stream object
mp.values().stream()-->stream object
mp.entrySet().stream()-->stream object
Getting only keys using Stream API?
****