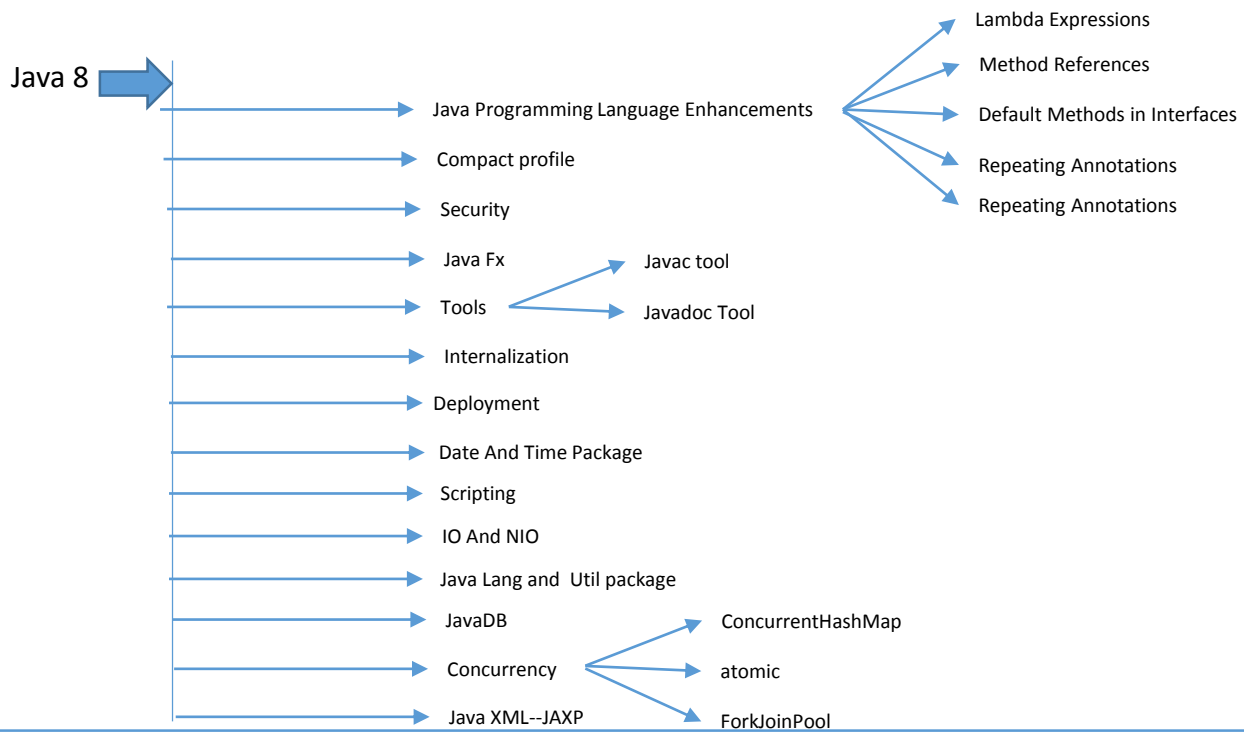

Anonymous inner Class with Java8 interfaces

```
package com.osp.java8;
public class AnonymousInnerClasses {
public static void main(String args[]){
SampleInterface sampleAnonymousClass = new
SampleInterface(){
public void printOne(){
System.out.println("print this its Anonymous  inner class
PrintOne Method");
}
public void print(){
System.out.println("print this its Anonymous  inner class");
}
};
sampleAnonymousClass.print();
sampleAnonymousClass.printOne();
sampleAnonymousClass.defaultMethod();//default method
interface utilization
}
}
```

```
package com.osp.java8;

public interface SampleInterface {
public void print();

public void printOne();
public default void defaultMethod(){
System.out.println("default Method Implementation");
}
}
```

```
package com.osp.java8;
public class SampleInterfaceImpl implements
SampleInterface {
@Override
public void print() {
System.out.println("print method in Sample Interface
Impl");
}
@Override
public void printOne() {
System.out.println("print method in Sample Interface
Impl");
}

public void defaultMedhod(){
SampleInterface.super.defaultMethod();
}
public static void main(String args[]){
SampleInterfaceImpl sample = new SampleInterfaceImpl();
sample.defaultMedhod();
}
}
```

```
package com.osp.java8;
@FunctionalInterface
public interface SampleFunctionalInterface {
public String sampleMethod(String arg1,String arg2);
default void test(){//BusnessLogic}
default void test2(){//BusnessLogic}
static void staticMethod(){//BusnessLogic}
static void staticMethod2(){//BusnessLogic}
}
We can have n number of default methods and static method in
function interfaces, but we should have only one abstract
interface, for lambda expressions @FunctionalInterface is
recommended
```

```
package com.osp.java8;
public class SampleInterfaceImpl implements SampleInterface {
@Override
public void print() {
System.out.println("print method in Sample Interface Impl");
}
@Override
public void printOne() {
System.out.println("print method in Sample Interface Impl");
}
public void defaultMedhod(){
SampleInterface.super.defaultMethod();
}
public static void main(String args[]){
        SampleInterfaceImpl sample = new SampleInterfaceImpl();
        sample.defaultMedhod();
        SampleFunctionalInterface fInterface=(String arg1,String arg2)->{
                System.out.println("Testing::::Lambda expressions "+arg1);
                System.out.println("Testing::::Lambda expressions "+arg2);
                String result=arg1+" "+arg2;
        return result;
        };
String result=fInterface.sampleMethod("Statement1- Madhava","Statement2 -Reddy");
System.out.println(result);
}
}
```

```
//Old way:
new Thread(new Runnable() {
@Override
public void run() {
System.out.println("Hello from
thread");
}
}).start();
//New way:
new Thread(
() -> System.out.println("Hello
from thread")
).start();

//Old way:
List<Integer> list =
Arrays.asList(1,2,3,4,5,6,7);
int sum = 0;
for(Integer n : list) {
int x = n * n;
sum = sum + x;
}
System.out.println(sum);
//New way:
List<Integer> list =
Arrays.asList(1,2,3,4,5,6,7);
int sum = list.stream().map(x ->
x*x).reduce((x,y) -> x + y).get();
System.out.println(sum);
```

```
(int a, int b) -> {  return a + b; }
() -> System.out.println("Hello World");
(String s) -> { System.out.println(s); }
() -> 42
() -> { return 3.1415 };
```

```
List<Integer> list =
 Arrays.asList(1, 2, 3, 4, 5, 6, 7);
list.forEach(n -> System.out.println(n));
//or we can use :: double colon operator in Java 8
list.forEach(System.out::println);
```

```java
package com.osp.java8;
public class MethodReferenceExample {
public static void main(String args[]){
UserFactory userFactory = new UserFactory(){
@Override
public User createUser(int id,String name){
return new User(id,name);
}
};
System.out.println(userFactory.createUser(1, "00name"));
}
}
```

```java
package com.osp.java8;
public class User {
int id;
String name;
public int getId() {
return id;
}
```

```java
package com.osp.java8;
public interface UserFactory {
public User createUser(int id,
String name);
}
```

```java
//Lambda Way of implementation
UserFactory userFactory1= (int id,String name)-> new
User(id,name);
System.out.println(userFactory1.createUser(2, "01name"));
```

Method Reference

```java
//Method reference Way of implementation
UserFactory userFactory2 = User::new;
System.out.println(userFactory2.createUser(2, "03name"));
```

```java
package com.osp.java8;
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
public class MethodReferenceExample {
public static void main(String args[]){
List<Integer> numbers=
Arrays.asList(1,5,2,55,3,535,64645,12124,7);
numbers.sort(new Comparator<Integer>() {
@Override
public int compare(Integer o1, Integer o2) {
return o1.compareTo(o2);
}
});
```

```java
//Lambda Way of implementation
numbers.sort((Integer o1,Integer o2)-
>o1.compareTo(o2));
System.out.println(numbers);
//Method Reference way of implementation
numbers.sort(Integer::compareTo);
}
```

```java
}
```

```java
package com.osp.java8;
import java.util.function.BiFunction;
public class MethodReferenceExample4 {
public static void main(String args[]) {
BiFunction<Integer, Integer, Integer> summation = new
BiFunction<Integer, Integer, Integer>() {
public Integer apply(Integer i1, Integer i2) {
return i1.sum(i1, i2);
}
};
System.out.println(summation.apply(10, 100));
```

```java
// Method Reference Way of implementation
BiFunction<Integer, Integer, Integer> summation1 =
Integer::sum;
System.out.println(summation1.apply(10, 20));
// Lambda Way of implementation
BiFunction<Integer, Integer, Integer> summation2 =
(Integer i1, Integer i2) -> i1.sum(i1, i2);
System.out.println(summation2.apply(2000, 3000));
}
```

```java
}
```

Concurrent Hash Map

```java
package com.osp.java8;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentMap;
public class ConcurentHahMapExample {
public static void main(String args[]){
ConcurrentMap<String,String> concurentHashMap = new
ConcurrentHashMap<String,String>();
//The ConcurrentHashMap class also implements the
ConcurrentMap interface, which contains some new
methods to provide truly atomic functionality:
concurentHashMap.putIfAbsent("10", "Ten");
concurentHashMap.putIfAbsent("9", "Nine");
concurentHashMap.remove("9", "Nine");
concurentHashMap.replace("10", "Ten..Ten");
System.out.println(concurentHashMap);
}
}
```

```java
package com.osp.java8;
import java.util.concurrent.atomic.AtomicInteger;
public class AtomicIntegerExample {
private static AtomicInteger at = new AtomicInteger(0);
static class MyRunnable implements Runnable {
private int myCounter;
private int myPrevCounter;
private int myCounterPlusFive;
private boolean isNine;
public void run() {
myCounter = at.incrementAndGet();
System.out.println("Thread " +
Thread.currentThread().getId() + "  / Counter : " +
myCounter);
myPrevCounter = at.getAndIncrement();
System.out.println("Thread " +
Thread.currentThread().getId() + " / Previous : " +
myPrevCounter);
myCounterPlusFive = at.addAndGet(5);
System.out.println("Thread " +
Thread.currentThread().getId() + " / plus five : " +
myCounterPlusFive);
isNine = at.compareAndSet(9, 3);
if (isNine) {
System.out.println("Thread " +
Thread.currentThread().getId()
+ " / Value was equal to 9, so it was updated to " +
at.intValue());
}
}
}
public static void main(String[] args) {
Thread t1 = new Thread(new MyRunnable());
Thread t2 = new Thread(new MyRunnable());
t1.start();
t2.start();
}
}
```

Atomic

```java
package com.osp.java8;

import java.time.Instant;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.Month;
import java.time.ZoneId;
import java.util.Set;
import java.time.Month;
import java.time.Period;
public class DateExample {
public static void main(String args[]){
LocalDate curDate =
LocalDate.now(ZoneId.of("Asia/Kolkata"));
System.out.println(curDate.toString());//2017-05-14
LocalDate specificDate =
LocalDate.of(1984,Month.AUGUST,26);
System.out.println(specificDate.toString());//1984-08-26
LocalTime currentTime= LocalTime.now();
System.out.println(currentTime.toString());

Set<String> zoneIds=ZoneId.getAvailableZoneIds();
for(String zoneId:zoneIds){
LocalDate zoneDate = LocalDate.now(ZoneId.of(zoneId));
LocalDateTime localDateTime =
LocalDateTime.now(ZoneId.of(zoneId));

System.out.println(zoneId+"
\t"+zoneDate+"\t"+localDateTime);
}
Instant instant = Instant.now();//Machine readable date
System.out.println(instant);
}
}
```

Time Zone

```java
 LocalDate date2 = curDate.plus(1,
ChronoUnit.MONTHS);//ToAdd One Month to Current Date
    System.out.println(date2);
```

```java
 Period period = Period.between(curDate, date2);
System.out.println("Period: " + period); //Period: P1Y2M
```

```java
 import java.time.ZonedDateTime;
    ZonedDateTime zonedDateTime =
ZonedDateTime.ofInstant(now, currentZone);
    System.out.println("Zoned date: " + zonedDateTime);
```

```java
package com.osp.java8.streams;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Stream;
public class StreamExample {
public static void main(String args[]){
List<String> list= populateList();
Map<String,String> map= populateMap();
map.forEach((k,v)->System.out.println("key-->"+k+"
Value"+v));//Displaying Map Using for Each
list.stream().forEach(x->System.out.println(x));
System.out.println("------sorting results--------------");
list.stream().sorted((String o1,String o2)-
>o1.compareTo(o2)).forEach(x->System.out.println(x));
list.stream().sorted(String::compareTo).forEach(x-
>System.out.println(x));
System.out.println("----------Map results--------------");
map.entrySet().stream().forEach(x->System.out.println(x));
map.keySet().stream().forEach(x->System.out.println(x));
map.values().stream().forEach(x->System.out.println(x));
//Stream iterate
Stream.iterate(0, i->i+1).limit(200).
forEach(System.out::println);
Stream.generate(()->{return
Math.random();}).limit(20).forEach(printVal-
>System.out.println("Random value="+printVal));
}
public static List<String> populateList(){
List<String> sampleList= new ArrayList<String>();
sampleList.add("one");
sampleList.add("four");
sampleList.add("five");
sampleList.add("three");
sampleList.add("six");
sampleList.add("two");
return sampleList;
}
public static Map<String, String>populateMap(){
Map<String,String> sampleMap= new
HashMap<String,String>();
sampleMap.put("one3","threeValue");
sampleMap.put("one5","fiveValue");
sampleMap.put("one4","fourValue");
sampleMap.put("one1","oneValue");
sampleMap.put("Two2", "Two Value");
return sampleMap;
}
}
```

```java
package com.osp.java8.streams;

import java.util.ArrayList;
import java.util.List;

public class FilterExample {
public static void main(String args[]){
List<Person> personList= populatePerson();
personList.stream().filter(person-
>person.getCountry().equals("India"))
.map((Person person)-> {
person.setId(person.getId().toUpperCase());
return "Person Id ="+person.getId()+" Person Name=
"+person.getCountry();
})
.forEach(person->System.out.println(person));
personList.stream().filter(person-
>!person.getCountry().equals("India"))
.map((Person person)-> {
person.setId(person.getId().toUpperCase());
return "Person Id ="+person.getId()+" Person Name=
"+person.getCountry();
})
.forEach(person->System.out.println(person));
}
public static List<Person> populatePerson(){
Person person1= new Person("Person1","US");
Person person2= new Person("Person2","India");
Person person3= new Person("Person3","US");
Person person4= new Person("Person4","India");
Person person5= new Person("Person5","US");
Person person6= new Person("Person6","India");

List<Person> personList= new ArrayList<Person>();
personList.add(person1);
personList.add(person2);
personList.add(person3);
personList.add(person4);
personList.add(person5);
personList.add(person6);

return personList;
}
}
```

```java
package com.osp.java8.streams;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

public class Partitioning {
public static void main(String args[]){
List<Person> personList= populatePerson();
Map <Boolean,List<Person>> personPartition =personList.stream().
            collect(Collectors.partitioningBy(t->t.getCountry().equals("India")));
personPartition.forEach((k,v)->System.out.println("Key:"+k+"  "+ ((List<Person>)v).stream().map(s->s.getId()+" :
"+s.getCountry()).collect(Collectors.joining(","))));

Map <Boolean,List<Person>> personGrouping =personList.stream().
            collect(Collectors.groupingBy(t->t.getCountry().equals("India")));
personGrouping.forEach((k,v)->System.out.println("Key:"+k+"  "+ ((List<Person>)v).stream().map(s->s.getId()+" :
"+s.getCountry()).collect(Collectors.joining(","))));
System.out.println(personList.stream().
                                collect(Collectors.groupingBy(t->t.getCountry(),Collectors.counting())));
Map<Object, List<Person>> test = personList.stream().
                                    collect(Collectors.groupingBy(t->t.getCountry()));
                test.forEach((k,v)->System.out.println(":"+k+"  "+ ((List<Person>)v).stream().map(s->s.getId()+" :
                                "+s.getCountry()).collect(Collectors.joining(","))));
}
```

```java
List<Person> indianPersonList= personList.stream().filter(person-
>!person.getCountry().equals("India")).collect(Collectors.toList());
Set<Person> indianPersonSet= personList.stream().filter(person->!person.getCountry().
                                equals("India")).collect(Collectors.toSet());

Map<Object, List<Person>> testToList = personList.stream().
            collect(Collectors.groupingBy(t->t.getCountry(),Collectors.toList()));
Map<Object, Set<Person>> testToSet = personList.stream().
            collect(Collectors.groupingBy(t->t.getCountry(),Collectors.toSet()));
```

- forEach
- map
- filter
- limit
- sorted
- ParallelStream
- Collectors
- Statistics

```java
List<Integer> numbers = Arrays.asList(3, 2, 2, 3, 7, 3, 5);
IntSummaryStatistics stats = integers.stream().mapToInt((x) -> x).summaryStatistics();

System.out.println("Highest number in List : " + stats.getMax());
System.out.println("Lowest number in List : " + stats.getMin());
System.out.println("Sum of all numbers : " + stats.getSum());
System.out.println("Average of all numbers : " + stats.getAverage());
```