JAVA 8 Features:-

Java Anonymous inner class

1. **interface** Eatable{
2. **void** eat();
3. }
4. **class** TestAnnonymousInner1{
5. **public** **static** **void** main(String args[]){
6. Eatable e=**new** Eatable(){
7. **public** **void** eat(){System.out.println("nice fruits");}
8. };
9. e.eat();
10. }
11. }

Lambda expressions :-

Lambda expression helps us to write our code in functional style. It provides a clear and concise way to implement SAM interface(Single Abstract Method) by using an expression.

Lambda expression only work on functional interface..

Functional interface are those interface which has only one abstract method.

**package** com.sayan;

**public** **interface** Student {

**void** show(**int** a,**int** b);

}

**package** com.sayan;

**public** **class** Demo {

**public** **static** **void** main(String[] args) {

Student s=(a,b)->{

System.***out***.println(a+b);

};

s.show(0, 7);

}

}

What is comparator?

Comparator is an interface where we can use our custom method for sorting.

**public** **class** Demo {

**public** **static** **void** main(String[] args) {

List<Employee> li=**new** ArrayList<Employee>();

li.add(**new** Employee(12,300));

li.add(**new** Employee(14,2500));

li.add(**new** Employee(11,600));

li.add(**new** Employee(16,1000));

li.add(**new** Employee(22,1200));

Comparator<Employee> com=**new** Comparator<Employee>() {

@Override

**public** **int** compare(Employee o1, Employee o2) {

// **TODO** Auto-generated method stub

**if**(o1.getSalary()>o2.getSalary())

**return** 1;

**else**

**return** -1;

}

};

Collections.*sort*(li, com);

**for**(Employee e:li)

System.***out***.println(e.age+" "+e.salary);

}

}

# Java Method References

Java provides a new feature called method reference in Java 8. Method reference is used to refer method of functional interface. It is compact and easy form of lambda expression. Each time when you are using lambda expression to just referring a method, you can replace your lambda expression with method reference. In this tutorial, we are explaining method reference concept in detail.

## **Types of Method References**

There are following types of method references in java:

1. Reference to a static method.
2. Reference to an instance method.
3. Reference to a constructor.

## **1) Reference to a Static Method**

1. **interface** Sayable{
2. **void** say();
3. }
4. **public** **class** MethodReference {
5. **public** **static** **void** saySomething(){
6. System.out.println("Hello, this is static method.");
7. }
8. **public** **static** **void** main(String[] args) {
9. // Referring static method
10. Sayable sayable = MethodReference::saySomething;
11. // Calling interface method
12. sayable.say();
13. }
14. }

# Java Base64 Encode and Decode

Java provides a class Base64 to deal with encryption. You can encrypt and decrypt your data by using provided methods. You need to import java.util.Base64 in your source file to use its methods.

# ava Default Methods

Java provides a facility to create default methods inside the interface. Methods which are defined inside the interface and tagged with default are known as default methods. These methods are non-abstract methods.

# Java forEach loop

Java provides a new method forEach() to iterate the elements. It is defined in Iterable and Stream interface. It is a default method defined in the Iterable interface.

# Java Collectors

Collectors is a final class that extends Object class. It provides reduction operations, such as accumulating elements into collections, summarizing elements according to various criteria, etc

# Java StringJoiner

Java added a new final class StringJoiner in java.util package. It is used to construct a sequence of characters separated by a delimiter.

### **Java StringJoiner Example**

1. // importing StringJoiner class
2. **import** java.util.StringJoiner;
3. **public** **class** StringJoinerExample {
4. **public** **static** **void** main(String[] args) {
5. StringJoiner joinNames = **new** StringJoiner(","); // passing comma(,) as delimiter
7. // Adding values to StringJoiner
8. joinNames.add("Rahul");
9. joinNames.add("Raju");
10. joinNames.add("Peter");
11. joinNames.add("Raheem");
13. System.out.println(joinNames);
14. }
15. }

Output:

Rahul,Raju,Peter,Raheem

# Java Optional Class

Java introduced a new class Optional in jdk8. It is a public final class and used to deal with NullPointerException in Java application.

**Stream In Java**

. *There uses of Stream in Java are mentioned below:*

1. *Stream API is a way to express and process collections of objects.*
2. *Enable us to perform operations like filtering, mapping,reducing and sorting.*

* A stream is not a data structure instead it takes input from the Collections, Arrays or I/O channels.
* Streams don’t change the original data structure, they only provide the result as per the pipelined methods.

**1. map()**

The map method is used to return a stream consisting of the results of applying the given function to the elements of this stream.

List number = Arrays.asList(2,3,4,5);

List square = number.stream().map(x->x\*x).collect(Collectors.toList());

**2. filter()**

The filter method is used to select elements as per the Predicate passed as an argument.

List names = Arrays.asList("Reflection","Collection","Stream");

List result = names.stream().filter(s->s.startsWith("S")).collect(Collectors.toList());

Filter(predicate) //predicate which is giving true and false.

**3. sorted()**

The sorted method is used to sort the stream.

List names = Arrays.asList("Reflection","Collection","Stream");

List result = names.stream().sorted().collect(Collectors.toList());