JavaSE 8 Features

The important features of JavaSE 8 are lambda expressions, methods references, default methods, functional interface, java 8 date/time, stream classes etc.

- Java 8 Date/Time API (Java 8)
- Lambda Expressions (Java 8)
- Method References (Java 8)
- Functional Interfaces (Java 8)
- Stream (Java 8)
- Base64 Encode Decode (Java 8)
- Default Methods (Java 8)
- forEach method(Java 8)
- Collectors(Java 8)
- StringJoiner(Java 8)
- Optional class (Java 8)
- Nashorn JavaScript (Java 8)
- Parallel Array Sorting (Java 8)
- Type Inference (Java 8)
- Method Parameter Reflection (Java 8)
- Type annotations and repeating annotations (Java 8)
- Java JDBC Improvements (Java 8)
- Java IO Improvement (Java 8)
- Java Concurrency Improvement (Java 8)

Methods of Java LocalDate

Method	**Description**
LocalDateTime atTime(int hour, int minute)	It is used to combine this date with a time to create a LocalDateTime.
int compareTo(ChronoLocalDate other)	It is used to compares this date to another date.
boolean equals(Object obj)	It is used to check if this date is equal to another date.

String format(DateTimeFormatter formatter)	It is used to format this date using the specified formatter.
int get(TemporalField field)	It is used to get the value of the specified field from this date as an int.
boolean isLeapYear()	It is used to check if the year is a leap year, according to the ISO proleptic calendar system rules.
LocalDate minusDays(long daysToSubtract)	It is used to return a copy of this LocalDate with the specified number of days subtracted.
LocalDate minusMonths(long monthsToSubtract)	It is used to return a copy of this LocalDate with the specified number of months subtracted.
static LocalDate now()	It is used to obtain the current date from the system clock in the default time-zone.

```
import java.time.LocalDate;

public class LocalDateExample1 {
  public static void main(String[] args) {
    LocalDate date = LocalDate.now();
    LocalDate yesterday = date.minusDays(1);
    LocalDate tomorrow = yesterday.plusDays(2);
    System.out.println("Today date: "+date);
    System.out.println("Yesterday date: "+yesterday);
    System.out.println("Tomorrow date: "+tomorrow);
  }
}
```

Output:

```
Today date: 2017-01-13
Yesterday date: 2017-01-12
```

```
Tomorrow date: 2017-01-14
```

```java
import java.time.LocalDate;
public class LocalDateExample2 {
  public static void main(String[] args) {
    LocalDate date1 = LocalDate.of(2017, 1, 13);
    System.out.println(date1.isLeapYear());
    LocalDate date2 = LocalDate.of(2016, 9, 23);
    System.out.println(date2.isLeapYear());
  }
}
```

Output:

```
false
true
```

```java
import java.time.LocalDate;
// String to LocalDate in java 8
public class LocalDateExample5
{
    public static void main(String ar[])
    {
        // Example 1
        String dInStr = "2011-09-01";
        LocalDate d1 = LocalDate.parse(dInStr);
        System.out.println("String to LocalDate : " + d1);
        // Example 2
        String dInStr2 = "2015-11-20";
        LocalDate d2 = LocalDate.parse(dInStr2);
        System.out.println("String to LocalDate : " + d2);
    }
}
```

Output:

```
String to LocalDate : 2011-09-01
String to LocalDate : 2015-11-20
```

Java MonthDay class

Java MonthDay class is an immutable date-time object that represents the combination of a month and day-of-month. It inherits Object class and implements the Comparable interface.

Methods of Java MonthDay

Method	**Description**
LocalDate atYear(int year)	It is used to combine this month-day with a year to create a LocalDate.
String format(DateTimeFormatter formatter)	It is used to format this month-day using the specified formatter.
int get(TemporalField field)	It is used to get the value of the specified field from this month-day as an int.
boolean isValidYear(int year)	It is used to check if the year is valid for this month-day.
static MonthDay now()	It is used to obtain the current month-day from the system clock in the default time-zone.
static MonthDay of(int month, int dayOfMonth)	It is used to obtain an instance of MonthDay.
ValueRange range(TemporalField field)	It is used to get the range of valid values for the specified field.

Java Lambda Expressions

Lambda expression is a new and important feature of Java which was included in Java SE 8. It provides a clear and concise way to represent one method interface using an expression. It is very useful in collection library. It helps to iterate, filter and extract data from collection.

The Lambda expression is used to provide the implementation of an interface which has functional interface. It saves a lot of code. In case of lambda expression, we don't need to define the method again for providing the implementation. Here, we just write the implementation code.

```
(argument-list) -> {body}
```

Java lambda expression is consisted of three components.
1) Argument-list: It can be empty or non-empty as well.
2) Arrow-token: It is used to link arguments-list and body of expression.
3) Body: It contains expressions and statements for lambda expression.

Without Lambda Expression

```
interface Drawable{
    public void draw();
}
public class LambdaExpressionExample {
    public static void main(String[] args) {
        int width=10;

                                                                                   //
without lambda, Drawable implementation using anonymous class

        Drawable d=new Drawable(){
            public void draw(){System.out.println("Drawing "+width);}
        };
        d.draw();
    }
}
```

Java Lambda Expression Example

```
@FunctionalInterface  //It is optional
interface Drawable{
    public void draw();
}

public class LambdaExpressionExample2 {
    public static void main(String[] args) {
        int width=10;

        //with lambda
        Drawable d2=()->{
            System.out.println("Drawing "+width);
        };
        d2.draw();
    }
}
```

Java Lambda Expression Example: No Parameter

```
() -> {
//Body of no parameter lambda
}

interface Sayable{
    public String say();
}
public class LambdaExpressionExample3{
public static void main(String[] args) {
    Sayable s=()->{
        return "I have nothing to say.";
    };
    System.out.println(s.say());
}
}
```

Java Lambda Expression Example: Single Parameter

```
(p1) -> {
//Body of single parameter lambda
}
```

```
interface Sayable{
    public String say(String name);
}

public class LambdaExpressionExample4{
    public static void main(String[] args) {

        // Lambda expression with single parameter.
        Sayable s1=(name)->{
            return "Hello, "+name;
        };
        System.out.println(s1.say("Sonoo"));

        // You can omit function parentheses
        Sayable s2= name ->{
            return "Hello, "+name;
        };
        System.out.println(s2.say("Sonoo"));
    }
}
```

Java Lambda Expression Example: Multiple Parameters

```
interface Addable{
    int add(int a,int b);
}

public class LambdaExpressionExample5{
    public static void main(String[] args) {

        // Multiple parameters in lambda expression
        Addable ad1=(a,b)->(a+b);
        System.out.println(ad1.add(10,20));

        // Multiple parameters with data type in lambda expression
        Addable ad2=(int a,int b)->(a+b);
        System.out.println(ad2.add(100,200));
    }
}
```

Java Lambda Expression Example: with or without return keyword

```
interface Addable{
    int add(int a,int b);
}

public class LambdaExpressionExample6 {
    public static void main(String[] args) {

        // Lambda expression without return keyword.
        Addable ad1=(a,b)->(a+b);
        System.out.println(ad1.add(10,20));

        // Lambda expression with return keyword.
        Addable ad2=(int a,int b)->{
                            return (a+b);
                            };
        System.out.println(ad2.add(100,200));
    }
}
```

Java Lambda Expression Example: Foreach Loop

```java
import java.util.*;
public class LambdaExpressionExample7{
    public static void main(String[] args) {
         
        List<String> list=new ArrayList<String>();
        list.add("ankit");
        list.add("mayank");
        list.add("irfan");
        list.add("jai");
         
        list.forEach(
            (n)->System.out.println(n)
        );
    }
}
```

Java Lambda Expression Example: Multiple Statements

```java
@FunctionalInterface
interface Sayable{
    String say(String message);
}

public class LambdaExpressionExample8{
    public static void main(String[] args) {
     
        // You can pass multiple statements in lambda expression
        Sayable person = (message)-> {
            String str1 = "I would like to say, ";
            String str2 = str1 + message;
            return str2;
        };
            System.out.println(person.say("time is precious."));
    }
}
```

Java Lambda Expression Example: Creating Thread

```
public class LambdaExpressionExample9{
    public static void main(String[] args) {

        //Thread Example without lambda
        Runnable r1=new Runnable(){
            public void run(){
                System.out.println("Thread1 is running...");
            }
        };
        Thread t1=new Thread(r1);
        t1.start();
        //Thread Example with lambda
        Runnable r2=()->{
                System.out.println("Thread2 is running...");
        };
        Thread t2=new Thread(r2);
        t2.start();
    }
}
```

Java Lambda Expression Example: Comparator

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
class Product{
    int id;
    String name;
    float price;
    public Product(int id, String name, float price) {
        super();
        this.id = id;
        this.name = name;
        this.price = price;
    }
}
public class LambdaExpressionExample10{
    public static void main(String[] args) {
        List<Product> list=new ArrayList<Product>();

        //Adding Products
        list.add(new Product(1,"HP Laptop",25000f));
        list.add(new Product(3,"Keyboard",300f));
        list.add(new Product(2,"Dell Mouse",150f));

        System.out.println("Sorting on the basis of name...");

        // implementing lambda expression
        Collections.sort(list,(p1,p2)->{
        return p1.name.compareTo(p2.name);
        });
        for(Product p:list){
            System.out.println(p.id+" "+p.name+" "+p.price);
        }

    }
}
```