Oracle's Java 8 release was a watershed moment in the history of the world's most popular development platform. In addition to a significant improvement to the Java programming paradigm as a whole, the JVM, Java language, and libraries were all updated in a coordinated manner. Many new features were included in this update, including enhanced simplicity of use, increased productivity and security, and overall better system performance.

Top Java 8 Features with Examples

The following Java 8 features will be discussed briefly, with each being explained via the use of basic and simple examples.

Functional Interfaces And Lambda Expressions

In Java 8, a new notion called functional interfaces was introduced. A Functional Interface is an interface that has exactly one abstract method. To designate an interface as a Functional Interface, we don’t need to use the @FunctionalInterface annotation.

```
@FunctionalInterface
public interface FunctionalInterface_one
{
    public void firstInt_method();

    @Override
    public String toString(); //Overridden from Object class

    @Override
    public boolean equals(Object obj); //Overridden from Object class
}
```

An anonymous function may be defined as a Lambda Expression (or function) (a function with no name and an identifier). Lambda Expressions are defined precisely where they are required, often as a parameter to another function.

Lambda Expressions, on the other hand, express instances of Functional Interfaces from a different viewpoint. Lambda Expressions implement functional interfaces by implementing the single abstract function provided in the functional interface.

A basic example of the Lambda Expression is:
(x,y) -> x+y

```
interface MyInterface
{
     void abstract_func(int x,int y);

     default void default_Fun()
    {
         System.out.println("This is default method");
    }
}

class Main
{
     public static void main(String args[])
    {
        //lambda expression
        MyInterface fobj = (int x, int y)->System.out.println(x+y);

        System.out.print("The result = ");
        fobj.abstract_func(5,5);
        fobj.default_Fun();
    }
}
```

```
public class Main {
     public static void main(String[] args) {
        List<String> subList = new ArrayList<String>();
        subList.add("Carrot");
        subList.add("Potato");
        subList.add("Cauliflower");
        subList.add("LadyFinger");
        subList.add("Tomato");
        System.out.println("------------Vegetable List-------------");
        subList.forEach(sub -> System.out.println(sub));
   }
}
```

Output:

————Vegetable List————
Carrot
Potato
Cauliflower
LadyFinger
Tomato

Optional Class

The following program demonstrates the use of the Optional class.

```
import java.util.Optional;
public class Main{

   public static void main(String[] args) {
        String[] str = new String[10];
        Optional<String>checkNull =
                       Optional.ofNullable(str[5]);
        if (checkNull.isPresent()) {
            String word = str[5].toLowerCase();
            System.out.print(str);
         } else
           System.out.println("string is null");
    }
}
```

Output:
String is null

To verify whether the string is null in this application, we utilize the Optional class's "ofNullable" attribute. If it is, the user receives the relevant message.

This ensures that the code created for previous versions is compatible with the newer interfaces (binary compatibility).

Let's understand the Default Method with an example:

```
import java.util.Optional;
interface interface_default {
     default void default_method(){
         System.out.println("We are default method of interface");
    }
}
class derived_class implements interface_default{

}
class Main{
   public static void main(String[] args){
        derived_class obj1 = new derived_class();
        obj1.default_method();
```


Output:

We are the default method of interface

Java Stream API For Bulk Data Operations On Collections

Another significant feature in Java 8 is the Stream API. The Stream API is used to handle a collection of items and allows many iterations. A Stream is a collection of items (elements) that enables you to combine multiple techniques to achieve your goals.

The java.util.stream package in Java 8 introduces a new Streams API that enables you to process components of Java Collections in parallel. Because Java is inherently sequential, and there is no direct way to implement parallel processing at the library level, the stream API will fill that void. You may use Java's Stream API to filter items of a collection based on a defined condition. If you have a list of orders, for example, you may combine purchase and sell orders, filter orders by amount and price, and so on.

Java Date Time API

Under the package java.time, Java 8 offers a new date-time API. The following are the most prominent classes among them:

- Local: Simplified date-time API with no timezone management complexity.
- Zoned: specialized date-time API that can handle several time zones.

DateTimeFormatter	Formatter for displaying and parsing date-time objects

Dates

In Java 8, the Date class has been deprecated. The following are the new courses that have been introduced:

- A date is defined by the LocalDate class. It is devoid of any indication of time or time zone.
- A time is defined by the LocalTime class. It doesn't have a date or time-zone representation.
- A date-time is defined by the LocalDateTime class. It doesn't have a time-zone representation

To combine time-zone information with date functions, you may utilize Lambda's OffsetDate, OffsetTime, and OffsetDateTime classes. Another class – "ZoneId" – is used to express the timezone offset here.

Collection API Improvements

The Collection API in Java 8 now includes the following new methods. A few of them are listed below.

Java IO Improvements

The following are some of the IO enhancements made in Java 8:

- Files.list (Path dir): Returns a lazily filled stream, each element of which represents a directory entry.
- Files.lines (Path path): Reads all the lines from a stream.
- Files.find (): Returns a stream filled by a path after searching for files in the file tree rooted at a provided beginning file and many more.
- BufferedReader.lines (): Returns a stream containing all of the elements of BufferedReader's lines and much more
- Buffered

Miscellaneous Core API Improvements

- ThreadLocal's static function with initial (Supplier supplier) allows you to build an instance quickly.
- The default and static methods for natural ordering, reverse order, and other operations have been added to the "Comparator" interface.
- The min (), max (), and sum () methods are available in the Integer, Long, and Double wrapper classes.

encoding class in Java.util.Base64.

Three Base64 encoders and decoders are provided in this class:

- The output is mapped to a set of characters between A-Za-z0-9+/ in this version. The encoder does not add a line feed to the output, and the decoder rejects any character other than the above.
- The filename safe is mapped to the set of characters between A-Za-z0-9+/, and the output is the URL.
- The output of this sort of encoder is mapped to a MIME-friendly format.

Conclusion

As we can see, Java 8 offers a number of important features. Other programming languages, such as Scala, have inspired some of the features. Java must improve in order to avoid becoming outdated, and although certain features are rudimentary in comparison to other programming languages.

FAQs

Q1: What are the characteristics of Java 8?

Q2: Why is Java 8 still popular?

Ans: The fact that Java 8 is an LTS (Long-Term Support) version is one of the main reasons for its continued popularity. Regrettably, not all Java versions are LTS versions! Only Java 8 (2014) and Java 11 (2018) have been recognized as having LTS since the policy was implemented. All subsequent versions, including Java 14 and the anticipated Java 15 (September 2020), will be LTS-free.

Q3: What changed in Java 8?

Ans: Some of the significant productivity enhancements are lambda expressions, the Streams API, and new methods on existing classes.

Additional Resources

- Learn Java
- Online Java Compiler
- Java MCQ
- Practice Coding
- Java Interview Questions
- How To Become A Java Developer
- Best Java IDE
- Java Projects
- Java 9 Features
- Java 11 New Features