Day Period Support Added to java.time Formats

With the DateTimeFormatter class, you can format date values of the Java Date/Time API, e.g., LocalDate, LocalTime, LocalDateTime, or Instant, Year, and YearMonth.

In Java 16, the list of available format characters has been extended by the letter "B", which stands for a prolonged form of the time of day:

Code:

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Locale;

public class DayTime {
 public static void main(String[] args) {
 String dtf = DateTimeFormatter.ofPattern("EEEE, MMMM d, yyyy, h:mm a",
Locale.US)
 .format(LocalDateTime.now());
 System.out.println(dtf);

 String dtf1 = DateTimeFormatter.ofPattern("EEEE, MMMM d, yyyy, h:mm B",
Locale.US)
 .format(LocalDateTime.now());
 System.out.println(dtf1);
 }
}
```

Output:

```
Sunday, December 31, 2023, 6:46 PM

Sunday, December 31, 2023, 6:46 in the evening
```

Java12 String New Methods

1.) Stream.toList() : It is a fourth variant that also generates an unmodifiable list: This method is implemented as a default method in the Stream interface and is overridden by a stream-specific optimization in most stream implementations.

2.) Stream.mapMulti() : It was introduced as a more efficient, imperative alternative to the declarative flatMap(): While with flatMap(), we specify *which* data we want to merge, with mapMulti() we implement *how* to merge this data.

Code:

```
import java.util.function.Consumer;
import java.util.stream.Collectors;
import java.util.stream.Stream;
import java.util.*;

public class StringNewMethods12 {
 public static void main(String[] args) {
 // ArrayList:
 System.out.println(Stream.of("foo", "bar", "baz").collect(Collectors.toList()));
 // ImmutableCollections$ListN:
 System.out.println(Stream.of("foo", "bar",
"baz").collect(Collectors.toUnmodifiableList()));
 //Java 12:
 System.out.println(Stream.of("foo", "bar", "baz").toList());

 Stream<List<Integer>> stream =
 Stream.of(
 List.of(1, 2, 3),
 List.of(4, 5, 6),
 List.of(7, 8, 9));

 List<Integer> list = stream.flatMap(List::stream).toList();
 System.out.println(list);

 Stream<List<Integer>> stream1 =
 Stream.of(
 List.of(1, 2, 3),
 List.of(4, 5, 6),
 List.of(7, 8, 9));

 List<Integer> list1 = stream1
 .mapMulti(
 (List<Integer> numbers, Consumer<Integer> consumer) ->
 numbers.forEach(number -> consumer.accept(number)))
```

```
 .toList();

 System.out.println(list1);
 }
}
```

Output:

```
[foo, bar, baz]

[foo, bar, baz]

[foo, bar, baz]

[1, 2, 3, 4, 5, 6, 7, 8, 9]

[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Invoke Default Methods From Proxy Instances

As an enhancement to the default method in Interfaces, with the release of Java 16, support has been added to *java.lang.reflect.InvocationHandler* invoke default methods of an interface via a dynamic proxy using reflection.

Code:

```
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.lang.reflect.Proxy;

import static java.lang.ClassLoader.getSystemClassLoader;

public class DefaultProxyInstance {

 public static void main(String[] arg) throws NoSuchMethodException {
 Object proxy = Proxy.newProxyInstance(getSystemClassLoader(), new
Class<?>[] { HelloWorld.class },
 (prox, method, args) -> {
 if (method.isDefault()) {
 return InvocationHandler.invokeDefault(prox, method, args);
 }
 return prox;
 }
```

```
 );
 Method method = proxy.getClass().getMethod("hello");
 System.out.println(method.getName());
 }
}
```

Output:

hello

- **JEP 394: Pattern Matching for *instanceof***
- **JEP 395: Records**
- **JEP 397: Sealed Classes (Second Preview)**