Compact Number Formatting

- Java 12 extends number formatting APIs to provide support for locale-sensitive compact number formatting.
- Now, numbers like 1000 (for example) can be formatted as “1K” (short style) or “1 thousand” (long style).
- CompactNumberFormat is a concrete subclass of NumberFormat that formats a decimal number in its compact form.
- This class is based on SPI Pattern I.e. we don’t need to directly use this class but instead we can get its instance via new factory methods of NumberFormat.

New enum NumberFormat.Style

This enum represents the style for formatting a number within a given NumberFormat instance.

```
public enum Style {

        SHORT,
        LONG

}
```

For example :

```
NumberFormatter formatter = NumberFormat.getCompactNumberInstance(Locale.US,
                                                        NumberFormat.Style.SHORT)

String formattedString = formatter.format(25000L)      // 25K
```

Constructor Details

```
public CompactNumberFormat(String decimalPattern,
        DecimalFormatSymbols symbols,
        String[] compactPatterns)
```

Creates a CompactNumberFormat using the given decimal pattern, decimal format symbols and compact patterns.

To obtain the instance of CompactNumberFormat with the standard compact patterns for a Locale and Style, it is recommended to use the factory methods given by NumberFormat for compact number formatting. For example, NumberFormat.getCompactNumberInstance(Locale, Style).

Parameters:

decimalPattern - a decimal pattern for general number formatting

symbols – the set of symbols to be used

compactPatterns – an array of compact number patterns.

Throws:

NullPointerException – if any of the given arguments is null

IllegalArgumentException – if the given decimalPattern or the compactPatterns array contains an invalid pattern or if a null appears in the array of compact patterns

Compact Number Patterns

The compact number patterns are represented in a series of patterns where each pattern is used to format a range of numbers.

An example of SHORT styled compact number patterns for the US locale is {"", "", "", "0K", "00K", "000K", "0M", "00M", "000M", "0B", "00B", "000B", "0T", "00T", "000T"} , ranging from 10^0 to 10^{14}.

Pattern at index 3 ("0K") is used for formatting number >= 1000 and number < 10000, pattern at index 4 ("00K") is used for formatting number >=10000 and number <100000 so on and so forth.

In most of the locales, patterns with range 10^0–10^2 are empty strings, which implicitly means a special pattern "0".

Compact Formatting

```
import java.text.NumberFormat;
import java.util.List;
import java.util.Locale;
import java.util.stream.IntStream;

public class CompactNumberFormatExample {
	public static void main(String[] args) {
		formatForLocale(Locale.US);
	}

	private static void formatForLocale(Locale locale) {
		List<Integer> numbers = List.of(1000, 1000000, 1000000000);
		System.out.printf("-- SHORT format for locale=%s --%n",
locale);
		numbers.stream().forEach((num) -> {
			NumberFormat nf =
NumberFormat.getCompactNumberInstance(locale, NumberFormat.Style.SHORT);
			String format = nf.format(num);
			System.out.println(format);
		});
		System.out.printf("-- LONG format for locale=%s --%n",
locale);
```

```
            numbers.stream().forEach((num) -> {
                  NumberFormat nf =
NumberFormat.getCompactNumberInstance(locale, NumberFormat.Style.LONG);
                  String format = nf.format(num);
                  System.out.println(format);
            });
      }
}
```

Output :

```
-- SHORT format for locale=en_US --
1K
1M
1B
-- LONG format for locale=en_US --
1 thousand
1 million
1 billion
```

Rounding

CompactNumberFormat provides rounding modes defined in RoundingMode for formatting. By default, it uses RoundingMode.HALF_EVEN.

```
import java.text.NumberFormat;
import java.util.List;
import java.util.Locale;

public class CompactNumberFormatDefaultRounding {
      public static void main(String[] args) {
            formatForLocale(Locale.US);
      }

      private static void formatForLocale(Locale locale) {
            List<Integer> numbers = List.of(1500, 1500000, 1200000000);
            System.out.printf("-- SHORT format for locale=%s --%n",
locale);
            numbers.stream().forEach((num) -> {
                  NumberFormat nf =
NumberFormat.getCompactNumberInstance(locale, NumberFormat.Style.SHORT);
                  String format = nf.format(num);
                  System.out.println(format);
            });
            System.out.printf("-- LONG format for locale=%s --%n",
locale);
            numbers.stream().forEach((num) -> {
```

```
                NumberFormat nf =
NumberFormat.getCompactNumberInstance(locale, NumberFormat.Style.LONG);
                String format = nf.format(num);
                System.out.println(format);
            });
      }
}
```

Output :

```
-- SHORT format for locale=en_US --
2K
2M
1B
-- LONG format for locale=en_US --
2 thousand
2 million
1 billion
```

We can set rounding mode by calling NumberFormat.setRoundingMode(RoundingMode).

```
import java.math.RoundingMode;
import java.text.NumberFormat;
import java.util.List;
import java.util.Locale;

public class CompactNumberFormatExplicitRounding {
      public static void main(String[] args) {
            formatForLocale(Locale.US);
      }

      private static void formatForLocale(Locale locale) {
            List<Integer> numbers = List.of(1500, 1500000, 1200000000);
            System.out.printf("-- SHORT format for locale=%s --%n",
locale);
            numbers.stream().forEach((num) -> {
                NumberFormat nf =
NumberFormat.getCompactNumberInstance(locale, NumberFormat.Style.SHORT);
                nf.setRoundingMode(RoundingMode.HALF_DOWN);
                String format = nf.format(num);
                System.out.println(format);
            });
            System.out.printf("-- LONG format for locale=%s --%n",
locale);
            numbers.stream().forEach((num) -> {
```

```
                NumberFormat nf =
NumberFormat.getCompactNumberInstance(locale, NumberFormat.Style.LONG);
                nf.setRoundingMode(RoundingMode.HALF_DOWN);
                String format = nf.format(num);
                System.out.println(format);
            });
      }
}
```

Output :

```
-- SHORT format for locale=en_US --
1K
1M
1B
-- LONG format for locale=en_US --
1 thousand
1 million
1 billion
```

Parsing

The default parsing behaviour does not allow a grouping separator until 'grouping used' is set to true by using setGroupingUsed(boolean). By default, 'grouping used' is set to 'false'.

```
import java.text.NumberFormat;
import java.text.ParseException;
import java.util.Locale;

public class CompactNumberFormatParseWithGrouping {
      public static void main(String[] args) throws ParseException {
            parseWithGrouping(false);
            parseWithGrouping(true);
      }

      private static void parseWithGrouping(boolean grouping) throws
ParseException {
            System.out.printf("-- grouping=%s  ---%n", grouping);
            NumberFormat nf =
NumberFormat.getCompactNumberInstance(Locale.US,
NumberFormat.Style.SHORT);
            nf.setGroupingUsed(grouping);
            Number num = nf.parse("1,00K");
            System.out.println(num);
            num = nf.parse("1,00M");
```

```
            System.out.println(num);
            num = nf.parse("1,00B");
            System.out.println(num);
      }
}
```

Output :

```
-- grouping=false  ---
1
1
1
-- grouping=true  ---
100000
100000000
100000000000
```