As per JavaDocs, *"LocalTime is an immutable date-time object that represents a time, often viewed as hour-minute-second. Time is represented to nanosecond precision. For example, the value "13:45.30.123456789" can be stored in a LocalTime"*.

In other words, the `LocalTime` represents time without a date. An instance of `LocalTime` can be created from the system clock or by using the `now()`, `parse()` and `of()` methods.

Let's look at some of the utilities provided by this class.

a) Getting the current time

We can get the current time by using the static `now()` method in the `LocalTime` class.

```
import java.time.LocalTime;

class DateTimeDemo {
    public static void main( String args[] ) {
        LocalTime time = LocalTime.now();
        System.out.println(time);
    }
}
```

Run Save Reset

b) Getting a specific time using of() method

We can get a specific time by using the static `of()` method in the `LocalTime` class. This method has three overloaded versions.

Each of them is shown in the example below.

```
import java.time.LocalTime;

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

        // of(int hour, int minute)
        LocalTime time = LocalTime.of(11, 25);
        System.out.println(time);

        // of(int hour, int minute, int second)
        time = LocalTime.of(11, 25, 03);
        System.out.println(time);

        // of(int hour, int minute, int second, int nanoOfSecond)
        time = LocalTime.of(11, 25, 04, 323);
        System.out.println(time);

    }
}
```

Run Save Reset

c) Getting a specific time using parse() method

We can get a specific time by using the static `parse()` method in the `LocalTime` class. This method has two overloaded versions.

Each of them is shown in the example below.

```
import java.time.LocalTime;
import java.time.format.DateTimeFormatter;

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


        // parse(CharSequence text)
        LocalTime time = LocalTime.parse("08:27");
        System.out.println(time);

        // parse(CharSequence text, DateTimeFormatter formatter)
        time = LocalTime.parse("08:27", DateTimeFormatter.ofPattern("HH:mm"));
        System.out.println(time);

    }
}
```

Run Save Reset

d) Adding seconds, minutes and hours to a given time.

We can use a whole range of the addition operations to add seconds, minutes and hours to a given time.

```
import java.time.LocalTime;
import java.time.temporal.ChronoUnit;

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

        // Adding 4 seconds to the given time.
        LocalTime time = LocalTime.parse("12:54:53").plusSeconds(4);
        System.out.println(time);

        // Adding 10 minutes to the given time.
        time = LocalTime.parse("12:54:53").plusMinutes(10);
        System.out.println(time);

        // Adding 2 hours to the given time.
        time = LocalTime.parse("12:54:53").plusHours(2);
        System.out.println(time);

        // Adding 4 minutes to the given time.
        time = LocalTime.parse("12:54:53").plus(4, ChronoUnit.MINUTES);
        System.out.println(time);

    }
}
```

Run Save Reset

e) Getting minute from time

We can get the value of minutes using `getMinute()` method.

```
import java.time.LocalTime;

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

        int minute = LocalTime.parse("07:45").getMinute();

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

Run Save Reset

f) Checking if time is before or after a given time.

We can check if a time is before or past another given time by using the `isBefore()` and `isAfter()` method.

```
import java.time.LocalTime;

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

        boolean isBefore = LocalTime.parse("06:23")
                .isBefore(LocalTime.parse("07:50"));
        System.out.println(isBefore);

        boolean isAfter = LocalTime.parse("06:23")
                .isAfter(LocalTime.parse("07:50"));
        System.out.println(isAfter);

    }
}
```

Run Save Reset