Timers

The Timer class is used when we want to perform time-related activities, such as scheduling a task to run at a specific time or repeating a task after a specific time interval.

When we create a timer, we tell it which code to run using a class called TimerTask.

This class implements a method called run, which defines the code that is executed when the Timer is triggered.

This code is run in a different thread than the code that created the Timer.

To implement a Timer, we will create a TimerTask with code to run when the Timer is triggered. The example below demonstrates how we can create a TimerTask that prints "Hello!" each time the Timer is triggered.

```
import java.util.Timer;
import java.util.TimerTask;

class TimerCode extends TimerTask {
    public void run() {
        System.out.println("Hello!");
    }
}
```

Once we have implemented our TimerTask, we can set up a Timer to run the task as required.

Timer scheduling

We can schedule a TimerTask to execute using the schedule method of a Timer object.
This method has many different modes, but to start, we will look at how to schedule a task to run once at a specific date and time.

Suppose we have some code we wish to run 2 seconds after the program starts. To do so, we need to get the current date and time and add 2 seconds to it.
Once we have that time, we can use our Timer to schedule the code execution.
The code below shows how this can be done using the TimerTask we defined previously.

```
public class Main {
    public static void main(String[] args) {
        Timer timer = new Timer();
        TimerTask task = new TimerCode();
        LocalDateTime timeToExecute = LocalDateTime.now().plusSeconds(2);
        Date execTime =
Date.from(timeToExecute.atZone(ZoneId.systemDefault()).toInstant());
        timer.schedule(task, execTime);
    }
}
```

In this example, we first create a LocalDateTime object, which stores the current time on the user's system plus two seconds.

As such, we obtain a time that is two seconds after the time the application was started. We then convert it to a Date object so it can be used with the schedule method.

When we convert the LocalDateTime to a Date, we specify the timezone using ZoneId.systemDefault().

This will ensure that the Date matches the timezone of the person running the application. This code results in our text "Hello" printing to the screen 2 seconds after the application starts.

Fixed-rate scheduling

In the last section, we saw how we could use the schedule method to run a task once at a specific time and date. In this section, we will see how we can create a task that starts at a particular time and date and then repeats at a fixed interval.
There are two ways in which we can set up a fixed-rate schedule. The first is to specify a delay from when the program starts and an interval at which the task repeats. For example, if we wanted to start the task 2 seconds after the program begins and repeat it every 3 seconds, we would use the code shown below.

```
public class Main {
    public static void main(String[] args) {
        Timer timer = new Timer();
        TimerTask task = new TimerCode();

        timer.schedule(task, 2000, 3000);
    }
}
```

Note that the schedule method expects times in milliseconds, so we use 2000 milliseconds to represent 2 seconds and 3000 milliseconds for 3 seconds. If we wanted to start the execution of our code at a specific date and time like before, we could replace the second argument with a Date object. The code below shows how we can execute our task at fixed intervals of 4 seconds starting at a specific time.

```
public class Main {
    public static void main(String[] args) {
        Timer timer = new Timer();
        TimerTask task = new TimerCode();

        LocalDateTime timeToExecute = LocalDateTime.now().plusSeconds(2);
        Date execTime =
Date.from(timeToExecute.atZone(ZoneId.systemDefault()).toInstant());
        timer.schedule(task, execTime, 4000);
    }
}
```

Canceling and exceptions

Once we schedule a fixed-rate TimerTask, we might eventually want to stop it. To do so, we need to call the cancel method of the TimerTask we wish to stop.

```
public class Main {
    public static void main(String[] args) {
        Timer timer = new Timer();
        TimerTask task = new TimerCode();

        timer.schedule(task, 2000, 3000);
        task.cancel();
    }
}
```

We can also use the cancel method of our Timer object to cancel all tasks associated with the Timer. This is ideal if you wish to cancel multiple tasks at the same time. The following example demonstrates how we can use the cancel method with our Timer object.

```
public class Main {
    public static void main(String[] args) {
        Timer timer = new Timer();
        TimerTask task = new TimerCode();
        TimerTask task2 = new TimerCode();

        timer.schedule(task2, 0, 2000);
        timer.schedule(task, 0, 1000);
        timer.cancel();
    }
}
```

In this example, task and task2 are stopped as soon as timer.cancel() is called. In addition to canceling tasks, we want to consider what happens if a task crashes unexpectedly due to an exception. If we created a TimerTask that throws an exception, we would see that the TimerTask stops running, and our program crashes when the exception is thrown. Ideally, we would like to be able to recover from any occurring exceptions. To that end, we can wrap our TimerTask code in a try-catch block.

```
class TimerCode extends TimerTask {
    public void run() {
        try {
            throw new IllegalArgumentException();
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

When this code throws the exception, it will print a message and run again at the next scheduled time. This is ideal for situations where we are looking for a file that doesn't exist yet. If the file is not found, we catch the exception and try again later when the file might have become available. Sometimes, we want to cancel a TimerTask if it encounters an exception. To do so, we can use the cancel method discussed earlier.

```
class TimerCode extends TimerTask {
    public void run() {
        try {
            throw new IllegalArgumentException();
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
            this.cancel();
        }
    }
}
```

With this code, our TimerTask will cancel as soon as an exception occurs.