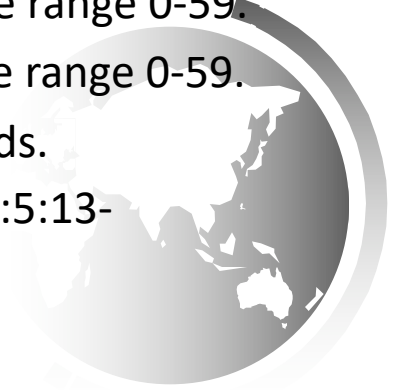
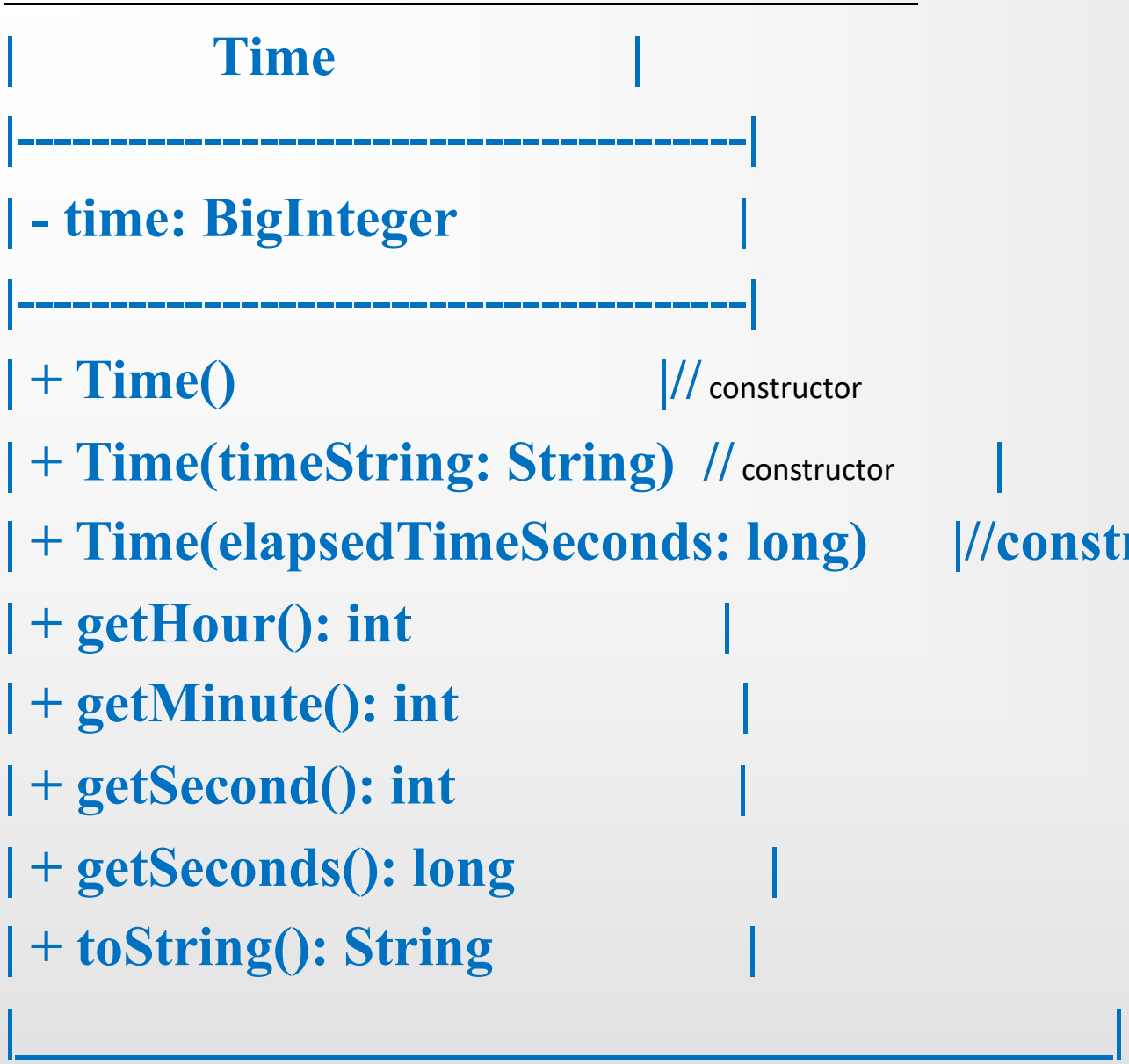
Lab 12
Object Oriented Thinking

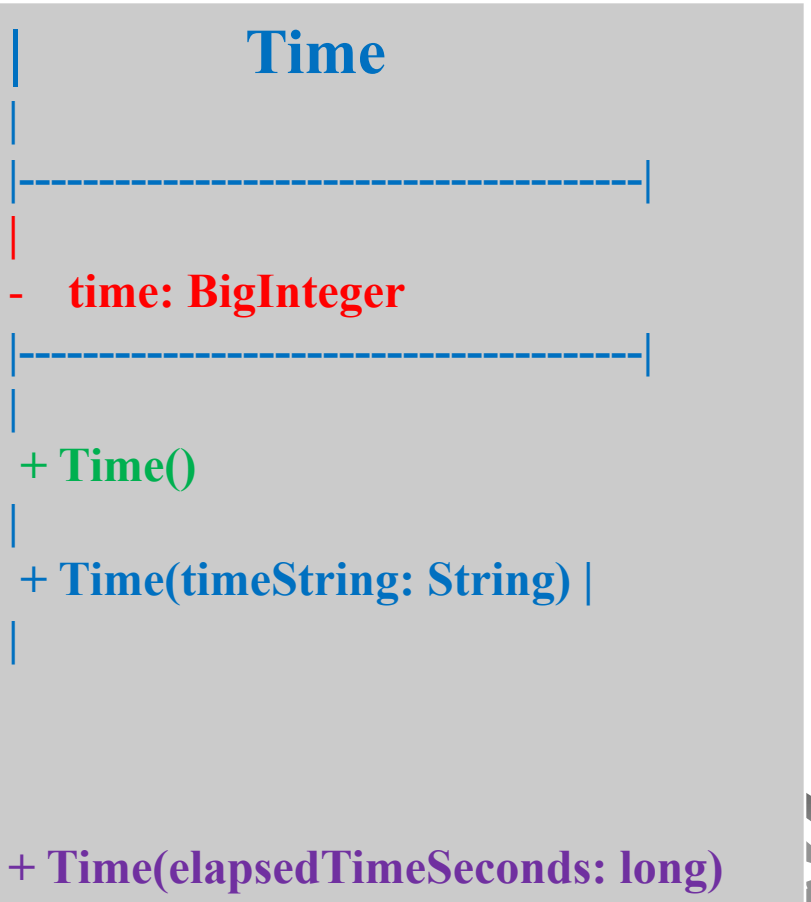
- **Activity 1:** Implement a class named **Time** for encapsulating a time. The class
- contains the following:
- – A data field of the **BigInteger** *time* that stores the elapsed time in milliseconds
- since midnight, Jan 1, 1970.
- – A no-arg constructor that constructs a **Time** for the current system time.
- – A constructor with the specified time string to create a **Time**. A time string format is “yyyy:mm:dd-hh:mm:ss”
  such as “2022:5:13-14:40:20”.
- – A constructor with the specified elapsed time in seconds since midnight, Jan 1, 1970.
- – The *getHour()* method that returns the current hour in the range 0-23.
- – The *getMinute()* method that returns the current minute in the range 0-59.
- – The *getSecond()* method that returns the current second in the range 0-59.
- – The *getSeconds()* method that returns the elapsed total seconds.
- – The *toString()* method that returns a string time such as “2022:5:13-14:40:20”.
- Write a Driver class to test **Time** class.



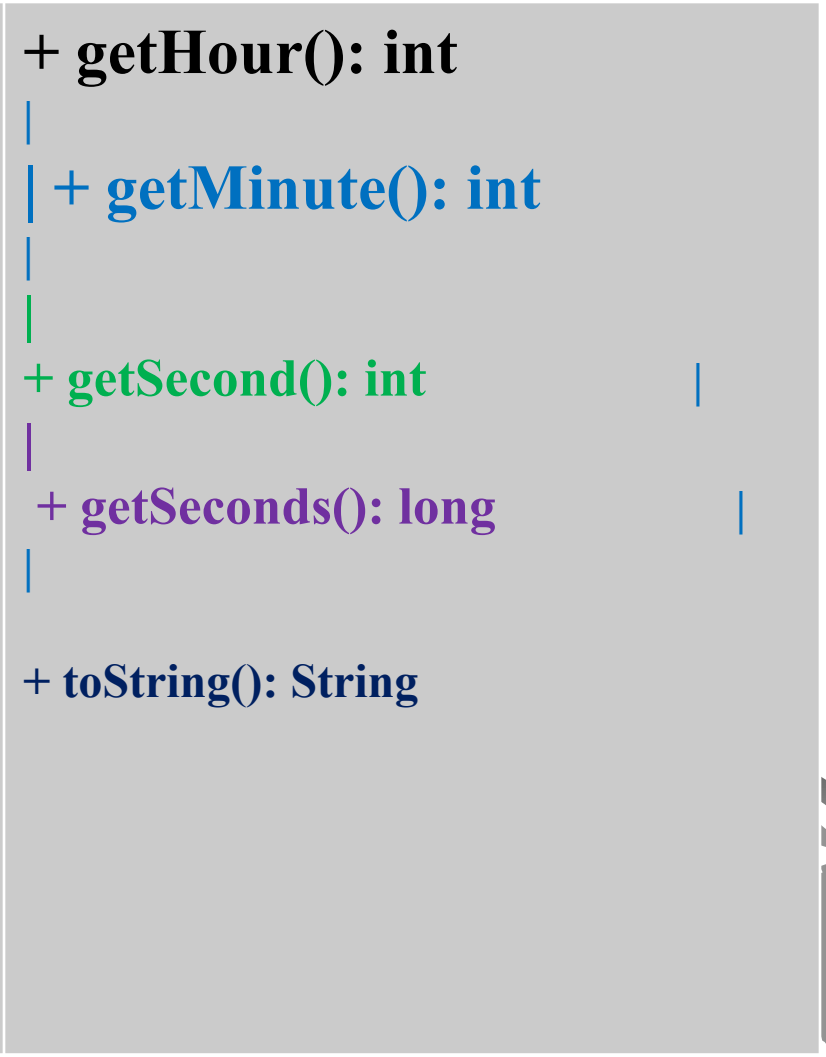


The class
contains the following:

- A data field of the BigInteger *time* that stores the elapsed time in milliseconds
  since midnight, Jan 1, 1970.
- A no-arg constructor that constructs a Time for the current system time.
- A constructor with the specified time string to create a Time. A time string format is "yyyy:mm:dd-hh:mm:ss"
  such as "2022:5:13-14:40:20".
- A constructor with the specified elapsed time in seconds since midnight, Jan 1, 1970.



- The *getHour()* method that returns the current hour in the range 0-23.
- The *getMinute()* method that returns the current minute in the range 0-59.
- The *getSecond()* method that returns the current second in the range 0-59.
- The *getSeconds()* method that returns the elapsed total seconds.
- The *toString()* method that returns a string time such as "2022:5:13-14:40:20".
- Write a Driver class to test Time class.



```
                Time
|------------------------------------
| - time: BigInteger
|
|------------------------------------
```

```java
import java.math.BigInteger;
import java.time.format.DateTimeFormatter;

public class Time {

private BigInteger time;
```

+ Time()	public Time() { time = BigInteger.*valueOf*(System.*currentTimeMillis*()); }
+ Time(timeString: String) |	public Time(String timeString) { // Parse timeString manually String[] parts = timeString.split("[-:]"); int year = Integer.*parseInt*(parts[0]); int month = Integer.*parseInt*(parts[1]); int day = Integer.*parseInt*(parts[2]); int hour = Integer.*parseInt*(parts[3]); int minute = Integer.*parseInt*(parts[4]); int second = Integer.*parseInt*(parts[5]); **// Calculate milliseconds since epoch** long milliseconds = LocalDateTime.*of*(year, month, day, hour, minute, second) .toEpochSecond(ZoneOffset.*UTC*) * 1000; time = BigInteger.*valueOf*(milliseconds);
+ Time(elapsedTimeSeconds: long)	public Time(long elapsedTimeSeconds) { time = BigInteger.*valueOf*(elapsedTimeSeconds * 1000); **}**

- The "epoch" is a reference point in time from which all other times are measured. It's a common concept in computing and refers to a specific moment in time that serves as a starting point for a time scale.
- In many computer systems, especially those based on Unix-like operating systems, the epoch is defined as midnight (00:00:00) on January 1, 1970, Coordinated Universal Time (UTC).
- UTC=GMT

Activity 2: Design two classes: Flight[1] and Itinerary[2]. The Flight class stores the information about a flight with the following members:

– A data field named *flightNo* of the String type with getter method.

– A data field named *departureTime* of the Time type (The one created in Activity 1) with getter and setter methods.

– A data field named *arrivalTime* of the Time type with getter and setter methods.

– A constructor that creates a Flight with the specified number, *departureTime*, and *arrivalTime*.

– A method named *getFlightTime()* that returns the flight time in minutes.

The Itinerary class stores the information about itinerary with the following members:

– A data field named *flights* of Flight[] type. The array contains the flights for the itinerary.

– A constructor that creates an Itinerary with the specified flights.

– A method named *getTotalTime()* that returns the total travel time in minutes from the departure time of the first flight to the arrival time of the last flight in the itinerary.

Implement these two classes and a Driver class to test these classes.

```
_____________________________________
|            Flight                 |
|-----------------------------------|
| - flightNo: String                |
| - departureTime: Time             |
| - arrivalTime: Time               |
|-----------------------------------|
| + Flight(flightNo: String,        |// constructor
|          departureTime: Time,     |
|          arrivalTime: Time)       |
| + getFlightNo(): String           |//methods
| + getDepartureTime(): Time        |
| + setDepartureTime(departureTime: Time)|
| + getArrivalTime(): Time          |
| + setArrivalTime(arrivalTime: Time)  |
| + getFlightTime(): int            |
|___________________________________|


_____________________________________
|            Itinerary              |
|-----------------------------------|
| - flights: Flight[]               |
|-----------------------------------|
| + Itinerary(flights: Flight[])    |
| + getTotalTime(): int             |
|___________________________________|
```

Class **Flight**

```
import java.time.LocalTime;


class Flight {
private String flightNo;
private LocalTime departureTime;
private LocalTime arrivalTime;


// Constructor
public Flight(String flightNo, LocalTime
departureTime, LocalTime arrivalTime) {
this.flightNo = flightNo;
this.departureTime = departureTime;
this.arrivalTime = arrivalTime;
}
```

Methods

```
// Getter for flight number
public String getFlightNo() {
return flightNo;
}
// Getter for departure time
public LocalTime getDepartureTime() {
return departureTime;
}
// Setter for departure time
public void setDepartureTime(LocalTime departureTime) {
this.departureTime = departureTime;
}
// Getter for arrival time
public LocalTime getArrivalTime() {
return arrivalTime;
}
// Setter for arrival time
public void setArrivalTime(LocalTime arrivalTime) {
this.arrivalTime = arrivalTime;
}

// Method to calculate flight time in minutes
public long getFlightTime() {
return departureTime.until(arrivalTime,
java.time.temporal.ChronoUnit.MINUTES);
}
}
```

class Itinerary

```
class Itinerary {
private Flight[] flights;
// Constructor
public Itinerary(Flight[] flights) {
this.flights = flights;
}

// Method to calculate total travel time in minutes
public long getTotalTime() {
if (flights.length == 0) {
return 0; // No flights in the itinerary
}
// Departure time of the first flight
LocalTime departureTime = flights[0].getDepartureTime();
// Arrival time of the last flight
LocalTime arrivalTime = flights[flights.length -
1].getArrivalTime();
// Calculate total travel time in minutes
return departureTime.until(arrivalTime,
java.time.temporal.ChronoUnit.MINUTES);
}
}
```

Driver

```
import java.time.LocalTime;

public class Driver {
public static void main(String[] args) {
// Create flights
Flight flight1 = new Flight("ABC123", LocalTime.of(8, 0),
LocalTime.of(10, 0));
Flight flight2 = new Flight("XYZ456", LocalTime.of(11, 0),
LocalTime.of(13, 0));
Flight[] flights = { flight1, flight2 };

// Create itinerary
Itinerary itinerary = new Itinerary(flights);

// Test getTotalTime method
System.out.println("Total travel time: " +
itinerary.getTotalTime() + " minutes");
}
}
```

```
java.time.temporal.ChronoUnit.MINUTES);
```

```
import static java.time.temporal.ChronoUnit.MINUTES;

//Your code...

long minutesDifference =startTime.until(endTime, MINUTES);
```

In Java's java.time.temporal.ChronoUnit enum, MINUTES is one of the predefined enum constants.
It represents a unit of time equal to 60 seconds.