The Functional Java Standard Library

On this page

The lib Package

This is the core standard library for Functional Java. It contains a selection of data types and functions that are useful in many programming scenarios. To import this package, use the following import statements:

```
import comp1110.lib.*;
import comp1110.lib.Date;
import static comp1110.lib.Functions.*;
```

General

Records

Pair

Pair

```
/**
 * Represents a pair of two values of given types.
 * Examples:
 * - Pair(5, "Hello")
 * - Pair(16.0, 32.5)
 * - Pair(Pair("", true), 0)
 * @param first The first value in the pair
 * @param second The second value in the pair
 */
record Pair<S, T>(
    S first,
    T second) {}
// TEMPLATE:
// { ... pair.first() ... pair.second() ... }
```

Itemizations

Maybe

Maybe

```java
/**
 * Represents a value that may or may not be present.
 * It is one of:
 * - Nothing, where no value is present
 * - Something, where some value is present
 */
sealed interface Maybe<T>permits Nothing, Something{}

// TEMPLATE:
// {
//     ...
//     return ... switch(maybe) {
//         case Nothing<T>() ->  ... ;
//         case Something<T>(var element) ->
//              ... element ... ;
//     } ...;
// }

/**
 * Represents the absence of a value
 * Examples:
 * - Nothing<T>()
 */
record Nothing<T>(
) implements Maybe<T>{}
/**
 * Represents the presence of a value of type T
 * Examples:
 * - Something<Integer>(5)
 * - Something<String>("Hello")
 * @param element The value that is present
 */
record Something<T>(
    T element) implements Maybe<T>{}
```

Functions

GetVersion ; CheckVersion ; DefaultCaseError ; Equals ; Compare ; Min ; Max ; GreaterThan ; LessThan ; ToString ; IsIntString ; StringToInt ; IsLongString ; StringToLong ; IsFloatString ; StringToFloat ; IsDoubleString ; StringToDouble ; StringToBoolean ; Length ; RandomNumber ; Default ; Join

GetVersion

```
/**
 * Returns the current version of the COMP1110 Standard Library
 *
 * Examples:
 * given:
 *  expect: "2025S1-1"
 * given:
 *  expect: "2025S1-2"
 * @return the current version of the COMP1110 Standard Library
 */
String GetVersion();
```

CheckVersion

```
/**
 * Checks the compatibility of the used library
 * with a particular version.
 * Throws an exception if not compatible.
 *
 * Examples:
 * given: "2025S1-1"
 *  expect: the program continues normally,
 *          except if breaking changes have been made to the library
 * given: "foobar"
 *  expect: the program will show an error message and exit
 *
 * @param version The required version of the library
 */
void CheckVersion(String version);
```

DefaultCaseError

```
/**
 * Use this to end the program if are doing a case distinction
 * that requires a default case which you know to be
 * impossible (under given preconditions and invariants).
 * The program will display an error message.
 * The function nominally returns a value of any type you ask
 * for, even though it never will.
 *
 * Examples:
 * given: (nothing)
 *  expect: The program ends with an error message
```

```
     *
     * @param message A message to display when the program ends.
     *          This is in addition to generic information about the
     *          kind and location of the error.
     *        OPTIONAL, default = ""
     */
    <T> T DefaultCaseError(String message);
```

Equals

```
    /**
     * Returns whether two given values are equal
     *
     * Examples:
     * given: "Hello", "World"
     *  expect: false
     * given: "COMP", "COMP"
     *  expect: true
     * given: 42, 42
     *  expect: true
     *
     * @param o1 the first value to be compared
     * @param o2 the second value to be compared
     * @return true if the values are equal, false if not
     */
    boolean Equals(Object o1, Object o2);
```

Compare

```
    /**
     * Compares two values of a comparable type T.
     * Comparable Types are:
     *   - String
     *   - the number types (int, double, ...)
     *   - booleans
     *   - Date
     *   - DateTime
     *
     * Examples:
     * given: 5, 3
     *  expect: A positive integer
     * given: "Alpha", "Omega"
     *  expect: A negative integer
```

```
     * given: true, true
     *  expect: 0
     *
     * @param left the first value to compare
     * @param right the second value to compare
     * @return A negative integer if left < right
     *         A positive integer if left > right
     *         0 if left equals right
     */
    <T> int Compare(T left, T right);
```

Min

```
    /**
     * Returns the smaller of two given values of
     * a comparable type T.
     * Comparable Types are:
     *   - String
     *   - the number types (int, double, ...)
     *   - booleans
     *   - Date
     *   - DateTime
     *
     * Examples:
     * given: 5, 3
     *  expect: 3
     * given: "Alpha", "Omega"
     *  expect: "Alpha"
     * given: true, true
     *  expect: true
     *
     * @param left the first value to compare
     * @param right the second value to compare
     * @return The smaller value, or the left argument
     *         if they are equal.
     */
    <T> T Min(T left, T right);
```

Max

```
    /**
     * Returns the larger of two given values of
     * a comparable type T.
```

```
 * Comparable Types are:
 *   - String
 *   - the number types (int, double, ...)
 *   - booleans
 *   - Date
 *   - DateTime
 *
 * Examples:
 * given: 5, 3
 *  expect: 5
 * given: "Alpha", "Omega"
 *  expect: "Omega"
 * given: true, true
 *  expect: true
 *
 * @param left the first value to compare
 * @param right the second value to compare
 * @return The larger value, or the left argument
 *         if they are equal.
 */
<T> T Max(T left, T right);
```

GreaterThan

```
/**
 * Checks whether a comparable value is greater than another
 * Comparable Types are:
 *   - String
 *   - the number types (int, double, ...)
 *   - booleans
 *   - Date
 *   - DateTime
 *
 * Examples:
 * given: 5, 3
 *  expect: true
 * given: "Alpha", "Omega"
 *  expect: false
 * given: true, true
 *  expect: false
 *
 * @param greater the potentially greater value
 * @param smaller the potentially smaller value
```

```
 * @return True if greater > smaller, false if not
 */
<T> boolean GreaterThan(T greater, T smaller);
```

LessThan

```
/**
 * Checks whether a comparable value is less than another
 * Comparable Types are:
 *   - String
 *   - the number types (int, double, ...)
 *   - booleans
 *   - Date
 *   - DateTime
 *
 * Examples:
 * given: 5, 3
 *  expect: false
 * given: "Alpha", "Omega"
 *  expect: true
 * given: true, true
 *  expect: false
 *
 * @param smaller the potentially smaller value
 * @param greater the potentially greater value
 * @return True if smaller < greater, false if not
 */
<T> boolean LessThan(T smaller, T greater);
```

ToString

```
/**
 * Returns a String representation of a given value
 *
 * Examples:
 * given: 5
 *  expect: "5"
 * given: "Hello"
 *  expect: "Hello"
 * given: A Date representing 2025-02-18
 *  expect: "2025-02-18"
 *
 * @param o Any value
```

```
 * @return A String representation of the given value
 */
String ToString(Object o);
```

IsIntString

```
/**
 * Checks whether a String represents an int
 *
 * Examples:
 * given: 15
 *  expect: true
 * given: Hello
 *  expect: false
 *
 * @param str The String that may represent an int
 * @return true if the String represents an int, false if not
 */
boolean IsIntString(String str);
```

StringToInt

```
/**
 * Converts a String to an Integer.
 *
 * Examples:
 * given: "-15"
 *  expect: -15
 * given: "1337"
 *  expect: 1337
 *
 * @param str The String representing an Integer
 * @return The Integer represented by the String
 * @implSpec
 *   Precondition: The given String must represent an Integer.
 */
int StringToInt(String str);
```

IsLongString

```
/**
 * Checks whether a String represents a long
 *
```

```
 * Examples:
 * given: 15
 *  expect: true
 * given: Hello
 *  expect: false
 *
 * @param str The String that may represent a long
 * @return true if the String represents a long, false if not
 */
boolean IsLongString(String str);
```

StringToLong

```
/**
 * Converts a String to a Long.
 *
 * Examples:
 * given: "-15"
 *  expect: -15
 * given: "1337"
 *  expect: 1337
 *
 * @param str The String representing a Long
 * @return The Long represented by the String
 * @implSpec
 *   Precondition: The given String must represent a Long.
 */
long StringToLong(String str);
```

IsFloatString

```
/**
 * Checks whether a String represents a float
 *
 * Examples:
 * given: 15.0
 *  expect: true
 * given: Hello
 *  expect: false
 *
 * @param str The String that may represent a float
 * @return true if the String represents a float, false if not
```

```
 */
boolean IsFloatString(String str);
```

StringToFloat

```
/**
 * Converts a String to a Float.
 *
 * Examples:
 * given: "-15.0"
 *  expect: -15.0f
 * given: "1337.3"
 *  expect: 1337.3f
 *
 * @param str The String representing a Float
 * @return The Float represented by the String
 * @implSpec
 *   Precondition: The given String must represent a Float.
 */
float StringToFloat(String str);
```

IsDoubleString

```
/**
 * Checks whether a String represents a double
 *
 * Examples:
 * given: 15.0
 *  expect: true
 * given: Hello
 *  expect: false
 *
 * @param str The String that may represent a double
 * @return true if the String represents a double, false if not
 */
boolean IsDoubleString(String str);
```

StringToDouble

```
/**
 * Converts a String to a Double.
 *
 * Examples:
```

```
     * given: "-15.0"
     *  expect: -15.0
     * given: "1337.3"
     *  expect: 1337.3
     *
     * @param str The String representing a Double
     * @return The Double represented by the String
     * @implSpec
     *   Precondition: The given String must represent a Double.
     */
    double StringToDouble(String str);
```

StringToBoolean

```
    /**
     * Converts a String to a Boolean.
     *
     * Examples:
     * given: "true"
     *  expect: true
     * given: "false"
     *  expect: false
     *
     * @param str The String representing a Boolean
     * @return The Boolean represented by the String
     * @implSpec
     *   Precondition: The given String must represent an Boolean.
     */
    boolean StringToBoolean(String str);
```

Length

```
    /**
     * Returns the length of an Array
     *
     * Examples:
     * given: {}
     *  expect: 0
     * given: {1,2,3,4,5}
     *  expect: 5
     *
     * @param arr The Array whose length we want to determine
     * @return The length of the Array
```

```
 */
<T> int Length(T[] arr);
```

RandomNumber

```
/**
 * Returns a random number
 *
 * Examples:
 * given: 5, 100
 *  expect: a number between 5 (inclusive) and 100 (exclusive)
 * given: 100
 *  expect: a number between 0 (inclusive) and 100 (exclusive)
 * given: (no arguments)
 *  expect: any integer between 2^(-31) and (2^31)-1
 *
 * @param min The minimum number to return (>= 0)
 *        OPTIONAL, default = 0 if max given, else Integer.MIN_VALUE
 * @param max The exclusive upper bound for the random
 *          number. If only one argument is specified,
 *          it counts for this argument. (>= 0)
 *        OPTIONAL, default = Integer.MAX_VALUE (around 2 billion)
 * @return A random integer between the specified bounds
 */
int RandomNumber(int min, int max);
```

Default

```
/**
 * Returns either the value contained in a Something case
 * of a Maybe value, or the default value if the Maybe
 * value is Nothing()
 *
 * Examples:
 * given: Something(50), 100
 *  expect: 50
 * given: Nothing(), 100
 *  expect: 100
 *
 * @param maybe The Maybe-value whose content we want to extract
 * @param else The default value to use if maybe is Nothing
 */
<T> T Default(Maybe<T> maybe, T else);
```

Join

```
/**
 * Flattens levels of Maybes. If both outer levels are Something,
 * then returns the inner Something. Else returns Nothing.
 *
 * Examples:
 * given: Something(Something(50))
 *  expect: Something(50)
 * given: Something(Nothing())
 *  expect: Nothing()
 * given: Nothing()
 *  expect: Nothing()
 *
 * @param maybe The Maybe-value that we want to flatten
 */
<T> Maybe<T> Join(Maybe<Maybe<T>> maybe);
```

DateTime

Opaque Data Types

Date ; DateTime

Date

```
/**
 * This type represents calendar dates.
 * Dates are associated with a timezone, by default
 * the local timezone, and represent some time at a
 * particular date. Use the functions further below
 * to create and compute with Dates.
 */
// This is an opaque type whose definition does not matter to you
```

DateTime

```
/**
 * This type represents time/date combinations.
 * DateTimes are associated with a timezone, by default
 * the local timezone, and can also be used whereever
 * Date values are expected. Use the functions further
 * below to create and compute with DateTimes.
```

```
 */
// This is an opaque type whose definition does not matter to you
```

Functions

CurrentDate ; CurrentTime ; IsDateValid ; GetDate ; GetDateTime ; GetYear ; GetMonth ; GetDay ; GetDayOfWeek ; GetHour ; GetMinute ; GetSecond ; GetMillisecond ; GetTimeZone ; AddYears ; AddMonths ; AddDays ; AddHours ; AddMinutes ; AddSeconds ; AddMilliseconds ; WithTimeZone ; InTimeZone ; ToDate ; YearsBetween ; MonthsBetween ; DaysBetween ; HoursBetween ; MinutesBetween ; SecondsBetween ; MillisecondsBetween ; FormatDate ; ParseDate ; ParseDateTime

CurrentDate

```
/**
 * Returns the current Date
 *
 * Examples:
 * given: "Australia/Sydney" at 2025-02-16 19:00 UTC
 *  expect: [2025-02-17, 00:00:00.000+11:00@Australia/Sydney]
 * given: "America/New_York" at 2025-02-16 02:00 UTC
 *  expect: [2025-02-15, 00:00:00.000-05:00@America/New_York]
 *
 * @param timeZone A time zone descriptor, such as Australia/Sydney
 *        OPTIONAL, default = The system's local timezone
 * @return The current date in the given time zone
 */
Date CurrentDate(String timeZone);
```

CurrentTime

```
/**
 * Returns the current Time and Date
 *
 * Examples:
 * given: "Australia/Sydney" at 2025-02-16 19:00:00.000 UTC
 *  expect: [2025-02-17, 06:00:00.000+11:00@Australia/Sydney]
 * given: "America/New_York" at 2025-02-16 02:00:00.000 UTC
 *  expect: [2025-02-15, 21:00:00.000-05:00@America/New_York]
 *
 * @param timeZone A time zone descriptor, such as Australia/Sydney
 *        OPTIONAL, default = The system's local timezone
 * @return The current time and date in the given time zone
```

```
 */
DateTime CurrentTime(String timeZone);
```

IsDateValid

```
/**
 * Checks whether a particular date is valid
 *
 * Examples:
 * given: 2025, 1, 17
 *  expect: true
 * given: 2025, 2, 29, "Australia/Sydney"
 *  expect: false
 * given: 2025, 11, 31
 *  expect: false
 *
 * @param year The year
 * @param month The month
 * @param day The day of the month
 * @param timeZone The zime zone of the date
 *        OPTIONAL, default = The system's local time zone
 * @return True if the month/day combination is valid
 *         for the year, false if not
 */
boolean IsDateValid(int year, int month, int day, String timeZone);
```

GetDate

```
/**
 * Constructs a Date value for a particular date
 *
 * Examples:
 * given: 2025, 1, 17
 *  expect: [2025-02-17, 00:00:00.000+-XX:XX@XXX]
 *          XX:XX@XXX depending on your system's local time zone
 * given: 2025, 1, 18, "Australia/Sydney"
 *  expect: [2025-01-18, 00:00:00.000+11:00@Australia/Sydney]
 *
 * @param year The year
 * @param month The month
 * @param day The day of the month
 * @param timeZone The zime zone of the date
 *        OPTIONAL, default = The system's local time zone
```

```
 * @return A Date representing the start of the specified
 *         day in the specified time zone
 */
Date GetDate(int year, int month, int day, String timeZone);
```

GetDateTime

```
/**
 * Constructs a DateTime value for a particular date and time
 *
 * Examples:
 * given: 2025, 1, 17, 15, 20, 0
 *  expect: [2025-02-17, 15:20:00.000+-XX:XX@XXX]
 *          XX:XX@XXX depending on your system's local time zone
 * given: 2025, 1, 18, 5, 45, 13, 253"Australia/Sydney"
 *  expect: [2025-01-18, 05:45:13.253+11:00@Australia/Sydney]
 *
 * @param year The year
 * @param month The month
 * @param day The day of the month
 * @param hour The hour of the day
 * @param minute The minute of the hour
 * @param second The second of the minute
 * @param millisecond The millisecond of the second
 *        OPTIONAL, default = 0
 * @param timeZone The zime zone of the date and time
 *        OPTIONAL, default = The system's local time zone
 * @return A DateTime representing the specified point in time
 */
DateTime GetDateTime(int year, int month, int day, int hour, int minu
```

GetYear

```
/**
 * Returns the year of a Date or DateTime value
 *
 * Examples:
 * given: [2025-05-13 ...]
 *  expect: 2025
 * given: [1925-01-14 ...]
 *  expect: 1925
 *
```

```
 * @param date The given date
 * @return The year of the given date
 */
int GetYear(Date date);
```

GetMonth

```
/**
 * Returns the month of a Date or DateTime value
 *
 * Examples:
 * given: [2025-05-13 ...]
 *  expect: 5
 * given: [1925-01-14 ...]
 *  expect: 1
 *
 * @param date The given date
 * @return The month of the given date
 */
int GetMonth(Date date);
```

GetDay

```
/**
 * Returns the day of a Date or DateTime value
 *
 * Examples:
 * given: [2025-05-13 ...]
 *  expect: 13
 * given: [1925-01-14 ...]
 *  expect: 14
 *
 * @param date The given date
 * @return The day of the given date
 */
int GetDay(Date date);
```

GetDayOfWeek

```
/**
 * Returns the day of the week of a Date or DateTime value
 * Monday is represented as 1, Sunday is represented as 7
 *
```

```
 * Examples:
 * given: [2025-05-13 ...]
 *  expect: 2
 * given: [1925-01-14 ...]
 *  expect: 3
 *
 * @param date The given date
 * @return The day of the week of the given date
 * @implSpec
 *   Postcondition: The returned value is between 1 and 7 (inclusive)
 */
int GetDayOfWeek(Date date);
```

GetHour

```
/**
 * Returns the hour of a DateTime value
 *
 * Examples:
 * given: [2025-05-13 22:34:15.532-05:00@...]
 *  expect: 22
 * given: [1925-01-14 06:47:01.003+11:00@...]
 *  expect: 6
 *
 * @param dateTime The given DateTime
 * @return The hour of the given DateTime
 */
int GetHour(DateTime dateTime);
```

GetMinute

```
/**
 * Returns the minute of a DateTime value
 *
 * Examples:
 * given: [2025-05-13 22:34:15.532-05:00@...]
 *  expect: 34
 * given: [1925-01-14 06:47:01.003+11:00@...]
 *  expect: 47
 *
 * @param dateTime The given DateTime
 * @return The minute of the given DateTime
```

```
 */
int GetMinute(DateTime dateTime);
```

GetSecond

```
/**
 * Returns the second of a DateTime value
 *
 * Examples:
 * given: [2025-05-13 22:34:15.532-05:00@...]
 *  expect: 15
 * given: [1925-01-14 06:47:01.003+11:00@...]
 *  expect: 1
 *
 * @param dateTime The given DateTime
 * @return The second of the given DateTime
 */
int GetSecond(DateTime dateTime);
```

GetMillisecond

```
/**
 * Returns the millisecond of a DateTime value
 *
 * Examples:
 * given: [2025-05-13 22:34:15.532-05:00@...]
 *  expect: 532
 * given: [1925-01-14 06:47:01.003+11:00@...]
 *  expect: 3
 *
 * @param dateTime The given DateTime
 * @return The millisecond of the given DateTime
 */
int GetMillisecond(DateTime dateTime);
```

GetTimeZone

```
/**
 * Returns the time zone of a Date or DateTime value
 *
 * Examples:
 * given: [2025-05-13 22:34:15.532-05:00@America/New_York]
 *  expect: America/New_York
```

```
     * given: [1925-01-14 06:47:01.003+11:00@Australia/Sydney]
     *  expect: Australia/Sydney
     *
     * @param date The given Date or DateTime
     * @return The time zone of the given Date or DateTime
     */
    String GetTimeZone(Date date);
```

AddYears

```
    /**
     * Adds a number of years to a Date or DateTime
     *
     * Examples:
     * given: [2025-05-13 ...], 5
     *  expect: [2030-05-13 ...]
     * given: [2025-02-02 ...], -2
     *  expect: [2023-02-02 ...]
     *
     * @param date A Date or DateTime
     * @param years The number of years to add
     * @return A Date or DateTime with the given years added
     */
    T AddYears(T date, long years);
```

AddMonths

```
    /**
     * Adds a number of months to a Date or DateTime
     *
     * Examples:
     * given: [2025-05-13 ...], 5
     *  expect: [2025-10-13 ...]
     * given: [2025-02-02 ...], -2
     *  expect: [2024-12-02 ...]
     *
     * @param date A Date or DateTime
     * @param months The number of months to add
     * @return A Date or DateTime with the given months added
     */
    T AddMonths(T date, long months);
```

AddDays

```
/**
 * Adds a number of days to a Date or DateTime
 *
 * Examples:
 * given: [2025-05-13 ...], 5
 *  expect: [2025-05-18 ...]
 * given: [2025-02-02 ...], -2
 *  expect: [2025-01-31 ...]
 *
 * @param date A Date or DateTime
 * @param days The number of days to add
 * @return A Date or DateTime with the given days added
 */
T AddDays(T date, long days);
```

AddHours

```
/**
 * Adds a number of hours to a DateTime
 *
 * Examples:
 * given: [2025-05-13 15:30:24.612...], 5
 *  expect: [2025-05-13 20:30:24.612...]
 * given: [2025-02-02 21:01:59.129...], -2
 *  expect: [2025-02-02 19:01:59.129...]
 *
 * @param dateTime A DateTime
 * @param hours The number of hours to add
 * @return A DateTime with the given hours added
 */
DateTime AddHours(DateTime dateTime, long hours);
```

AddMinutes

```
/**
 * Adds a number of minutes to a DateTime
 *
 * Examples:
 * given: [2025-05-13 15:30:24.612...], 5
 *  expect: [2025-05-13 15:35:24.612...]
 * given: [2025-02-02 21:01:59.129...], -2
 *  expect: [2025-02-02 20:59:59.129...]
 *
```

```
     * @param dateTime A DateTime
     * @param minutes The number of minutes to add
     * @return A DateTime with the given minutes added
     */
    DateTime AddMinutes(DateTime dateTime, long minutes);
```

AddSeconds

```
    /**
     * Adds a number of seconds to a DateTime
     *
     * Examples:
     * given: [2025-05-13 15:30:24.612...], 5
     *  expect: [2025-05-13 15:30:29.612...]
     * given: [2025-02-02 21:01:59.129...], -2
     *  expect: [2025-02-02 21:01:57.129...]
     *
     * @param dateTime A DateTime
     * @param seconds The number of seconds to add
     * @return A DateTime with the given seconds added
     */
    DateTime AddSeconds(DateTime dateTime, long seconds);
```

AddMilliseconds

```
    /**
     * Adds a number of milliseconds to a DateTime
     *
     * Examples:
     * given: [2025-05-13 15:30:24.612...], 5
     *  expect: [2025-05-13 15:30:24.617...]
     * given: [2025-02-02 21:01:59.129...], -2
     *  expect: [2025-02-02 21:01:59.127...]
     *
     * @param dateTime A DateTime
     * @param milliseconds The number of milliseconds to add
     * @return A DateTime with the given milliseconds added
     */
    DateTime AddMilliseconds(DateTime dateTime, long milliseconds);
```

WithTimeZone

```
/**
 * Returns a new Date or DateTime that represents the
 * same local time or date in the new time zone as
 * the given date
 *
 * Examples:
 * given: [2025-05-13 22:34:15.532-05:00@America/New_York],
 *        "Australia/Sydney"
 *  expect: [2025-05-13 22:34:15.532+11:00@Australia/Sydney]
 * given: [1925-01-14 06:47:01.003+11:00@Australia/Sydney],
 *        "America/New_York"
 *  expect: [1925-01-14 06:47:01.003-05:00@America/New_York]
 *
 * @param date A Date or DateTime
 * @return A new Date or DateTime with just the time zone
 *         component adjusted
 */
T WithTimeZone(T date);
```

InTimeZone

```
/**
 * Returns a new Date or DateTime that represents the
 * same point in time in a given time zone
 *
 * Examples:
 * given: [2025-05-13 22:34:15.532-05:00@America/New_York],
 *        "Australia/Sydney"
 *  expect: [2025-05-14 16:34:15.532+11:00@Australia/Sydney]
 * given: [1925-01-14 06:47:01.003+11:00@Australia/Sydney],
 *        "America/New_York"
 *  expect: [1925-01-13 14:47:01.003-05:00@America/New_York]
 *
 * @param date A Date or DateTime
 * @return A new Date or DateTime in the new time zone
 */
T InTimeZone(T date);
```

ToDate

```
/**
 * Returns the Date component of a DateTime
 *
```

```
 * Examples:
 * given: [2025-05-13 22:34:15.532-05:00@America/New_York]
 *  expect: [2025-05-13 00:00:00.000-05:00@America/New_York]
 * given: [1925-01-14 06:47:01.003+11:00@Australia/Sydney]
 *  expect: [1925-01-14 00:00:00.000+11:00@Australia/Sydney]
 *
 * @param dateTime The DateTime to be truncated
 * @return A Date representing the Date component of the DateTime
 */
Date ToDate(DateTime dateTime);
```

YearsBetween

```
/**
 * Calculates the number of full years between two Dates
 *
 * Examples:
 * given: [2025-05-13 22:34:15.532-05:00@America/New_York]
 *        [2026-05-13 22:34:15.532-05:00@America/New_York]
 *  expect: 1
 * given: [2025-05-13 22:34:15.532-05:00@America/New_York]
 *        [2026-05-13 21:34:15.532-05:00@America/New_York]
 *  expect: 0
 *
 * @param first The first date
 * @param second The second date
 * @return The number of years between the first date and the second.
 *         May be negative if the second date is before the first.
 */
long YearsBetween(Date first, Date second);
```

MonthsBetween

```
/**
 * Calculates the number of full months between two Dates
 *
 * Examples:
 * given: [2025-05-13 22:34:15.532-05:00@America/New_York]
 *        [2025-06-13 22:34:15.532-05:00@America/New_York]
 *  expect: 1
 * given: [2025-05-13 22:34:15.532-05:00@America/New_York]
 *        [2025-06-13 21:34:15.532-05:00@America/New_York]
```

```
     *  expect: 0
     *
     * @param first The first date
     * @param second The second date
     * @return The number of months between the first date and the second
     *         May be negative if the second date is before the first.
     */
    long MonthsBetween(Date first, Date second);
```

DaysBetween

```
    /**
     * Calculates the number of full days between two Dates
     *
     * Examples:
     * given: [2025-05-13 22:34:15.532-05:00@America/New_York]
     *        [2025-05-14 22:34:15.532-05:00@America/New_York]
     *  expect: 1
     * given: [2025-05-13 22:34:15.532-05:00@America/New_York]
     *        [2025-05-14 21:34:15.532-05:00@America/New_York]
     *  expect: 0
     *
     * @param first The first date
     * @param second The second date
     * @return The number of days between the first date and the second.
     *         May be negative if the second date is before the first.
     */
    long DaysBetween(Date first, Date second);
```

HoursBetween

```
    /**
     * Calculates the number of full hours between two Dates
     *
     * Examples:
     * given: [2025-05-13 22:34:15.532-05:00@America/New_York]
     *        [2025-05-13 23:34:15.532-05:00@America/New_York]
     *  expect: 1
     * given: [2025-05-13 22:34:15.532-05:00@America/New_York]
     *        [2025-05-13 23:33:15.532-05:00@America/New_York]
     *  expect: 0
     *
```

```
 * @param first The first date
 * @param second The second date
 * @return The number of hours between the first date and the second.
 *         May be negative if the second date is before the first.
 */
long HoursBetween(Date first, Date second);
```

MinutesBetween

```
/**
 * Calculates the number of full minutes between two Dates
 *
 * Examples:
 * given: [2025-05-13 22:34:15.532-05:00@America/New_York]
 *        [2025-05-13 22:35:15.532-05:00@America/New_York]
 *  expect: 1
 * given: [2025-05-13 22:34:15.532-05:00@America/New_York]
 *        [2025-05-13 22:35:12.532-05:00@America/New_York]
 *  expect: 0
 *
 * @param first The first date
 * @param second The second date
 * @return The number of minutes between the first date and the secor
 *         May be negative if the second date is before the first.
 */
long MinutesBetween(Date first, Date second);
```

SecondsBetween

```
/**
 * Calculates the number of full seconds between two Dates
 *
 * Examples:
 * given: [2025-05-13 22:34:15.532-05:00@America/New_York]
 *        [2025-05-13 22:34:16.532-05:00@America/New_York]
 *  expect: 1
 * given: [2025-05-13 22:34:15.532-05:00@America/New_York]
 *        [2025-05-13 22:34:16.122-05:00@America/New_York]
 *  expect: 0
 *
 * @param first The first date
 * @param second The second date
```

```
 * @return The number of seconds between the first date and the secon
 *         May be negative if the second date is before the first.
 */
long SecondsBetween(Date first, Date second);
```

MillisecondsBetween

```
/**
 * Calculates the number of full milliseconds between two Dates
 *
 * Examples:
 * given: [2025-05-13 22:34:15.532-05:00@America/New_York]
 *        [2025-05-13 22:34:15.533-05:00@America/New_York]
 *  expect: 1
 * given: [2025-05-13 22:34:15.532-05:00@America/New_York]
 *        [2025-05-13 23:34:16.532-05:00@America/New_York]
 *  expect: 1000
 *
 * @param first The first date
 * @param second The second date
 * @return The number of milliseconds between the first date and the
 *         May be negative if the second date is before the first.
 */
long MillisecondsBetween(Date first, Date second);
```

FormatDate

```
/**
 * Formats a Date or DateTime into a String
 *
 * Examples:
 * given: [2025-02-13 22:34:15.532-05:00@America/New_York],
 *        "d MMM yy HH:mm:ssxxx'@'VV"
 *  expect: 13 Feb 25 22:34:15-05:00@America/New_York
 * given: [2025-07-13 22:34:15.532-05:00@America/New_York],
 *        "d/M"
 *  expect: "13/7"
 *
 * @param date A Date or DateTime to format
 * @param pattern A pattern to format the Date or DateTime
 * @return A formatted version of the given Date or DateTime
```

```
 */
String FormatDate(Date date, String pattern);
```

Pattern Strings are explained in the JDK Documentation.

ParseDate

```
/**
 * Parses a Date from a String
 *
 * Examples:
 * given: "2025-02-17",
 *        "yyyy-MM-dd"
 *  expect: [2025-02-17 00:00:00.000+XX:XX@XXX]
 *          where XX:XX@XXX depends on your system's local time zone
 * given: "1/4/25",
 *        "d/M/yy",
 *        "Australia/Sydney"
 *  expect: [2025-04-01 00:00:00.000+11:00@Australia/Sydney]
 *
 * @param text A String representing a date
 * @param pattern A pattern describing the format of the String
 * @param timeZone An identifier for a time zone.
 *          e.g. "Australia/Sydney"
 *        OPTIONAL, default = Your local system time zone
 * @return The Date represented by the given String
 * @implSpec
 *   Precondition: The String needs to represent a valid date
 *                 in a valid format
 */
Date ParseDate(String text, String pattern, String timeZone);
```

Pattern Strings are explained in the JDK Documentation.

ParseDateTime

```
/**
 * Parses a DateTime from a String
 *
 * Examples:
 * given: "2025-02-17 09:00",
 *        "yyyy-MM-dd HH:mm"
 *  expect: [2025-02-17 09:00:00.000+XX:XX@XXX]
 *          where XX:XX@XXX depends on your system's local time zone
```

```
     * given: "1/4/25 14:45:13+11:00 in Australia/Sydney",
     *        "d/M/yy HH:mm:ssxxx 'in' VV"
     *  expect: [2025-04-01 14:15:13.000+11:00@Australia/Sydney]
     *
     * @param text A String representing a date and time
     * @param pattern A pattern describing the format of the String
     * @return The DateTime represented by the given String
     * @implSpec
     *   Precondition: The String needs to represent a valid date
     *                 and time in a valid format
     */
    DateTime ParseDateTime(String text, String pattern);
```

Pattern Strings are explained in the JDK Documentation.

HigherOrder

Functions

Bind ; Map ; BuildList ; Sort ; Fold ; FoldLeft ; Map ; Map ; Filter

Bind

```
    /**
     * Applies a function to a Maybe value if it is Something,
     * otherwise returns Nothing
     *
     * Examples:
     * given: Something(50), x -> Something(x+10)
     *  expect: Something(60)
     * given: Something(50), x -> Nothing()
     *  expect: Nothing()
     * given: Nothing(), x -> Something(x+10)
     *  expect: Nothing()
     *
     * @param maybe The Maybe-value to whose content we want to
     *          apply a function
     * @param fun The function we want to apply to the value in the Maybe
     */
    <T, U> Maybe<U> Bind(Maybe<T> maybe, Function<T, Maybe<U>> fun);
```

Map

```
/**
 * Applies a function to a Maybe value if it is Something,
 * otherwise returns Nothing
 *
 * Examples:
 * given: Something(50), x -> x+10
 *  expect: Something(60)
 * given: Nothing(), x -> x+10
 *  expect: Nothing()
 *
 * @param maybe The Maybe-value to whose content we want to
 *          apply a function
 * @param fun The function we want to apply to the value in the Maybe
 */
<T, U> Maybe<U> Map(Maybe<T> maybe, Function<T, U> fun);
```

BuildList

```
/**
 * Creates a list of a given length based on a generator function
 *
 * Examples:
 * given: 0, x->x
 *  expect: []
 * given: 4, x->x*2
 *  expect: [0, 2, 4, 6]
 *
 * @param length The length of the new list
 * @param generator A function that, given the index of an element
 *          in the list, produces the element
 * @return A new list with elements constructed using the generator
 */
<T> ConsList<T> BuildList(int length, Function<Integer, T> generator)
```

Sort

```
/**
 * Sorts a given list, using a given comparison function.
 *
 * Examples:
 * given: [], (x,y)->0
```

```
     *  expect: []
     * given: [4, 1, 3, 2], (x,y)->0
     *  expect: [4, 1, 3, 2]
     * given: [4, 1, 3, 2], (x,y) -> -Compare(x,y)
     *  expect: [4, 3, 2, 1]
     *
     * @param list The list to be sorted
     * @param comparator The function to use to compare any two elements
     *          of T. Its behavior should be like for the
     *          Compare function, i.e. return 0 if the elements
     *          are equal, a negative integer if the first
     *          element is "less" than the second, and a
     *          positive integer otherwise.
     * @return A version of the given list, sorted using the comparator
     */
    <T> ConsList<T> Sort(ConsList<T> list, BiFunction<T, T, Integer> comp
```

Fold

```
    /**
     * Combines the values of a list with a given aggregator
     * function, following a normal recursive template.
     *
     * Examples:
     * given: (x, y) -> x + y, 0, []
     *  expect: 0
     * given: (x, y) -> x + y, 0, [1, 2, 3]
     *  expect: 6
     * given: (x, y) -> x * y, 1, [1, 2, 3]
     *  expect: 6
     * given: (x, y) -> x + y, "", ["A", "B", "C"]
     *  expect: ABC
     *
     * @param aggregator The function used to combine the elements of
     *          the list. The first argument is an element of
     *          the list, the second argument is an intermediate
     *          result of having already combined some elements
     *          of the list.
     * @param start The starting value of the combination result
     * @param list The list whose elements should be combined
     * @return The result of combining the elements of the list
```

```
 */
<S, T> T Fold(BiFunction<S, T, T> aggregator, T start, ConsList<S> li
```

FoldLeft

```
/**
 * Combines the values of a list with a given aggregator
 * function, combining values from the start of the list.
 *
 * Examples:
 * given: (x, y) -> x + y, 0, []
 *  expect: 0
 * given: (x, y) -> x + y, 0, [1, 2, 3]
 *  expect: 6
 * given: (x, y) -> x * y, 1, [1, 2, 3]
 *  expect: 6
 * given: (x, y) -> x + y, "", ["A", "B", "C"]
 *  expect: CBA
 *
 * @param aggregator The function used to combine the elements of
 *          the list. The first argument is an element of
 *          the list, the second argument is an intermediate
 *          result of having already combined some elements
 *          of the list.
 * @param start The starting value of the combination result
 * @param list The list whose elements should be combined
 * @return The result of combining the elements of the list
 */
<S, T> T FoldLeft(BiFunction<S, T, T> aggregator, T start, ConsList<S
```

Map

```
/**
 * Applies a function to each element of a list, and produces
 * a list of the results of those function calls
 *
 * Examples:
 * given: x -> x + 1, []
 *  expect: []
 * given: x -> x + 1, [1, 2, 3]
 *  expect: [2, 3, 4]
 * given: x -> ToString(x), [1, 2]
```

```
 *  expect: ["1", "2"]
 *
 * @param mapper The function to convert values from S to T
 * @param list The list of arguments to the mapper
 * @return A list of the results of applying the mapper
 *         to each element of the given list.
 */
<S, T> ConsList<T> Map(Function<S, T> mapper, ConsList<S> list);
```

Map

```
/**
 * Applies a two-argument function to each pair of elements of two
 * equal-length lists, and produces a list of the results of those
 * function calls
 *
 * Examples:
 * given: (x, y) -> x + y, [], []
 *  expect: []
 * given: (x, y) -> x + y, [1, 2, 3], [4, 5, 6]
 *  expect: [5, 7, 9]
 * given: (x, y) -> ToString(x), [1, 2], [3, 4]
 *  expect: ["1", "2"]
 *
 * @param mapper The function apply to values of types S and T
 * @param list1 The first list of arguments
 * @param list2 The second list of arguments
 * @return A list of the results of applying the mapper
 *         to each pair of elements of the given lists.
 * @implSpec
 *   Precondition: list1 and list2 must be of equal length
 */
<S, T, U> ConsList<U> Map(BiFunction<S, T, U> mapper, ConsList<S> lis
```

Filter

```
/**
 * Filters a list based on a given predicate
 *
 * Examples:
 * given: x -> x%2 == 0, [1,2,3,4,5]
 *  expect: [2,4]
```

```
 * given: x -> Length(x) > 0, [[], [1,2,3], [5]]
 *  expect: [[1,2,3], [5]]
 *
 * @param predicate The predicate that elements of the list need
 *          to satisfy to stay in the filtered list
 * @param list The list we want to filter
 * @return A list that contains only those elements of the given
 *         list where the predicate returns true.
 */
<T> ConsList<T> Filter(Predicate<T> predicate, ConsList<T> list);
```

Lambdas

Functions

Curry ; UnCurryFunction ; CurryPredicate ; UnCurryPredicate ; PairFunction ; UnpairFunction ; Compose ; ComposeAnd ; ComposeOr ; BiComposeAnd ; BiComposeOr

Curry

```
/**
 * Converts a two-argument BiFunction to its curried equivalent
 *
 * Examples:
 * given: (x, y) -> x + y
 *  expect: x -> y -> x + y (effectively)
 *
 * @param fun The function to be curried
 * @return The curried version of the given function
 */
<A1, A2, R> Function<A1, Function<A2, R>> Curry(BiFunction<A1, A2, R>
```

UnCurryFunction

```
/**
 * Converts a curried two-argument function to a BiFunction
 *
 * Examples:
 * given: x -> y -> x + y
 *  expect: (x, y) -> x + y (effectively)
 *
 * @param fun The function to be uncurried
 * @return The uncurried version of the given function
```

```
 */
<A1, A2, R> BiFunction<A1, A2, R> UnCurryFunction(Function<A1, Functi
```

CurryPredicate

```
/**
 * Converts a two-argument BiPredicate to its curried equivalent
 *
 * Examples:
 * given: (x, y) -> x + y == 5
 *  expect: x -> y -> x + y == 5 (effectively)
 *
 * @param fun The predicate to be curried
 * @return The curried version of the given predicate
 */
<A1, A2> Function<A1, Predicate<A2>> CurryPredicate(BiPredicate<A1, A
```

UnCurryPredicate

```
/**
 * Converts a curried two-argument function to a BiPredicate
 *
 * Examples:
 * given: x -> y -> x + y == 5
 *  expect: (x, y) -> x + y == 5 (effectively)
 *
 * @param fun The predicate to be uncurried
 * @return The uncurried version of the given function/predicate
 */
<A1, A2> BiPredicateA1, A2> UnCurryPredicate(Function<A1, Predicate<A
```

PairFunction

```
/**
 * Converts a two-argument BiFunction to a function
 * that accepts a pair of arguments.
 *
 * Examples:
 * given: (x, y) -> x + y
 *  expect: x -> x.first() + x.second() (effectively)
 *
```

```
 * @param fun The function to convert to a single-argument function
 * @return The pair-curried version of the given function
 */
<A1, A2, R> Function<Pair<A1, A2>, R> PairFunction(BiFunction<A1, A2,
```

UnpairFunction

```
/**
 * Converts a function on pairs to a BiFunction
 *
 * Examples:
 * given: x -> x.first() + y.second()
 *  expect: (x, y) -> x + y (effectively)
 *
 * @param fun The function to have its arguments expanded
 * @return The pair-uncurried version of the given function
 */
<A1, A2, R> BiFunction<A1, A2, R> UnpairFunction(Function<Pair<A1, A2
```

Compose

```
/**
 * Composes to functions to a new function that first
 * applies the first function to its argument, and then
 * the second function to the output of the first call.
 *
 * Examples:
 * given: StringToInt, x -> x + 1
 *  expect: x -> StringToInt(x) + 1 (effectively)
 *
 * @param fun1 The first function to compose
 * @param fun2 The second function to compose
 * @return The composed function
 */
<T, U, R> Function<T, R> Compose(Function<T, U> fun1, Function<U, R>
```

ComposeAnd

```
/**
 * Composes two predicates to a new predicate that
 * holds if both composed predicates hold on its
```

```
     * argument
     *
     * Examples:
     * given: x -> x > 50, x -> x < 100
     *  expect: x -> x > 50 && x < 100 (effectively)
     *
     * @param pred1 The first predicate to compose
     * @param pred2 The second predicate to compose
     * @return The composed predicate
     */
    <T> Predicate<T> ComposeAnd(Predicate<T> pred1, Predicate<T> pred2);
```

ComposeOr

```
    /**
     * Composes two predicates to a new predicate that
     * holds if either of the composed predicates holds
     * on its argument
     *
     * Examples:
     * given: x -> x > 50, x -> x < -50
     *  expect: x -> x > 50 || x < -50 (effectively)
     *
     * @param pred1 The first predicate to compose
     * @param pred2 The second predicate to compose
     * @return The composed predicate
     */
    <T> Predicate<T> ComposeOr(Predicate<T> pred1, Predicate<T> pred2);
```

BiComposeAnd

```
    /**
     * Composes two predicates on different types to
     * a new BiPredicate that holds if both composed
     * predicates hold on their respective arguments
     *
     * Examples:
     * given: x -> x > 50, x -> Equals(x, "A")
     *  expect: (x, y) -> x > 50 && Equals(y, "A") (effectively)
     *
     * @param pred1 The first predicate to compose
     * @param pred2 The second predicate to compose
     * @return The composed predicate
```

```
 */
<T, U> BiPredicate<T, U> BiComposeAnd(Predicate<T> pred1, Predicate<U
```

BiComposeOr

```
/**
 * Composes two predicates on different types to
 * a new BiPredicate that holds if either of the
 * composed predicates holds on their respective
 * argument
 *
 * Examples:
 * given: x -> x > 50, x -> Equals(x, "A")
 *  expect: (x, y) -> x > 50 || Equals(y, "A") (effectively)
 *
 * @param pred1 The first predicate to compose
 * @param pred2 The second predicate to compose
 * @return The composed predicate
 */
<T, U> BiPredicate<T, U> BiComposeOr(Predicate<T> pred1, Predicate<U>
```

Lists

Provides support for lists. We'll use the following notation in examples:

- `[]` describes the empty list `Nil<>()`
- `x :: l` describes a `Cons<>(x, l)`
- `[a, b, c]` describes a list, represented by `Cons<>(a, Cons<>(b, Cons<>(c, Nil<>())))`

Itemizations

ConsList

ConsList

```
/**
 * Represents lists of values.
 * It is one of:
 * - The list is empty
 * - A non-empty list, consisting of at least one element,
 *    plus some other list
 */
```

```
sealed interface ConsList<T>permits Nil, Cons{}

// TEMPLATE:
// {
//     ...
//     return ... switch(conslist) {
//         case Nil<T>() ->  ... ;
//         case Cons<T>(var element, var rest) ->
//              ... element ... [recursiveCall](... rest ...) ... ;
//     } ...;
// }

/**
 * Represents the empty list
 * Examples:
 * - Nil<Integer>()
 * - Nil<String>()
 */
record Nil<T>(
) implements ConsList<T>{}
/**
 * Represents a non-empty list consisting of at least one element
 * Examples:
 * - Cons<String>("Hello", Nil<String>())
 * - Cons<Integer>(1, Cons<Integer>(2, Nil<Integer>()))
 * @param element The first element of the list
 * @param rest The rest of the list, which may be empty
 */
record Cons<T>(
    T element,
    ConsList<T> rest) implements ConsList<T>{}
```

Functions

MakeList ; BuildList ; BuildList ; IsEmpty ; First ; Rest ; Length ; Nth ; Append ; Sort ; Zip ; Unzip

MakeList

```
/**
 * Creates a new ConsList with the given elements
 *
 * Examples:
 * given: [no arguments]
```

```
     *  expect: []
     * given: 1, 2
     *  expect: [1,2]
     *
     * @param elements Arbitrarily many elements that form the new list
     * @return A ConsList that contains the given elements
     */
    <T> ConsList<T> MakeList(T... elements);
```

BuildList

```
    /**
     * Creates a new ConsList<Integer> of given length
     * with elements ranging from 0 up to (excluding) length
     *
     * Examples:
     * given: 0
     *  expect: []
     * given: 4
     *  expect: [0, 1, 2, 3]
     *
     * @param length The length of the new list
     * @return A new list of integers from 0 up to (excluding) lenght
     */
    ConsList<Integer> BuildList(int length);
```

BuildList

```
    /**
     * Creates a list of a given length of all the same value
     *
     * Examples:
     * given: 0, "A"
     *  expect: []
     * given: 4, true
     *  expect: [true, true, true, true]
     *
     * @param length The length of the new list
     * @param value  - will be used as every element in the list
     * @return A new list of length times value
     */
    <T> ConsList<T> BuildList(int length, T value);
```

IsEmpty

```
/**
 * Returns whether or not a list is empty
 *
 * Examples:
 * given: []
 *  expect: true
 * given: [1, 2, 3]
 *  expect: false
 *
 * @param list A possibly empty list
 * @return true if the list is empty, false if not
 */
<T> boolean IsEmpty(ConsList<T> list);
```

First

```
/**
 * Returns the first element of a list
 *
 * Examples:
 * given: [1]
 *  expect: 1
 * given: ["B", "C", "A", "D"]
 *  expect: "B"
 *
 * @param list A list - must be non-empty
 * @return The first element of the given list
 */
<T> T First(ConsList<T> list);
```

Rest

```
/**
 * Returns the rest of a list, past the first element
 *
 * Examples:
 * given: [1]
 *  expect: []
 * given: ["B", "C", "A", "D"]
 *  expect: ["C", "A", "D"]
 *
 * @param list A list - must be non-empty
```

```
     * @return The rest of the given list, past the first element
     */
    <T> ConsList<T> Rest(ConsList<T> list);
```

Length

```
    /**
     * Returns the length of a given list
     *
     * Examples:
     * given: []
     *  expect: 0
     * given: [1, 2, 3]
     *  expect: 3
     *
     * @param list The list whose length should be computed
     * @return The length of the given list
     */
    <T> int Length(ConsList<T> list);
```

Nth

```
    /**
     * Returns the nth element of a given list
     *
     * Examples:
     * given: [1], 0
     *  expect: 1
     * given: [1,2,3,4], 3
     *  expect: 4
     *
     * @param list A list whose element we want to extract
     * @param index The 0-based index of an element in the list
     * @return The element at the given index in the given list
     * @implSpec
     *   Precondition: 0 <= index < Length(list)
     */
    <T> T Nth(ConsList<T> list, int index);
```

Append

```
    /**
     * Appends two lists
```

```
     *
     * Examples:
     * given: [], []
     *  expect: []
     * given: [1, 2, 4], [3, 6]
     *  expect: [1, 2, 4, 3, 6]
     *
     * @param list1 The first list
     * @param list2 The list to be appended to the first list
     * @return A new list that is the result of appending the given ones
     */
    <T> ConsList<T> Append(ConsList<T> list1, ConsList<T> list2);
```

Sort

```
    /**
     * Sorts a given list of a comparable type T.
     * Comparable Types are:
     *   - String
     *   - the number types (int, double, ...)
     *   - booleans
     *   - Date
     *   - DateTime
     *
     * Examples:
     * given: []
     *  expect: []
     * given: [4, 1, 3, 2]
     *  expect: [1, 2, 3, 4]
     *
     * @param list The list to be sorted
     * @return A version of the given list, sorted by the standard
     *         ordering of its elements (as with the Compare function)
     */
    <T> ConsList<T> Sort(ConsList<T> list);
```

Zip

```
    /**
     * Given two lists of equal length, produces a list
     * of pairs of their elements
     *
     * Examples:
```

```
 * given: [1,2,3], [4,5,6]
 *  expect: [Pair(1,4), Pair(2,5), Pair(3,6)]
 * given: [9, 10], ["A", "B"]
 *  expect: [Pair(9, "A"), Pair(10, "B")]
 *
 * @param list1 The first list
 * @param list2 The second list
 * @return A list of pairs of elements at the same indices
 *         as in the original lists
 */
<S, T> ConsList<Pair<S, T>> Zip(ConsList<S> list1, ConsList<T> list2)
```

Unzip

```
/**
 * Given a list of Pairs, returns a pair of lists
 * containing the first and second elements of the
 * Pairs, respectively
 *
 * Examples:
 * given: [Pair(1,4), Pair(2,5), Pair(3,6)]
 *  expect: Pair([1,2,3], [4,5,6])
 * given: [Pair(9, "A"), Pair(10, "B")]
 *  expect: Pair([9, 10], ["A", "B"])
 *
 * @param list1 The list of pairs
 * @return A pair of lists of the first and second elements of the pa
 */
<S, T> Pair<ConsList<S>, ConsList<T>> Unzip(ConsList<Pair<S, T>> list
```

Maps

A map (a.k.a. dictionary) is a data structure that associates keys to values, and explicitly stores these associations as key-value pairs. In a map, any two pairs must have different keys. There might be different keys associated to the same value, though. The order of key-value pairs is undefined in a map, meaning that one should not assume any particular order. We will be working with two different implementations of maps: ConsList-based maps and Hash maps.

ConsList-based maps rely on `ConsList<Pair<K,V>>` to store the association of keys and values, with `K` being the type of the map keys, and `V` that of the map values. Besides, such

maps are immutable, meaning that functions operating on them return new maps instead of modifying them in-place

Hash maps are of type `Map<K,V>`. They are implemented under the hood smartly using the so-called hash functions to perform fast key look-ups, thus the name Hash maps. Besides, Hash maps are stateful maps (a.k.a. mutable) meaning that the functions operating on them modify the input maps in-place, instead of returning whole new maps.

The functions below provide support to perform operations on both kind of maps. We'll use the following notation in examples:

- `{}` denotes either: (1) the `ConsList`-based empty map, represented by `Nil<Pair<K,V>>()`, or (2) the empty Hash map
- `{k1:v1, k2:v2}` denotes either: (1) a `ConsList`-based map with two key-value pairs represented by either `Cons<>(Pair<>(k1,v1), Cons<>(Pair<>(k2,v2), Nil<>()))` or `Cons<>(Pair<>(k2,v2), Cons<>(Pair<>(k1,v1), Nil<>()))`, or (2) a Hash map with two key-value pairs

Functions

MakeConsMap ; MakeHashMap ; Put ; Put ; Remove ; Remove ; Get ; Get ; ContainsKey ; ContainsKey ; GetKeys ; GetKeys

MakeConsMap

```
/**
 * Creates a new ConsList-based map with the given key-value pairs.
 * If there is more than one key-value pair with the same key
 * in the given key-value pairs, the right-most one is the one that
 * prevails in the resulting map
 *
 * Examples:
 * given: [no arguments]
 *  expect: {}
 * given: Pair<>("COMP1110",91), Pair<>("COMP6710",139)
 *  expect: {"COMP1110": 91, "COMP6710": 139}
 * given: Pair<>("COMP1110",91), Pair<>("COMP1110",89)
 *  expect: {"COMP1110": 89}
 *
 * @param pairs Arbitrarily many pairs that form the new
 *              ConstList-based map
 * @return A ConsList-based map that contains the given pairs
 */
<K,V> ConsList<Pair<K,V>> MakeConsMap(Pair<K,V>... pairs);
```

MakeHashMap

```
/**
 * Creates a new Hash map with the given key-value pairs.
 * If there is more than one key-value pair with the same key
 * in the given key-value pairs, the right-most one is the one that
 * prevails in the resulting map
 *
 * Examples:
 * given: [no arguments]
 *  expect: {}
 * given: Pair<>("COMP1110",91), Pair<>("COMP6710",139)
 *  expect: {"COMP1110": 91, "COMP6710": 139}
 * given: Pair<>("COMP1110",91), Pair<>("COMP1110",89)
 *  expect: {"COMP1110": 89}
 *
 * @param pairs Arbitrarily many pairs that form the new
 *              Hash map
 * @return A new Hash map that contains the given pairs
 */
<K,V> Map<K,V> MakeHashMap(Pair<K,V>... pairs);
```

Put

```
/**
 * Given a (1) ConsList-based map, (2) a key, and (3) a value,
 * returns a new ConsList-based map with a new key-value
 * pair (2)-(3) added to (1) if (2) is not present in (1)
 * or the value associated to (2) updated with (3) if (2)
 * is present in (1)
 *
 * Examples:
 * given: {"COMP1110": 91, "COMP6710": 139}, "COMP1110", 89
 *  expect: {"COMP1110": 89, "COMP6710": 139}
 * given: {"COMP1110": 91, "COMP6710": 139}, "COMP1730", 223
 *  expect: {"COMP1110": 89, "COMP6710": 139, "COMP1730": 223}
 *
 * @param map A ConstList-based map
 * @param key The key of the key-value pair
 * @param value The value of the key-value pair
 * @return A new ConsList-based map with a new key-value map added
 *         to the input map if the given key is not present
 *         in the input map, or with the value associated to key
 *         updated with the given value if the key is present in
```

```
     *         the input map
     */
    <K,V> ConsList<Pair<K,V>> Put(ConsList<Pair<K,V>> map, K key, V value
```

Put

```
    /**
     * Given a (1) Hash map, (2) a key, and (3) a value,
     * updates the stateful map (1) in-place with a new key-value
     * pair (2)-(3) added to (1) if (2) is not present in (1)
     * or the value associated to (2) updated with (3) if (2)
     * is present in (1)
     *
     * Examples:
     * given: {"COMP1110": 91, "COMP6710": 139}, "COMP1110", 89
     *  expect: {"COMP1110": 89, "COMP6710": 139}
     * given: {"COMP1110": 91, "COMP6710": 139}, "COMP1730", 223
     *  expect: {"COMP1110": 89, "COMP6710": 139, "COMP1730": 223}
     *
     * @param map A Hash map to be updated in-place
     * @param key The key of the key-value pair
     * @param value The value of the key-value pair
     * @return void
     */
    <K,V> void Put(Map<K,V> map, K key, V value);
```

Remove

```
    /**
     * Given (1) a ConsList-based map, and (2) a key,
     * returns a new ConsList-based map where the key-value
     * pair associated to (2) is removed from the input map.
     * If (2) is not present in (1), it just returns a copy
     * of the input map.
     *
     * Examples:
     * given: {"COMP1110": 91, "COMP6710": 139}, "COMP1110"
     *  expect: {"COMP6710": 139}
     * given: {"COMP1110": 91, "COMP6710": 139}, "COMP6730"
     *  expect: {"COMP1110": 91, "COMP6710": 139}
     *
     * @param map A ConstList-based map
```

```
 * @param key The key of the key-value pair to be removed
 * @return A new ConsList-based map with a key-value pair
 *         removed from the input map if the given key is present
 *         in the input map, or a copy of the input map otherwise
 */
<K,V> ConsList<Pair<K,V>> Remove(ConsList<Pair<K,V>> map, K key);
```

Remove

```
/**
 * Given (1) a Hash map, and (2) a key,
 * updates the stateful map (1) in-place such that the key-value
 * pair associated to (2) is removed from (1).
 * If (2) is not present in (1), it leaves (1) untouched.
 *
 * Examples:
 * given: {"COMP1110": 91, "COMP6710": 139}, "COMP1110"
 *  expect: {"COMP6710": 139}
 * given: {"COMP1110": 91, "COMP6710": 139}, "COMP6730"
 *  expect: {"COMP1110": 91, "COMP6710": 139}
 *
 * @param map A Hash map to be updated in-place
 * @param key The key of the key-value pair to be removed
 * @return void
 */
<K,V> void Remove(Map<K,V> map, K key);
```

Get

```
/**
 * Given (1) a ConsList-based map, and (2) a key,
 * returns the Something<V>(value) associated to (2)
 * if the key is present in (1) or Nothing<V>() otherwise
 *
 * Examples:
 * given: {"COMP1110": 91, "COMP6710": 139}, "COMP1110"
 *  expect: Something<Integer>(91)
 * given: {"COMP1110": 91, "COMP6710": 139}, "COMP6730"
 *  expect: Nothing<Integer>()
 *
 * @param map A ConstList-based map
 * @param key The key whose associated value being looked-up
 * @return The value associated to the given key as
```

```
     *         Something<V>(value) if the key is present
     *         in the input map, and Nothing<V>() otherwise
     */
    <K,V> Maybe<V> Get(ConsList<Pair<K,V>> map, K key);
```

Get

```
    /**
     * Given (1) a Hash map, and (2) a key,
     * returns the Something<V>(value) associated to (2)
     * if the key is present in (1) or Nothing<V>() otherwise
     *
     * Examples:
     * given: {"COMP1110": 91, "COMP6710": 139}, "COMP1110"
     *  expect: Something<Integer>(91)
     * given: {"COMP1110": 91, "COMP6710": 139}, "COMP6730"
     *  expect: Nothing<Integer>()
     *
     * @param map A Hash map
     * @param key The key whose associated value being looked-up
     * @return The value associated to the given key as
     *         Something<V>(value) if the key is present
     *         in the input map, and Nothing<V>() otherwise
     */
    <K,V> Maybe<V> Get(Map<K,V> map, K key);
```

ContainsKey

```
    /**
     * Given (1) a ConsList-based map, and (2) a key,
     * returns true if (2) is present in (1), and false otherwise
     *
     * Examples:
     * given: {"COMP1110": 91, "COMP6710": 139}, "COMP1110"
     *  expect: true
     * given: {"COMP1110": 91, "COMP6710": 139}, "COMP6730"
     *  expect: false
     *
     * @param map A ConsList-based map
     * @param key The key being looked-up
     * @return true if the key is present in the input map,
     *         and false otherwise
```

```
 */
<K,V> boolean ContainsKey(ConsList<Pair<K,V>> map, K key);
```

ContainsKey

```
/**
 * Given (1) a Hash map, and (2) a key,
 * returns true if (2) is present in (1), and false otherwise
 *
 * Examples:
 * given: {"COMP1110": 91, "COMP6710": 139}, "COMP1110"
 *  expect: true
 * given: {"COMP1110": 91, "COMP6710": 139}, "COMP6730"
 *  expect: false
 *
 * @param map A Hash map
 * @param key The key being looked-up
 * @return true if the key is present in the input map,
 *         and false otherwise
 */
<K,V> boolean ContainsKey(Map<K,V> map, K key);
```

GetKeys

```
/**
 * Given a ConsList-based map, returns a list with the keys
 * present in the map
 *
 * Examples:
 * given: {}
 *  expect: []
 * given: {"COMP1110": 91, "COMP6710": 139}
 *  expect: ["COMP1110","COMP6710"]
 *
 * @param map A ConsList-based map
 * @return A list with the keys present in the input map
 */
<K,V> ConsList<K> GetKeys(ConsList<Pair<K,V>> map);
```

GetKeys

```
/**
 * Given a Hash map, returns a list with the keys
```

```
 * present in the map
 *
 * Examples:
 * given: {}
 *  expect: []
 * given: {"COMP1110": 91, "COMP6710": 139}
 *  expect: ["COMP1110","COMP6710"]
 *
 * @param map A Hash map
 * @return A list with the keys present in the input map
 */
<K,V> ConsList<K> GetKeys(Map<K,V> map);
```

Math

Constants

PI ; E

PI

```
/**
 * Represents the mathematical constant pi
 */
double PI;
```

E

```
/**
 * Represents the mathematical constant e
 */
double E;
```

Functions

Abs ; Exp ; Pow ; Sqrt ; Round ; RoundInt ; Ceil ; CeilInt ; Floor ; FloorInt ; Log ; Log10 ; Sin ; Cos ; Tan ; ASin ; ACos ; ATan

Abs

```
/**
 * Converts a numeric value to its absolute value
 *
 * Examples:
 * given: 15
```

```
     *  expect: 15
     * given: -5.2
     *  expect: 5.2
     *
     * @param value An int, long, double, or float
     * @return The absolute value of the given numeric value,
     *         in the same type
     */
    T Abs(T value);
```

Exp

```
    /**
     * Returns Euler's number E raised to the given value
     *
     * Examples:
     * given: 0.0
     *  expect: 1.0
     * given: 2.0
     *  expect: 7.38905...
     *
     * @param value The power to which to raise E
     * @return E raised to value
     */
    double Exp(double value);
```

Pow

```
    /**
     * Raises the first argument to the power of the second argument
     *
     * Examples:
     * given: 4.0, 2.0
     *  expect: 16.0
     * given: 7.0, 0.0
     *  expect: 1.0
     *
     * @param base The base of the exponentiation
     * @param exponent The exponent
     * @return The base to the power of the exponent
     */
    double Pow(double base, double exponent);
```

Sqrt

```
/**
 * Calculates the square root of a given value
 *
 * Examples:
 * given: 4.0
 *  expect: 2.0
 * given: 1.0
 *  expect: 1.0
 *
 * @param value A value >=0 of which to calculate the square root
 * @return The square root of value
 */
double Sqrt(double value);
```

Round

```
/**
 * Rounds a given floating point number to a long
 *
 * Examples:
 * given: 4.5
 *  expect: 5l
 * given: 7.1
 *  expect: 7l
 *
 * @param number The value to be rounded
 * @return A long, rounded with ties going to positive infinity
 */
long Round(double number);
```

RoundInt

```
/**
 * As Round above, but returns an int
 *
 * Examples:
 * given: 4.5
 *  expect: 5
 * given: 7.1
 *  expect: 7
 *
 * @param number The value to be rounded
```

```
 * @return An integer, rounded with ties going to positive infinity
 */
int RoundInt(double number);
```

Ceil

```
/**
 * Rounds a given floating point number up to a long
 *
 * Examples:
 * given: 4.5
 *  expect: 5l
 * given: 7.1
 *  expect: 8l
 *
 * @param number The value to be rounded up
 * @return A long, rounded up towards positive infinity
 */
long Ceil(double number);
```

CeilInt

```
/**
 * As Ceil above, but returns an int
 *
 * Examples:
 * given: 4.5
 *  expect: 5
 * given: 7.1
 *  expect: 8
 *
 * @param number The value to be rounded up
 * @return An integer, rounded up towards positive infinity
 */
int CeilInt(double number);
```

Floor

```
/**
 * Rounds a given floating point number down to a long
 *
 * Examples:
 * given: 4.5
```

```
 *  expect: 41
 * given: 7.1
 *  expect: 71
 *
 * @param number The value to be rounded down
 * @return A long, rounded down towards negative infinity
 */
long Floor(double number);
```

FloorInt

```
/**
 * As Floor above, but returns an int
 *
 * Examples:
 * given: 4.5
 *  expect: 4
 * given: 7.1
 *  expect: 7
 *
 * @param number The value to be rounded down
 * @return An integer, rounded down towards negative infinity
 */
int FloorInt(double number);
```

Log

```
/**
 * Computes the natural logarithm of a given number
 *
 * Examples:
 * given: E
 *  expect: 1.0
 * given: E*E
 *  expect: 2.0
 *
 * @param number The number whose natural logarithm we want to comput
 * @return The natural logarithm of the given number
 */
double Log(double number);
```

Log10

```
/**
 * Computes the logarithm (base 10) of a given number
 *
 * Examples:
 * given: 10.0
 *  expect: 1.0
 * given: 100.0
 *  expect: 2.0
 *
 * @param number The number whose logarithm (base 10) we want to comp
 * @return The logarithm (base 10) of the given number
 */
double Log10(double number);
```

Sin

```
/**
 * Computes the trigonometric sine of an angle
 *
 * Examples:
 * given: PI
 *  expect: 0.0
 * given: PI/2.0
 *  expect: 1.0
 *
 * @param angle The angle in radians (i.e. PI = 180 degrees)
 * @return The sine of the given angle
 */
double Sin(double angle);
```

Cos

```
/**
 * Computes the trigonometric cosine of an angle
 *
 * Examples:
 * given: PI
 *  expect: -1.0
 * given: PI/2.0
 *  expect: 0.0
 *
 * @param angle The angle in radians (i.e. PI = 180 degrees)
```

```
 * @return The cosine of the given angle
 */
double Cos(double angle);
```

Tan

```
/**
 * Computes the trigonometric tangent of an angle
 *
 * Examples:
 * given: PI
 *  expect: 0.0
 * given: PI/4.0
 *  expect: 1.0
 *
 * @param angle The angle in radians (i.e. PI = 180 degrees)
 * @return The tangent of the given angle
 */
double Tan(double angle);
```

ASin

```
/**
 * Computes the trigonometric arc sine of an angle
 *
 * Examples:
 * given: 0.0
 *  expect: PI
 * given: 1.0
 *  expect: PI/2.0
 *
 * @param angle The angle in radians (i.e. PI = 180 degrees)
 * @return The arc sine of the given angle
 *         Value is between -PI/2 and PI/2
 */
double ASin(double angle);
```

ACos

```
/**
 * Computes the trigonometric arc cosine of an angle
 *
 * Examples:
```

```
 * given: -1.0
 *  expect: PI
 * given: 0.0
 *  expect: PI/2.0
 *
 * @param angle The angle in radians (i.e. PI = 180 degrees)
 * @return The arc cosine of the given angle
 *         Value is between 0.0 and PI
 */
double ACos(double angle);
```

ATan

```
/**
 * Computes the trigonometric arc tangent of an angle
 *
 * Examples:
 * given: 0.0
 *  expect: PI
 * given: 1.0
 *  expect: PI/4.0
 *
 * @param angle The angle in radians (i.e. PI = 180 degrees)
 * @return The arc tangent of the given angle
 *         Value is between -PI/2 and PI/2
 */
double ATan(double angle);
```

Strings

Functions

Format ; Concatenate ; SubString ; Contains ; StartsWith ; EndsWith ; Replace ; Length ; IndexOf ; LastIndexOf ; Trim ; UpperCase ; LowerCase ; GetCharAt

Format

```
/**
 * A wrapper for Java's and therefore C's format capabilities.
 * Advanced users only!
 *
 * Examples:
 * given: "$ %(,.2f", 6217.58
 *  expect: "$ (6,217.58)"
```

```
     * given: "Hello, %s!", "John"
     *  expect: "Hello, John!"
     *
     * @param format A format string (see explanation below)
     * @param args Arbitrarily many more arguments to be formatted into t
     * @return A formatted String
     */
    String Format(String format, Object... args);
```

This is a Wrapper for Java's String formatting functionality. You don't technically need it, and it is here as a convenience feature for some advanced users. You'll need to consult the documentation of format strings.

Concatenate

```
    /**
     * Concatenates a number of Strings
     *
     * Examples:
     * given: "a", "b", "c"
     *  expect: "abc"
     * given: "Hello, ", "World!"
     *  expect: "Hello, World!"
     *
     * @param strings The strings to concatenate
     * @return The String resulting from concatenating the arguments
     */
    String Concatenate(String... strings);
```

SubString

```
    /**
     * Returns a part of a given String
     *
     * Examples:
     * given: "Hello", 3
     *  expect: "lo"
     * given: "Hello", 1, 4
     *  expect: "ell"
     *
     * @param string The string whose part we want to extract
     * @param startIndex The index (>=0, <Length(String)) of the first
     *          character to include in the output.
```

```
 * @param endIndex The index (>=0, <=Length(String)) of the first
 *          character to *not* be included in the output.
 *        OPTIONAL, default = Length(string)
 * @return The part of the String between the given indices.
 */
String SubString(String string, int startIndex, int endIndex);
```

Contains

```
/**
 * Checks whether a String contains another String
 *
 * Examples:
 * given: "Hello,", "He"
 *  expect: true
 * given: "Hello", "W"
 *  expect: false
 *
 * @param haystack The larger String, in which we are searching
 * @param needle The String we are searching for
 * @return true if haystack contains needle, false if not
 */
boolean Contains(String haystack, String needle);
```

StartsWith

```
/**
 * Checks whether a String starts with another String
 *
 * Examples:
 * given: "Hello", "Hell"
 *  expect: true
 * given: "Hello", "ello"
 *  expect: false
 *
 * @param string The larger String whose start we are looking at
 * @param prefix The prefix we are looking for
 * @return True if string starts with prefix, false if not
 */
boolean StartsWith(String string, String prefix);
```

EndsWith

```
/**
 * Checks whether a String ends with another String
 *
 * Examples:
 * given: "Hello", "Hell"
 *  expect: false
 * given: "Hello", "ello"
 *  expect: true
 *
 * @param string The larger String whose end we are looking at
 * @param postfix The postfix we are looking for
 * @return True if string ends with postfix, false if not
 */
boolean EndsWith(String string, String postfix);
```

Replace

```
/**
 * Replaces a String with another String in a given String
 *
 * Examples:
 * given: "Hello", "Hi", "Hello World!"
 *  expect: "Hi World!"
 * given: "ell", "", "Hello, hello, hello!"
 *  expect: "Ho, ho, ho!"
 * given: "x", "y", "Hello"
 *  expect: "Hello"
 *
 * @param replace The String to be replaced
 * @param withString The String to replace "replace" with
 * @param inString The overall String in which "withString" should
 *          replace "replace"
 * @return "inString" with all occurences of "replace"
 *         replaced with "withString"
 */
String Replace(String replace, String withString, String inString);
```

Length

```
/**
 * Returns the length of a String
 *
 * Examples:
```

```
 * given: ""
 *  expect: 0
 * given: "Hello"
 *  expect: 5
 *
 * @param string The String whose length we want to determine
 * @return The length of the String
 */
int Length(String string);
```

IndexOf

```
/**
 * Returns the start index of a String in a given String
 *
 * Examples:
 * given: "o", "Hello"
 *  expect: 4
 * given: "x", "Hello"
 *  expect: -1
 * given: "ell", "Well well well", 4
 *  expect: 6
 *
 * @param find The String to find
 * @param inString The String in which to find the other String
 * @param fromIndex An index before which occurrences of find
 *          should be ignored
 *        OPTIONAL, default = 0
 * @return The index (>=0) of the first occurence of find in inString
 *         after fromIndex, or -1 if it does not occur.
 */
int IndexOf(String find, String inString, int fromIndex);
```

LastIndexOf

```
/**
 * Returns the last start index of a String in a given String
 *
 * Examples:
 * given: "oo", "Hellooo"
 *  expect: 5
 * given: "x", "Hellooo"
```

```
 *  expect: -1
 * given: "ell", "Well well well", 10
 *  expect: 6
 *
 * @param find The String to find
 * @param inString The String in which to find the other String
 * @param fromIndex An index after which occurrences of find
 *          should be ignored
 *        OPTIONAL, default = Length(inString)
 * @return The index (>=0) of the last occurence of find in inString
 *         before fromIndex, or -1 if it does not occur.
 */
int LastIndexOf(String find, String inString, int fromIndex);
```

Trim

```
/**
 * Removes whitespace from the ends of a String
 *
 * Examples:
 * given: "   a   "
 *  expect: "a"
 * given: "Hello "
 *  expect: "Hello"
 *
 * @param string The String that should be trimmed
 * @return A version of the given String without
 *         preceding or trailing whitespace
 */
String Trim(String string);
```

UpperCase

```
/**
 * Converts all applicable characters of a String to upper case
 *
 * Examples:
 * given: "Hello"
 *  expect: "HELLO"
 * given: "3445"
 *  expect: "3445"
 *
 * @param string The String that should be converted to upper case
```

```
 * @return A version of the given String with all upper case letters
 */
String UpperCase(String string);
```

LowerCase

```
/**
 * Converts all applicable characters of a String to lower case
 *
 * Examples:
 * given: "Hello"
 *  expect: "hello"
 * given: "3445"
 *  expect: "3445"
 *
 * @param string The String that should be converted to lower case
 * @return A version of the given String with all lower case letters
 */
String LowerCase(String string);
```

GetCharAt

```
/**
 * Returns the character at a given index in a String
 *
 * Examples:
 * given: "Hello", 2
 *  expect: 'l'
 * given: "World", 0
 *  expect: 'W'
 *
 * @param string The String in which to look for a character
 * @param index The index (>=0, < Length(string)) of the character
 * @return The character at the given index of the given String
 */
char GetCharAt(String string, int index);
```

The Testing Package

This package provides basic testing functionality.

Each of your tests should be in its own function, named `test[FunctionName][testName]`. These test functions should use the assertion

functions `testEqual`, `testNotEqual`, `testTrue`, and `testFalse` in this package to assert things about the results of the test. For example:

```
/** tests some simple examples of sums */
void testSumOfNumbersSimple() {
    testEqual(10, sumOfNumbers(3,7));
    testEqual(-5, sumOfNumbers(-10,5));
}
```

You then run these testing functions from your main functions, using the `runAsTest` function from this package as a wrapper, as in

```
void test() {
    runAsTest(this::testSumOfNumbersSimple);
    ...
}
```

To run your tests, you can either call the test function from `main`, or you can run

```
java --enable-preview comp1110.testing.Test yourfilename.java
```

This will just run the `test` function.

To import this package, use the following import statements:

```
import static comp1110.testing.Comp1110Unit.*;
```

Testing

Functions

runAsTest ; testEqual ; testNotEqual ; testTrue ; testFalse

runAsTest

```
/**
 * Runs a given function as a test. The test succeeds if all
 * assertions within the function succeed, and no errors occur.
 *
 * @param testFun a void function with no arguments that runs a test
 */
void runAsTest(Runnable testFun);
```

testEqual

```
/**
 * Tests whether two given objects are equal
 * (i.e. either both are null or both are not
 * null and Equals(o1,o2)==true). Prints given
 * message if test fails.
 *
 * Examples:
 * given: "Hello", "World"
 *  expect: Test fails, default message displayed
 * given: 5, 5
 *  expect: Test succeeds
 *
 * @param o1 An object to compare. By convention,
 *           this is the expected value.
 * @param o2 An object to compare. By convention,
 *           this is the actual value.
 * @param message A message to display if the test fails.
 *           You can use $0; and $1; to include the
 *           values of o1 and o2, respectively.
 *        OPTIONAL, default = "$0; and $1; are not equal!"
 */
void testEqual(Object o1, Object o2, String message);
```

testNotEqual

```
/**
 * Tests whether two given objects are not equal
 * (i.e. either one is null and the other is not,
 * or both are not null and Equals(o1,o2)==false).
 * Prints given message if test fails.
 *
 * Examples:
 * given: "Hello", "World"
 *  expect: Test succeeds
 * given: 5, 5
 *  expect: Test fails, default message displayed
 *
 * @param o1 An object to compare.
 * @param o2 An object to compare.
 * @param message A message to display if the test fails.
 *           You can use $0; and $1; to include the
 *           values of o1 and o2, respectively.
 *        OPTIONAL, default = "$0; and $1; are equal!"
```

```
 */
void testNotEqual(Object o1, Object o2, String message);
```

testTrue

```
/**
 * Tests whether a given boolean is true.
 * Prints given message if boolean is false.
 *
 * Examples:
 * given: true
 *  expect: Test succeeds
 * given: false
 *  expect: Test fails, displays error message
 * given: 5 == 2
 *  expect: Test fails, displays error message
 *
 * @param b The boolean value that should be true.
 * @param message A message to display if the test fails.
 *        OPTIONAL, default = "Expected true, but got false!"
 */
void testTrue(boolean b, String message);
```

testFalse

```
/**
 * Tests whether a given boolean is false.
 * Prints given message if boolean is true.
 *
 * Examples:
 * given: false
 *  expect: Test succeeds
 * given: true
 *  expect: Test fails, displays error message
 * given: 5 == 2
 *  expect: Test succeeds
 *
 * @param b The boolean value that should be false.
 * @param message A message to display if the test fails.
 *        OPTIONAL, default = "Expected false, but got true!"
 */
void testFalse(boolean b, String message);
```

The Universe Package

To import this package, use the following import statements:

```
import comp1110.universe.*;
import static comp1110.universe.Colour.*;
import static comp1110.universe.Image.*;
import static comp1110.universe.Universe.*;
```

Images

Opaque Data Types

Colour ; Image

Colour

```
/**
 * This type represents colours.
 * There are a number of colour constants you can use:
 * RED, GREEN, BLUE, WHITE, BLACK, CYAN, YELLOW, MAGENTA,
 * GRAY, DARK_GRAY, LIGHT_GRAY, ORANGE, PINK
 * You can also use the MakeColour() function.
 */
// This is an opaque type whose definition does not matter to you
```

Image

```
/**
 * This type represents various images.
 * Each image has a width and height, and a "pin"
 * position, which is used by default to position
 * the image relative to others. Pins are initialized
 * to be in the middle of a shape (rounded down).
 * These values can be retrieved using the functions:
 * GetImageWidth, GetImageHeight, GetImagePinX, and
 * GetImagePinY, respectively.
 */
// This is an opaque type whose definition does not matter to you
```

Enumerations

FontStyle ; Mode ; TextAlign

FontStyle

```
/**
 * Represents a font style for text rendering
 */
enum FontStyle {
    /** Normal text style
     */
    PLAIN,
    /** Italic text
     */
    ITALIC,
    /** Bold text
     */
    BOLD,
    /** Bold italic text
     */
    ITALICBOLD
}
// TEMPLATE:
// {
//     return ... switch(fontstyle) {
//         case PLAIN -> ...;
//         case ITALIC -> ...;
//         case BOLD -> ...;
//         case ITALICBOLD -> ...;
//     } ...;
// }
```

Mode

```
/**
 * Represents a mode of drawing a geometrical shape
 */
enum Mode {
    /** Indicates that a geometrical shape should be filled
     */
    SOLID,
    /** Indicates that only the outline of the shape should be drawn
     */
    OUTLINE
}
// TEMPLATE:
// {
//     return ... switch(mode) {
```

```
//         case SOLID -> ...;
//         case OUTLINE -> ...;
//     } ...;
// }
```

TextAlign

```
/**
 * Describes a desired text alignment
 */
enum TextAlign {
    /** LEFT alignment for left-to-right languages
     *  RIGHT alighnment for right-to-left languages
     */
    DEFAULT,
    /** Align text on the left-hand side of the text field
     */
    LEFT,
    /** Align text on the right-hand side of the text field
     */
    RIGHT,
    /** Align text on the center of the text field
     */
    CENTER,
    /** Justify the text within the given maxWidth parameter
     */
    JUSTIFY
}
// TEMPLATE:
// {
//     return ... switch(textalign) {
//         case DEFAULT -> ...;
//         case LEFT -> ...;
//         case RIGHT -> ...;
//         case CENTER -> ...;
//         case JUSTIFY -> ...;
//     } ...;
// }
```

Functions

GetImageWidth ; GetImageHeight ; GetImagePinX ; GetImagePinY ; Intersects ; MovePinBy ; MovePinTo ; Rectangle ; Circle ; Ellipse ; Polygon ; Text ; DrawLine ; FromFile ; SavePNG ;

SaveJPEG ; Overlay ; OverlayXY ; Place ; PlaceXY ; Rotate ; Scale ; ScaleX ; ScaleY ; Mask ; ShowImage

GetImageWidth

```
/**
 * Returns the width of an image, in pixels.
 *
 * Examples:
 * given: Circle(5, RED)
 *  expect: 10
 * given: Rectangle(25, 50, BLUE)
 *  expect: 25
 *
 * @param image the image whose width should be returned
 * @return The width of the given image, in pixels
 * @implSpec
 *   Postcondition: Image widths are always nonnegative
 */
int GetImageWidth(Image image);
```

GetImageHeight

```
/**
 * Returns the height of an image, in pixels.
 *
 * Examples:
 * given: Circle(5, RED)
 *  expect: 10
 * given: Rectangle(25, 50, BLUE)
 *  expect: 50
 *
 * @param image the image whose height should be returned
 * @return The height of the given image, in pixels
 * @implSpec
 *   Postcondition: Image heights are always nonnegative
 */
int GetImageHeight(Image image);
```

GetImagePinX

```
/**
 * Returns the x position of an image's pin,
 * relative to its top left corner
```

```
 *
 * Examples:
 * given: Circle(5, RED)
 *  expect: 5
 * given: Rectangle(25, 50, BLUE)
 *  expect: 12
 *
 * @param image the image whose pin's x-coordinate should be returned
 * @return The x-coordinate of the given image's pin
 */
int GetImagePinX(Image image);
```

GetImagePinY

```
/**
 * Returns the y position of an image's pin,
 * relative to its top left corner
 *
 * Examples:
 * given: Circle(5, RED)
 *  expect: 5
 * given: Rectangle(25, 50, BLUE)
 *  expect: 25
 *
 * @param image the image whose pin's y-coordinate should be returne
 * @return The y-coordinate of the given image's pin
 */
int GetImagePinY(Image image);
```

Intersects

```
/**
 * Tests whether two images placed at certain coordinates
 * would intersect. This is reasonably precise for basic shapes,
 * but for images and text a rectangular bounding box is assumed.
 *
 * Examples:
 * given: Circle(5, RED), 10, 10, Circle(5, BLUE), 18, 18
 *  expect: false
 * given: Rectangle(10, 10, RED), 10, 10,
 *        Rectangle(10, 10, BLUE), 18, 18
 *  expect: true
```

```
 *
 * @param image1 The first image
 * @param x1 the x-coordinate of the first image (at its pin)
 * @param x1 the y-coordinate of the first image (at its pin)
 * @param image2 The second image
 * @param x2 the x-coordinate of the second image (at its pin)
 * @param x2 the y-coordinate of the second image (at its pin)
 * @return True if the two images would intersect if placed
 *         as given, false if not
 */
boolean Intersects(Image image1, int x1, int x1, Image image2, int x2
```

MovePinBy

```
/**
 * Adjusts the position of an image's pin
 *
 * Examples:
 * given: An image whose pin is at (10,-2), x=5, y=3
 *  expect: An image like the given one with a pin at (15, 1)
 *
 * @param image The image whose pin should be moved
 * @param x The amount by which the x-coordinate should move
 * @param y The amount by which the y-coordinate should move
 * @return A new image that is like the given one, except that
 *         the position of the pin was adjusted
 */
Image MovePinBy(Image image, int x, int y);
```

MovePinTo

```
/**
 * Adjusts the position of an image's pin
 *
 * Examples:
 * given: An image whose pin is at (10,-2), x=5, y=3
 *  expect: An image like the given one with a pin at (5, 3)
 *
 * @param image The image whose pin should be moved
 * @param x The new x-coordinate of the pin
 * @param y The new y-coordinate of the pin
 * @return A new image that is like the given one, except that
```

```
     *         the position of the pin was adjusted
     */
    Image MovePinTo(Image image, int x, int y);
```

Rectangle

```
    /**
     * Creates an image of a rectangle
     *
     * Examples:
     * given: 10, 10, RED
     *  expect: An image of a 10x10 square filled red
     * given: 20, 25, BLUE, OUTLINE
     *  expect: An image of a 20x25 rectangular blue outline
     *
     * @param width The width of the rectangle, in pixels
     * @param height The height of the rectangle, in pixels
     * @param colour The colour of the rectangle
     * @param mode The fill mode of the rectangle
     *        OPTIONAL, default = SOLID
     * @return An image of a rectangle based on the given parameters
     */
    Image Rectangle(int width, int height, Colour colour, Mode mode);
```

Circle

```
    /**
     * Creates an image of a circle
     *
     * Examples:
     * given: 10, RED
     *  expect: An image of a circle of radius 10 filled red
     * given: 20, BLUE, OUTLINE
     *  expect: An image of a circular blue outline with radius 20
     *
     * @param radius The radius of the circle, in pixels
     * @param colour The colour of the circle
     * @param mode The fill mode of the circle
     *        OPTIONAL, default = SOLID
     * @return An image of a circle based on the given parameters
     */
    Image Circle(int radius, Colour colour, Mode mode);
```

Ellipse

```
/**
 * Creates an image of an ellipse
 *
 * Examples:
 * given: 10, 10, RED
 *  expect: An image of a circle of radius 5 filled red
 * given: 20, 25, BLUE, OUTLINE
 *  expect: An image of a elliptical 20x25 blue outline
 *
 * @param width The width of the ellipse, in pixels
 * @param height The height of the ellipse, in pixels
 * @param colour The colour of the ellipse
 * @param mode The fill mode of the ellipse
 *        OPTIONAL, default = SOLID
 * @return An image of a ellipse based on the given parameters
 */
Image Ellipse(int width, int height, Colour colour, Mode mode);
```

Polygon

```
/**
 * Creates an image of a polygon.
 * In contrast to the other image functions, which put
 * the pin at the center of the image, a polygon is
 * constructed by coordinates relative to the pin.
 *
 * Examples:
 * given: RED, Pair(5,5), Pair(5,9), Pair(8,5)
 *  expect: An image of a right triangle filled red
 * given: BLUE, OUTLINE, Pair(-3, -3), Pair(-3, 3),
 *        Pair(3, 3), Pair(3, -3)
 *  expect: An image of a 6x6 square blue outline
 *
 * @param colour The colour of the polygon
 * @param mode The fill mode of the polygon
 *        OPTIONAL, default = SOLID
 * @param points Coordinates for the corners of the polygon,
 *          as many as you like (at least three), separated
 *          by commas
 * @return An image of a polygon based on the given parameters
```

```
 */
Image Polygon(Colour colour, Mode mode, Pair<Integer, Integer>... poi
```

Text

```
/**
 * Creates an image of text. Most parameters are
 * optional, but need to be specified in order up
 * to the last parameter that is needed, except for
 * maxWidth, which can always be omitted.
 *
 * Examples:
 * given: "Hello World!", 15
 *  expect: An image of the text "Hello World!", rendered
 *          in 15pt Arial
 *
 * @param text The text that should be rendered.
 *          May include line breaks ('\n').
 * @param fontSize The size of the text, in points
 * @param maxWidth A non-negative integer if the text should include
 *          line-breaks to avoid exceeding maxWidth, or -1 if
 *          the text may get arbitrarily wide
 *        OPTIONAL, default = -1
 * @param colour The colour with which the text should be rendered
 *        OPTIONAL, default = BLACK
 * @param fontName The name of the font that should be used
 *        OPTIONAL, default = Arial
 * @param style The style of the font that should be used
 *        OPTIONAL, default = FontStyle.PLAIN
 * @param align The alignment of the text, relevant if multi-line
 *        OPTIONAL, default = TextAlign.DEFAULT
 * @return The text rendered as an Image according to the parameters
 */
Image Text(String text, int fontSize, int maxWidth, Colour colour, St
```

DrawLine

```
/**
 * Draws a line onto an existing image.
 *
 * Examples:
 * given: Rectangle(50, 50, WHITE), 50, 0, 0, 50, BLACK, 5
```

```
     *  expect: A white 50x50 rectangle with a black diagonal line
     *
     * @param base The base image on which the line should be drawn
     * @param x1 The x-coordinate of the first end of the line
     * @param y1 The y-coordinate of the first end of the line
     * @param x1 The x-coordinate of the second end of the line
     * @param y1 The y-coordinate of the second end of the line
     * @param colour The colour of the line
     * @param thickness The thickness of the line
     *        OPTIONAL, default = 1
     * @return A new image that is like the given image, but
     *         with a line drawn on top according to parameters
     */
    Image DrawLine(Image base, int x1, int y1, int x1, int y1, Colour col
```

FromFile

```
    /**
     * Loads an image from a file.
     *
     * Examples:
     * given: car.png
     *  expect: Assuming car.png exists where the application was run,
     *          an Image that when drawn draws the contents of car.png
     *
     * @param path A path to an image file (PNG or JPEG)
     * @return An Image representing the imgage contained in the file
     */
    Image FromFile(String path);
```

SavePNG

```
    /**
     * Saves an image to a PNG file. If the given
     * file already exists, it is overwritten.
     *
     * Examples:
     * given: Rectangle(400, 400, GREEN), "box.png"
     *  expect: "box.png" in the application's working
     *          directory now contains a 400x400 green square
     *
     * @param image The Image to be saved to a file
```

```
     * @param path The path of the file
     */
    void SavePNG(Image image, String path);
```

SaveJPEG

```
    /**
     * Saves an image to a JPEG file. If the given
     * file already exists, it is overwritten.
     *
     * Examples:
     * given: Rectangle(400, 400, GREEN), "box.jpg"
     *  expect: "box.jpg" in the application's working
     *          directory now contains a 400x400 green square
     *
     * @param image The Image to be saved to a file
     * @param path The path of the file
     */
    void SaveJPEG(Image image, String path);
```

Overlay

```
    /**
     * Overlays two images on top of each other, using
     * pin positions to position the images relative to
     * each other. The resulting image is sized to
     * have space for both images as they are layed out.
     *
     * Examples:
     * given: Rectangle(10, 500, GREEN), Rectangle(500, 10, GREEN)
     *  expect: A green cross in a 500x500 image
     *
     * @param base The base image. Parts of it may be covered by the
     *          other image.
     * @param top The top image. Parts of it may cover the other
     *          image.
     * @return A combination of both images.
     */
    Image Overlay(Image base, Image top);
```

OverlayXY

```
/**
 * Overlays two images on top of each other, using
 * the pin position of the top image and x/y coordinates
 * relative to the bottom image's top left corner to
 * position the two relative to each other.  The
 * resulting image is sized to have space for both
 * images as they are layed out.
 *
 * Examples:
 * given: Rectangle(10, 500, GREEN), Rectangle(500, 10, GREEN), 250,
 *  expect: A sideways green T in a 500x500 image
 *
 * @param base The base image. Parts of it may be covered by the
 *          other image.
 * @param top The top image. Parts of it may cover the other
 *          image.
 * @param x The x-coordinate from the top-left corner of base
 *          at which to place the pin of the top image
 * @param y The y-coordinate from the top-left corner of base
 *          at which to place the pin of the top image
 * @return A combination of both images.
 */
Image OverlayXY(Image base, Image top, int x, int y);
```

Place

```
/**
 * Places an image on top of another, using
 * pin positions to position the images relative to
 * each other. The resulting image has the same size
 * as the base image, with parts of the placed image
 * potentially cut off.
 *
 * Examples:
 * given: Rectangle(10, 500, GREEN), Rectangle(500, 10, RED)
 *  expect: A green 10x500 rectangle, with a 10x10 red
 *          square in the middle.
 *
 * @param base The base image. Parts of it may be covered by the
 *          other image.
 * @param top The top image. Parts of it may cover the other
 *          image, and parts of it may be cut off.
```

```
 * @return A combination of both images.
 */
Image Place(Image base, Image top);
```

PlaceXY

```
/**
 * Places an image on top of another, using the pin
 * position of the top image and x/y coordinates
 * relative to the bottom image's top left corner to
 * position the two relative to each other. The
 * resulting image has the same size as the base
 * image, with parts of the placed image
 * potentially cut off.
 *
 * Examples:
 * given: Rectangle(10, 500, GREEN), Rectangle(500, 10, RED), 5, 495
 *  expect: A green 10x500 rectangle, with a 10x10 red
 *          square at the bottom.
 *
 * @param base The base image. Parts of it may be covered by the
 *          other image.
 * @param top The top image. Parts of it may cover the other
 *          image, and parts of it may be cut off.
 * @param x The x-coordinate from the top-left corner of base
 *          at which to place the pin of the top image
 * @param y The y-coordinate from the top-left corner of base
 *          at which to place the pin of the top image
 * @return A combination of both images.
 */
Image PlaceXY(Image base, Image top, int x, int y);
```

Rotate

```
/**
 * Returns a rotated version of an image.
 * Rotation is measured in radians.
 * A full circle = 2*PI.
 *
 * Examples:
 * given: Rectangle(50,50, RED), PI/4
 *  expect: An image of a red diamond
 *
```

```
 * @param image The image to be rotated.
 * @param radians The radians by which the image should be rotated
 * @return An image like the given one, rotated by given radians
 */
Image Rotate(Image image, double radians);
```

Scale

```
/**
 * Returns a scaled version of an image.
 *
 * Examples:
 * given: Rectangle(50,50, RED), 2.0
 *  expect: An image of a 100x100 red rectangle
 *
 * @param image The image to be scaled.
 * @param scale The scaling factor
 * @return An image like the given one, scaled by the given factor
 */
Image Scale(Image image, double scale);
```

ScaleX

```
/**
 * Returns a version of an image that is scaled along the x-axis.
 *
 * Examples:
 * given: Rectangle(50,50, RED), 2.0
 *  expect: An image of a 100x50 red rectangle
 *
 * @param image The image to be scaled.
 * @param scale The x-scaling factor
 * @return An image like the given one, scaled on
 *         the x-axis by the given factor
 */
Image ScaleX(Image image, double scale);
```

ScaleY

```
/**
 * Returns a version of an image that is scaled along the y-axis.
 *
 * Examples:
```

```
 * given: Rectangle(50,50, RED), 2.0
 *  expect: An image of a 50x100 red rectangle
 *
 * @param image The image to be scaled.
 * @param scale The y-scaling factor
 * @return An image like the given one, scaled on
 *         the y-axis by the given factor
 */
Image ScaleY(Image image, double scale);
```

Mask

```
/**
 * Masks a given image to the shape of another one.
 * This may cut off parts by the image and affects
 * its intersection behavior.
 *
 * Examples:
 * given: Rectangle(50, 50, RED), Circle(27, BLUE)
 *  expect: A red rectangle with some part of its edges cut off.
 *
 * @param image The image to be masked
 * @param mask The image to use as the mask
 * @return A masked version of the given image
 */
Image Mask(Image image, Image mask);
```

ShowImage

```
/**
 * Shows an image in a new window.
 *
 * Examples:
 * given: Rectangle(50, 50, GRAY)
 *  expect: A window filled with a 50x50 gray rectangle
 *
 * @param image The image to be shown
 * @param title A title for the window
 *        OPTIONAL, default = "Image"
 */
void ShowImage(Image image, String title);
```

World

Enumerations

KeyEventKind ; MouseEventKind

KeyEventKind

```
/**
 * Represents a kind of keyboard event
 */
enum KeyEventKind {
    /** A key has been pressed down, but not yet released
     */
    KEY_PRESSED,
    /** A key that had been pressed down has been released
     */
    KEY_RELEASED,
    /** For keys that result in text, this represents the direct
     *  result of having entered that text. That is, the
     *  corresponding KeyEvent will distinguish "A" from "a",
     *  depending on whether the SHIFT key has also been pressed.
     */
    KEY_TYPED
}
// TEMPLATE:
// {
//     return ... switch(keyeventkind) {
//         case KEY_PRESSED -> ...;
//         case KEY_RELEASED -> ...;
//         case KEY_TYPED -> ...;
//     } ...;
// }
```

MouseEventKind

```
/**
 * Represents a kind of mouse event
 */
enum MouseEventKind {
    /** The left mouse button has been clicked (down+up)
     */
    LEFT_CLICK,
    /** The right mouse button has been clicked (down+up)
     */
    RIGHT_CLICK,
```

```
        /** The middle mouse button has been clicked (down+up)
         */
        MIDDLE_CLICK,
        /** The left mouse button has been pressed down
         */
        LEFT_BUTTON_DOWN,
        /** The right mouse button has been pressed down
         */
        RIGHT_BUTTON_DOWN,
        /** The middle mouse button has been pressed down
         */
        MIDDLE_BUTTON_DOWN,
        /** The left mouse button has been released
         */
        LEFT_BUTTON_UP,
        /** The right mouse button has been released
         */
        RIGHT_BUTTON_UP,
        /** The middle mouse button has been released
         */
        MIDDLE_BUTTON_UP,
        /** The mouse has been moved
         */
        MOUSE_MOVE,
        /** The mouse has entered the window
         */
        MOUSE_ENTER,
        /** The mouse has left the window
         */
        MOUSE_LEAVE
    }
    // TEMPLATE:
    // {
    //     return ... switch(mouseeventkind) {
    //         case LEFT_CLICK -> ...;
    //         case RIGHT_CLICK -> ...;
    //         case MIDDLE_CLICK -> ...;
    //         case LEFT_BUTTON_DOWN -> ...;
    //         case RIGHT_BUTTON_DOWN -> ...;
    //         case MIDDLE_BUTTON_DOWN -> ...;
    //         case LEFT_BUTTON_UP -> ...;
    //         case RIGHT_BUTTON_UP -> ...;
    //         case MIDDLE_BUTTON_UP -> ...;
```

```
//         case MOUSE_MOVE -> ...;
//         case MOUSE_ENTER -> ...;
//         case MOUSE_LEAVE -> ...;
//     } ...;
// }
```

Records

KeyEvent ; MouseEvent

KeyEvent

```
/**
 * A KeyEvent records a concrete keyboard event,
 * e.g. a key press or release of a specific key
 * Examples:
 * - KeyEvent(KEY_TYPED, "A")
 * - KeyEvent(KEY_TYPED, "a")
 * - KeyEvent(KEY_PRESSED, "Space")
 * - KeyEvent(KEY_RELEASED, "F10")
 * @param kind The kind of event. See the KeyEventKind enumeration.
 * @param key The key that was pressed. Depending on the event kind,
 *          this may be the name of a key, e.g. "Space" (for
 *          press/release events), or a string representation of
 *          what was typed.
 */
record KeyEvent(
    KeyEventKind kind,
    String key) {}
// TEMPLATE:
// { ... keyevent.kind() ... keyevent.key() ... }
```

MouseEvent

```
/**
 * A MouseEvent records a concrete mouse event,
 * e.g. a button click/press/release or mouse move
 * Examples:
 * - MouseEvent(LEFT_CLICK, 15, 237)
 * - MouseEvent(RIGHT_BUTTON_DOWN, 211, 853)
 * - MouseEvent(MOUSE_MOVE, 500, 12)
 * - MouseEvent(MOUSE_LEAVE, 0, 63)
 * @param kind The kind of event. See the MouseEventKind enumeration.
 * @param x The x-coordinate of the mouse at the time of the event.
```

```
 * @param y The y-coordinate of the mouse at the time of the event.
 */
record MouseEvent(
    MouseEventKind kind,
    int x,
    int y) {}
// TEMPLATE:
// { ... mouseevent.kind() ... mouseevent.x() ... mouseevent.y() ...
```

Functions

BigBang

BigBang

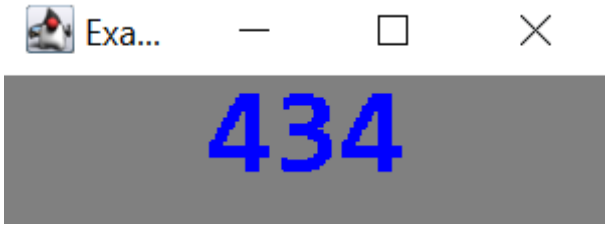
```
/**
 * Examples: see below
 *
 * @param name the name of the window
 * @param startState the initial value for the world state
 * @param drawFunction a function that represents the current world s
 * @param stepFunction a function that returns a new world state afte
 * @param keyFunction a function that returns a new world state after
 * @param mouseFunction a function that returns a new world state af
 * @param endPredicate returns whether the world state indicates tha
 */
<State> void BigBang(String name, State startState, Function<State, I
```

BigBang is used to start a World program, which is an interactive gui program that draws images on the screen based on world states, which themselves can evolve via clock ticks, keyboard events, and mouse events. The clock tick frequency is 30 clock ticks/second. This function is modeled after the Universe teachpack in HtDP (that is, if you are reading the book, this library works in very similar ways to what is described there).

The arguments `keyFunction`, `mouseFunction`, and `endPredicate` are optional - you do not have to specify them, but when you do, you need to keep them in the same order. `stepFunction` is also optional, but needs to be present if either `keyFunction` or `mouseFunction` are present.

The world state can be any data type you like, but it needs to be the same data type throughout the program. You need to write functions that provide the necessary functionality for each argument that you are using - at the very minimum, a drawing function, but in most cases at least also a stepping function.

Below is an example program that simply uses an integer as a state. Each clock tick increases the integer by one, and the program draws the integer as blue text on a gray background as shown in the following image:




```
import comp1110.universe.*;
import comp1110.lib.*;
import static comp1110.lib.Functions.*;
import static comp1110.universe.Colour.*;
import static comp1110.universe.Image.*;
import static comp1110.universe.Universe.*;

/**
 * Draws the world state as blue text on gray background
 * Examples:
 * given: 0 expect: an image of a blue "0" centered on a gray backgro
 * given: 15 expect: an image of a blue "15" centered on a gray backg
 * Strategy: Combining Simpler Functions
 * @param state the current world state (a counter value)
 * @return an image representing the world state (the counter value
 */
Image draw(int state) {
    Image text = Text(ToString(state), 40, -1, BLUE, "Consolas", Font
    return Place(Rectangle(200, 50, GRAY), text);
}

/**
 * Increases the counter representing the world state by one.
 * Examples:
 * given: 0 expect: 1
 * given: 15 expect: 16
 * Strategy: Simple Expression
 * @param state the world state at the start of the tick
 * @return the world state after the tick
 */
int step(int state) {
    return state+1;
}
```

```
void main() {
    BigBang("Example", 0, this::draw, this::step);
}

void test() {
}
```

Version History

Version	Date	Comments
2025S1-7	10/03/2025 09:00	Released definition of Maps and associated functions Used for U2. P2 may use older versions of the libraries, but can upgrade
2025S1-6	03/03/2025 08:00	Released definition of Cons-lists and associated functions Used for P2. U1 Part 4 may use older versions of the libraries, except for higher-level list functions - in that case, please upgrade to this version
2025S1-5	27/02/2025 20:30	Fixed a bug in the implementation of OverlayXY If you are not using this function, you do not need to update
2025S1-4	26/02/2025 21:30	Fixed a bug in the implementation of EndsWith If you are not using this function, you do not need to update
2025S1-3	24/02/2025 17:00	Addressed a mismatch in key names between MacOS, Linux, and Windows This is an important update, please replace any older versions of the library
2025S1-2	23/02/2025 21:00	Added key missing testing functions, in particular runAsTest This is an important update, please replace any older versions of the library

Acknowledgement of Country

The Australian National University acknowledges, celebrates and pays our respects to the Ngunnawal and Ngambri people of the Canberra region and to all First Nations Australians on whose traditional lands we meet and work, and whose cultures are among the oldest continuing cultures in human history.