# OOP - exercises

## Task 1

### Point2D class

Implement the `Point2D` class. It should contain:

- two float fields : `x`, `y`
- non-arguments constructor which will set `x`, `y` fields to `0`
- two-arguments constructor: `float x`, `float y`
- `getter` methods which will be responsible for returning `x`, `y` fields values
- `getXY` method which will return `x`, `y` values as two-element array
- `setter` methods which will be responsible for setting `x`, `y` fields values
- `setXY` method which will be responsible for settting `x` and `y`
- `toString` method which should return string in the following format: `(x, y)`

### Point3D class

Using the `Point2D` class implement the `Point3D` class. It should extend the `Point2D` class. It should contain:

- private float field: `z`
- three-arguments constructor: `float x`, `float y`, `float z`
- `getter` method which will be responsible for returning the `z` field value
- `getXYZ` method which will return `x`, `y`, `z` values as three-element array
- `setter` method which will be responsible for setting the `z` field value
- `setXYZ` method which will be responsible for setting `x`, `y`, `z`

- `toString` method which should return string in the following format: `(x, y, z)`

Please provide an example usage of above implementation.

# Task 2

## Person class

Implement the `Person` abstract class. It should contain:

- two String fields: `name`, `address`
- non-arguments constructor which will set `name`, `address` fields as empty strings
- two-arguments constructor: `String name`, `String address`
- `getter` methods which will be responsible for returning `name`, `address` fields values
- `setter` methods which will be responsible for setting `name`, `address` fields values
- `toString` method which should return string in the following format: `?->?`, where `?` is the name and adress value accordingly

## Student class

Implement the `Student` class. It should extend the `Person` class.
Implementation should meet the below criteria:

- three fields: type of study, year of study, study price
- three-arguments constructor: type of study, year of study, study price
- `getter` methods which will be responsible for returning declared fields
- `setter` methods which will be responsible for setting declared fields
- `toString` method which should return details information about a student

## Staff class

Implement the `Lecturer` class. It should extend the `Person` class.
Implementation should meet the below criteria:

- two fields: specialization, salary
- two-arguments constructor: specialization, salary
- `getter` methods which will be responsible for returning declared fields
- `setter` methods which will be responsible for setting declared fields
- `toString` method which should return detials information about a lecturer

Please provide an example usage of above implementation.

# Task 3

## Shape class

Implement the `Shape` class. It should contain:

- color information
- information if color should fill a shape
- non-arguments constructor which will set: the `color` field to `unknown` and the `isFilled` field to `false`
- two-arguments constructor: `color`, `isFilled`
- `getter` methods which will be responsible for returning declared fields
- `setter` methods which will be responsible for setting declared fields
- `toString` method which should return the following information: `Shape with color of ? and filled/NotFilled`, where ? is a placeholder for the color and the `filled` / `not filled` info should be returned depending on the `isFilled` field

## Circle class

Implement the `Circle` class. It should extend the `Shape` class. Implementation should meet the below criteria:

- information about the radius value
- non-arguments constructor which will set: the `color` field value to `unknown`, the `isFilled` field value to `false` and the `radius` field value to `1`
- three-arguments constructor: `color`, `isFilled`, `radius`
- `getter` methods which will be responsible for returning the `radius` value
- `setter` methods which will be responsible for setting the `radius` value
- `getArea` method which will be responsible for calculating the surface area
- `getPerimeter` method which will be responsible for calculating the circuit
- `toString` method which should return the following information: `Circle with radius=? which is a subclass off y`, where ? is a placeholder for the radius value and the `y` info should be a result of the `toString` execution from the base class

## Rectangle class

Implmenent the `Rectangle` class. It should extend the `Shape` class. Implementation should meet the below criteria:

- information about the width and length value as a `double` type
- non-arguments constructor which will set: the `color` field value to `unknown`, the `isFilled` field value to `false` and the `width` and `length` field value to `1`
- four-arguments constructor: `color`, `isFilled`, `width` i `length`
- `getter` methods which will be responsible for returning the `width` and `length` value
- `setter` methods which will be responsible for setting the `width` and `length` value
- `getArea` method which will be responsible for calculating the surface area
- `getPerimeter` method which will be responsible for calculating the circuit
- `toString` method which should return the following information: `Rectangle with width=? and length=? which is a subclass off y`, where ? is a placeholder for the width and length value accordingly and the `y` info should be a result of the `toString` execution from the base class

### Square class

Implement the `Square` class which will extend the `Rectangle` class. It should not add any new field or feature, but it should force square behaviour on the base class methods.

Please provide an example usage of above implementation.

## Task 4

Modify implementation prepared as a scope of task 2. Refactor the following functionality:

- make the `Shape` class abstract
- change the `Shape` class fields access modifiers to `protected`
- add two abstract methods: `getArea` and `getPerimeter`

Each class which is extending the `Shape` class directly/indirectly should override abstract methods from base class.

Please provide an example usage of above implementation.

## Task 5

Implement the `Line` class which will contain (as a composition) the instance of two `Point2D` object from task 1. Those points should be the start and end point of the line. In addition, this class should implement:

- two-arguments constructor: begin point, end point
- four-arguments constructor accepting 4 parameters: coordinates of the start and end point
- `getter` method which will be responsibe for returning start and end points
- `setter` method which will be responsible for setting start and end points
- the method responsible for calculating the length of the line based on the set points

- the method responsible for returning the coordinates of the point being the center of the created line

Please provide an example usage of above implementation.

# Task 6

Implement the `Movable` interface which will contain a definition of common methods for `MovablePoint` and `MovableCircle` classes.

- `void moveUp()`
- `void moveDown()`
- `void moveLeft()`
- `void moveRigth()`

## MovablePoint class

The `MovablePoint` class should implement the `Movable` interface and should contain 4 fields: `int`: `x`, `y`, `xSpeed`, `ySpeed`. The `x`, `y` fields should define the coordinates of point and the `xSpeed`, `ySpeed` fields should determine how much the appropriate coordinates should change.

- the `moveUp()` and the `moveDown()` method should increase/decrease the `y` coordinate by provided `ySpeed` value
- the `moveLeft()` and the `moveRight()` method should increase/decrease the `x` coordinate by provided `xSpeed` value

## MovableCircle class

The `MovableCircle` class should implement the `Movable` interface and should contain (as a composition) instance of `MovablePoint` class. It should contain fields necessery to define circle radius.

- the `moveUp()` and the `moveDown()` method should increase/decrease the `y` coordinate from `MovablePoint` instance by provided `ySpeed` value

- the `moveLeft` and the `moveRight()` method should increase/decrease the `x` coordinate from `MovablePoint` instance by provided `xSpeed` value

Please provide an example usage of above implementation. Please consdier to present polymorphism as well.

# Task 7

Implement the `GeometricObject` interface which should contain common behaviours definition for each sub class:

- `double getPerimeter()`
- `double getArea()`

## Circle class

The `Circle` class should implement the `GeometricObject` interface and it should contain the `radius` field. Methods from `GeometricObject` interface should be implemented according to the mathematic rules.

## Resizable interface

The `Resizable` interface should declare the `resize(int percent)` method which will be responsible for rescaling objectes which are implementing created interface.

## ResizableCircle class

The `ResizableCircle` class should implement `Resizable` interace. The `resize` method from interface should reduce the radius of the circle in percentage.

Please provide an example usage of above implementation.

# Exceptions - exercises

## Task 1

Create the `divide` method which has to divide the two numbers that are the attributes of the method. In case the second parameter of the method is 0, a non-default exception should be thrown: `CannotDivideBy0Exception`.

## Task 2

### BookRepository class

Create the `BookRepository` class, which will be responsible for:

- adding `Book` objects
- removing `Book` objects
- searching for objects of the `Book` type with the indicated name
- searching for objects of the `Book` type with the indicated id
- removing objects of the `Book` type based on the provided `id`

### Book class

The `Book` class should include the following fields:

- id
- title
- author
- year of release

### NoBookFoundException

In case of lack of searched results an exception should be thrown. This exception should accept the `String` parameter object with information about which elements could not be found.

# Classes and interfaces - exercises

## Task 1

Create the `UserValidator` class which with the `validateEmails` method will be responsible for validating user data such as: email, alternative email. Within the scope of the `validateEmails` method, please create the local `Email` class which will be responsible for formatting the provided email. Validation should cover the following scenarios:

- if the given email address is empty or it is null, set the value to `unknown`
- if the given email address does not meet the email criteria, set the value to `unknown` (use regular expressions)

## Task 2

### Movie class

Create the `Movie` class which will cover fields: title, director, year of release, genre, distributor. This class should contain a default constructor and `getter` and `setter` methods. Please consider creating `toString` method which will be responsible for returning info about a specific object.

### MovieCreator class

Create the static nested `MovieCreateor` class. It should contain:

- class fields similar to `Movie` class
- methods which will be responsible for setting movie values. Each method should return an object instance of the object for which the method is being called
- the `createMovie` method will create the instance of the `Movie` class. It will return it as a method results

# Task 3

## Car class

Create the `Car` class which will store information about a car make and type. It should contain `getter` and `setter` methods.

## Engine class

Implement the `Engine` class which will be a nested non-static class under the `Car` class. This class should contain the field: engine type and `setEngine` method which will set a type based on the car type. If the car type is `economy`, then the engine type should be set to `diesel`. If the car type is `luxury`, then the engine type should be defined as `electric`. Otherwise, the engine type should be defined as `petrol`.

# Task 4

## Validator interface

Create the `Validator` interface, which will include the `boolean validate(T input)` method.

## User class

Create the `User` class which will include: * fields: name, last name, age, login, password * default constructor * `setter` and `getter` methods * `setter` methods should accept as method params: value for the field and the `Validator` interface instance * `setter` methods should execute the `validate` method based on the instance of the transferred object. The parameter passed to the `validate` method should be the value of the argument

## Anonymous class

Instances of the `Validator` class should be implemented as anonymous class. Implementation should meet the below criteria:

- name validation: the name cannot be empty or null, it should start with a capital letter
- last name validation: the last name cannot be empty or null, it should start with a capital letter
- age validation: the value should be between 0 and 150
- login validation: the field value should contain 10 characters
- password validation: it should contain the `!` character

Please present the solution described above on an example.

# Enumerated types - exercises

## Task 1

Crete an enum `Weekday` with constants `MONDAY`, `TUESDAY`, ... `SUNDAY`. The enum should contain `boolean isWeekDay` and `boolean isHoliday` methods. The `isHoliday` method should return the opposite result to the call of the `isWeekDay` method. Please implement the `whichIsGreater` method as a scope of the enum class. This method should accept an object of `Weekday` type. This method should display information that the indicated day of the week is the predecessor or successor of the day of the week passed as the method argument. Please consider using the `compareTo` method.

Present the solution described above with an example.

## Task 2

Create the `PackageSize` enum class with constants `SMALL`, `MEDIUM`, `LARGE`. Enum should include two parameters in the constructor:

- minimum package size in `cm`
- maximum package size in `cm`

The `PackageSize` enum class should adopt the static `getPackageSize` method. This method should accept package size and as a result it should return a specific `PackageSize` object based on the package size passed.

## Task 3

Create the `TemperatureConvert` enum class with constants `C_F`, `C_K`, `K_C`, `F_C`, `F_K`, `K_F`. Enum should adopt a constructor that includes three parameters:

- input temperature unit
- output temperature unit
- the `Converter` interface instance (with a `float convert(float tempIn)` method) - the instance should define the necessary calculations for temperature conversion

The `TemperatureConvert` enum class should adopt the static `convertTemperature` method which will include the following params:

- input temperature unit
- output temperature unit
- value of the temperature

This method should return the converted value. To find the right constant from set of enum values, use the `values()` method.

# Collections - exercises

## Task 1

Implement the `SDAArrayList<T>` class which will implements `ArrayList<T>` logic. For this purpose please implement following methods:

- add
- remove
- get
- display

## Task 2

### Author class

Implement the `Author` class which will contains fields: name, last name, gender. Please consider all available methods and constructor parameters. Please prepare `hashCode` and `equals` implementation.

### Book class

Implement the `Book` class which will contains: title, price, year of release, author lists, genre (represented as enum class) fields. Please consider all necessery methods and constructor parameters. Please prepare `hashCode` and `equals` implementation.

### BookService class

Implement the `BookService` class which will include book lists and it needs to cover following methods:

- adding books to the list

- removing books from the list
- returning a list of all books
- returning a list of books by `Fantasy` type
- returning a list of books released before `1999`
- returning the most expensive book
- returning the cheapest book
- returning a book writted by `3 authors`
- returning a list of books sorted by parameter: `ascending/descending`
- veryfing if a book is on the list
- returning a list of books written by provided author

## Task 3

Based on `100` element array with randomly selected elements from the range `0-50` please implement following features:

- returning a list of unique elements
- returning a list of elements that have been repeated in the generated array at least once

## Task 4

Based on `Task 2` please implement a method which will be responsible for returing unique key-value pairs. The key should be represented as a book genre, value need to contain a title.

## Task 5

Based on `Task 2`, implement a method that will be responsible for creating a stack of books sorted from highest to lowest price.

# Functional programming - exercises

## Task 1

Using the functional programming mechanisms based on the given structure, please provide:

- a list of all episodes
- a list of all videos
- a list of all season names
- a list of all season numbers
- a list of all episode names
- a list of all episode numbers
- a list of all video names
- a list of all url addresses for each video
- only episodes from even seasons
- only videos from even seasons
- only videos from even episodes and seasons
- only Clip videos from even episodes and odd seasons
- only Preview videos from odd episodes and even seasons

```
enum VideoType {
    CLIP, PREVIEW, EPISODE
}

class Video {
    public String title;
    public String url;
    public VideoType videoType;

    public Video(String title, String url, VideoType videoType) {
        this.title = title;
        this.url = url;
```

```
        this.videoType = videoType;
    }
}

class Episode {
    public String episodeName;
    public int episodeNumber;
    List<Video> videos;

    public Episode(String episodeName, int episodeNumber,
List<Video> videos) {
        this.episodeName = episodeName;
        this.episodeNumber = episodeNumber;
        this.videos = videos;
    }
}

class Season {
    public String seasonName;
    public int seasonNumber;
    List<Episode> episodes;

    public Season(String seasonName, int seasonNumber,
List<Episode> episodes) {
        this.seasonName = seasonName;
        this.seasonNumber = seasonNumber;
        this.episodes = episodes;
    }
}
```

# Generic types - exercises

## Task 1

Create the `Pair` class which, based on generic types, will allow to store any pair of objects.

## Task 2

Design the `countIf` generic method wich, based on an array of elements of any type will count the number of elements meeting the condition using an functional interface. Any interface implemented anonymously can be a function.

## Task 3

Design the generic `swap` method, which will be responsible for swapping the position of the selected elements of the array.

## Task 4

Create a class that will behave like a library for the following types of media:

- books
- newspapers
- movies

Please provide a solution for generic types. For data collection, use any array or Collection API class.

## Task 5

Create a class that will behave like a pet house for the following animals:

- cat
- dog

Please provide a solution for generic types. For data collection, use any array or Collection API class.

# Java IO - exercises

## Task 1

Create a solution which will display all files/directories included in the provided directory.

## Task 2

Prepare a solution which will read and display a file line by line.

## Task 3

Prepare a solution which will add a string to the specified file.

## Task 4

Prepare a solution which will return the longest word from the provided file.

## Task 5

Create a CSV parser:

```
John,Smith,23
Sam,Johnson,40
Andrew,Manly,43
```

With the file above the program should return the three-element list of objects of the User type with fields: name, surname, age.

## Task 6

Create a program which will provide following features based on the `Movie` class objects:

- adding objects
- returning object lists

The `Movie` class should contains fields: `title`, `genre`, `director`, `year of release`. Adding objects should send their serialized form to a file. Displaing object list should enable deserialization of the text file to convert individual lines to `Movie` objects.

# Parallel and concurrent programming - exercises

## Task 1

Write a program that in parallel will find even numbers in two intervals: `1000-2000` and `14300-17800`.

## Task 2

Write a program that will solve the problem below.

On the road between the towns `A` and `B` there is a bridge on which there can be only one car. Implement a mechanism that will allow synchronized access by a car objects to object of the `Bridge` class.

The `Car` class should contain the following data:

- car name
- car type

The `Bridge` class can contain the following method:

- driveThrough, which will accept as parameter the object of the `Car` class. The journey should take 5s.

## Task 3

Write a program which will execute two independent sorting algorithms on two separate threads. The main goal of the implementation is to return information about the algorithm that has completed faster.

## Task 4

Write a program which will synchronize access to a bank account. If any cyclical Internet service wants to charge the account with a higher amount than currently available, then the thread should be suspended. When additional money will be transfered to the account, the thread should be raised.

## Task 5

Write a data structure that will allow you to navigate the array in two directions:

- forward ( `next()` )
- backwards ( `prev()` )

The data structure should store the currently searched index. Please take care of its additional synchronization.

# Reflection API basics - exercises

## Task 1

Implement the `Student` class with the following details:

- the class should contain fields: first name, last name, no. index, field of study
- the class should include a non-arguments constructor and a constructor that accepts the value for each field as an argument
- `setter` methods
- `getter` methods

Display the following information using the reflection mechanism:

- available methods
- available fields
- available constructors

## Task 2

Using the `Student` class for the task no. 1 please implement following operations using the reflection mechanism:

- object creation with passing all required parameters
- calling methods of type `getter`
- direct modification of private field values