PROJEKT PIERWSZY

KLASA POINT:

Zdefiniuj klasę Point posiadającą dwa publiczne, ostateczne pola x, y. Napisz konstruktor ustawiający te wartości.

```
import java.lang.Math;
class Point {
 public final double x; Publiczne ostateczne pola
 public final double y;
 public Point(double x, double y) { alt+insert ->tworzenie getterów, konstruktorów
 this.x = x;
 this.y = y;
 }
```

Statyczna funkcja

```
 public static Point middlePoint(Point p1, Point p2){
 return new Point(
 (p1.x + p2.x)/2,
 (p1.y + p2.y)/2
 );
 }
}
```

KLASA SEGMENT

Zdefiniuj klasę Segment reprezentującą odcinek, posiadającą dwa prywatne punkty klasy Point. Wygeneruj akcesory i mutatory klasy Segment.

Napisz publiczną metodę, która zwraca długość odcinka. W kolejnym kroku usuń mutatory i utwórz konstruktor ustawiający te pola na wartości swoich dwóch parametrów.

Zdefiniuj w klasie Segment publiczną metodę toSvg(), która zwróci napis zawierający kod języka znacznikowego SVG pozwalający wyświetlić tę linię.

```
import java.util.Locale;
class Segment {
 private Point startPoint;
 private Point endPoint;
```

```
 public Segment(Point startPoint, Point endPoint) {
 this.startPoint = startPoint;
 this.endPoint = endPoint;
 }
 public Point getStartPoint() {
 return startPoint;
 }
 public void setStartPoint(Point startPoint) {
 this.startPoint = startPoint;
 }
 public Point getEndPoint() {
 return endPoint;
 }
 public void setEndPoint(Point endPoint) {
 this.endPoint = endPoint;
 }
 public double length() {
 double deltaX = endPoint.x - startPoint.x;
 double deltaY = endPoint.y - startPoint.y;
 return Math.sqrt(Math.pow(deltaX, 2) + Math.pow(deltaY, 2));
 }
```

Format pliku svg, Locale.English (aby były . a nie , pomiędzy liczbami)
alt+enter ->dodajemy biblioteke

```
 public String toSvg() {
 String svgRep = "<svg height=\"240\" width=\"500\"
xmlns=\"http://www.w3.org/2000/svg\">\n";
 svgRep+=String.format(Locale.ENGLISH,"<line x1=\"%f\" y1=\"%f\" x2=\"%f\"
y2=\"%f\" style=\"stroke:red;fill:black;stroke-width:5\"/>\n</svg>",
startPoint.x, endPoint.x, startPoint.y, endPoint.y);
 return svgRep;
 }
```

Napisz funkcję (metodę klasy głównej), która przyjmie: obiekt *segment* klasy Segment oraz obiekt *point* klasy Point. Funkcja powinna zwrócić odcinek prostopadły do *segment*, rozpoczynający się w punkcie *point* o długości równej odcinkowi *segment*. Następnie zmodyfikuj tę metodę tak, aby zwracała tablicę dwóch możliwych do konstrukcji linii oraz przenieś tę metodę jako statyczną do klasy Segment. Szczególne przypadki należy zignorować.

```
POWALONA FUNKCJA NIE POLECAM
 public static Segment [] perpendicularSegment(Segment segment, Point point)
{
 double a;
 a = (segment.startPoint.y - segment.endPoint.y) / (segment.startPoint.x -
segment.endPoint.x);
 double b;
 a=-1/a;
 b=point.y-a*point.x;
 double x0 = point.x;
 double y0 = point.y;
 double r = segment.length();
 double root =
Math.sqrt(-y0*y0+(2*a*x0+2*b)*y0-a*a*x0*x0-2*a*b*x0+(a*a+1)*r*r-b*b);
 double x1 = -(root-a*y0-x0+a*b)/(a*a+1);
 double y1 = -(a*root-a*a*y0-a*x0-b)/(a*a+1);
```

```
 double x2 = (root+a*y0+x0-a*b)/(a*a+1);
 double y2 = (a*root+a*a*y0+a*x0+b)/(a*a+1);
 return new Segment[]{new Segment(point, new Point(x2, y2)), new
Segment(point, new Point(x1, y1))};
 }
 //Przeciąż metodę klasy Line zwracającą prostopadły odcinek tak,
 //aby przyjmowała jako dodatkowy argument długość zwracanego odcinka.
 public static Segment [] perpendicularSegment(Segment segment, Point point,
double lengthSeg) {
 double a;
 a = (segment.startPoint.y - segment.endPoint.y) / (segment.startPoint.x -
segment.endPoint.x);
 double b;
 a=-1/a;
 b=point.y-a*point.x;
 double x0 = point.x;
 double y0 = point.y;
 double root =
Math.sqrt(-y0*y0+(2*a*x0+2*b)*y0-a*a*x0*x0-2*a*b*x0+(a*a+1)*lengthSeg*lengthS
eg-b*b);
 double x1 = -(root-a*y0-x0+a*b)/(a*a+1);
 double y1 = -(a*root-a*a*y0-a*x0-b)/(a*a+1);
 double x2 = (root+a*y0+x0-a*b)/(a*a+1);
 double y2 = (a*root+a*a*y0+a*x0+b)/(a*a+1);
 return new Segment[]{new Segment(point, new Point(x2, y2)), new
Segment(point, new Point(x1, y1))};
 }
}
```

KLASA POLYGON

```
import java.util.Arrays;
import java.util.Locale;
```

Zdefiniuj klasę Polygon posiadającą prywatną tablicę punktów. Konstruktor tej klasy powinien przyjmować tablicę punktów. Napisz publiczną metodę toSvg() działającą analogicznie jak w poprzednim zadaniu.

(zadanie na drugich labach)
Zmodyfikuj klasę Polygon dodając do jej konstruktora argument Style i modyfikując jej metodę toSvg(), aby uwzględniała styl. Dopuść możliwość pominięcia stylu przy konstrukcji. Wówczas należy narysować przezroczysty (fillColor) wielokąt, z czarnym obrysem (strokeColor) o grubości jednego piksela (strokeWidth).

```
public class Polygon extends Shape { (WAŻNE, DZIEDZICZENIE PRZYKŁAD!!)
 private Point[] points;
// private Style style;
 public Point[] getPoints() {
 return points;
 }
 public void setPoints(Point[] points) {
 this.points = points;
 }
```

```
public Polygon(){}
public Polygon(Point[] points) {
this.points = points;
}
public Polygon(Point[] points, Style style) {
this.points = points;
this.style = style;
}
```

W klasie Polygon napisz konstruktor kopiujący, wykonujący głęboką kopię obiektu. **(WAŻNE, GŁĘBOKA KOPIA!!)**

```
public Polygon(Polygon other) {
this.points = new Point[other.points.length];
for (int i = 0; i < other.points.length; i++) {
this.points[i] = new Point(other.points[i].x, other.points[i].y);
}
}
public String toSvg() {
// String pol = "<svg viewBox=\"0 0 200 100\" xmlns=\"http://www.w3.org/2000/svg\">\n";
String pol = "";
pol += "<polygon points=\"";
for (Point p : points) {
pol += String.format(Locale.ENGLISH, "%f,%f ", p.x, p.y); //alt+enter - importowanie biblioteki
}
pol += "\"";
if (style == null) {
pol += " style=\"fill:none;stroke:green;stroke-width:1\"";
} else {
pol += ' ';
pol += style.toSvg();
}
pol += "/>";
// pol += "\n</svg>";
return pol;
}
```

Napisz publiczną, statyczną metodę wytwórczą klasy Polygon o nazwie square. Funkcja powinna przyjąć jako argumenty: obiekt Line, obiekt Style i zwrócić wielokąt będący kwadratem, którego przekątną jest dany odcinek.

```
public static Polygon square(Segment line, Style style) {
Point points[] = new Point[4];
points[0] = line.getStartPoint();
points[1] = new Point(line.getEndPoint().x, line.getStartPoint().y);
points[2] = line.getEndPoint();
points[3] = new Point(line.getStartPoint().x, line.getEndPoint().y);
Polygon polygon = new Polygon(points);
return polygon;
}
}
```

Zdefiniuj klasę Style o finalnych, publicznych polach klasy String: fillColor, strokeColor oraz Double: strokeWidth. Napisz trójargumentowy konstruktor ustawiający te wartości.

KLASA STYLE

```
public class Style {
 private final String fillColor;
 private final String strokeColor;
 private final double strokeWidth;


 public Style(String fillColor, String strokeColor, double strokeWidth) {
 this.fillColor = fillColor;
 this.strokeColor = strokeColor;
 this.strokeWidth = strokeWidth;
 }
```

Napisz publiczną metodę toSvg() zwracającą napis, będący opcją style, którą można umieścić np. w tagu <polygon>.

```
 public String toSvg(){
 String styleFor=String.format(Locale.ENGLISH,"style=\"stroke:%s;fill:%s;stroke-width:%f\"",
strokeColor, fillColor, strokeWidth);
 return styleFor;
 }
 public String getFillColor() {
 return fillColor;
 }
 public String getStrokeColor() {
 return strokeColor;
 }
 public double getStrokeWidth() {
 return strokeWidth;
 }
}
```

KLASA SVGSCENE

```
import java.io.BufferedWriter;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.util.ArrayList;
import java.util.List;
```

Napisz klasę SvgScene posiadającą prywatną listę obiektów Polygon. Napisz metodę, która przyjmuje obiekt klasy Polygon oraz dodaje go do listy w obiekcie SvgScene. Napisz funkcję save(String), która utworzy plik HTML w ścieżce danej argumentem i zapisze do niego reprezentacje wszystkich wielokątów znajdujących się na kanwie.

Zmodyfikuj klasę SvgScene, aby posiadała tablicę obiektów klasy Shape i korzystając z polimorfizmu zapisz w niej obiekty typu Polygon i Ellipse.

```
public class SvgScene {
// private List<Polygon> polygons = new ArrayList<>(); //gdyby byl konstruktor, to new
ArrayList<>() tam by byl
```

```
 private List<Shape> shapes=new ArrayList<>(); //polimorfizm ->po prostu zamiast Polygon to wszedzie dac Shape(bo juz tak mozemy)

 public void addShape(Shape shape){
 shapes.add(shape);
 }

 //ZAPISYWANIE DO PLIKU WAŻNE!!! ZAPISYWANIE DO PLIKU HTML.
public void saveToFile(String filePath){
 try {
 BufferedWriter writer = new BufferedWriter(new FileWriter(filePath));
 writer.write(toSvg());
 writer.close();
 System.out.println("Plik " + filePath + " został poprawnie zapisany!");
 } catch (IOException e) {
 System.err.println("Wystąpił błąd podczas zapisywania do pliku: " +
e.getMessage());
 }
}
```

KLASA SHAPE

Utwórz klasę abstrakcyjną Shape, która otrzyma jako pole, obiekt klasy Style. Uczyń pole tego obiektu chronionym. Utwórz publiczny konstruktor, który ustawia to pole. Napisz abstrakcyjną metodę toSvg(). Zmodyfikuj klasę Polygon, aby dziedziczyła po klasie Shape.

```
public abstract class Shape {
 protected Style style; //protected - chronione; private - tylko tej klasy, public - wszystkie pakiety, protected - w ramach danego pakietu jest widoczny
 public Shape(){};
 public Shape(Style style) {
 this.style = style;
 }


//Zmodyfikuj klasę Polygon, aby dziedziczyła po klasie Shape.
 public abstract String toSvg();
}

import java.util.Locale;


public class Ellipse extends Shape{
 private Point middle;
 private double radius1;
 private double radius2;
 private Style style;


 public Ellipse(Point middle, double radius1, double radius2, Style style){
 this.radius1=radius1;
 this.radius2=radius2;
 this.middle=middle;
 this.style=style;
 }
 //W jej implementacji metody toSvg() powinno znaleźć się rysowanie z użyciem tagu <ellipse>.
 //Zmodyfikuj klasę SvgScene, aby posiadała tablicę obiektów klasy Shape i korzystając z
```

```
polimorfizmu zapisz w niej obiekty typu Polygon i Ellipse.
 @Override
 public String toSvg() {
 String code= String.format(Locale.ENGLISH, "<ellipse cx=\"%f\" cy=\"%f\" rx=\"%f\" ry=\"%f\"",
middle.x, middle.y, radius1, radius2);
 if (style == null) {
 code += " style=\"fill:none;stroke:green;stroke-width:1\"";
 } else {
 code += ' ';
 code += style.toSvg();
 }
 code += "/>";
 return code;
 }
}
```

KLASA ELIPSE

Napisz klasę Ellipse dziedziczącą po Shape, posiadającą prywatne pola: środek elipsy (Point), długości promieni i styl.
W jej implementacji metody toSvg() powinno znaleźć się rysowanie z użyciem tagu <ellipse>.

```
import java.util.Locale;
public class Ellipse extends Shape{
 private Point middle;
 private double radiusX;
 private double radiusY;
 private Style style;
 public Ellipse(Point middle, double radiusX, double radiusY, Style style){
 this.radiusX=radius1;
 this.radiusY=radius2;
 this.middle=middle;
 this.style=style;
 }
 //W jej implementacji metody toSvg() powinno znaleźć się rysowanie z użyciem tagu <ellipse>.
 //Zmodyfikuj klasę SvgScene, aby posiadała tablicę obiektów klasy Shape i korzystając z
polimorfizmu zapisz w niej obiekty typu Polygon i Ellipse.
 @Override
 public String toSvg() {
 String code= String.format(Locale.ENGLISH, "<ellipse cx=\"%f\" cy=\"%f\" rx=\"%f\" ry=\"%f\"",
middle.x, middle.y, radiusX, radiusY);
 if (style == null) {
 code += " style=\"fill:none;stroke:green;stroke-width:1\"";
 } else {
 code += ' ';
 code += style.toSvg();
 }
 code += "/>";
 return code;
 }
}
```

Co jest w pliku svg?

```
<svg viewBox="0 0 200 1000" xmlns="http://www.w3.org/2000/svg">
 <polygon points="100.000000,20.000000 180.000000,60.000000 180.000000,140.000000 "
style="stroke:green;fill:red;stroke-width:6.000000"/>
</svg>
```

Co jest w pliku HTML?

```
<!DOCTYPE html>
<html>
<body>
<svg viewBox="0 0 2000 1000" xmlns="http://www.w3.org/2000/svg">
<polygon points="100.000000,20.000000 180.000000,60.000000
180.000000,140.000000 100.000000,180.000000 20.000000,140.000000
20.000000,60.000000 "
style="stroke:red;fill:#ffff00;stroke-width:20.000000"/><polygon
points="300.000000,97.000000 12.000000,109.000000
48.000000,222.000000 "
style="stroke:blue;fill:green;stroke-width:20.000000"/><polygon
points="300.000000,300.000000 300.000000,200.000000
200.000000,200.000000 200.000000,300.000000 "
style="stroke:gray;fill:black;stroke-width:20.000000"/><ellipse
cx="100.000000" cy="20.000000" rx="40.000000" ry="300.000000"
style="fill:none;stroke:green;stroke-width:1"/></svg>
</body>
</html>
```

LABY 3Zadanie 0.

Z klasy Shape usuń wszystkie pola i uczyń ją interfejsem. Przemianuj klasę Point na Vec2.

```
public interface Shape {
 String toSvg();
}
```

```
Z boku, gdzie mamy src; Klikamy Prawym przyciskiem na Vec2, Refactor, Rename
```

Zadanie 1

Napisz klasę SolidFilledPolygon dziedziczącą po Polygon. Klasa powinna posiadać prywatne pole String color ustawiane, obok tablicy punktów, w konstruktorze. Przemodeluj funkcję toSvg w interfejsie Shape tak, aby możliwe było przekazanie jej parametru typu String, który zostanie umieszczony w tagu rysowanego obiektu. Wykorzystaj poniższy kod:
"<polygon points=\"%s\" %s />"

W klasie SolidFilledPolygon zdefiniuj metodę toSvg, która nadpisze metodę klasy nadrzędnej. Wewnątrz tej metody wywołaj metodę toSvg klasy nadrzędnej, przekazując jej jako parametr napis powstały ze sformatowania:
"fill=\"%s\" %s "
kolejno kolorem i parametrem napisowym.

1.Napisz klasę SolidFilledPolygon dziedziczącą po Polygon. Klasa powinna posiadać prywatne pole String color ustawiane, obok tablicy punktów, w konstruktorze.

```
import java.util.ArrayList;
public class SolidFilledPolygon extends Polygon{
 private String color;
 public SolidFilledPolygon(String color, ArrayList<Vec2> points, Style style){
 super(points,style);
 this.color=color;
 }
```

```
Pamiętaj! Musisz stworzyć pusty konstruktor w Polygon, aby klasa dziedziczona
umożliwiła nam utworzenie konstruktora w klasie dziedziczącej,
z dodatkowym argumentem
```

```
W klasie Polygon:

public Polygon(){};
```

2. Przemodeluj funkcję toSvg w interfejsie Shape tak, aby możliwe było przekazanie jej parametru typu String, który zostanie umieszczony w tagu rysowanego obiektu. Wykorzystaj poniższy kod:
"<polygon points=\"%s\" %s />"

```
public interface Shape {
 String toSvg(String string);
}
```

2. W klasie SolidFilledPolygon zdefiniuj metodę toSvg, która nadpisze metodę klasy nadrzędnej. Wewnątrz tej metody wywołaj metodę toSvg klasy nadrzędnej, przekazując jej jako parametr napis powstały ze sformatowania:
"fill=\"%s\" %s "
kolejno kolorem i parametrem napisowym.

```
        public      String       toSvg(String string1){
 String code=String.format("fill=\"%s\"       %s ",color,string1);
        //wywolanie metody z klasy nadrzednej
 return      super.toSvg(code);
        }
}
```

```
Pamiętaj!
Trzeba zmienić także w klasie Polygon metodę toSvg ->aby argumentem był String

W klasie Polygon:
```

```
public String toSvg(String string) {
 String code = "\n";
 code += "\t";
 code += "<polygon points=\"";
 for (Vec2 p : points){
 code += p.getX() + "," + p.getY() + " ";
 }
 code += "\"";
 code += getStyle().toSvg();
 if (string !=null){
 code+="/>";
 }
 else{
 code+=' ' + string;
 code+="/>";
 }
 code += "\n";
 return code;
}
```

Trzeba dodać warunek, gdy napis jest pusty

Zastanów się, jakie konsekwencje dla struktury programu miałoby stworzenie analogicznej klasy dziedziczącej po klasie Ellipse oraz próba dodawania innych parametrów do tagu.

Zadanie 2.
Zdefiniuj klasę ShapeDecorator implementującą interfejs Shape, która posiadać będzie chronione pole Shape decoratedShape. Pole to powinno być ustawiane w konstruktorze. Nadpisz metodę toSvg w taki sposób, by zawierała wywołanie tej samej metody na rzecz obiektu decoratedShape.

Po klasie ShapeDecorator podziedzicz nową klasę SolidFillShapeDecorator. Klasa ta powinna posiadać prywatne pole String color. W konstruktorze ma przyjmować obiekt klasy Shape oraz kolor wypełnienia typu String. W jej metodzie toSvg wywołaj metodę toSvg na rzecz decoratedShape, z parametrami jak w zadaniu 1.

Utwórz dwa obiekty klasy SolidFillShapeDecorator tak, aby parametrem jednego był obiekt wielokąta, a drugiego elipsy.

Zadanie 3.
Utwórz klasę StrokeShapeDecorator posiadającą prywatne pola String color i double width, które powinny być ustawione w konstruktorze. Wywołaj metodę toSvg podobnie jak w zadaniu 2. formatując napis
"stroke=\"%s\" stroke-width=\"%f\" "
kolorem i grubością obrysu. Przetestuj udekorowanie tą klasą obiektów będących wynikiem poprzedniego zadania.

Zadanie 4.
Utwórz klasę TransformationDecorator odpowiadającą za wpisanie w wyświetlany tab informacji o przekształceniach afinicznych: translacji, rotacji i skalowaniu. Na potrzeby każdego z tych działań stwórz prywatne pola:

- boolean translate, Vec2 translateVector,
- boolean rotate, double rotateAngle, Vec2 rotateCenter,
- boolean scale, Vec2 scaleVector.

Wewnątrz klasy TransformationDecorator zdefiniuj publiczną klasę Builder. Zdefiniuj w niej prywatne pola, jednakowe z polami w klasie zewnętrznej oraz pole Shape shape. Wartości logiczne powinny być fałszywe. Napisz po jednej metodzie ustawiającej parametry transformacji i zmieniającej wartość logiczną na prawdziwą na znak, że transformacja ma się wykonać. Funkcje powinny zwracać obiekt klasy Builder z wprowadzonymi zmianami. Napisz w klasie Builder metodę build, która utworzy obiekt TransformationDecorator, przekazując mu jako parametr obiekt shape, a następnie ustawi wszystkim polom w tym obiekcie wartości zapisane w obiekcie Buildera i zwróci tak stworzony obiekt.

W klasie TransformationDecorator nadpisz metodę toSvg tak, aby poskładać w niej napis definiujący transformację z elementów:
"translate(%f %f) ", translateVector.x, translateVector.y
"rotate(%f %f %f) ", rotateAngle, rotateCenter.x, rotateCenter.y
"scale(%f %f) ", scaleVector.x, scaleVector.y

Umieść je w we własności “transform”:
"transform=\"%s\" %s", result, parameters.

Przetestuj tworzenie klasy TransformationDecorator za pomocą całości lub części dostępnych transformacji.

Uzyskanie możliwości zastosowania filtra oraz wypełnienia obiektu gradientem wymaga zapisania stosownego kodu w tagu <defs>, którego zawartość nie będzie wprost renderowana. Lokalne obiekty w SVG mogą być identyfikowane za pomocą unikalnych nazw (id), a odwoływać można się do nich za pomocą składni “url(#id)”.

Zadanie 5a.
W klasie SvgScene utwórz prywatne, statyczne pole SvgScene instance, początkowo równe null. Napisz akcesor do tego pola. Jeżeli znajduje się tam null, należy je zainicjalizować.

Zadanie 5b.
Dodaj do klasy SvgScene tablicę String defs[] oraz metodę dodającą elementy do tej tablicy, wzorując się na tablicy shapes i metodzie addShape. W metodzie saveHtml uwzględnij dopisanie tagów <defs> do wynikowego pliku.

Zdefiniuj klasę DropShadowDecorator dziedziczącą po ShapeDecorator. Jej zadaniem jest udekorowanie obiektu Shape rzucanym cieniem. Jest to realizowane przez umieszczenie w tagu <defs> sformatowanego kodu:

\t<filter id=\"f%d\" x=\"-100%%\" y=\"-100%%\" width=\"300%%\" height=\"300%%\">\n" +
"\t\t<feOffset result=\"offOut\" in=\"SourceAlpha\" dx=\"5\" dy=\"5\" />\n" +
"\t\t<feGaussianBlur result=\"blurOut\" in=\"offOut\" stdDeviation=\"5\" />\n" +
"\t\t<feBlend in=\"SourceGraphic\" in2=\"blurOut\" mode=\"normal\" />\n" +
"\t</filter>", index

oraz w metodzie toSvg:

"filter=\"url(#f%d)\" ", index

gdzie w obu przypadkach index jest liczbą całkowitą, unikalną dla tego filtra. Unikalność indeksu zagwarantuj przy użyciu prywatnego, statycznego pola klasy.

Zadanie 6.
Łącząc wiedzę wyniesioną z zadania 4 i 5 zdefiniuj klasę GradientFillShapeDecorator dziedziczącą po ShapeDecorator, której celem jest wypełnienie kształtu poziomym, barwnym gradientem.

Gradient wymaga umieszczenia w tagu <defs> napisu rozpoczynającego się od:
"\t<linearGradient id=\"g%d\" >\n", index
a następnie dla każdego koloru i jego położenia:
\t\t<stop offset=\"%f\" style=\"stop-color:%s\" />\n", stop.offset, stop.color,
gdzie stop.offset jest liczbą zmiennoprzecinkową z przedziału 0-1, a stop.color napisem. Definicję gradientu zamyka:
"\t</linearGradient>"

Wewnątrz klasy zdefiniuj klasę Builder. W klasie Builder stwórz metodę, która przyjmuje offset i kolor, a której wielokrotne wywołania pozwalają stworzyć tablicę tych wartości definiującą przebieg gradientu.

W metodzie toSvg klasy zewnętrznej wykorzystaj sformatowany napis:
"fill=\"url(#g%d)\" ", index

Rozważ, jakie modyfikacje należałoby poczynić w programie wynikowym, aby możliwa była rezygnacja z singletonowej postaci klasy SvgScene.

PROJEKT 2

Skorzystaj z udostępnionego pliku zawierającego opisy przykładowych osób tworzących ze sobą związki.

W kolejnych kolumnach pliku opisane są:

1. imię i nazwisko,

2. data narodzin,

3. data śmierci (pusta jeśli żyje),

4. imię rodzica (puste jeśli nieznany),

5. imię rodzica (puste jeśli nieznany).

Zadanie 1.

Napisz klasę Person, w której znajdować będą się dane odpowiadające wierszowi pliku. Na tym etapie pomiń wczytywanie rodziców. Napisz metodę wytwórczą fromCsvLine() klasy Person przyjmującą jako argument linię opisanego pliku.

Metoda wytwórcza to metoda, która służy do tworzenia i zwracania instancji obiektów w ramach danej klasy lub interfejsu.

```
import java.time.LocalDate;

import java.time.format.DateTimeFormatter;


public class Person {
```

```
    private String name;  // Imię osoby

    private LocalDate birthDate, deathDate;  // Daty urodzenia i śmierci osoby


    public Person(String name, LocalDate birthDate, LocalDate deathDate) {

        this.name = name;

        this.birthDate = birthDate;

        this.deathDate = deathDate;

    }


    public static Person fromCsvLine(String line){

        String[] parts = line.split("," ,-1);   // Metoda split dzieli linię na części, używając przecinka jako separatora

        String name = parts[0].trim();  // Pobiera imię osoby


        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd.MM.yyyy"); // Tworzy obiekt DateTimeFormatter z określonym wzorcem daty


        String birthPart = parts[1];  // Pobiera datę urodzenia osoby

        String deathPart = parts[2];  // Pobiera datę śmierci osoby


        LocalDate death = null;

        LocalDate birth = null;


        if(!deathPart.isEmpty()){

            death = LocalDate.parse(deathPart, formatter);  // Przekształca datę śmierci na obiekt LocalDate

        }

        birth = LocalDate.parse(birthPart, formatter);  // Przekształca datę urodzenia na obiekt LocalDate


        return new Person(name, birth, death);    // Tworzy nowy obiekt Person i go zwraca

    }
```

Zadanie 2.

Napisz metodę fromCsv(), która przyjmie ścieżkę do pliku i zwróci listę obiektów typu Person.

```
public static List<Person> fromCsv(String filePath) throws
IOException {

   BufferedReader bufferedReader = new BufferedReader(new
FileReader(filePath)); // Tworzy BufferedReader do czytania z pliku

   List<Person> people = new ArrayList<>();  // Tworzy listę osób

   List<PersonWithParentsNames> parentsNames = new ArrayList<>();
// Tworzy listę imion rodziców



   String line;



   bufferedReader.readLine(); // Pomija pierwszą linię pliku
(nagłówek)



   // Czyta plik linia po linii

   while ((line = bufferedReader.readLine()) != null){



   }



   bufferedReader.close();  // Zamyka BufferedReader

   return people;  // Zwraca listę osób

}
```

Zadanie 3.

Napisz klasę NegativeLifespanException. Rzuć jej obiekt jako wyjątek, jeżeli data śmierci osoby jest wcześniejsza niż data urodzin. Wywołanie metody getMessage() powinno wyświetlić stosowną informację zawierającą przyczyny rzucenia wyjątku.

```
// Definiujemy klasę wyjątku NegativeLifespanException, która
rozszerza klasę Exception

public class NegativeLifespanException extends Exception{

   // Konstruktor wyjątku

   public NegativeLifespanException(Person person){

       // Wywołanie konstruktora klasy nadrzędnej z sformatowanym
komunikatem o błędzie

       super(String.format("%s, urodził(a) się %s, później niż
umarła %s", person.getName(), person.getBirthDate(),
person.getDeathDate()));

   }

}
```

dodajemy metodę do klasy Person:

```
private void validateLifespan() throws NegativeLifespanException{

   // Sprawdza, czy osoba nie żyje dłużej niż żyła

   if (deathDate != null && deathDate.isBefore(birthDate)){

       // Rzuca wyjątek, jeśli osoba zmarła przed urodzeniem

       throw new NegativeLifespanException(this);

   }

}
```

Zadanie 4.

Napisz klasę AmbiguousPersonException. Rzuć jej obiekt jako wyjątek, jeżeli w pliku pojawi się więcej niż jedna osoba o tym samym imieniu i nazwisku. Wywołanie metody getMessage() powinno wyświetlić stosowną informację zawierającą przyczyny rzucenia wyjątku.

```
// Definiujemy klasę wyjątku AmbiguousPersonException, która
rozszerza klasę RuntimeException

public class AmbiguousPersonException extends RuntimeException{

   // Konstruktor wyjątku

   public AmbiguousPersonException(String name){

       // Wywołanie konstruktora klasy nadrzędnej z sformatowanym
komunikatem o błędzie

       super(String.format("Osoba: %s pojawiła się kolejny raz w
pliku.", name));

   }

}
```

dodajemy do klasy Person

```
private void validatePerson(List<Person> people) throws
AmbiguousPersonException {

   for (Person person : people){

       // Sprawdza, czy istnieje już osoba o takim samym imieniu

       if(person.name.equals(this.name)){

           // Rzuca wyjątek, jeśli istnieje już osoba o takim samym
imieniu

           throw new AmbiguousPersonException(name);

       }

   }

}
```

Zadanie 5.

Niech rodzice będą przechowywani w klasie Person w postaci listy obiektów Person. Zmodyfikuj metodę fromCsv, by w obiektach dzieci ustawiała referencje do obiektów rodziców.

```
private List<Person> parents = new ArrayList<>();  // Lista rodziców osoby
```

tworzymy klasę PersonWithParentsNames:

```
import java.util.HashMap;

import java.util.List;

import java.util.Map;



// Klasa reprezentująca osobę wraz z imionami jej rodziców

public class PersonWithParentsNames {

   // Osoba

   public final Person person;

   // Imiona rodziców

   public final String [] parentNames;



   // Konstruktor klasy

   public PersonWithParentsNames(Person person, String[] parents) {

       this.person = person;

       this.parentNames = parents;

   }



   // Metoda tworząca obiekt klasy na podstawie linii CSV

   public static PersonWithParentsNames fromCsvLine(String line){

       // Tworzenie osoby na podstawie linii CSV

       Person person = Person.fromCsvLine(line);

       // Podział linii na części

       String[] parts = line.split(",", -1); //-1 powoduje, że puste pola na końcu linii nie będą ignorowane, będzie utworzona pusta składowa tablicy
```

```
      // Tablica z imionami rodziców

       String[] parents = new String[2];

       // Wypełnianie tablicy imionami rodziców

       for (int i = 0; i < 2; ++i) {

           if(!parts[i + 3].isEmpty()){

               parents[i] = parts[i+3];

           }

       }

       // Zwracanie nowego obiektu klasy

       return new PersonWithParentsNames(person, parents);

   }



   // Metoda łącząca osoby z ich rodzicami

   public static void linkRelatives(List<PersonWithParentsNames> people)
throws UndefinedParentException{

       // Mapa osób

       Map<String, PersonWithParentsNames> peopleMap = new HashMap<>();



       /*Mapa to struktura danych, która przechowuje pary klucz-wartość. W
tym przypadku kluczem jest String, a wartością jest obiekt klasy
PersonWithParentsNames.



       Każdy klucz w mapie jest unikalny, a do każdego klucza przypisana
jest dokładnie jedna wartość. Możemy dodać parę klucz-wartość do mapy,
możemy usunąć parę na podstawie klucza,

       możemy sprawdzić czy dany klucz istnieje w mapie, a także możemy
pobrać wartość przypisaną do danego klucza.



       HashMap to konkretna implementacja interfejsu Map w Javie.
Wykorzystuje ona tablicę do przechowywania danych i funkcję hashującą do
przypisywania kluczy do indeksów w tej tablicy.

       Dzięki temu, operacje takie jak dodawanie pary klucz-wartość,
usuwanie pary klucz-wartość, czy pobieranie wartości na podstawie klucza są
bardzo szybkie.*/
```

```
        // Wypełnianie mapy osobami
        for (PersonWithParentsNames personWithParentsNames : people){
            peopleMap.put(personWithParentsNames.person.getName(),
personWithParentsNames);
        }

        // Przypisywanie rodziców do osób
        for (PersonWithParentsNames personWithParentsNames : people){
            Person person = personWithParentsNames.person;

            // Przypisywanie rodziców do osoby
            for (int i = 0; i < 2; i++) {
                String parentName = personWithParentsNames.parentNames[i];
                if(parentName != null){
                    // Jeżeli rodzic jest w mapie, dodajemy go do osoby
                    if(peopleMap.containsKey(parentName)){
                        person.addParent(peopleMap.get(parentName).person);
                    } else {
                        // Jeżeli rodzica nie ma w mapie, rzucamy wyjątek
                        throw new UndefinedParentException(person,
parentName);
                    }
                }
            }
        }
    }
}
```

modyfikujemy metodę fromCsv:

```
public static List<Person> fromCsv(String filePath) throws
IOException, NegativeLifespanException, UndefinedParentException{

   BufferedReader bufferedReader = new BufferedReader(new
FileReader(filePath)); // Tworzy BufferedReader do czytania z pliku

   List<Person> people = new ArrayList<>();  // Tworzy listę osób

   List<PersonWithParentsNames> parentsNames = new ArrayList<>();
// Tworzy listę imion rodziców



   String line;



   bufferedReader.readLine(); // Pomija pierwszą linię pliku
(nagłówek)



   // Czyta plik linia po linii

   while ((line = bufferedReader.readLine()) != null){

       // Tworzy obiekt PersonWithParentsNames z linii CSV

       PersonWithParentsNames personWithParentsNames =
PersonWithParentsNames.fromCsvLine(line);

       // Sprawdza, czy osoba ma prawidłowy czas życia

       personWithParentsNames.person.validateLifespan();

       // Sprawdza, czy osoba jest prawidłowa

       personWithParentsNames.person.validatePerson(people);

       // Dodaje imiona rodziców do listy

       parentsNames.add(personWithParentsNames);

       // Dodaje osobę do listy osób

       people.add(personWithParentsNames.person);

   }



// Łączy krewnych

   PersonWithParentsNames.linkRelatives(parentsNames);
```

```
bufferedReader.close();  // Zamyka BufferedReader

    return people;  // Zwraca listę osób

}
```

Zadanie 6.

Napisz klasę ParentingAgeException. Rzuć jej obiekt jako wyjątek, jeżeli rodzic jest młodszy niż 15 lat lub nie żyje w chwili narodzin dziecka. Przechwyć ten wyjątek tak, aby nie zablokował dodania takiej osoby, a jedynie poprosił użytkownika o potwierdzenie lub odrzucenie takiego przypadku za pomocą wpisania znaku "Y" ze standardowego wejścia.

```
// Definiujemy klasę wyjątku ParentingAgeException, która rozszerza klasę Exception

public class ParentingAgeException extends Exception{

    // Osoba, której dotyczy wyjątek

    public final Person person;



    // Metoda pomocnicza do formatowania danych osoby

    private static String personAndLifespan(Person person){

        // Zwraca sformatowany ciąg znaków z imieniem osoby i datami urodzenia i śmierci (jeśli istnieją)

        return String.format("%s (%s%s)", person.getName(), person.getBirthDate(), person.getDeathDate() == null ? "" : " - " + person.getDeathDate());

    }

    // Konstruktor wyjątku

    public ParentingAgeException(Person person, Person parent){

        // Wywołanie konstruktora klasy nadrzędnej z sformatowanym komunikatem o błędzie

        super(String.format("Osoba %s nie może być rodzicem %s.", personAndLifespan(parent), personAndLifespan(person)));

        // Przypisanie osoby do pola klasy

        this.person = person;

    }

}
```

dodajemy do klasy Person metodę do sprawdzenia wieku rodzica:

```
private void validateParentingAge() throws ParentingAgeException {

   for(Person parent : parents){

       // Sprawdza, czy rodzic jest zbyt młody lub nie żyje w
momencie narodzin osoby

       if(birthDate.isBefore(parent.birthDate.plusYears(15)) ||
(parent.deathDate != null && birthDate.isAfter(parent.deathDate))){

           // Rzuca wyjątek, jeśli rodzic jest zbyt młody lub nie
żyje w momencie narodzin osoby

           throw new ParentingAgeException(this, parent);

       }

   }

}
```

edytujemy metodę fromCsv:

```
public static List<Person> fromCsv(String filePath) throws
IOException, NegativeLifespanException, UndefinedParentException{

   BufferedReader bufferedReader = new BufferedReader(new
FileReader(filePath)); // Tworzy BufferedReader do czytania z pliku

   List<Person> people = new ArrayList<>();  // Tworzy listę osób

   List<PersonWithParentsNames> parentsNames = new ArrayList<>();
// Tworzy listę imion rodziców



   String line;



   bufferedReader.readLine(); // Pomija pierwszą linię pliku
(nagłówek)

   // Czyta plik linia po linii

   while ((line = bufferedReader.readLine()) != null){
```

```
// Tworzy obiekt PersonWithParentsNames z linii CSV
        PersonWithParentsNames personWithParentsNames = PersonWithParentsNames.fromCsvLine(line);
        // Sprawdza, czy osoba ma prawidłowy czas życia
        personWithParentsNames.person.validateLifespan();
        // Sprawdza, czy osoba jest prawidłowa
        personWithParentsNames.person.validatePerson(people);
        // Dodaje imiona rodziców do listy
        parentsNames.add(personWithParentsNames);
        // Dodaje osobę do listy osób
        people.add(personWithParentsNames.person);
    }
    // Łączy krewnych
    PersonWithParentsNames.linkRelatives(parentsNames);

    try {
        // Sprawdza, czy wiek rodzicielstwa jest prawidłowy dla każdej osoby
        for (Person person : people) {
            person.validateParentingAge();
        }
    } catch (ParentingAgeException e){
        // W przypadku wyjątku prosi o potwierdzenie od użytkownika
        Scanner scanner = new Scanner(System.in);
        System.out.println(e.getMessage());
        System.out.println("Proszę o potwierdzenie [t/n]: ");
        String response = scanner.nextLine();
```

```
// Jeśli odpowiedź nie jest "t" ani "n", usuwa osobę z listy
        if(!response.equals("t") && !response.equals("n")){
            people.remove(e.person);
        }
    }
    bufferedReader.close();  // Zamyka BufferedReader
    return people;  // Zwraca listę osób
}
```

Zadanie 7.

W klasie Person napisz statyczne metody toBinaryFile i fromBinaryFile, które zapiszą i odczytają listę osób do i z pliku binarnego.

```
//statyczne metody toBinaryFile i fromBinaryFile, które zapiszą i
odczytają listę osób do i z pliku binarnego



public static void toBinaryFile(List<Person> personList, String
fileName) throws IOException {
    // Zapisuje listę osób do pliku binarnego
    try(
            // Tworzy strumień wyjściowy pliku
            FileOutputStream fos = new FileOutputStream(fileName);
            // Tworzy strumień wyjściowy obiektu
            ObjectOutputStream oos = new ObjectOutputStream(fos);
    ){
        // Zapisuje listę osób do pliku
        oos.writeObject(personList);
    }
}
```

```
public static List<Person> fromBinaryFile(String fileName) throws
IOException, ClassNotFoundException {

   // Odczytuje listę osób z pliku binarnego

   try(

            // Tworzy strumień wejściowy pliku

            FileInputStream fis = new FileInputStream(fileName);

            // Tworzy strumień wejściowy obiektu

            ObjectInputStream ois = new ObjectInputStream(fis);

   ){

       // Odczytuje listę osób z pliku i ją zwraca

       return (List<Person>) ois.readObject();

   }

}
```

DODATKOWO MAIN DO LAB 4:

```
import java.util.List;


public class Main {

   public static void main(String[] args) {

       // Inicjalizacja obiektu Person

       Person p = null;

       try {

           // Tworzenie obiektu Person z linii CSV

           p = Person.fromCsvLine("Anna
Dąbrowska,07.02.1930,22.12.1991,Ewa Kowalska,Marek Kowalski");

           // Wyświetlanie obiektu Person

           System.out.println(p.toString());
```

```
// Tworzenie listy obiektów Person z pliku CSV

            List<Person> personList =
Person.fromCsv("second_project\\family.csv");



            // Przechodzenie przez listę i wyświetlanie każdego
obiektu Person

            for (Person person : personList) {

                System.out.println(person);

            }

        } catch (Exception e){

            // Wyświetlanie komunikatu o błędzie, jeśli wystąpi
wyjątek

            System.err.println(e.getMessage());

        }

    }

}
```

i wynik projektu:

```
Person{name='Anna Dąbrowska', birthDate=1930-02-07,
deathDate=1991-12-22, parents = []}

Person{name='Marek Kowalski', birthDate=1899-05-15,
deathDate=1957-06-25, parents = []}

Person{name='Ewa Kowalska', birthDate=1901-11-03,
deathDate=1990-03-05, parents = []}

Person{name='Anna Dąbrowska', birthDate=1930-02-07,
deathDate=1991-12-22, parents = [Person{name='Ewa Kowalska',
birthDate=1901-11-03, deathDate=1990-03-05, parents = []},
Person{name='Marek Kowalski', birthDate=1899-05-15,
deathDate=1957-06-25, parents = []}]}

Person{name='Andrzej Kowalski', birthDate=1936-09-12,
deathDate=1990-06-25, parents = [Person{name='Ewa Kowalska',
birthDate=1901-11-03, deathDate=1990-03-05, parents = []},
Person{name='Marek Kowalski', birthDate=1899-05-15,
deathDate=1957-06-25, parents = []}]}

Person{name='Krystyna Dąbrowska', birthDate=1927-06-25,
deathDate=1940-04-08, parents = [Person{name='Ewa Kowalska',
birthDate=1901-11-03, deathDate=1990-03-05, parents = []},
```

```
Person{name='Marek Kowalski', birthDate=1899-05-15,
deathDate=1957-06-25, parents = []}]}

Person{name='Alicja Wiśniewska', birthDate=1963-10-18,
deathDate=2012-10-18, parents = [Person{name='Anna Dąbrowska',
birthDate=1930-02-07, deathDate=1991-12-22, parents =
[Person{name='Ewa Kowalska', birthDate=1901-11-03,
deathDate=1990-03-05, parents = []}, Person{name='Marek Kowalski',
birthDate=1899-05-15, deathDate=1957-06-25, parents = []}]}]}

Person{name='Tomasz Dąbrowski', birthDate=1966-01-24,
deathDate=null, parents = [Person{name='Anna Dąbrowska',
birthDate=1930-02-07, deathDate=1991-12-22, parents =
[Person{name='Ewa Kowalska', birthDate=1901-11-03,
deathDate=1990-03-05, parents = []}, Person{name='Marek Kowalski',
birthDate=1899-05-15, deathDate=1957-06-25, parents = []}]}]}

Person{name='Joanna Nowak', birthDate=1973-04-08, deathDate=null,
parents = [Person{name='Andrzej Kowalski', birthDate=1936-09-12,
deathDate=1990-06-25, parents = [Person{name='Ewa Kowalska',
birthDate=1901-11-03, deathDate=1990-03-05, parents = []},
Person{name='Marek Kowalski', birthDate=1899-05-15,
deathDate=1957-06-25, parents = []}]}]}

Person{name='Kacper Kowalski', birthDate=1970-07-15, deathDate=null,
parents = [Person{name='Andrzej Kowalski', birthDate=1936-09-12,
deathDate=1990-06-25, parents = [Person{name='Ewa Kowalska',
birthDate=1901-11-03, deathDate=1990-03-05, parents = []},
Person{name='Marek Kowalski', birthDate=1899-05-15,
deathDate=1957-06-25, parents = []}]}]}

Person{name='Elżbieta Kowalska', birthDate=1990-12-28,
deathDate=null, parents = [Person{name='Kacper Kowalski',
birthDate=1970-07-15, deathDate=null, parents =
[Person{name='Andrzej Kowalski', birthDate=1936-09-12,
deathDate=1990-06-25, parents = [Person{name='Ewa Kowalska',
birthDate=1901-11-03, deathDate=1990-03-05, parents = []},
Person{name='Marek Kowalski', birthDate=1899-05-15,
deathDate=1957-06-25, parents = []}]}]}]}

Person{name='Jan Kowalski', birthDate=1992-03-05, deathDate=null,
parents = [Person{name='Kacper Kowalski', birthDate=1970-07-15,
deathDate=null, parents = [Person{name='Andrzej Kowalski',
birthDate=1936-09-12, deathDate=1990-06-25, parents =
[Person{name='Ewa Kowalska', birthDate=1901-11-03,
deathDate=1990-03-05, parents = []}, Person{name='Marek Kowalski',
birthDate=1899-05-15, deathDate=1957-06-25, parents = []}]}]}]}
```