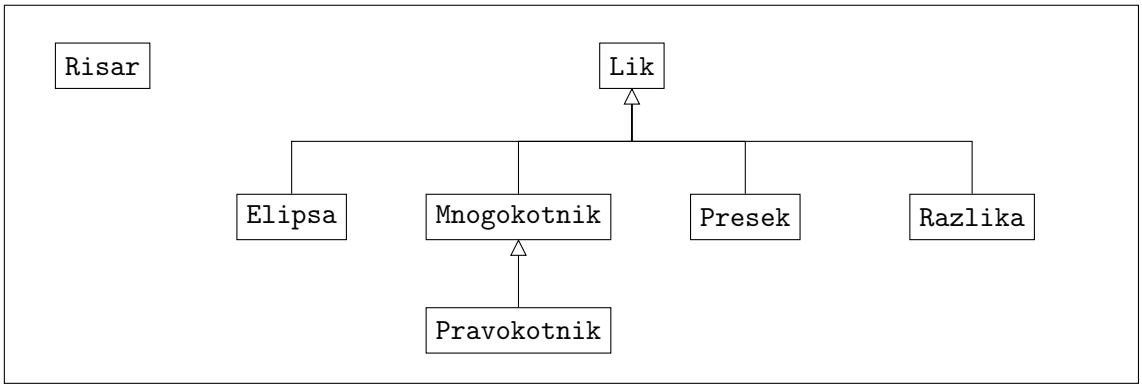
Rešitev 10. domače naloge (Risar)

Zgradba rešitve

Vsi razredi, ki jih bomo predstavili v nadaljevanju, nastopajo kot statični notranji razredi v razredu `Oddaja`. Zaradi enostavnosti bomo izpustili razred `Oddaja` in besedi `public static` v glavah posameznih razredov.

Rešitev tvori izvršilni razred `Risar` in hierarhija razredov, izpeljanih iz abstraktnega razreda `Lik`. Razreda `Elipsa` in `Mnogokotnik` sta izpeljana neposredno iz razreda `Lik`, razred `Pravokotnik` pa je podrazred razreda `Mnogokotnik`, saj je pravokotnik poseben primer mnogokotnika. Presek in razlika likov sta prav tako lika, zato sta tudi razreda `Presek` in `Razlika` podrazreda razreda `Lik`. Razredi, ki sestavljajo rešitev, so prikazani na sliki 1.



Slika 1: *Razredi, ki tvorijo rešitev, in hierarhični odnosi med njimi.*

Razred `Lik`

Lik je množica točk, množico pa lahko opredelimo z relacijo vsebovanosti. Relacijo vsebovanosti bomo predstavili z metodo `vsebuje`:

```
abstract class Lik {
    public abstract boolean vsebuje(int x, int y);
}
```

Objekt tipa `Lik` tako predstavlja lik, ki ga tvorijo vse točke (x, y), za katere metoda `vsebuje` vrne `true`. Za splošen lik metode seveda ne moremo definirati, lahko pa jo definiramo za elipso in mnogokotnik, pa tudi za presek in razliko likov.

Razred `Elipsa`

Točka (x, y) pripada elipsi s središčem (x_s, y_s), vodoravno polosjo a in navpično polosjo b natanko tedaj, ko velja

$$b^2(x - x_s)^2 + a^2(y - y_s)^2 \leqslant a^2b^2.$$

Gornjo neenačbo moramo zgolj prepisati v metodo `vsebuje`.

```
class Elipsa extends Lik {

    private int xSredisce, ySredisce, vodoravnaPolos, navpicnaPolos;

    public Elipsa(int xSredisce, int ySredisce, int vodoravnaPolos, int navpicnaPolos) {
        this.xSredisce = xSredisce;
        this.ySredisce = ySredisce;
        this.vodoravnaPolos = vodoravnaPolos;
        this.navpicnaPolos = navpicnaPolos;
    }

    @Override
    public boolean vsebuje(int x, int y) {
        int a2 = this.vodoravnaPolos * this.vodoravnaPolos;
        int b2 = this.navpicnaPolos * this.navpicnaPolos;
        int dx2 = (x - this.xSredisce) * (x - this.xSredisce);
        int dy2 = (y - this.ySredisce) * (y - this.ySredisce);

        return (b2 * dx2 + a2 * dy2 <= a2 * b2);
    }
}
```

Razred `Mnogokotnik`

Točka (x, y) pripada mnogokotniku z omejitvenimi parametri $((a_1, b_1, c_1), \ldots, (a_n, b_n, c_n))$ natanko tedaj, ko za vsak $i \in \{1, \ldots, n\}$ velja

$$a_i x + b_i y + c_i \leqslant 0.$$

Metoda `vsebuje` v zanki potuje po vrsticah tabele `omejitve` in za vsako omejitev preveri, ali je izpolnjena. Če ni, takoj vrne `false`, če se zanka izteče do konca, pa vrne `true`.

```
class Mnogokotnik extends Lik {

    private int[][] omejitve;

    public Mnogokotnik(int[][] omejitve) {
        this.omejitve = omejitve;
    }

    public boolean vsebuje(int x, int y) {
        for (int i = 0;  i < this.omejitve.length;  i++) {
            int a = this.omejitve[i][0];
            int b = this.omejitve[i][1];
            int c = this.omejitve[i][2];
            if (a * x + b * y + c > 0) {
                return false;
            }
        }
        return true;
    }
}
```

Razred `Pravokotnik`

Pravokotnik z zgornjim levim ogliščem (x_0, y_0), širino w in višino h tvorijo vse točke (x, y), za katere velja $x_0 \leqslant x \leqslant x_0 + w - 1$ in $y_0 \leqslant y \leqslant y_0 + h - 1$. Če želimo v razredu `Pravokotnik` izkoristiti »infrastrukturo«, ki nam jo ponuja njegov nadrazred `Mnogokotnik`, moramo omejitve prepisati v obliko $a_i x + b_i y + c_i \leqslant 0$:

$$\begin{aligned} -1x + 0y + x_0 &\leqslant 0 \\ 1x + 0y + (-x_0 - w + 1) &\leqslant 0 \\ 0x - 1y + y_0 &\leqslant 0 \\ 0x + 1y + (-y_0 - h + 1) &\leqslant 0 \end{aligned}$$

```
class Pravokotnik extends Mnogokotnik {

    public Pravokotnik(int xLevo, int yZgoraj, int sirina, int visina) {
        super(omejitve(xLevo, yZgoraj, sirina, visina));
    }

    private static int[][] omejitve(int xLevo, int yZgoraj, int sirina, int visina) {
        return new int[][]{
            {-1, 0, xLevo},
            {1, 0, -xLevo - sirina + 1},
            {0, -1, yZgoraj},
            {0, 1, -yZgoraj - visina + 1}
        };
    }
}
```

Razred `Presek`

Točka (x, y) je sestavni del preseka likov A in B natanko tedaj, ko pripada tako liku A kot liku B.

```
class Presek extends Lik {

    private Lik prvi;
    private Lik drugi;

    public Presek(Lik prvi, Lik drugi) {
        this.prvi = prvi;
        this.drugi = drugi;
    }

    public boolean vsebuje(int x, int y) {
        return prvi.vsebuje(x, y) && drugi.vsebuje(x, y);
    }
}
```

Razred `Razlika`

Točka (x, y) je sestavni del razlike likov A in B natanko tedaj, ko pripada liku A, obenem pa ne pripada liku B.

```
class Razlika extends Lik {

    private Lik prvi;
    private Lik drugi;

    public Razlika(Lik prvi, Lik drugi) {
        this.prvi = prvi;
        this.drugi = drugi;
    }

    public boolean vsebuje(int x, int y) {
        return prvi.vsebuje(x, y) && !drugi.vsebuje(x, y);
    }
}
```

Razred `Risar`

V razredu `Risar` vzdržujemo dvojiško matriko `slika` velikosti 100×100. Matrika služi kot platno, na katero rišemo like in njihove robove. Element matrike v vrstici y in stolpcu x ima vrednost `true` natanko tedaj, ko točka (x, y) pripada trenutni sliki.

```
class Risar {

    private static final int DX = 100;
    private static final int DY = 100;

    private boolean[][] slika;

    public Risar() {
        this.slika = new boolean[DY][DX];
    }

    public boolean[][] slika() {
        return this.slika;
    }
    ...
}
```

Lik narišemo tako, da za vsako točko koordinatne mreže preverimo, ali mu pripada. Če je odgovor pozitiven, nastavimo istoležni element matrike na `true`.

```
class Risar {
    ...
    public void narisiLik(Lik lik) {
        for (int y = 0;  y < DY;  y++) {
            for (int x = 0;  x < DX;  x++) {
                if (lik.vsebuje(x, y)) {
                    this.slika[y][x] = true;
                }
            }
        }
    }
    ...
}
```

Nekoliko več dela imamo z risanjem robov. Naj pojem *rob nivoja d* označuje množico točk, ki pripadajo robu debeline d, vendar pa ne pripadajo robu debeline $d-1$. Rob debeline d je potemtakem sestavljen iz roba nivoja 1, roba nivoja 2, ... in roba nivoja d. Rdeče točke na sliki 1 v besedilu naloge tvorijo rob nivoja 1, zelene sestavljajo rob nivoja 2, modre in vijoličaste pa rob nivoja 3 oziroma 4.

V metodi

`public void narisiRob(Lik lik, int debelina)`

bomo izdelali pomožno tabelo `rob` velikosti 100×100. Tabelo bomo napolnili tako, da bomo dosegli sledeče stanje:

- Vrednost elementa `rob[y][x]` je enaka 0 natanko tedaj, ko je točka (x, y) izven lika `lik`.
- Vrednost elementa `rob[y][x]` je enaka -1 natanko tedaj, ko točka (x, y) pripada liku `lik`, vendar pa ne pripada robu debeline `debelina`.
- Vrednost elementa `rob[y][x]` je enaka d natanko tedaj, ko točka (x, y) pripada robu nivoja d.

V metodi `narisiRob` najprej ustvarimo tabelo `rob` in vsem točkam, ki pripadajo liku, v tabeli pripišemo vrednost -1. Ostale elemente tabele `rob` pustimo na privzeti vrednosti 0.

```
class Risar {
    ...
    public void narisiRob(Lik lik, int debelina) {
        int[][] rob = new int[DY][DX];
        for (int y = 0;  y < DY;  y++) {
            for (int x = 0;  x < DX;  x++) {
                if (lik.vsebuje(x, y)) {
                    rob[y][x] = -1;
                }
            }
        }
        ...
    }
    ...
}
```

Sedaj s pomočjo metode `dopolniRob` pripišemo vrednost 1 vsem točkam, ki ležijo na robu nivoja 1. To so točke, ki pripadajo liku, vendar pa vsaj ena od njihovih sosed ne pripada liku. Nato pripišemo vrednost 2 vsem točkam na robu nivoja 2. To so točke znotraj lika, pri katerih vsaj ena od sosed pripada robu nivoja 1. Postopek ponavljamo, dokler ne popišemo celotnega roba debeline `debelina`.

```
class Risar {
    ...
    public void narisiRob(Lik lik, int debelina) {
        ...
        for (int d = 1;  d <= debelina;  d++) {
            dopolniRob(rob, d);
        }
        ...
    }
    ...
}
```

Sliko roba debeline `debelina` tvorijo točke, ki pripadajo kateremukoli od robov nivoja 1, 2, ..., `debelina`.

```
class Risar {
    ...
    public void narisiRob(Lik lik, int debelina) {
        ...
        for (int y = 0;  y < DY;  y++) {
            for (int x = 0;  x < DX;  x++) {
                if (rob[y][x] > 0 && rob[y][x] <= debelina) {
                    this.slika[y][x] = true;
                }
            }
        }
    }
    ...
}
```

Metoda `dopolniRob` nastavi na vrednost d vse elemente tabele `rob`, ki pripadajo točkam na robu nivoja d. Metoda predpostavlja, da smo točkam na robovih nižjih nivojev že pripisali vrednosti, točke znotraj lika, ki jim vrednosti v tabeli `rob` še nismo pripisali, pa imajo vrednost -1. Metoda pripiše vrednost d vsem točkam z vrednostjo -1, pri katerih vsaj ena od sosed pripada robu nivoja $d-1$, poleg tega pa vrednost d pripiše tudi točkam z vrednostjo -1, ki ležijo na robu platna. (Ta situacija pride v poštev samo pri $d = 1$, saj lahko točke na robu platna ležijo kvečjemu na robu nivoja 1.)

```
class Risar {
    ...
    private static void dopolniRob(int[][] rob, int d) {
        for (int y = 0;  y < DY;  y++) {
            for (int x = 0;  x < DX;  x++) {
                if (rob[y][x] == -1 && vsajEnSosed(rob, x, y, d - 1)) {
                    rob[y][x] = d;
                }
            }
        }
    }
    ...
}
```

V metodi `vsajEnSosed` nam bo prišla prav tabela relativnih položajev posameznih sosed neke točke. Vrednost v vrstici i in stolpcu 0 (oziroma 1) podaja razliko med koordinato x (oziroma y) i-te sosede neke točke in koordinato x (oziroma y) točke same.

```
class Risar {
    ...
    private static final int[][] ODMIKI_SOSEDOV = {
        {-1,  0},
        { 0, -1},
        { 0,  1},
        { 1,  0}
    };
    ...
}
```

Metoda `vsajEnSosed` vrne `true` natanko v primeru, če se podana točka (x, y) nahaja na robu platna ali pa če vsaj ena od njenih štirih sosed leži na robu nivoja d.

```
class Risar {
    ...
    private static boolean vsajEnSosed(int[][] rob, int x, int y, int d) {
        if (x == 0 || y == 0 || x == DX - 1 || y == DY - 1) {
            return true;
        }

        for (int i = 0;  i < ODMIKI_SOSEDOV.length;  i++) {
            int xSosed = x + ODMIKI_SOSEDOV[i][0];
            int ySosed = y + ODMIKI_SOSEDOV[i][1];
            if (rob[ySosed][xSosed] == d) {
                return true;
            }
        }
        return false;
    }
}
```