B-OOP 2025: Zadanie č. 4

Aktualizované 07.04.2025

Stiahnite si zdrojové kódy ku zadaniu 4. Nájdete v nich priečinky `src` a `test`. Majú nasledujúcu štruktúru:

```
z4.zip
├── src
│   ├── image
│   │   ├── RasterImage.java
│   │   ├── TextImage.java
│   │   └── Savable.java
│   └── manipulation
│       ├── Normalizable.java
│       ├── Rotatable.java
│       ├── Substitutable.java
│       └── RotationDirection.java
├── test
│   └── image
│       ├── RasterImageTest.java
│       └── TextImageTest.java
├── dude.png
├── dude_rotated_left.png
├── dude_rotated_right.png
├── dude_normalized.png
├── text
├── text_left
└── text_right
```

Vašou úlohou je implementovať jednoduchý editor obrázkov. Pre tento účel musíte dokončiť implementáciu tried:

- `RasterImage`,
- `TextImage`.

Trieda `RasterImage` je už čiastočne implementovaná.

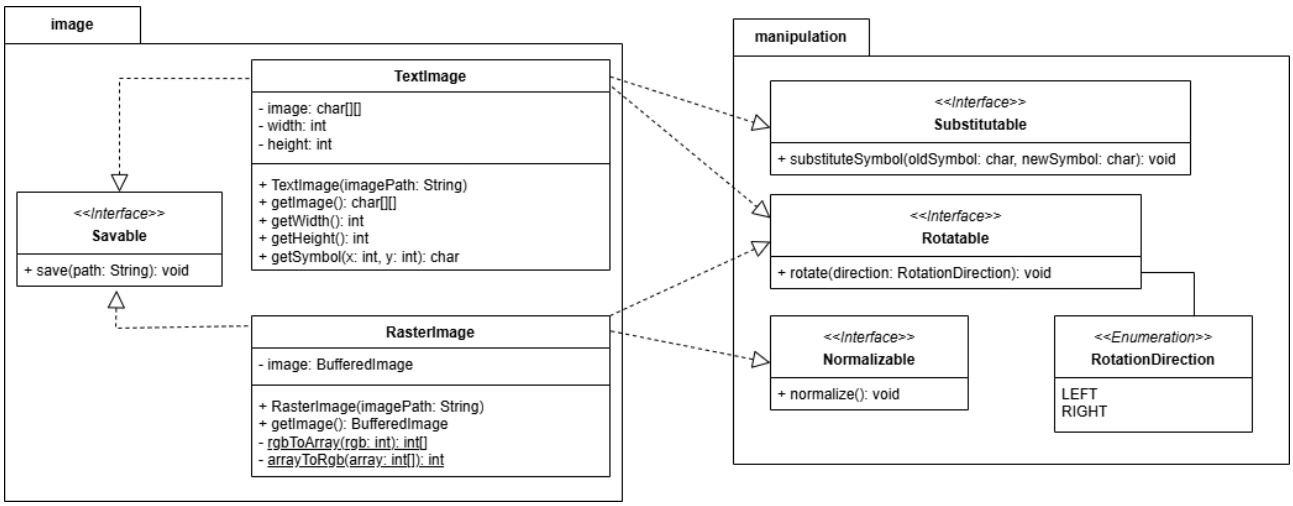
Triedy implementujte tak, aby spĺňali nižšie uvedený UML diagram tried, a zároveň aj dokumentačné komentáre, ktoré sa nachádzajú priamo v kóde. V dokumentačných komentároch predpisujeme, aké hodnoty môžu nadobúdať parametre, aké sú návratové hodnoty a aké výnimky majú byť vyhadzované.

Editor obrázkov bude zatiaľ podporovať dva formáty obrázkov:

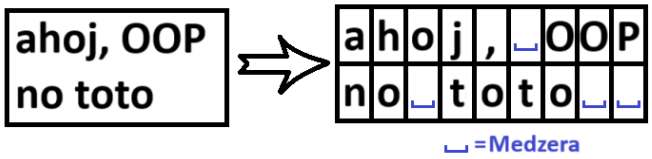
- PNG obrázky,
- textové obrázky (ascii art).

PNG obrázky zastrešuje trieda `RasterImage`. Táto trieda obaľuje triedu `BufferredImage` zo štandardnej knižnice (trieda `BufferedImage` podporuje aj iné formáty než PNG, ale môžete predpokladať, že pri testovaní triedy `RasterImage` budú použité iba obrázky vo formáte PNG). Textové obrázky zastrešuje trieda `TextImage`.

Konštruktor triedy `RasterImage` už je implementovaný. Trieda `TextImage` ukladá obrázok ako dvojrozmerné pole znakov. V konštruktore musíte načítať textový obrázok zo súboru a uložiť ho do tohto poľa. Jednotlivé riadky rozdelíte na znaky a pomocou nich naplníte pole `image`. Načítané riadky nesmú obsahovať znak konca riadku. Ak súbor končí prázdnymi riadkami, tieto riadky ignorujte. Jednotlivé riadky súboru môžu mať rôzne dĺžky. Predpokladajme, že v súbore je N neprázdnych riadkov a najdlhší riadok má M znakov (bez znaku konca riadku). Pole `image` bude mať veľkosť $N \times M$. Ak je i-ty riadok kratší ako M znakov, tak sa všetky jeho znaky (okrem znaku nového riadku) uložia do jednotlivých pozícií `image[i]` a zvyšné pozície sa zaplnia znakom medzera. Na obrázku 2 vidíme názorný príklad tejto situácie. Súbor obsahuje dva riadky: `ahoj, OOP` a



Obr. 1: UML diagram tried pre zadanie 4



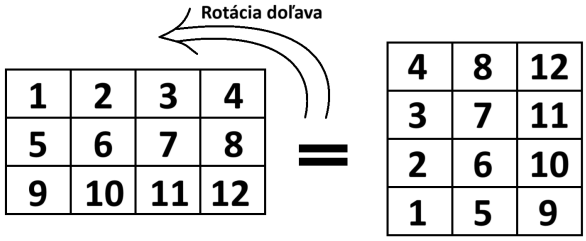
Obr. 2: Vyplnenie poľa `image` v `TextImage`

`no toto`. Najdlhší riadok má 9 znakov. Pole `image` má veľkosť 2×9. Riadok obsahujúci `no toto` sme doplnili dvomi medzerami.

Aj textové, aj PNG obrázky sú typy obrázkov, ale sú fundamentálne odlišné a preto nedáva veľký zmysel, aby dedili od spoločnej nadtriedy. Napriek tomu existujú operácie, ktoré dávajú zmysel v určitej forme vykonať nad oboma typmi obrázkov. Z toho dôvodu sme tieto operácie definovali v rozhraniach, ktoré budú tieto triedy implementovať.

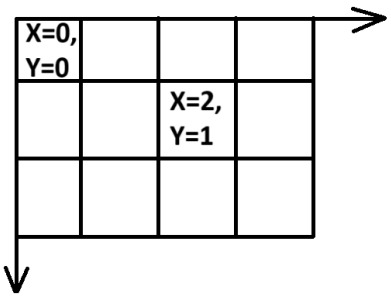
V prvom rade, obe triedy budú implementovať rozhranie `Savable`. Slúži na uloženie obrázku do súboru. V prípade `RasterImage` už je toto rozhranie implementované. Musíte ho však implementovať v `TextImage`. Metóda `save` uloží obsah poľa `image` po riadkoch do súboru.

`TextImage` aj `RasterImage` musia podporovať jednoduchú rotáciu. Za týmto účelom musíte implementovať rozhranie `Rotatable`. Obrázky je možné rotovať o 90° smerom vľavo (proti smeru chodu hodinových ručičiek) alebo vpravo (v smere chodu hodinových ručičiek). Na obrázku 3 je zobrazená rotácia obrázku vľavo. Pre názornosť sme očíslovali prvky poľa. V prípade `TextImage` je rotáciu možné dosiahnuť manipuláciou poľa `image`. V prípade `RasterImage` je potrebné manipulovať s `BufferedImage image`, ale logika rotácie bude v podstate identická, ako v prípade `TextImage`.



Obr. 3: Rotácia `TextImage` o 90° vľavo

Pre triedu `TextImage` dáva zmysel vykonať substitúciu znaku za iný znak. Napríklad, ak sa v ascii art vyskytuje znak '*', môžeme ho nahradiť znakom '.' v celom obrázku. Za týmto účelom implementujte rozhranie `Substitutable`. Pre `RasterImage` takáto operácia nedáva zmysel. Tento typ obrázku sa neskladá zo znakov,



Obr. 4: Znázornenie indexovania obrázkov pre obrázok veľkosti 2×4

ale z pixelov. Z toho dôvodu pre `RasterImage` rozhranie neimplementujte.

Posledným rozhraním je `Normalizable`. Toto rozhranie nedáva zmysel pre `TextImage`. V prípade `RasterImage` toto rozhranie implementujete. Rozhranie `Normalizable` pridáva možnosť vykonať normalizáciu obrázku. Normalizácia je operácia, ktorá slúži na vyhladenie intenzity pixelov. Často sa používa v počítačovom videní ako krok predspracovania obrázku. O normalizácii farebného priestoru sa dočítate napríklad na tejto stránke.

Pre potreby tohto zadania vám stačí vedieť, že normalizáciu počítame pre každý pixel samostatne nasledujúcim spôsobom:

```
// normalizuj pixel na koordinátoch (x,y)
R,G,B = získaj aktuálne hodnoty pre pixel (x,y)
sum = R+G+B
noveR = (R*255)/sum
noveG = (G*255)/sum
noveB = (B*255)/sum
nastav pixel (x,y) na nové hodnoty (noveR, noveG, noveB)
```

Pozor! Toto je len pseudokód. Zamyslite sa nad špeciálnymi prípadmi (ak nejaké existujú) a ošetrite ich. Pri výpočte hodnôt `noveR`, `noveG` a `noveB` zaokrúhľujte nadol.

Na úspešné zvládnutie zadania sa vám bude hodiť niekoľko tipov na prácu s triedou `BufferedImage`:

- Obrázky v počítačovej grafike majú počiatok súradnicovej osi v ľavom hornom rohu. Os y rastie smerom dole a os x rastie smerom doprava. To znamená, že pixel s koordinátami `x, y` sa nachádza na riadku y a stĺpci x. Ak má obrázok šírku `w` a výšku `h`, tak platí `0 <= x < w` a `0 <= y < h`. Názorná ukážka indexovania je viditeľná na obrázku 4.
- Normalizovať obrázok je možné bez vytvorenia jeho kópie, ale na rotáciu je zrejme potrebné vytvoriť nový pomocný obrázok. Nový, prázdny `BufferedImage`, ktorý má šírku `w` a výšku `h`, vytvoríte nasledujúcim spôsobom: `BufferedImage img = new BufferedImage(w, h, image.getType());` , kde `image` je `BufferedImage` v triede `RasterImage`.
- Metóda `getRGB` vracia `int`. Tento dátový typ je v jazyku Java veľký 4B a jednotlivé bajty sú nastavené na dopytované hodnoty. Nech `int rgb = image.getRGB(x, y);`. Potom:
  - `int alpha = (rgb » 24);`
  - `int red = (rgb » 16) & 0xFF;`
  - `int green = (rgb » 8) & 0xFF;`
  - `int blue = rgb & 0xFF;`

  Rovnakú logiku zachováva aj metóda `setRGB`.
- Šírku, resp. výšku `BufferedImage` je možné získať metódami `getWidth()`, resp. `getHeight()`.

Pri normalizácii neberte do úvahy kanál alfa, iba kanály R, G a B. V testoch ignorujeme hodnotu kanálu alfa. Na korektnú implementáciu normalizácie a rotácie v `RasterImage` nepotrebujete iné metódy než tie, ktoré sú uvedené v tomto dokumente.

Zdrojové kódy k zadaniu 4 obsahujú podmnožinu unit testov, ktoré budú použité na hodnotenie vašich zadaní, ako aj testovacie dáta pre tieto testy.

Pozor! Je dôležité, aby ste dodržali predpísané názvy tried a metód, ktoré vyplývajú z popisu zadania a z priloženého UML diagramu. Nemeňte štruktúru projektu! Ak to považujete za potrebné, môžete si v jednotlivých triedach vytvoriť dodatočné pomocné metódy a premenné. Môžete rozšíriť existujúce triedy o implementáciu dodatočných rozhraní. Nemeňte metódy, ktoré už sú implementované.

V AIS je vytvorené miesto odovzdania `OOP - Zadanie 4` do času precizovaného mailom. Do tohto miesta odovzdania nahráte **zip** archív (to znamená nie `rar`, ani `tar`...), ani žiadny iný formát než `zip`. Váš archív bude obsahovať priečinok `src` s nasledujúcou štruktúrou (za predpokladu, že ste nevytvorili dodatočné triedy):

```
archiv.zip
└── src
    └── image
        ├── RasterImage.java
        └── TextImage.java
```

Vaše zadanie bude hodnotené pomocou automatizovaných testov, aby sme si overili, či splňuje všetky náležitosti. Zadania, ktoré splnia uvedené náležitosti, a prejdú všetkými automatizovanými testami, budú ohodnotené 1 bodom.