Tema 2 - MiniCAD

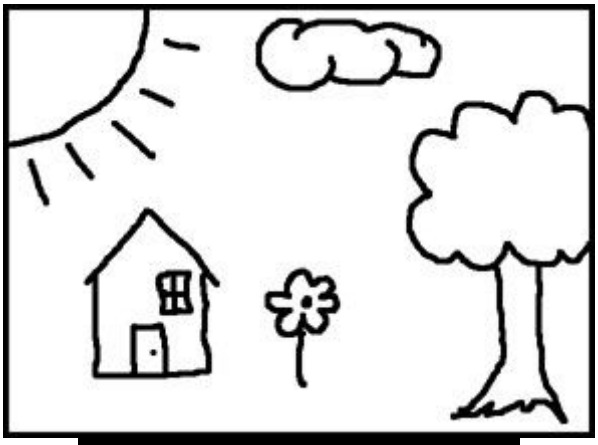
- Responsabili: Burdușelu Mihai, Ceapă Delia, Nica Cristi, Paraschivescu Tudor
- Deadline soft: 26.11.2017 23:55
- Deadline hard: 5.12.2017 23:55 (1 sapt + zilele libere)
- Versiune tester: v12-11-2017
- Data publicării: 6.11.2017
- Istoric modificări:
  - **07.11.2017** - Corectare fișiere de input cu forma corectă pentru CANVAS
  - **08.11.2017** - Corectare algoritm Bresenham pentru linie (updatarea erorii în while error > 0)
  - **11.11.2017** - Regenerarea fișierelor de input și de referință (toate formele care se colorează prin Flood Fill au acum centrul de greutate în interiorul CANVASului)
  - **12.11.2017** - Clarificări timpi de execuție (timeoutul setat pe VMChecker) și diagonală romb
  - **13.11.2017** - Specificări suplimentare pentru TRIANGLE și POLYGON (ordinea de desenare a liniilor)

Obiectivele temei

- Aprofundarea noțiunilor de moștenire, agregare și interfațare în contextul programării orientate pe obiecte.
- Utilizarea unor design patterns în contextul implementării unei aplicații reale, și anume:
  - Singleton Pattern
  - Factory Pattern
  - Visitor Pattern
- Respectarea unui **coding-style** adecvat.

Descriere

CAD [1] (**Computer-Aided Design** sau **Computer-Aided Design and Drafting**) reprezintă utilizarea tehnologiei pentru crearea unor designuri (în general 2D) și înlocuiește procesul de desenare manuală a unei piese cu unul automatizat. Unul dintre cele mai cunoscute astfel de programe este AutoCAD. Astfel de programe îmbunătățesc calitatea design-ului și productivitatea designer-ului.



Aflati mai multe despre CAD la [2]

Programul **MiniCAD** pe care urmează sa îl implementați va respecta principiile unui software CAD utilizat în viața de zi cu zi de către mii de companii și va putea produce forme geometrice diverse, în diferite poziții și în diferite culori. Pentru simplitate, nu va trebui sa implementați o interfață grafică pentru a realiza asta, ci doar să vă folosiți de cunoștințele acumulate în cadrul materiei și de câteva alte noțiuni adiționale simple.

Cerințe

Pentru această temă, va trebui să implementați folosind **Visitor Pattern** [3] un program MiniCAD care va putea desena forme geometrice. Pe baza datelor inițiale și a setului de instrucțiuni primite la intrare, acesta va produce la finalul execuției comenzilor un fișier PNG, reprezentând imaginea finală. În același timp, trebuie să aveți în vedere utilizarea **Factory Pattern** [4] și **Singleton Pattern** [5], pentru gestionarea creării fiecărei forme în parte.

Unul dintre scopurile temei este implementarea cât mai corectă a celor trei patternuri menționate.

Implementări doar parțiale sau greșite ale acestor patternuri pot fi depunctate. Lipsa implementării acestor patternuri (nici măcar încercarea sau implementarea parțială) va duce la punctarea temei cu **0 puncte**.

Date de intrare

Numele fișierului de input din care se citește este primit ca argument al programului. Fișierele de input vor avea următoarea structură:

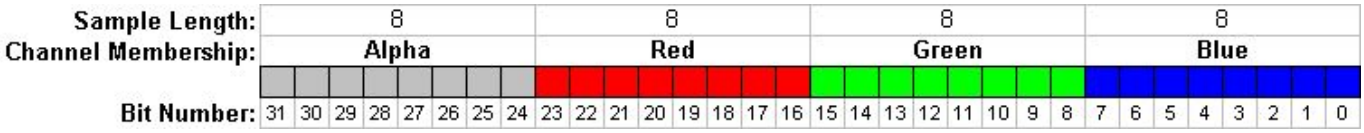
- Pe prima linie se va găsi un număr natural N, reprezentând numărul total de forme ce se vor afla în desen.
- Următoarele N linii vor conține informații despre fiecare dintre cele N forme: poziția unei forme, dimensiunile și culoarea acesteia.

Culorile pe care va trebui să le folosiți vor fi redate în formatul: "**#RGB A**", unde R, G și B reprezintă valorile pixelilor Red, Green și Blue în format hexa (256 de valori posibile: 00, 01, 02, ..., 09, 0A, 0B, ..., 0F, 10, 11, ..., FE, FF), iar A reprezintă valoarea Alpha (de opacitate) în format decimal, putând lua valori întregi de la 0 la 100.

Exemple de culori ce pot apărea în input:

- #110859 100
- #1F0ab9 80
- #0008Af 100

Spațiul de culori RGBA [6] este spațiul de culori ce are la bază aceste patru proprietăți specificate anterior: Red, Green, Blue și Alpha. Reprezentarea unei culori pe care o veți folosi în rezolvare este reprezentarea întreagă ARGB, și anume un număr întreg în care primul byte este canalul Alpha, al doilea canalul Red, al treilea canalul Green, iar al patrulea canalul Blue.



Desigur, fiecare formă este diferită, așa că va avea un format diferit. Posibilitățile sunt următoarele:

- Fundal:

CANVAS <înălțime> <lățime> #RGB A

Acesta va fi mereu primul în orice input, pentru a stabili dimensiunile imaginii.

- Linie:

LINE <coordonata x de start> <coordonata y de start> <coordonata x de final> <coordonata y de final> #RGB A

- Pătrat:

SQUARE <coordonata x a coltului stanga sus> <coordonata y a coltului stanga sus> <dimensiunea unei laturi> #RGB A #RGB A

Pentru toate formele ce vor avea 2 culori în linia de input (pătrat, dreptunghi, cerc, triunghi, romb, poligon), prima culoare este cea folosită pentru conturul formei, iar cea de-a doua pentru interiorul ei.

- Dreptunghi:

RECTANGLE <coordonata x a coltului stanga sus> <coordonata y a coltului stanga sus> <dimensiunea înălțimii> <dimensiunea lungimii> #RGB A #RGB A

- Cerc:

CIRCLE <coordonata x a centrului> <coordonata y a centrului> <raza> #RGB A #RGB A

- Triunghi:

TRIANGLE <coordonata x a primului punct> <coordonata y a primului punct> <coordonata x a celui de-al doilea punct> <coordonata y a celui de-al doilea punct> <coordonata x a celui de-al treilea punct> <coordonata y a celui de-al treilea punct> #RGB A #RGB A

Liniile ce formează conturul triunghiului se vor face în ordinea dată (linii de la P1 la P2, de la P2 la P3 și de la P3 la P1).

- Romb:

DIAMOND <coordonata x a centrului> <coordonata y a centrului> <dimensiunea diagonalei orizontale> <dimensiunea diagonalei verticale> #RGB A #RGB A

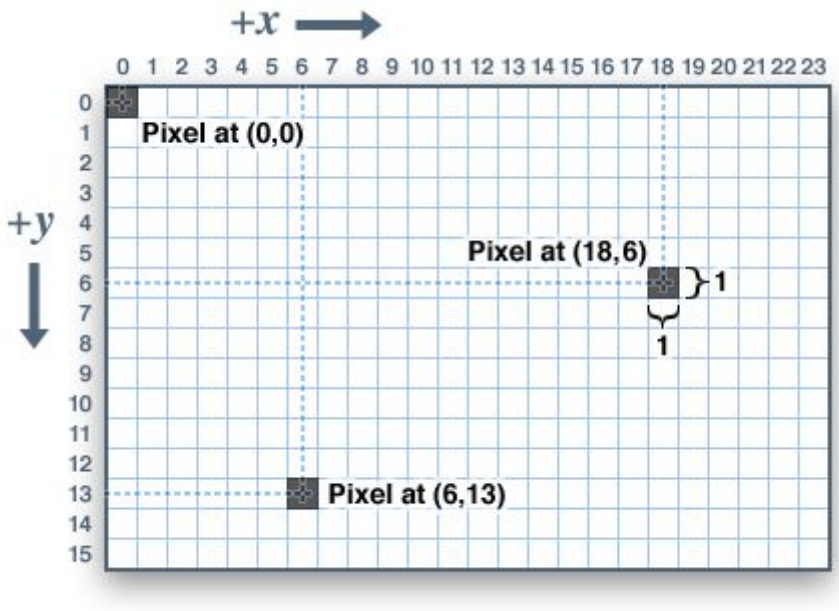
Pentru diagonalele cu lungime număr impar, semidiagonala se consideră rotunjirea prin lipsă a împărțirii pentru a lucra numai cu numere întregi.

- Poligon

POLYGON <numarul de puncte> (numarul de puncte) * (<coordonata x a unui punct> <coordonata y a unui punct>) #RGB A #RGB A

Liniile ce formează conturul poligonului se vor face în ordinea dată (linii de la P1 la P2, de la P2 la P3, ..., de la Pn-1 la Pn, de la Pn la P1).

Toate coordonatele pixelilor vor începe din colțul din stânga sus și axele se vor numerota astfel:



Acest set de intrări crește în dificultate gradat: 10 fișiere cu maxim 5 forme, 10 cu 5-15 forme, 10 cu 15-30 forme, 5 cu 30-45 si 5 cu 45-50. De ce? Pentru că, în acest mod, vă puteți pregăti pas cu pas pentru adevărata provocare: să desenați două animale mai speciale. Desigur, veți primi din nou inputul în același format și, dacă ați trecut cu bine de cele de până aici, nu ar trebui ca acestea să prezinte vreo problemă. De asemenea, vă invităm să admirați și voi rezultatul!

Exemplu input

Un exemplu de fișier de input alături de imaginea generată de acesta puteți găsi în folderul **media** al repository-ului de Github al temei 2 [7].

Datele de ieșire

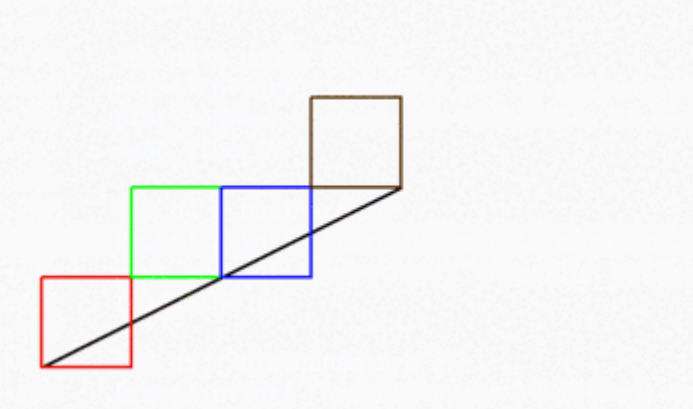
Outputul programului trebuie sa fie o imagine de tip **PNG** cu numele “**drawing.png**” care să prezinte formele descrise în input.

Algoritmii de desenare

Pentru a trece la formatul vizual al formelor, va trebui să folosiți subclasa BufferedImage [8] a clasei Image [9] . Clasa aceasta stochează matricea de pixeli a imaginii. În cazul de față, aceasta este foarte utilă deoarece metoda de scriere a clasei ImageIO [10] se folosește de metodele sale de Get și Set pentru a prelucra matricea de pixeli. Dacă ne gândim la o imagine ca la o matrice de pixeli, care văzută în ansamblu compune imaginea vizibilă nouă, atunci tot ce trebuie să facem pentru a crea o imagine este să ”pictăm” fiecare pixel accesând metoda sa de Set (setRGB) și aplicând, prin ea, o anumită culoare asupra pixelului respectiv. Aceasta primește ca parametru valoarea întreagă ARGB a culorii, explicată anterior. Desigur, fiecare formă este diferită, deci asupra fiecăreia va fi nevoie să procedați diferit când vine vorba de selectarea efectivă a pixelilor ce vor fi colorați. Astfel, vom prezenta în continuare algoritmii pe care va trebui să îi implementați pentru a realiza acest lucru.

- Linie: **Forma generală a Algoritmului lui Bresenham**

Algoritmul lui Bresenham [11] de desenare a liniilor este un algoritm care determină punctele care ar trebui selectate pentru a forma o linie cât mai apropiată de o linie dreaptă între două puncte ale unui raster n-dimensional. Este un algoritm de eroare incrementală și unul dintre cei mai vechi algoritmi dezvoltați în domeniul graficii computerizate.



Forma algoritmului depinde mult de unghiul și de panta dreptei. Pseudocodul pentru forma generalizată (adaptare din “Procedural Elements for Computer Graphics” [12] de David F. Rogers) a acestuia arată în felul următor:

bresenham_algorithm

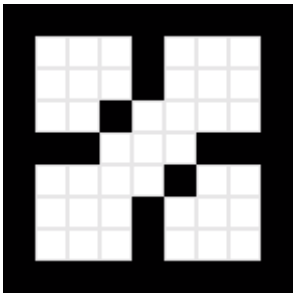
```
// Generalized Integer Bresenham's Algorithm for all quadrants
// all variables are assumed integer
// sign function returns -1, 0, 1 as its argument is < 0, = 0 or > 0
bresenham_algorithm(x1, y1, x2, y2)

    // initialize variables
    x = x1
    y = y1
    delta_x = abs(x2 - x1)
    delta_y = abs(y2 - y1)
    s1 = sign(x2 - x1)
    s2 = sign(y2 - y1)

    // interchange delta_x and delta_y, depending on the slope of the
line
    if delta_y > delta_x then
        interchange(delta_x, delta_y)
        interchanged is true
    else interchanged is false
    end if

    // initialize the error term to compensate for a nonzero intercept
    error = 2 * delta_y - delta_x

    for i =  to delta_x
        set_pixel(x, y)
```

```
        while error >
            if interchanged is true then x = x + s1
            else y = y + s2
            end if
            error = error - 2 * delta_x
        end while

        if interchanged is true then y = y + s2
        else x = x + s1
        end if

        error = error + 2 * delta_y
    end for
end
```

- Canvas, Pătrat, Dreptunghi: Reprezintă niște forme clar definite la care determinarea pixelilor de pe marginea sau din interiorul formei este trivială.
- Cerc: **Algoritmul lui Bresenham de desenare a cercurilor**

Pentru desenarea cercurilor, algoritmul folosit va fi tot cel al lui Bresenham, care se bazează tot pe erori incrementale și evaluarea unui factor de decizie. Aici [13] gasiți o explicație foarte bună a algoritmului ce trebuie folosit, alături de o prezentare adițională a altui algoritm de desenare a cercurilor (Mid-Point).

- Cerc, Triunghi, Romb, Poligon: **Flood Fill**



Umplerea acestor forme o puteți considera ca umplerea spațiului dintre mai multe laturi, ca în Microsoft Paint. Acest procedeu se numește Flood Fill [14] .

Recomandarea noastră la umplerea acestor 4 forme este pornirea din centrul de greutate al formei. **Centrul de greutate este garantat a fi în cadrul CANVAS-ului**.

Precizări

- Datele de intrare se consideră corecte din punct de vedere al sintaxei.

- Toate coordonatele, dimensiuni (laturi, lățime, rază, diagonală), valoare alpha se consideră a fi numere întregi.
- La suprapunerea a două forme, culorile nu se vor amesteca, acei pixeli rămânând de culoarea ultimei forme din input. Excepție de la această regulă face suprapunerea anumitor forme (triunghi, romb, poligon) cu aceeași culoare de contur, caz în care umplerea celei de-a doua forme se va opri la conturul primei forme. Nu este nevoie să vă preocupați efectiv cu aceste aspecte, ele sunt problema algoritmilor de desenare; este pur și simplu ceva ce ar trebui să aveți în vedere când analizați rezultatele.
- La ieșirea unor forme din canvas, trebuie sa aveți grijă să nu colorați decât acei pixeli care se află în limitele acestuia. **Hint**: puteți face verificarea de încadrare într-o funcție auxiliară (**evitați duplicarea de cod!**)

Structură arhivă

Arhiva pe care o veţi urca pe **VMChecker** va trebui să conţină în directorul rădăcină:

- fișierul de tip **Makefile**, avand numele `MiniCADMakefile`, care sa includa regulile build și clean
- fișierul `Main.java` (entry-point-ul aplicatiei voastre care nu se află în vreun pachet)
- alte fișiere **organizate** cu implementarea **voastră**
- fișierul README

Nu încărcați fișierele de test, checker-ul sau documente generate cu JavaDoc.

Mod de utilizare checker public

Checkerul de pe **VMChecker** va fi cel public din folderul **checker** al repository-ului de Github al Temei 2. După descărcarea locală a folderului **checker**, testarea o puteți face prin rularea scriptului din acest folder, după ce ați copiat sursa Main.java în folderul **checker** alături de celelalte fișiere sursă folosite în implementare.

Notare

- 90p teste publice (40 de teste a cate 2.25p fiecare)
- 10p README (care surprinde cele mai relevante detalii de implementare)

Checkstyle-ul va efectua doar o verificare, dar nu va putea aduce un punctaj adițional. **Totuși, la peste 30 de erori rezultate din acesta, se va scădea câte 1p pentru fiecare warning.**

Exemple punctare:

- 40 teste OK, README, 0 checkstyle warnings → 100p

- 40 teste OK, README, 30 checkstyle warnings → 100p
- 40 teste OK, README, 31 checkstyle warnings → 69p
- 40 teste OK, README, 55 checkstyle warnings → 45p

Rularea temei pe **VMChecker** cu cele 40 de teste trebuie să se încadreze în timpul de timeout de **150 de secunde**. Încercați astfel să nu lăsați pe ultima sută de metri rezolvarea, deoarece se pot forma cozi de așteptare pentru testarea pe **VMChecker**.

Temele vor fi testate împotriva plagiatului. Orice tentativă de copiere va duce la **anularea punctajului** de pe parcursul semestrului şi **repetarea materiei** atât pentru sursă(e) cât şi pentru destinație(ii), fără excepție.

You shall indeed not pass!

Linkuri utile

- [1] - CAD
- [2] - What is AutoCAD used for?
- [3] - Visitor Pattern
- [4] - Factory Pattern
- [5] - Singleton Pattern
- [6] - RGBA Color Space
- [7] - Repository Github Tema 2
- [8] - Documentație BufferedImage
- [9] - Documentație Image
- [10] - Documentație ImageIO
- [11] - Algoritmul lui Bresenham de desenare a liniilor
- [12] - "Procedural Elements for Computer Graphics" de David F. Rogers
- [13] - Algoritmi de generare a cercurilor
- [14] - Flood Fill