SFML - Crtanje i transformacije oblika

Objektno programiranje - 4. vježbe

Marko Živković

Prirodoslovno-matematički fakultet,
Sveučilište u Zagrebu

3. travnja 2026. godine

Crtanje - posebna klasa prozora

`sf::RenderWindow`

- klasa izvedena iz `sf::Window` (sve funkcije naslijeđene)

Primjer. U trenutnom kodu potrebne su minimalne promjene:

```
#include <SFML/Graphics.hpp>
...
  sf::RenderWindow prozor;
  prozor.create(sf::VideoMode(800,600),"Prozor");
  while (prozor.isOpen()) {
      sf::Event d;
      while (prozor.pollEvent(d)) {
          if (d.type == sf::Event::Closed) {
              prozor.close();
          }
      }
  }
```

Napomena uz prethodni `#include`

- dovoljno je samo uključiti datoteku **`SFML/Graphics.hpp`** (umjesto još i datoteke `SFML/Window.hpp`) jer u toj datoteci između ostalog piše:

```
#include <SFML/Window.hpp>
```

Čisti/crtaj/prikaži ciklus

- ne crtamo odmah na ekran nego u spremnik
- u prikladnom trenutku kopiramo to na ekran

```
while (prozor.isOpen()) {
    ...
    prozor.clear(sf::Color::Black);
    //prozor.draw(...);
    prozor.display();
}
```

- ne želimo da se crta jedno preko drugoga - prvo očistimo prozor
- `clear` prima enumerirani tip **`sf::Color`** (*default* je crna boja)
- nakon crtanja, prikažemo nacrtano `display` metodom

Boje - klasa `sf::Color`

- za manipuliranje **RGBA** bojama
- *defaultni* konstruktor daje crnu, neprozirnu boju

Primjer.

```
sf::Color crna;
prozor.clear(crna);
```

```
sf::Color::Color(Uint8 red, Uint8 green,
                 Uint8 blue, Uint8 alpha=255)
```

- crvena, zelena, plava i prozirnost komponenta - svaka cijeli broj iz raspona [0,255]

Ima i treći konstruktor: `sf::Color::Color(Uint32 color)`

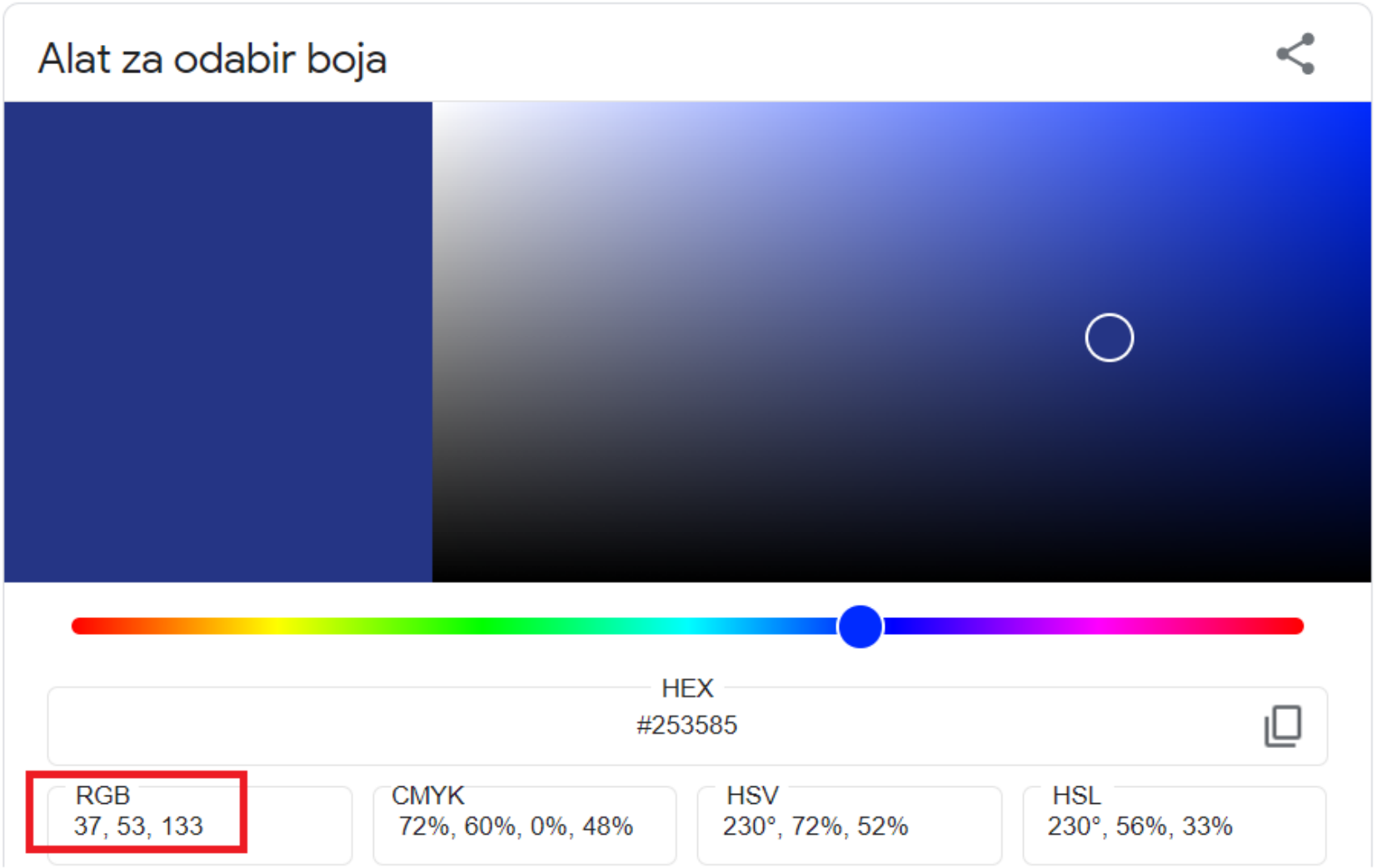
- parametar je 32-bitni nenegativni cijeli broj koji sadrži RGBA komponente (u tom redoslijedu)

Predefinirane boje

- `sf::Color::Black`
- `sf::Color::Blue`
- `sf::Color::Cyan`
- `sf::Color::Green`
- `sf::Color::Magenta`
- `sf::Color::Red`
- `sf::Color::Transparent`
- `sf::Color::White`
- `sf::Color::Yellow`

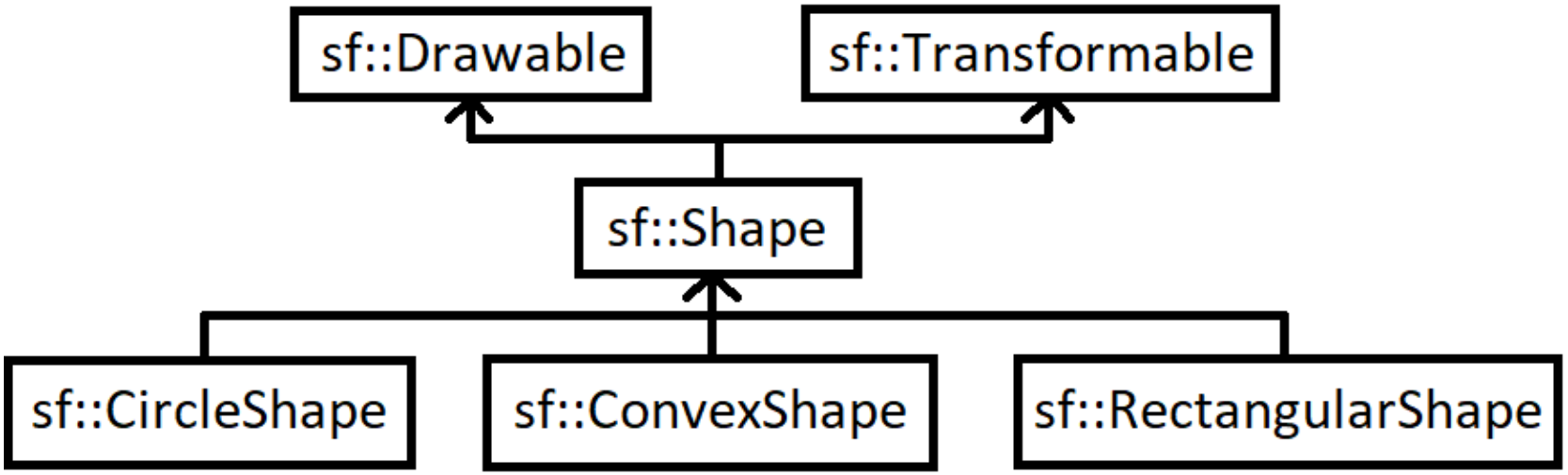
Boje pomoću RGB koda

- guglati Alat za odabir boja (*color picker*) i odabrati željenu boju
- za RGB 37, 53, 133 $\rightarrow$ `sf::Color boja(37,53,133);`



Crtanje oblika

- svaki oblika ima posebnu klasu
- većina svojstava ista jer imaju istu baznu klasu (**`sf::Shape`**)
- dijagram nasljeđivanja:



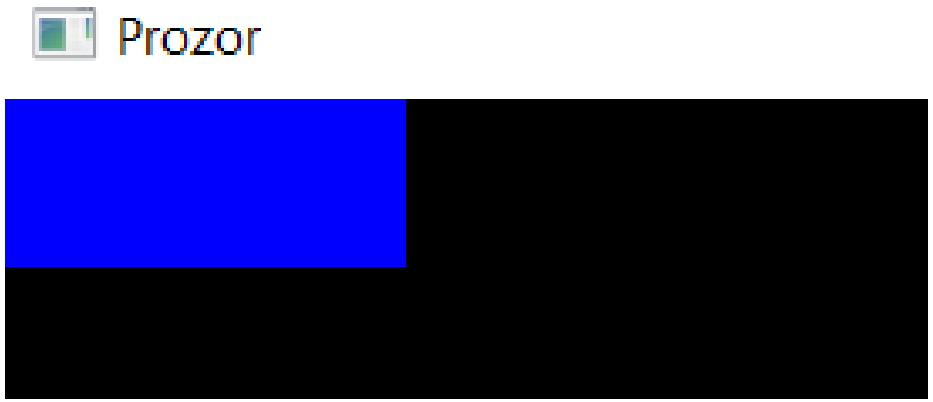
- `sf::Shape` je **apstraktna klasa** ($\Rightarrow$ mora se specijalizirati konkretnim oblikom)
- **`sf::Drawable`** - objekti koji se mogu crtati na ekran
- **`sf::Transformable`** - pruža funkcionalnosti pomicanja, skaliranja i rotiranja objekta

Klase oblika: pravokutnik

- klasa **`sf::RectangleShape`**
- konstruktor prima njegove dimenzije
- samo `setSize` je specifična za tu klasu

Primjer. Plavi pravokutnik dimenzija 120×50:

```
prozor.clear(sf::Color::Black);
sf::RectangleShape p(sf::Vector2f(120.f,50.f));
p.setFillColor(sf::Color::Blue);
prozor.draw(p);
prozor.display();
```



Klase oblika: pravokutnik - promjena veličine

- funkcija specifična za klasu `sf::RectangleShape` je `setSize`
- možemo nakon konstruktora mijenjati veličinu pravokutnika
- po *defaultu* je 0×0 - konstruktor ima ovakvu deklaraciju:

```
RectangleShape(const Vector2f &size=Vector2f(0,0))
```

Primjer. Umjesto prethodnog konstruktora može ovako:

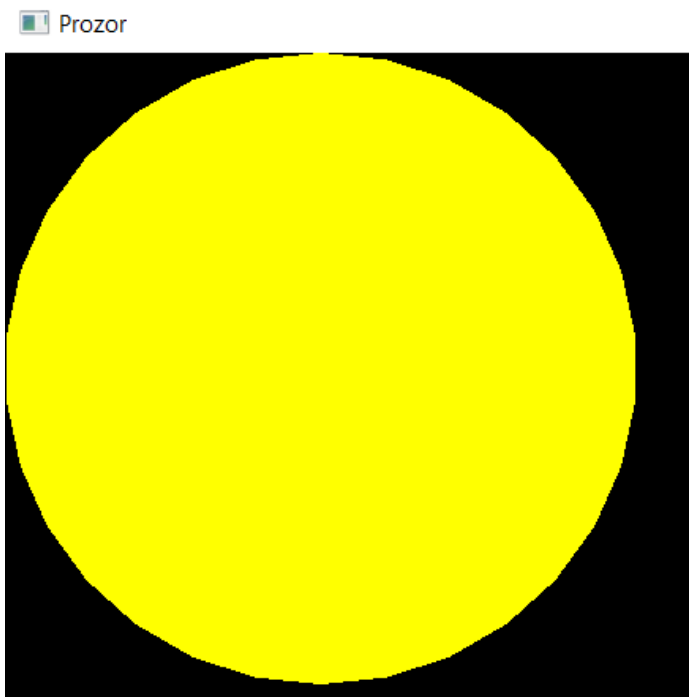
```
sf::RectangleShape p;
p.setSize(sf::Vector2f(120.f, 50.f));
```

Klase oblika: krug

- klasa **sf::CircleShape**
- konstrukor prima veličinu radijusa (polumjera)

Primjer.

```
prozor.clear(sf::Color::Black);
sf::CircleShape p(200.f);
p.setFillColor(sf::Color::Yellow);
prozor.draw(p);
prozor.display();
```



Klase oblika: krug - specifične funkcije i konstruktor

- konstruktor ima sljedeću deklaraciju:

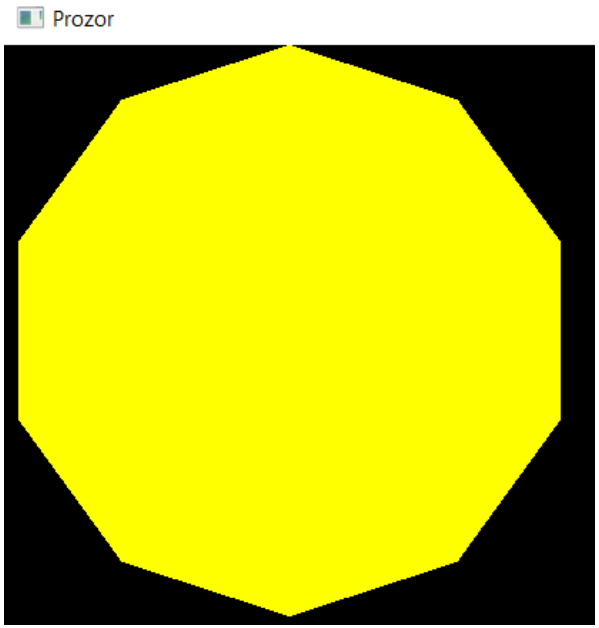
```
CircleShape(float radius=0,std::size_t pointCount=30)
```

- drugi (opcionalan) argument je broj stranica („kvaliteta" kruga; ne crta se savršen krug nego aproksimacija mnogokutom)
- za naknadno postavljanje radijus i broja stranica: funkcije `setRadius` i `setPointCount`

Primjer.

```
sf::CircleShape p;
p.setRadius(200.f);
p.setPointCount(10);
```

- uočite: dobili smo pravilni deseterokut

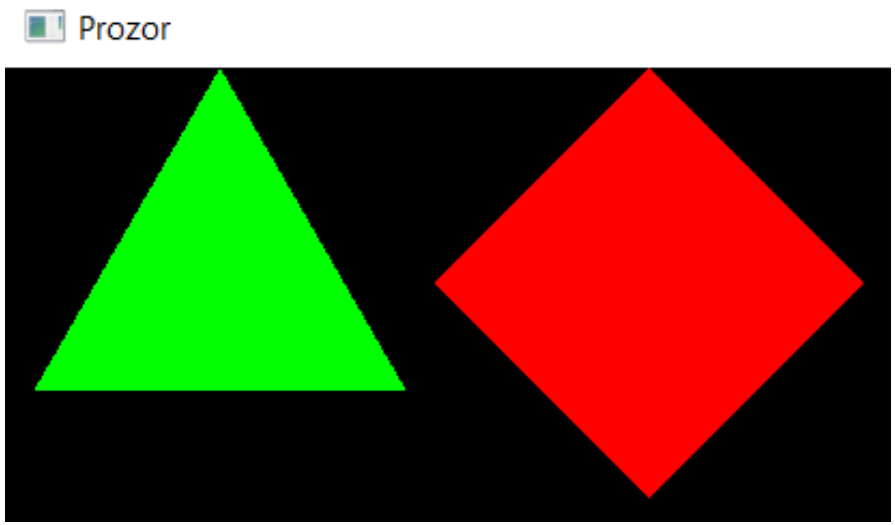


Pravilni mnogokuti

- ne postoji klasa za pravilne mnogokute - mogu se dobiti kao u prethodnom primjeru prilagodbom broja stranica

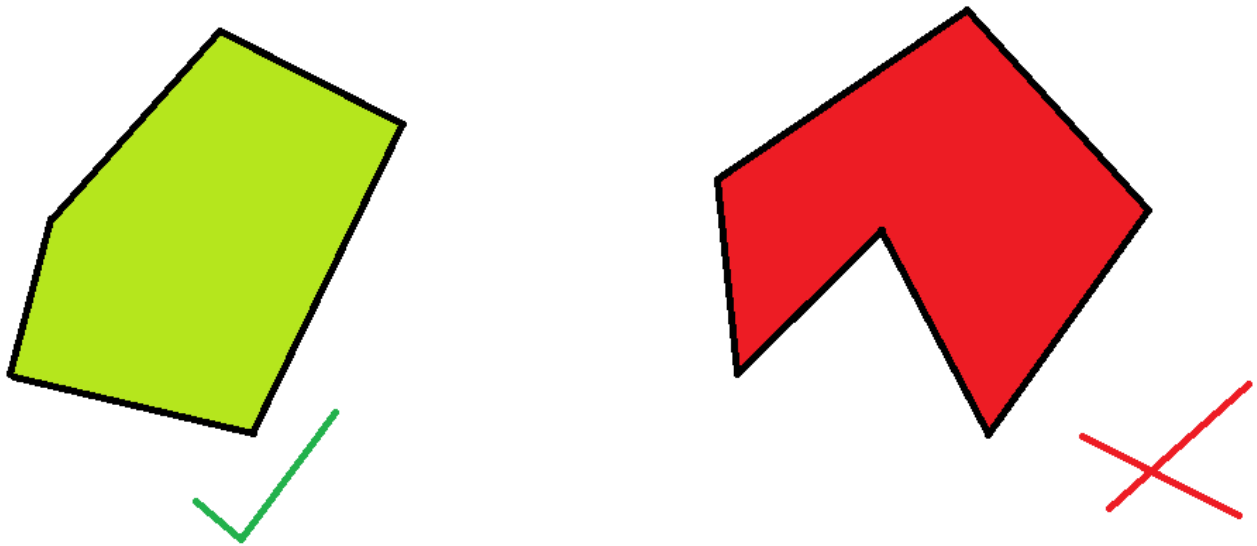
Primjer. Jednakostranični trokut i kvadrat:

```
sf::CircleShape t(100.f,3);
t.setFillColor(sf::Color::Green);
prozor.draw(t);
sf::CircleShape k(100.f,4);
k.setFillColor(sf::Color::Red);
k.move(200.f, 0.f);
prozor.draw(k);
```



Konveksni oblici

- klasa `sf::ConvexShape` - za crtanje konveksnih mnogokuta
- važno: **SFML u načelu ne može crtati konkavne mnogokute**
  $\Rightarrow$ treba ih crtati kao više konveksnih mnogokuta



- slika desno: konveksni mnogokut, slika lijevo: nekonveksni (konkavni) mnogokut

Pitanje. Kako biste nacrtali gornji desni mnogokut?

Konveksni oblici - crtanje

`sf::ConvexShape::ConvexShape(std::size_t pointCount=0)`

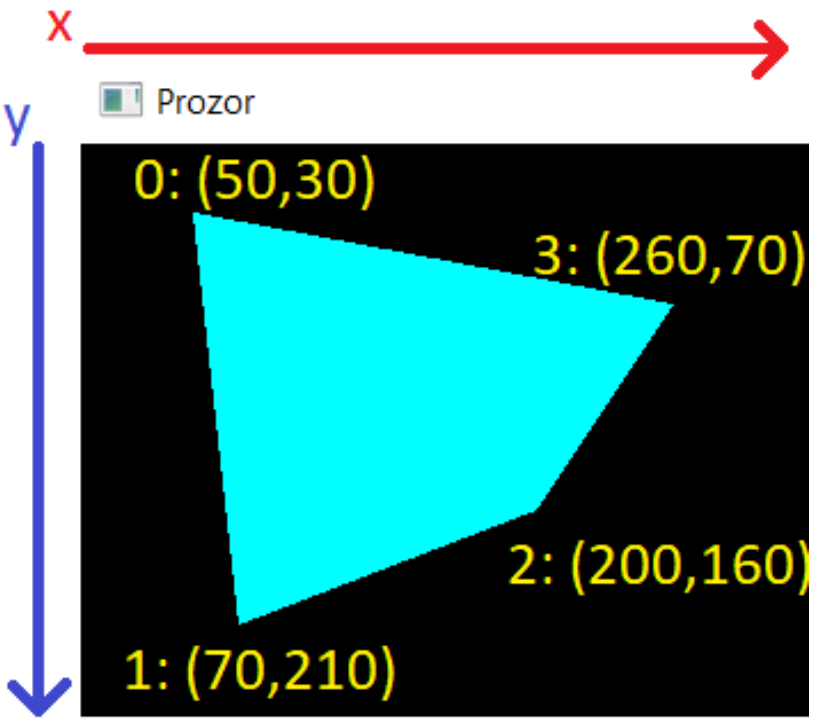
- konstruktor prima broj točaka mnogokuta (za valjani oblik treba biti $\geqslant 2$ točaka)
- funkcija `getPointCount()` vraća postavljeni broj točaka
- točke se nakon toga navode **po redu** (u smjeru kazaljke na satu ili u smjeru obratnom kazaljci na satu)
- **indeks mora biti u rasponu** $[0, \texttt{getPointCount()} - 1]$

Konveksni oblici - primjer

```
prozor.clear(sf::Color::Black);
sf::ConvexShape p(4);
p.setFillColor(sf::Color::Cyan);
p.setPoint(0,sf::Vector2f(50,30));
p.setPoint(1,sf::Vector2f(70,210));
p.setPoint(2,sf::Vector2f(200,160));
p.setPoint(3,sf::Vector2f(260,70));
prozor.draw(p);
prozor.display();
```

Napomena. Umjesto gornjeg konstruktora može ovako:

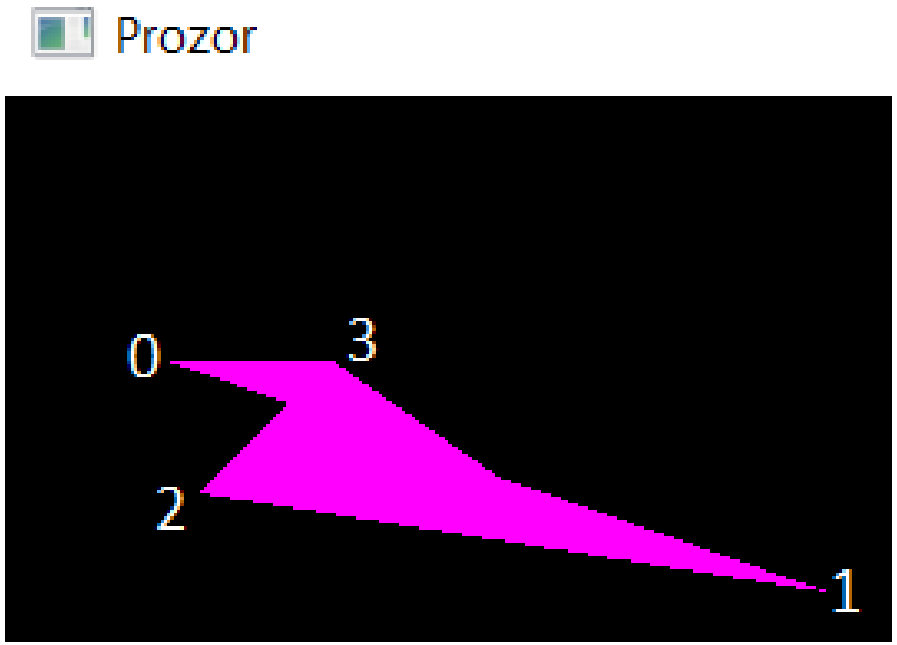
```
sf::ConvexShape p;
p.setPointCount(4);
```



Konveksni oblici - još jedan primjer

Pitanje. Što nije u redu u sljedećem primjeru?

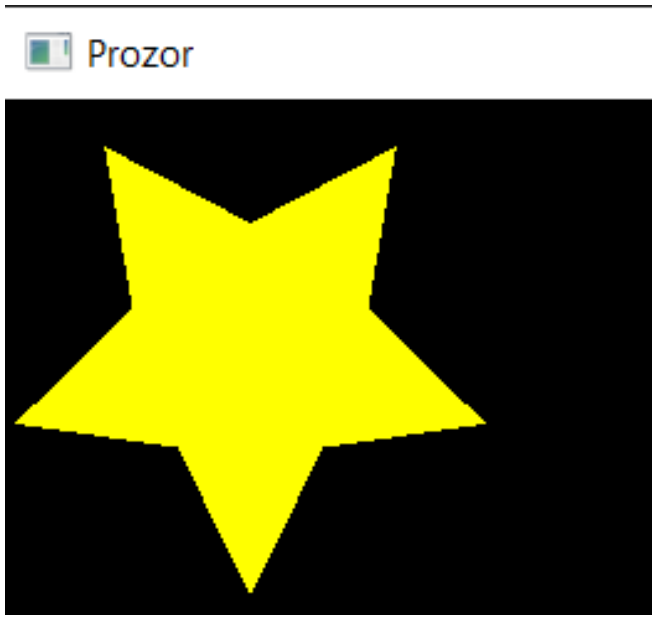
```
sf::ConvexShape p(4);
p.setFillColor(sf::Color::Magenta);
p.setPoint(0,sf::Vector2f(50,80));
p.setPoint(1,sf::Vector2f(250,150));
p.setPoint(2,sf::Vector2f(60,120));
p.setPoint(3,sf::Vector2f(100,80));
```



Triangle fans

- zapravo nije nužno crtanje konveksnih mnogokuta
- jedini zahtjev: ako crtamo linije iz težišta do svih navedenih točaka, one moraju biti nacrtane u istom redoslijedu
- konveksni oblici se zapravo crtaju pomoću trokuta
- zbog toga možemo, primjerice, nacrtati zvijezdu

Zadatak 1. Nacrtati zvijezdu sličnu onoj s donje slike.



Jedno od mogućih rješenja

```
sf::ConvexShape s(10);
s.setFillColor(sf::Color::Yellow);
float pi = float(atan(1)) * 4;
for (size_t k = 0; k <= 4; ++k) {
    float si = 2 * pi * k / 5 + pi / 2,
          co = 2 * pi * k / 5 + pi / 2;
    s.setPoint(2*k, sf::Vector2f
          (100*cos(co)+100,100*sin(si)+100));
    s.setPoint(2*k+1, sf::Vector2f
        (50*cos(co+pi/5)+100,50*sin(si+pi/5)+100));
}
prozor.draw(s);
```

- prikazano rješenje zahtijeva `#include<cmath>`

Obrub oblika

- što sve dobivamo od klase `sf::Shape` može se vidjeti u dokumentaciji klase (link) - posebno su to funkcije za obrub

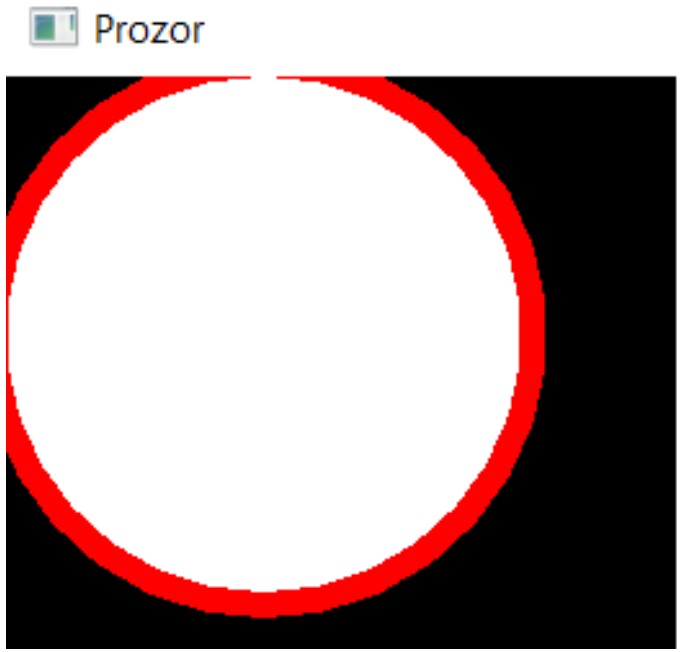
Primjer.

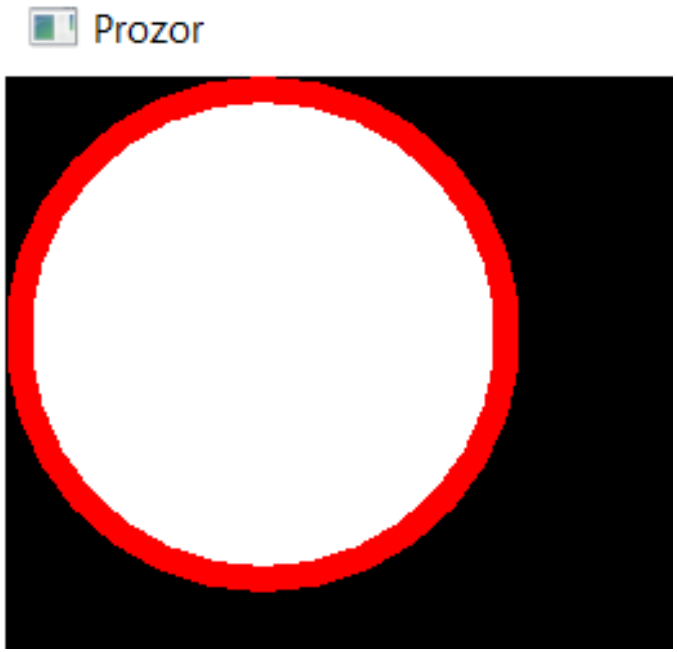
```
prozor.clear(sf::Color::Black);
sf::CircleShape s(100.f);
s.setFillColor(sf::Color::White);
s.setOutlineThickness(10.f);
s.setOutlineColor(sf::Color::Red);
prozor.draw(s);
```

- **`setOutlineThickness`** postavlja debljinu obruba
- **`setOutlineColor`** postavlja boju obruba

Obrub oblika

- po *defaultu* obrub je izvan oblika
- u prethodnom, radijus kruga je 100 piksela, a obrub je debljine 10 piksela $\Rightarrow$ dobili smo krug radijusa 110 piksela (slika lijevo)





- ako želimo da obrub bude dio oblika, postavimo mu negativnu debljinu (slika desno; ukupan radijus i dalje je 100 piksela):

```
s.setOutlineThickness(-10.f);
```

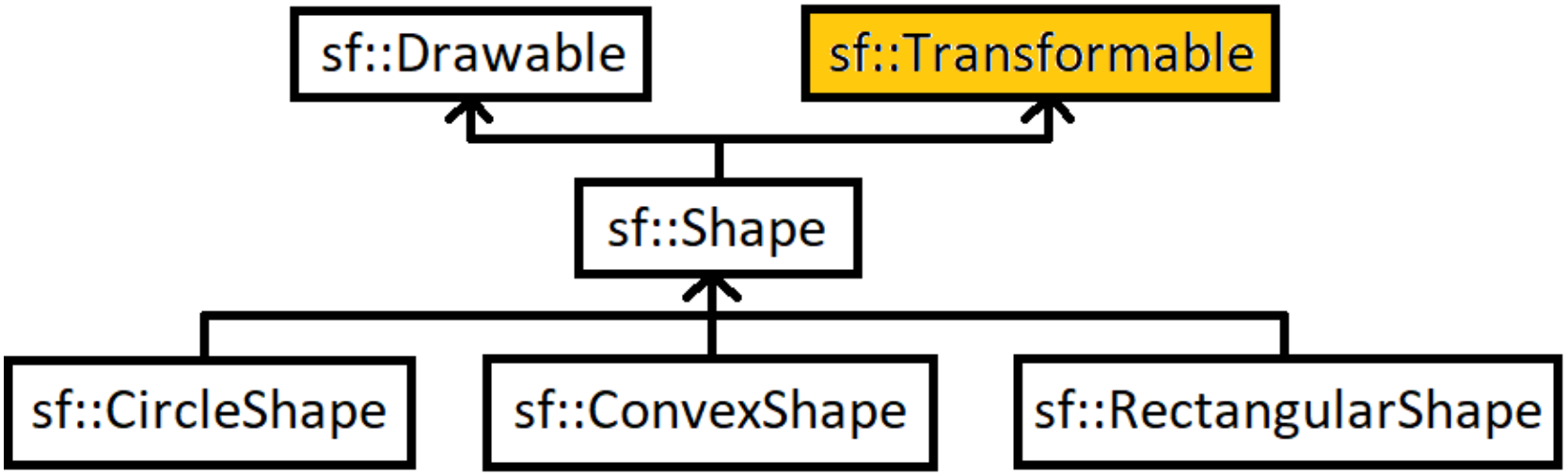
Kružnica

- crtanje samo obruba određene debljine
  $\rightarrow$ postavimo transparentnu ispunu oblika:

```
sf::CircleShape s(100.f);
s.setFillColor(sf::Color::Transparent);
s.setOutlineThickness(-10.f);
s.setOutlineColor(sf::Color::Red);
```

Tranformacija SFML objekata

- `sf::Tranformable` daje sučelje za transformacije

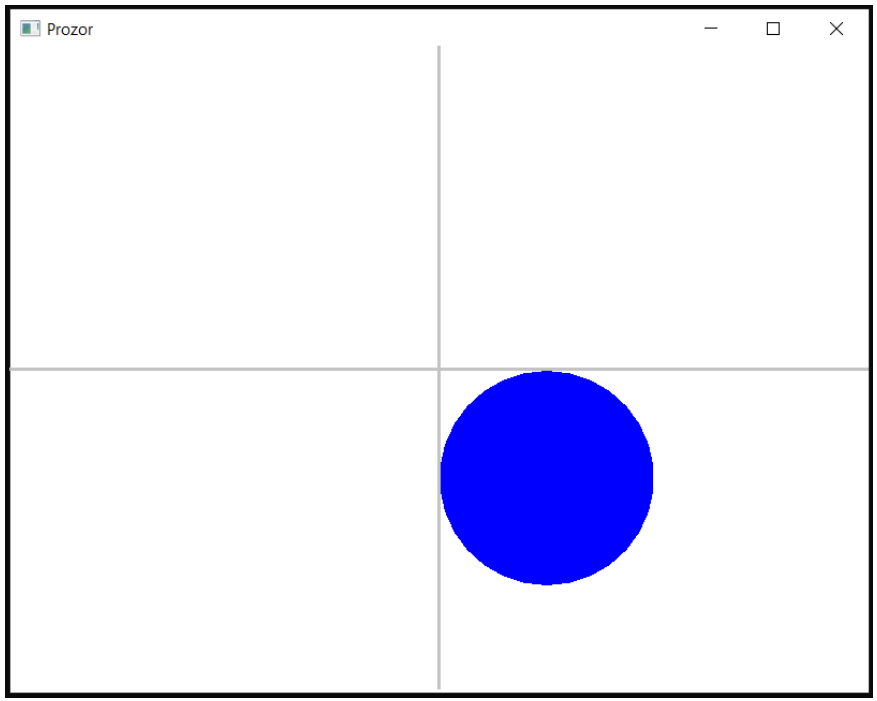


- četiri komponente transformacije (međusobno nezavisne):
  - pozicija
  - rotacija
  - skaliranje
  - ishodište

Postavljenje apsolutne pozicije objekta

Primjer. Za prozor dimenzije 800 $\times$ 600 želimo postaviti krug točno na sredinu tog prozora (tj. na koordinate 400 $\times$ 300):

```
prozor.clear(sf::Color::White);
sf::CircleShape s(100.f);
s.setFillColor(sf::Color::Blue);
s.setPosition(400, 300);
prozor.draw(s);
```

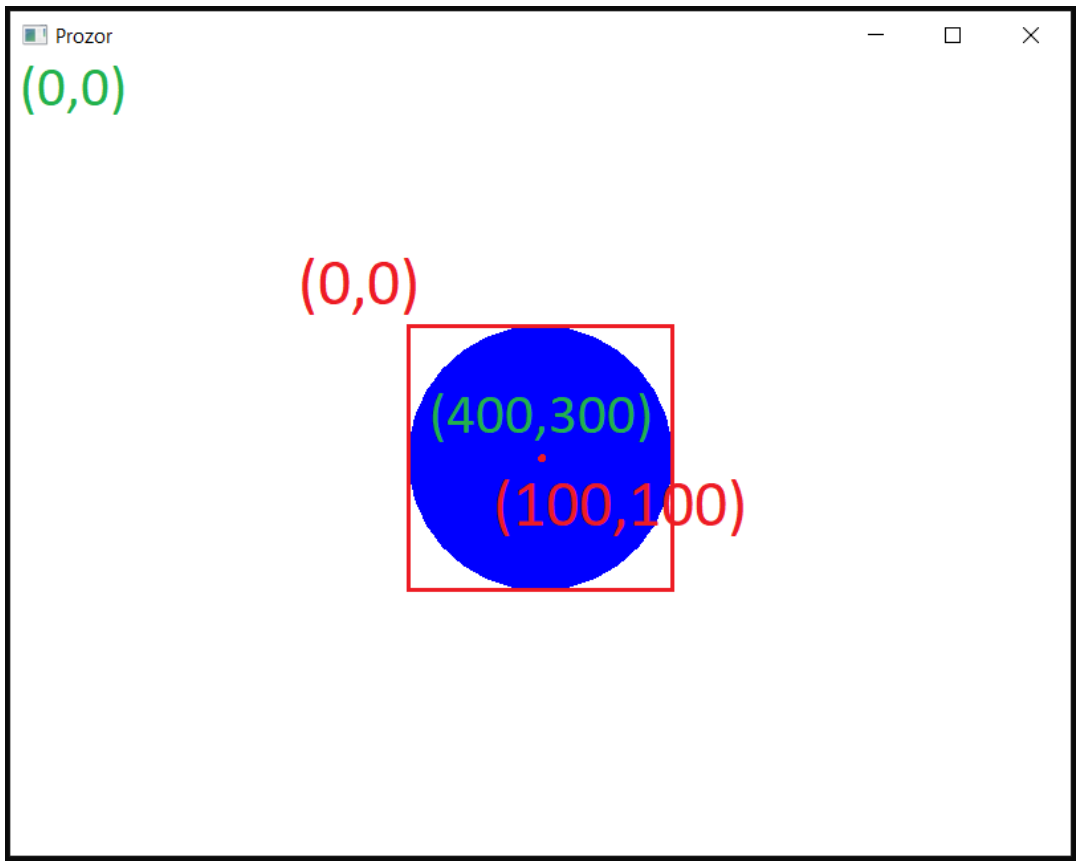


- po *defaultu*, objekti pozicionirani s obzirom na svoj lijevi gornji kut - zbog toga gornji kod nije postavio krug kako smo htjeli

Postavljanje ishodišta objekta

- po *defaultu* ishodište je gornji lijevi kut objekta: koordinate (0,0)
- možemo ga postaviti u središte objekta

```
sf::CircleShape s(100.f);
s.setFillColor(sf::Color::Blue);
s.setOrigin(100,100);
s.setPosition(400,300);
prozor.draw(s);
```



Dobivanje dimenzija objekta

- ako ne znamo radijus ili dimenzije, možemo koristiti `get...` funkcije
- prošli primjer mogli smo napisati ovako:

```
s.setOrigin(s.getRadius(),s.getRadius());
s.setPosition(prozor.getSize().x / 2.f,
              prozor.getSize().y / 2.f);
```

`set...` funkcije često imaju `get...` verzije

Primjeri.

`sf::Window`

- `void setPosition(const Vector2i &position)`
  `Vector2i getPosition() const`
- `void setSize (const Vector2u &size)`
  `Vector2u getSize () const`

`sf::CircleShape`

- `void setRadius (float radius)`
  `float getRadius() const`

`sf::RectangleShape`

- `void setSize (const Vector2f &size)`
  `const Vector2f& getSize() const`

... (nećemo ih navoditi sve i za sve klase)

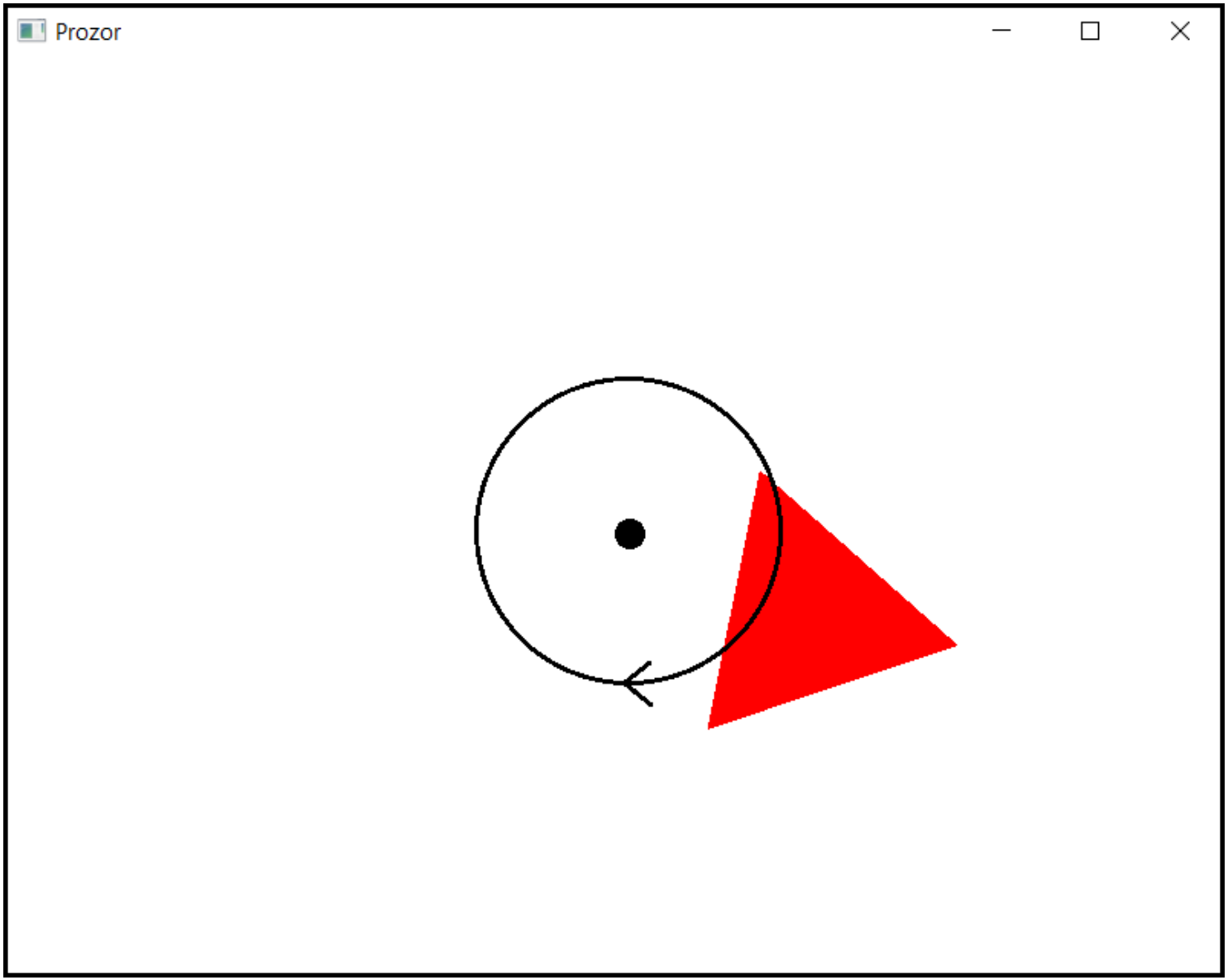
Rotacija

- **u stupnjevima**, **u smjeru kazaljke na satu**
  (ne obratno jer y-os u SFML-u pokazuje prema dolje)
- središnja točka za sve preostale transformacije je ishodište

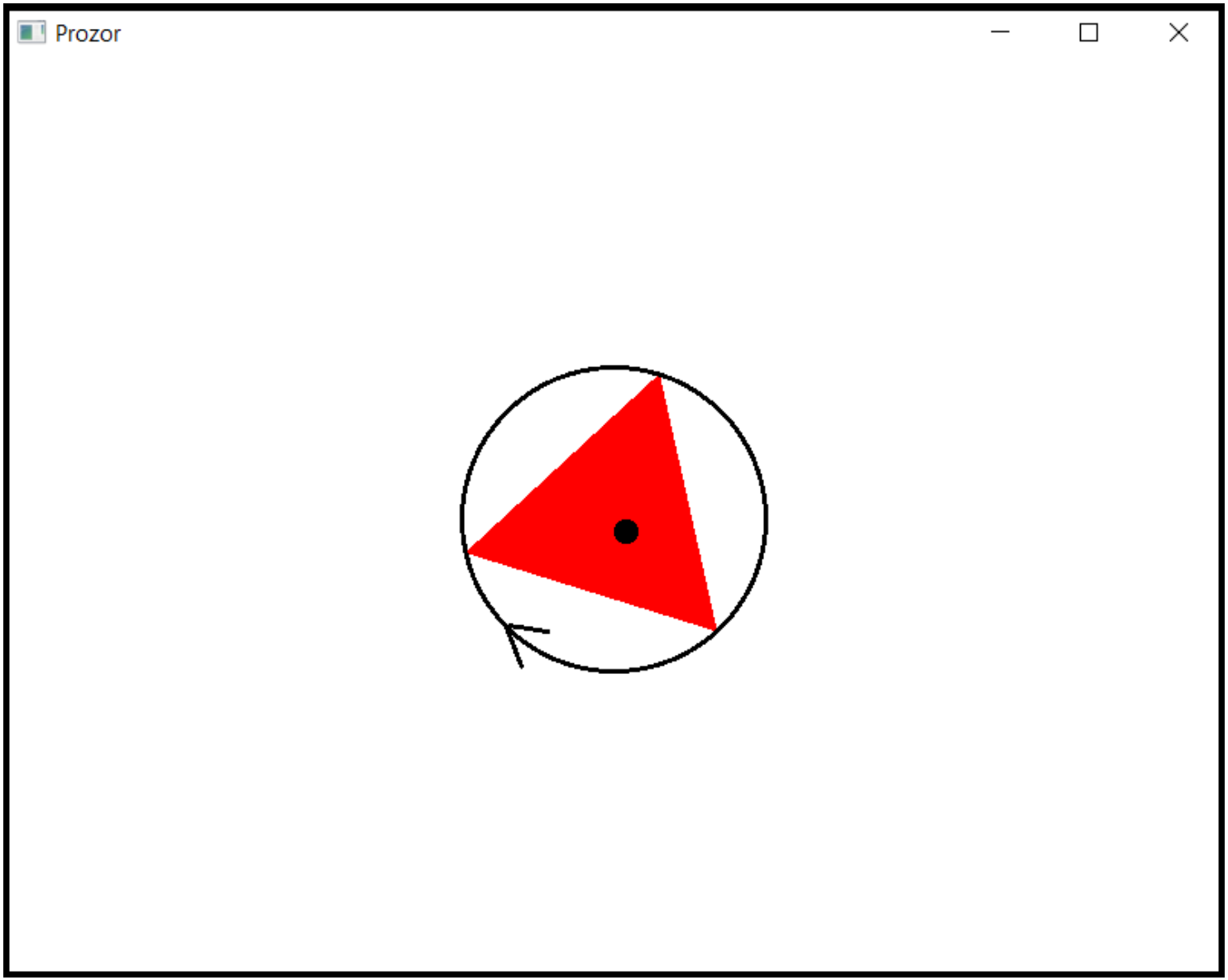
Primjer. Isprobati sa i bez zakomentirane linije (kod dostupan ovdje).

```
sf::CircleShape s(100.f,3);  //jednakostr. △
s.setFillColor(sf::Color::Red);
s.setPosition(400,300);
//s.setOrigin(100,100);
s.setRotation(1.f);
while(prozor.isOpen()) {
        ...
        prozor.clear(sf::Color::White);
        s.setRotation(s.getRotation() + 0.1f);
        prozor.draw(s);
        prozor.display();
}
```

Slika uz prethodni primjer (bez `setOrigin`)



Slika uz prethodni primjer (sa `setOrigin`)



Napomena o funkciji `getRotation`

- vraća broj u intervalu $[0, 360\rangle$ (veličina kuta)

Primjer. lako isto dobivamo pomoću:

```
s.setRotation(45.f - 360);
s.setRotation(45.f);
s.setRotation(45.f + 360);
```

sljedeće uvijek ispiše `45`:

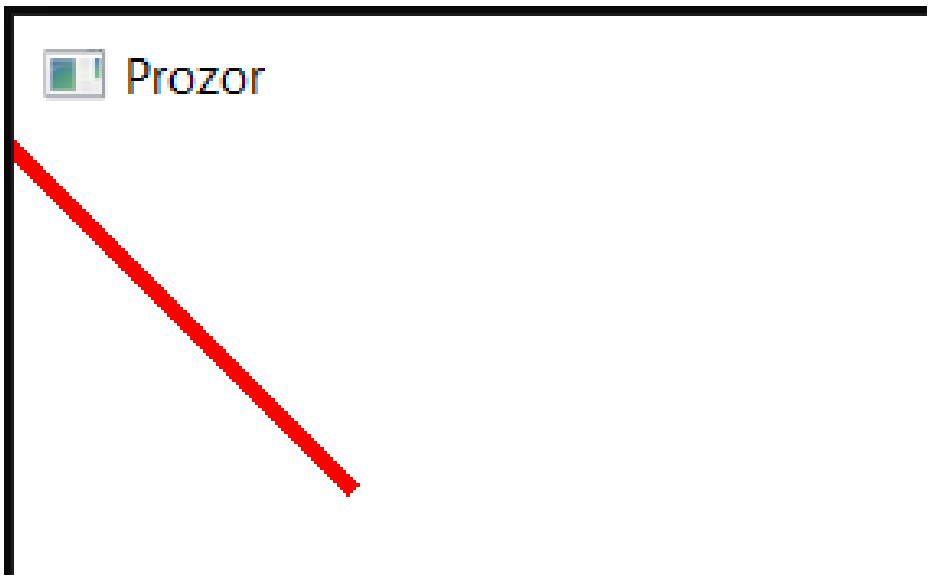
```
cout << s.getRotation() << endl;
```

Crtanje dužine određene debljine

- pomoću klase `sf::RectangleShape`

Primjer.

```
prozor.clear(sf::Color::White);
sf::RectangleShape linija(sf::Vector2f(150.f,5.f));
linija.setFillColor(sf::Color::Red);
linija.rotate(45.f);
prozor.draw(linija);
prozor.display();
```



Primjer. Promotrimo pobliže sliku dobivenu sljedećim kodom:

```
sf::ConvexShape s(3);
s.setFillColor(sf::Color::Blue);
s.setPoint(0, sf::Vector2f(50, 50));
s.setPoint(1, sf::Vector2f(600, 100));
s.setPoint(2, sf::Vector2f(200, 500));
prozor.draw(s);
```

Anti-aliasing

- kako bi dobili *anti-aliasing* oblike (tj. s glatkim rubovima) možemo to uključiti globalno pri stvaranju prozora

Primjer.

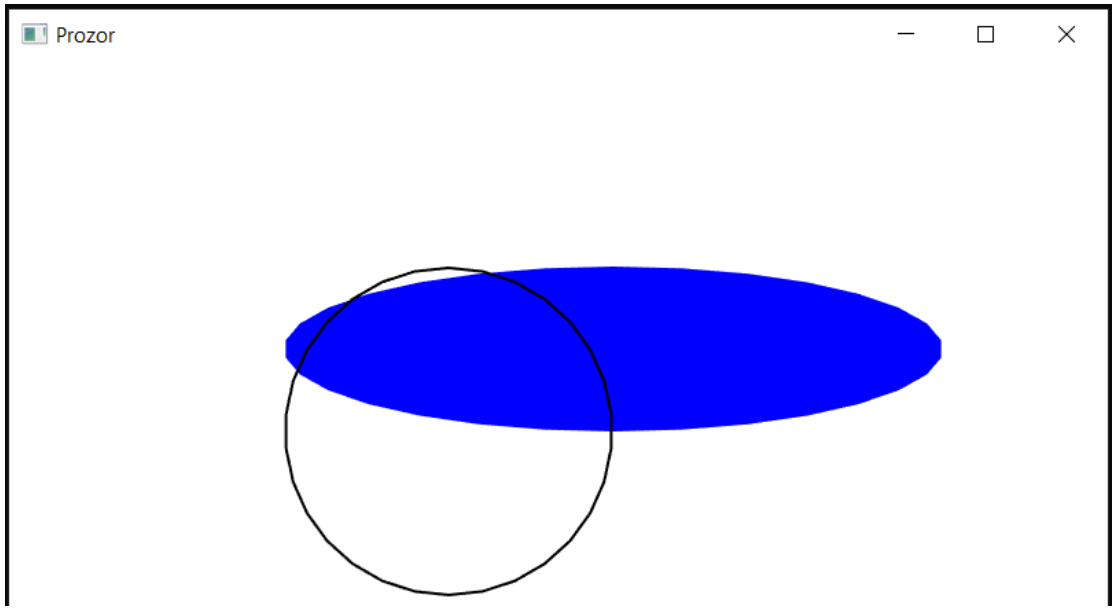
```
sf::RenderWindow prozor;
sf::ContextSettings postavke;
postavke.antialiasingLevel = 8;
prozor.create(sf::VideoMode(800,600),"Prozor",
                    sf::Style::Default,postavke);
```

Napomene. Ne možemo to postaviti samo za jedan oblik. Neke grafičke kartice mogu onemogućiti ili ne podržavati anti-aliasing.

Skaliranje

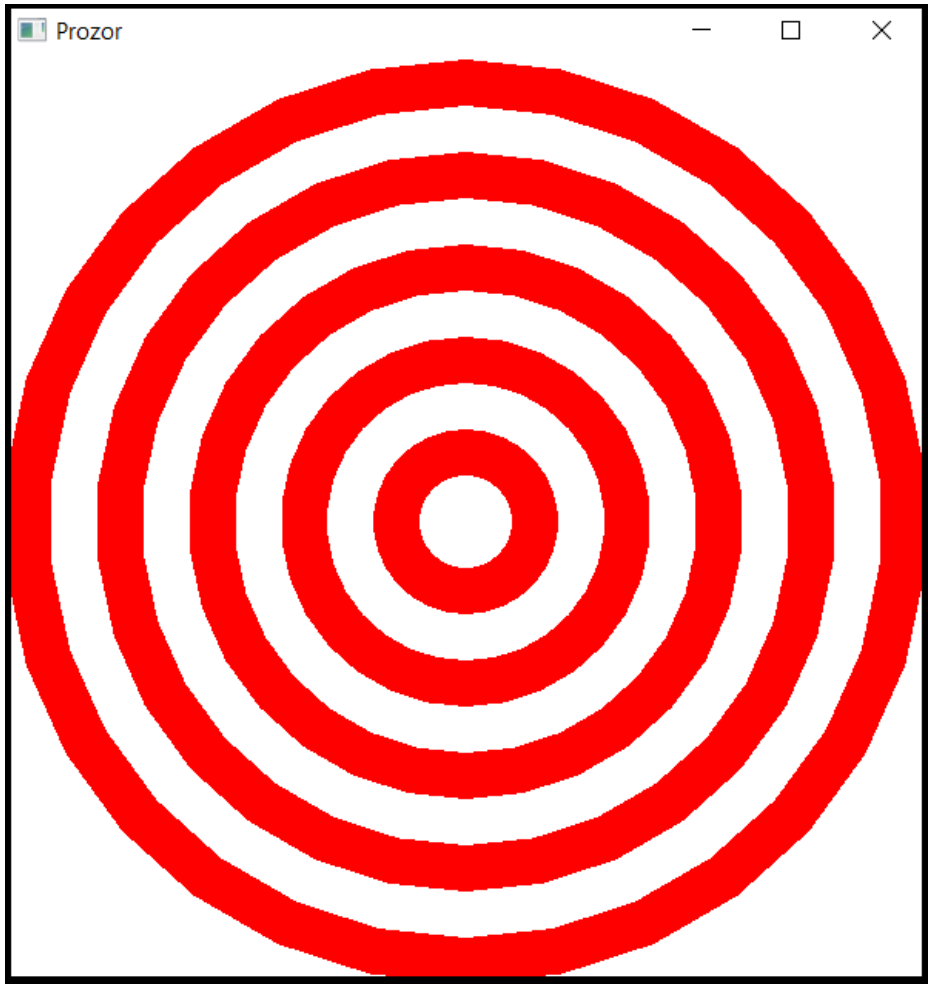
- po *defaultu* je 1 (i po x i po y osi)
- faktor skaliranja < 1 smanjuje objekt, a faktor > 1 ga povećava
- negativni faktori zrcale objekt

```
sf::CircleShape s(120);
s.setFillColor(sf::Color::Blue);
s.setPosition(200, 150);
s.setScale(2, 0.5);
prozor.draw(s);
```



Zadatak

Zadatak 2. Dobiti prikaz kao na sljedećoj slici koristeći skaliranje.



Rješenje (prikazan je dio unutar `main` funkcije)

```
sf::RenderWindow prozor;
prozor.create(sf::VideoMode(600,600),"Prozor");
sf::CircleShape c(300);
c.setPosition(prozor.getSize().x / 2.f,
              prozor.getSize().y / 2.f);
c.setOrigin(c.getRadius(), c.getRadius());
while (prozor.isOpen()) {
   ...  //while petlja za event (kao prije)
   prozor.clear(sf::Color::White);
   for (size_t i = 10; i >= 1; --i) {
      c.setFillColor((i % 2 == 0) ? sf::Color::Red :
                                    sf::Color::White);
      c.setScale(0.1 * i,0.1 * i);
      prozor.draw(c);
   }
   prozor.display();
}
```

Relativne transformacije

- funkcije `move`, `rotate`, `scale` pomiču, rotiraju i skaliraju objekt relativno obzirom na njega

Primjer.

(a) za `void sf::Transformable::move(float offsetX, float offsetY)` ekvivalentno je `objekt.move(a,b);` i

```
sf::Vector2f p = objekt.getPosition();
objekt.setPosition(p.x + a, p.y + b);
```

(b) za `void sf::Transformable::rotate(float angle)` ekvivalentno je `objekt.rotate(kut);` i

```
objekt.setRotation(objekt.getRotation() + kut);
```

(c) za `void sf::Transformable::scale(float factorX, float factorY)` ekvivalentno je `objekt.scale(a,b);` i

```
sf::Vector2f s = objekt.getScale();
objekt.setScale(s.x * a, s.y * b);
```

Napomena: dva argumenta ili jedan kao par

- neke funkcije koje zahtijevaju dva broja kao dva argumenta mogu primiti i par tih brojeva u jednom argumentu

Primjeri. Sljedeće funkcije imaju takve dvije verzije:

(1.)
```
void sf::Transformable::move(float x, float y)
void sf::Transformable::move(const Vector2f& p)
```

(2.)
```
void sf::Transformable::scale(float factorX,
                              float factorY)
void sf::Transformable::scale(const Vector2f&
                              factor)
```

(3.)
```
void sf::Transformable::setOrigin(float x,
                                  float y)
void sf::Transformable::setOrigin(const
                                  Vector2f& origin)
```

itd.

sf::Transformable::getTransform()

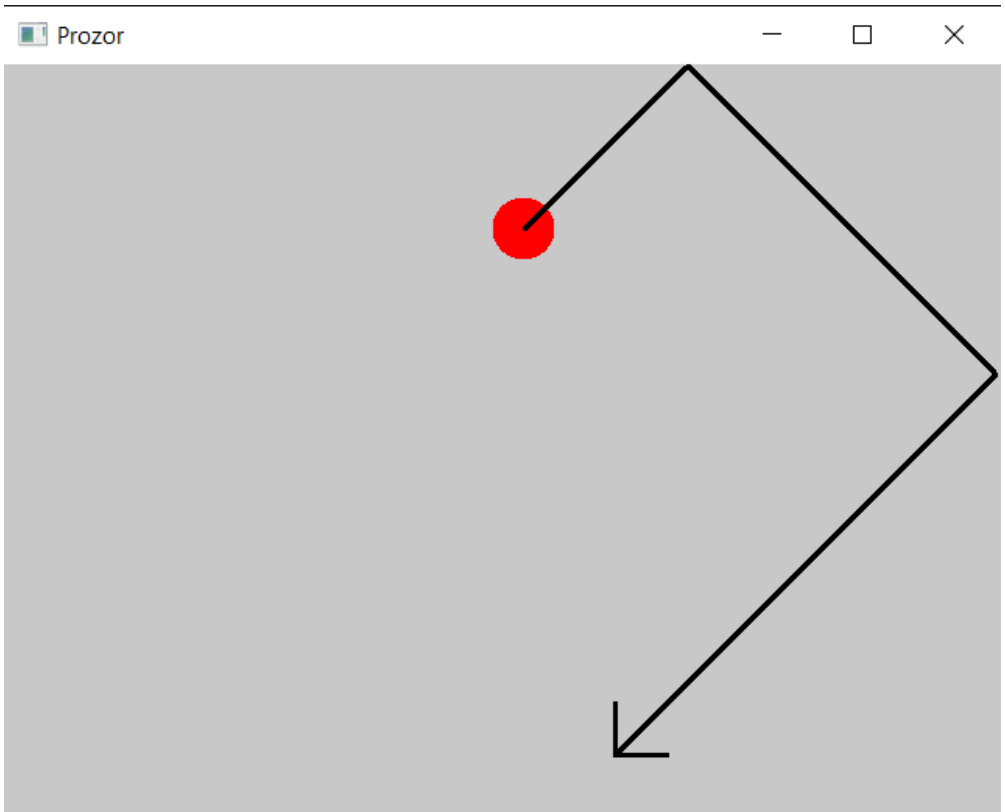
- Funkcija **sf::Transformable::getTransform()** vraća transformaciju tipa **sf::Transform**.
- **sf::Transform** interno sadrži 4x4 matricu transformacije.
- Može se koristiti npr. za transformiranje točaka pomoću funkcije `sf::Transform::transformPoint(float x, float y)` odnosno `sf::Transform::transformPoint(const Vector2f& point)`.
- Oblici neovisno pamte definiciju objekta i transformaciju, i svaki put kada treba nacrtati oblik izvode transformaciju. Zato npr. kod `sf::ConvexShape` funkcija **getPoint(i)** vraća originalne koordinate točke, bez transformacija. Ako želimo koordinate transformirane točke, trebamo pozvati

```
shape.getTransform().transformPoint(
        shape.getPoint(i))
```

Zadatak

Zadatak 3. Napraviti crveni krug radijusa 20 piksela koji se kreće po prozoru dimenzije 640×480 i odbija od njegovih rubova.

Prikaz (prikazan je i dio kretanja kruga nakon situacije sa slike):



Rješenje

```
sf::RenderWindow prozor(sf::VideoMode(640,480),
                        "Prozor");
float r = 20.f;
sf::CircleShape krug(r);
krug.setFillColor(sf::Color::Red);
krug.setOrigin(r,r);
sf::Vector2f pomak(0.1f, 0.1f);
while (prozor.isOpen()) {
    sf::Event event;
    while (prozor.pollEvent(event)) {
        if (event.type == sf::Event::Closed) {
            prozor.close();
        }
    }
```

Rješenje (nastavak `while` petlje s prethodnog slajda)

```
  if((krug.getPosition().x + r > prozor.getSize().x
        && pomak.x > 0) ||
     (krug.getPosition().x - r < 0 && pomak.x < 0)){
           pomak.x = -pomak.x;
  }
  if((krug.getPosition().y + r > prozor.getSize().y
        && pomak.y > 0) ||
     (krug.getPosition().y - r < 0 && pomak.y < 0)) {
           pomak.y = -pomak.y;
  }
  krug.move(pomak);
  prozor.clear(sf::Color(200, 200, 200, 255));
  prozor.draw(krug);
  prozor.display();
}
```

Napomena

- u kodu s prethodnog slajda mogli smo umjesto

```
krug.move(pomak);
```

napisati

```
krug.setPosition(krug.getPosition() + pomak);
```

- uočite da se u uokvirenom dijelu zbrajaju parovi!

Brzina

- Brzina kruga ovisi o brzini programa, što ovisi o performansama računala i eventualnim pozadinskim procesima.
- Za dosljedno određivanje brzine moramo uvesti mjerenje vremena:

```
sf::Clock clock;
float deltaTime = clock.restart().asSeconds();
```

Zadatak 4. Kao u prošlom zadatku, ali osigurajte da se krug kreće određenom brzinom neovisno o računalu.

Rješenje

```
  ...
sf::Vector2f brzina(500, 500); // uzmjesto pomak
sf::Clock clock;
while (prozor.isOpen()) {
    ...
    float deltaTime = clock.restart().asSeconds();
    krug.move(brzina * deltaTime);
    ...
    }
```