SFML - Prozori i crtanje oblika

Objektno programiranje - 2. vježbe

dr. sc. Sebastijan Horvat

Prirodoslovno-matematički fakultet,
Sveučilište u Zagrebu

19. ožujka 2025. godine

Zadatak

Zadatak. Napisati program koji početno otvara jedan prozor. Kad korisnik odluči zatvoriti neki od otvorenih prozora, otvoriti još jedan prozor. Kad ukupno bude otvoreno 5 prozora, na zahtjev korisnika za zatvaranje bilo kojeg od njih, zatvoriti ih sve (i tako završiti program). U naslovnim trakama prozora treba pisati `"Prozor"` + redni broj tog prozora.

Rješenje (1. dio)

- imat ćemo (najviše) 5 prozora, pa možemo koristiti polje `prozori` od 5 prozora
- kako bi znali koliko je od tih 5 prozora otvoreno, koristit ćemo varijablu brojač
- početno će biti otvoren samo jedan prozor (`prozor[0]`)

```
sf::Window prozori[5];
size_t brojac = 1;
prozori[0].create(sf::VideoMode(800,600),
          string("Prozor")+to_string(brojac));
```

Rješenje (2. dio)

- uočiti: početni prozor (`prozor[0]`) je otvoren od početka pa sve dok se ne krene zatvarati peti po redu otvoreni prozor

$\Rightarrow$ program se zapravo izvršava dok je prvi prozor otvoren

- `for` petljom prolazimo po svim otvorenim prozorima `prozor[i]` (takvih prozora ima u svakom trenutku ukupno `brojac`)
- za svaki otvoreni prozor, pogledamo njegove događaje (svaki prozor ima svoj red događaja koji su se nad njim dogodili!)

```
while (prozori[0].isOpen()) {
    sf::Event d;
    for(size_t i = 0; i < brojac; ++i)
        while (prozori[i].pollEvent(d)) {
            ...  kod sa sljedećeg slajda ...
        }
}
```

Rješenje (3. dio - kod unutar `while` petlje)

- ne zanimaju nas svi događaji, nego samo zahtjevi za zatvaranje prozora
- u slučaju takvog događaja, ako još nije otvoreno 5 prozora, otvaramo novi prozor
- inače je potrebno zatvoriti sve otvorene prozore (podsjetnik: zatvaranjem početnog prozora, završava vanjska `while` petlja i program će završiti)

```
if(d.type == sf::Event::Closed) {
  if(brojac < 5)
      prozori[brojac-1].create
        (sf::VideoMode(800,600),
         string("Prozor")+to_string(++brojac));
  else for(size_t j = 0; j < brojac; ++j)
         prozori[j].close();
}
```

Funkcije koje primaju prozore

- Primjer: želimo napisati funkciju `stvori_prozor` koji će otvoriti dobiveni prozor (i u naslovnoj traci ispisati `Prozor` i njegov redni broj)

```
...
size_t brojac = 1;
stvori_prozor(prozori[0], brojac);
while (prozori[0].isOpen()) {
    ...
    if (brojac < 5) {
        ++brojac;
        stvori_prozor(prozori[brojac-1], brojac);
    }
    ...
```

Prvi pokušaj takve funkcije i greška

```
void stvori_prozor(sf::Window prozor, int brojac){
     prozor.create(sf::VideoMode(800, 600),
          string("Prozor") + to_string(brojac));
}
```

- međutim, javlja se sljedeća greška (prikazan je dio prikaza u VS2022):

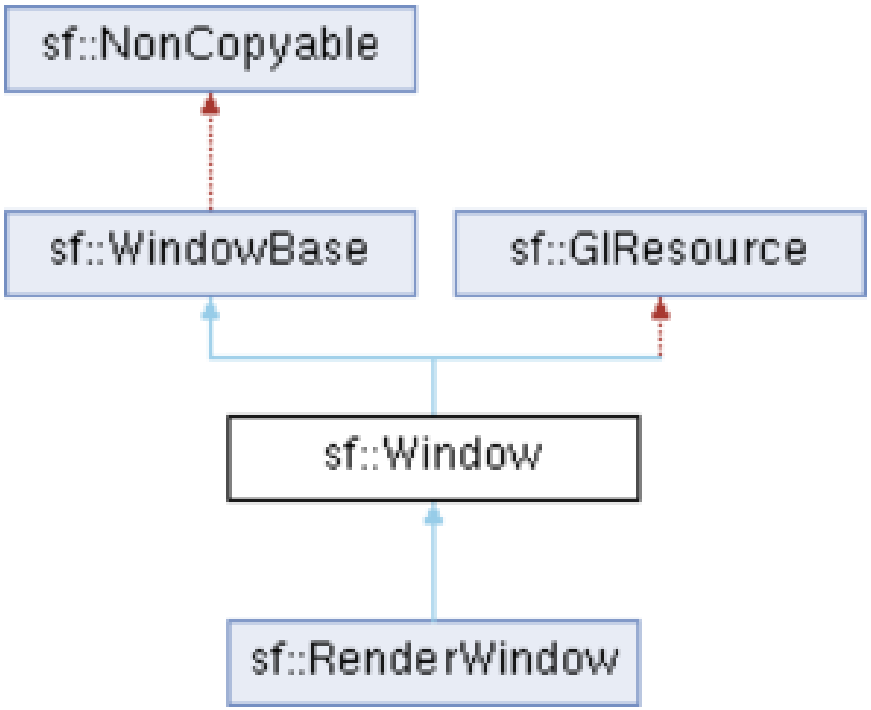
```
stvori_prozor(prozori[0], brojac);
```

function "sf::Window::Window (const sf::Window &)" (declared implicitly) cannot be referenced -- it is a deleted function

Objašnjenje - prozori se ne mogu kopirati

- s obzirom da su prozori izvedeni iz `sf::NonCopyable` nije ih moguće kopirati
- posljedica: funkcije umjesto prozora moraju primiti referencu ili pokazivač na njega

$\Rightarrow$ rješenje našeg problema:




```
void stvori_prozor(sf::Window &prozor,int brojac){
        ...
}
```

Klasa `sf::NonCopyable`

- klasa koja služi tome da svaka klasa koja je nasljeđuje nema mogućnost kopiranja
- to je postignuto na način da su konstruktor kopiranja i operator pridruživanja kopiranjem privatni

```
namespace sf {
    class SFML_SYSTEM_API NonCopyable {
    protected:
        NonCopyable() {}
        ~NonCopyable() {}
    private:
        NonCopyable(const NonCopyable&);
        NonCopyable& operator =(const NonCopyable&);
    };
}
```

- napomena: kako klasa ima konstruktor kopiranjem, kompajler neće automatski generirati *defaultni* konstruktor - zato je on eksplicitno definiran

Upotreba `sf::NonCopyable` za vlastite klase

- jedna od stvari o kojoj treba razmisliti prije pisanja vlastite klase jest treba li se ona moći kopirati
- kako bi stvorili klasu koja se ne može kopirati, dovoljno je naslijediti klasu `sf::NonCopyable`

Primjer:

```
class tocka : sf::NonCopyable {
public:
    int x, y;
    tocka(int a, int b)
        :  x(a), y(b) { }
};
void ispis(tocka T) {
    cout << T.x << "," << T.y << endl;
}
...
tocka A = { 1,2 };
ispis(A);    ✗
```

Koristili `sf::Event::Closed` - ostali događaji:

`sf::Event::`

- Resized
- LostFocus
- GainedFocus
- TextEntered
- KeyPressed
- KeyReleased
- MouseWheelScrolled
- MouseButtonPressed
- MouseButtonReleased
- MouseMoved
- MouseEntered
- MouseLeft
- JoystickButtonPressed
- JoystickButtonReleased
- JoystickMoved
- JoystickConnected
- JoystickDisconnected

Više informacija o pojedinom događaju na linku: link
Nekima od njih bavit ćemo se kasnije.

Napomena. `MouseWheelMoved` je od SFML-a 2.3. zamijenjen s `MouseWheelScrolled`.

Promjena postavki prozora

Promjena naslova prozora

```
prozor.setTitle("Novi naslov");
```

Promjena veličine prozora

```
prozor.setSize(sf::Vector2u(200,400));
```

- za manipulaciju dvodimenzionalnim vektorima (s dvije koordinate x i y): parametrizirana klasa `sf::Vector2<T>`
- tu klasu koristimo za objekte s dvije dimenzije - veličine, točke, brzine, ...
- napomena: za 3-dimenzionalne vektore postoji parametrizirana klasa `sf::Vector3<T>`

Implementacija klase `Vector2`

```
namespace sf {
    template <typename T>
    class Vector2 {
    public:
        Vector2();
        Vector2(T X, T Y);
        template <typename U>
        explicit Vector2(const Vector2<U>& vector);
        T x;
        T y;
    };
...
```

- koristimo najčešće specijalizacije:

```
typedef Vector2<int>          Vector2i;
typedef Vector2<unsigned int> Vector2u;
typedef Vector2<float>        Vector2f;
```

Operacije klase `Vector2`

- unarni minus

```
template <typename T>
Vector2<T> operator -(const Vector2<T>& right);
```

- operator += (slično za -=)

```
template <typename T>
Vector2<T>& operator +=(Vector2<T>& left,
                        const Vector2<T>& right);
```

- operator + (slično za -)

```
template <typename T>
Vector2<T> operator +(const Vector2<T>& left,
                      const Vector2<T>& right);
```

Operacije sa skalarom i usporedbe

- množenje skalarom (slično za dijeljenje skalarom / i /=):

```
template <typename T>
Vector2<T> operator *(const Vector2<T>& left,
                      T right);

template <typename T>
Vector2<T> operator *(T left,
                      const Vector2<T>& right);

template <typename T>
Vector2<T>& operator *=(Vector2<T>& left,
                        T right);
```

- provjera jednakosti (slično za !=)

```
template <typename T>
bool operator ==(const Vector2<T>& left,
                 const Vector2<T>& right);
```

Primjer upotrebe klase `Vector2f`

```
void ispis(sf::Vector2f &v) {
    cout << "(" << v.x << "," << v.y << ")" << endl;
}

int main() {
    sf::Vector2f v1(16.5f, 24.f);
    v1.x = 18.2f;
    ispis(v1);      //(18.2,24)
    sf::Vector2f v2 = v1 * 2.f;
    ispis(v2);      //(36.4,48)
    sf::Vector2f v3;
    v3 = v1 + v2;
    if(v2 != v3)
        ispis(v3);  //(54.6,72)
    return 0;
}
```

Pozicija prozora

`void setPosition(const Vector2i &position)`

- promjena pozicije prozora

`Vector2i sf::Window::getPosition() const`

- daje poziciju prozora (u pikselima)

Primjer.

```
prozor.setPosition(sf::Vector2i(40,100));
auto p = prozor.getPosition();
cout << "Pozicija prozora:  (" << p.x
     << "," « p.y « ")" << endl;
```

Još o mogućnostima rada s prozorima: link

Crtanje - posebna klasa prozora

`sf::RenderWindow`

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

Primjer. U trenutnom kodu potrebne su minimalne promjene:

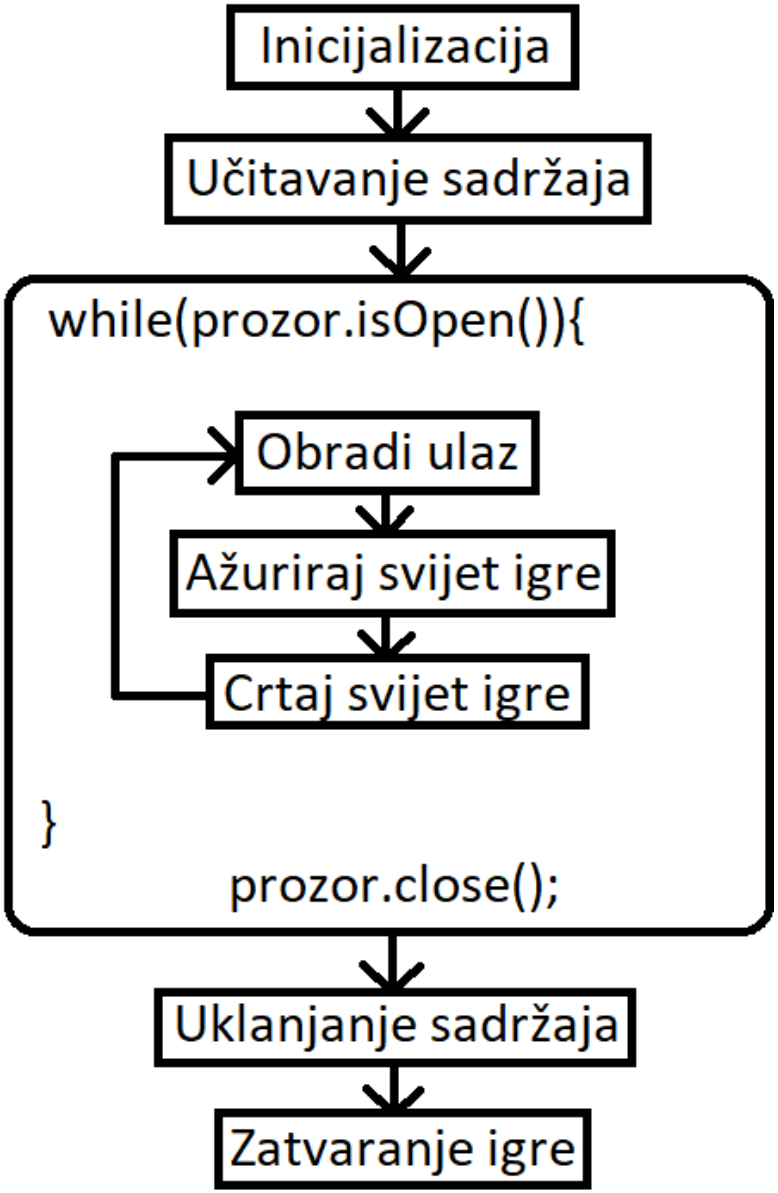
```
#include <SFML/Graphics.hpp>
...
  sf::RenderWindow prozor;
  prozor.create(sf::VideoMode(800,600),"Prozor");
  while (prozor.isOpen()) {
      sf::Event d;
      while (prozor.pollEvent(d)) {
          if (d.type == sf::Event::Closed) {
              prozor.close();
          }
      }
  }
```

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>
```

Dijagram tipične igre



Čisti/crtaj/prikaži ciklus

- ***double-buffering*** metoda - standardna u igrama
- 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)`

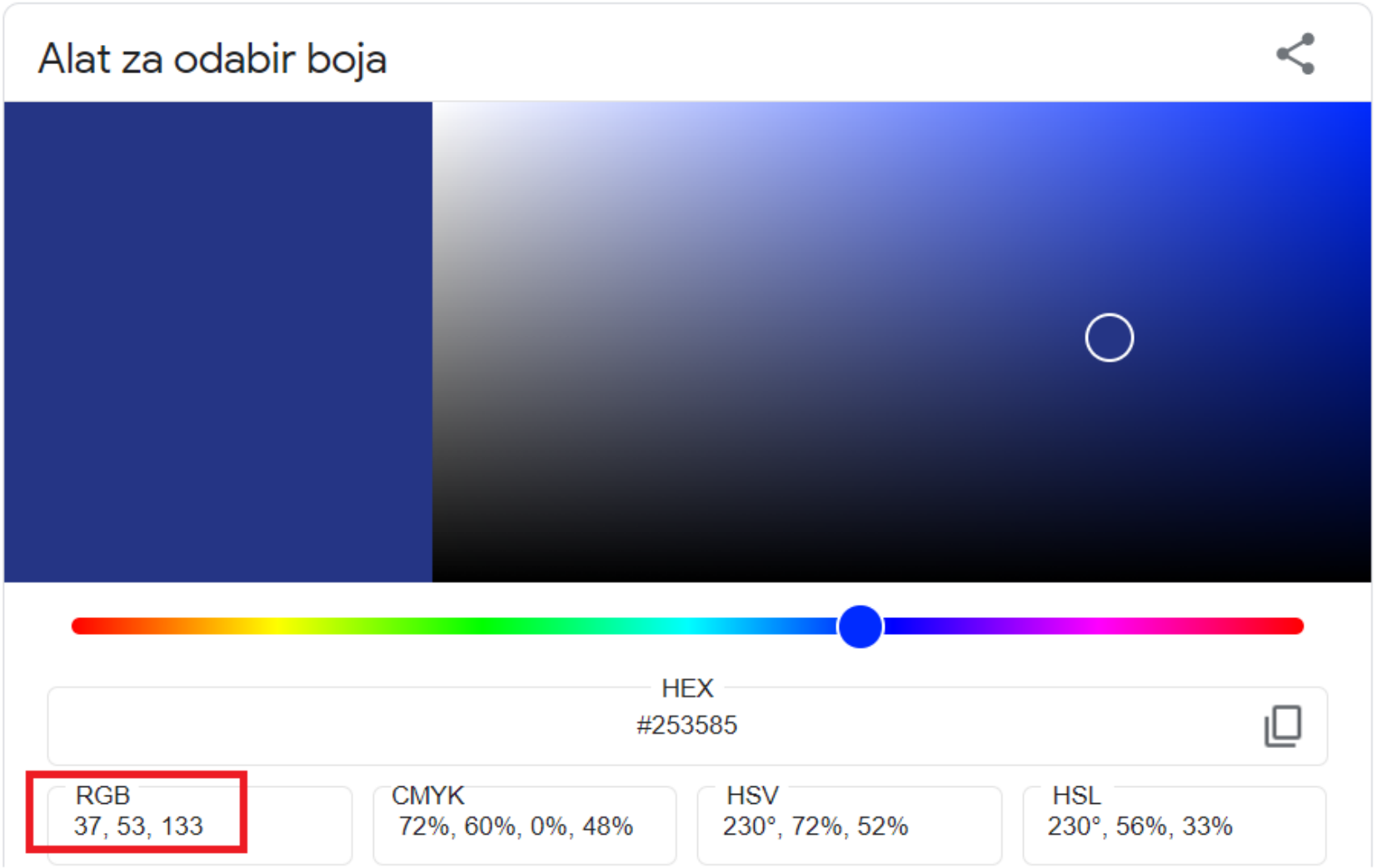
- parametar je 32-bitni nenegativni cijeli broj koji sadrži RGBA komponente (u tom redoslijedu)

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`

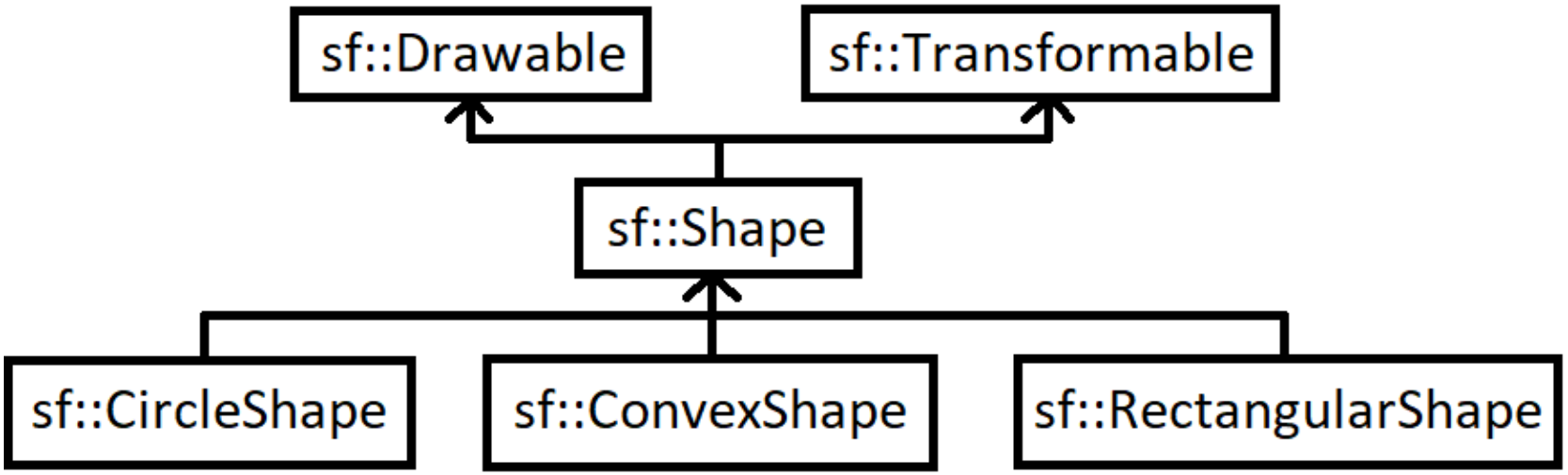
Boje pomoću RGB koda

- 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:



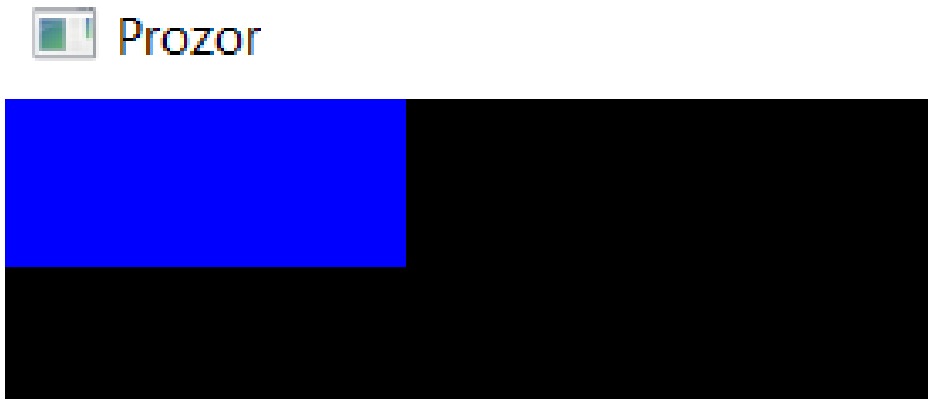
- `sf::Shape` je **apstraktna klasa** ($\Rightarrow$ mora se specijalizirati konkretnim oblikom)
- **`sf::Drawable`** - objekti koji se mogu crtati na ekran
- **`sf::Transformable`** - pruža funkcionalnosti pomicanja, skaliranja i rotiranja objekta

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:

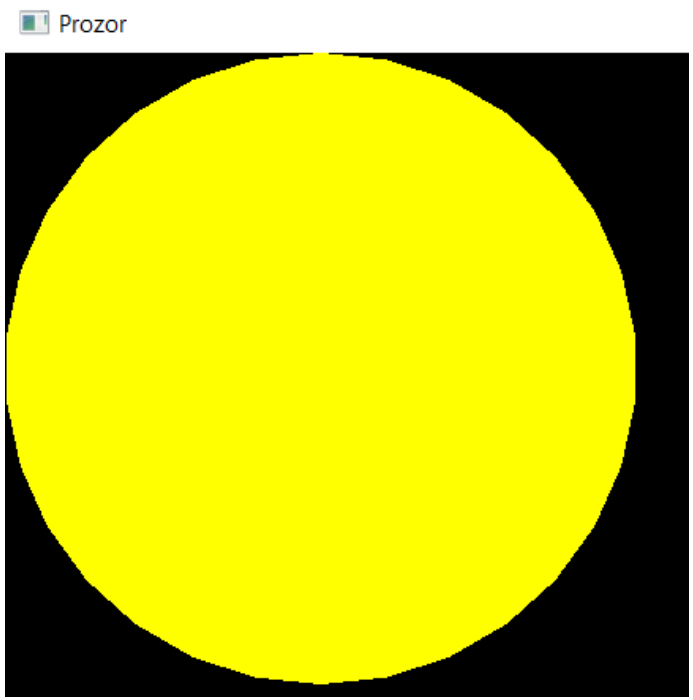
```
sf::RectangleShape p;
p.setSize(sf::Vector2f(120.f, 50.f));
```

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:

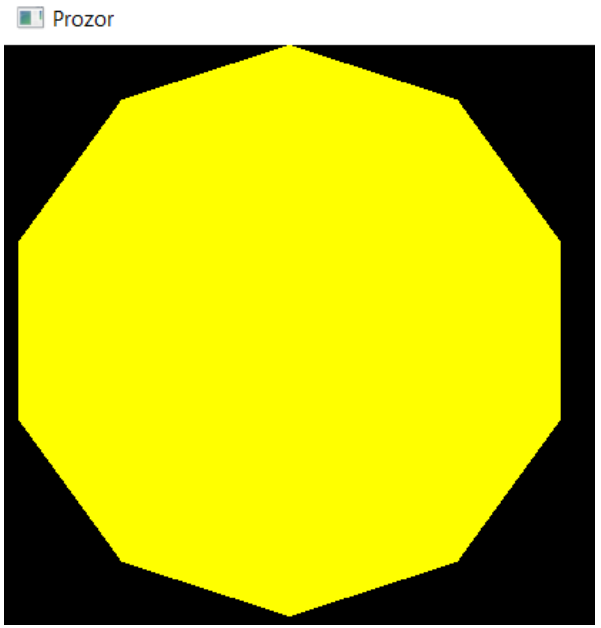
```
CircleShape(float radius=0,std::size_t pointCount=30)
```

- 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

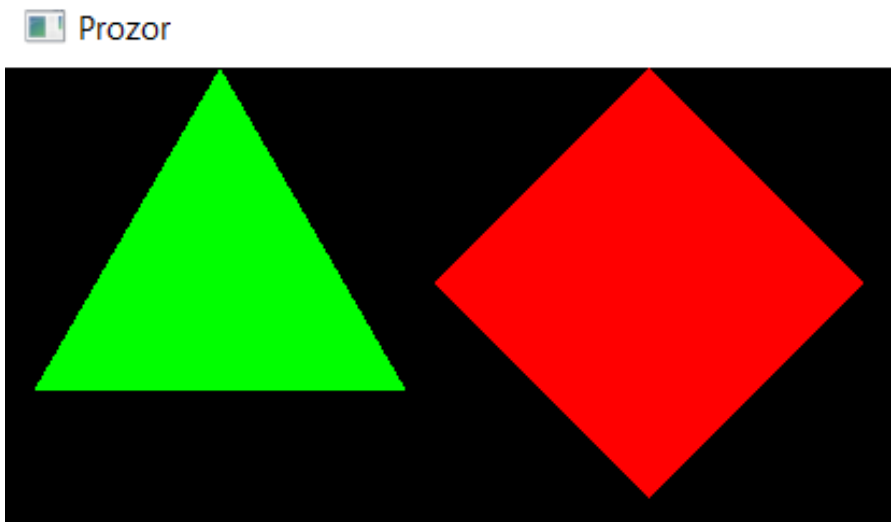


Pravilni mnogokuti

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

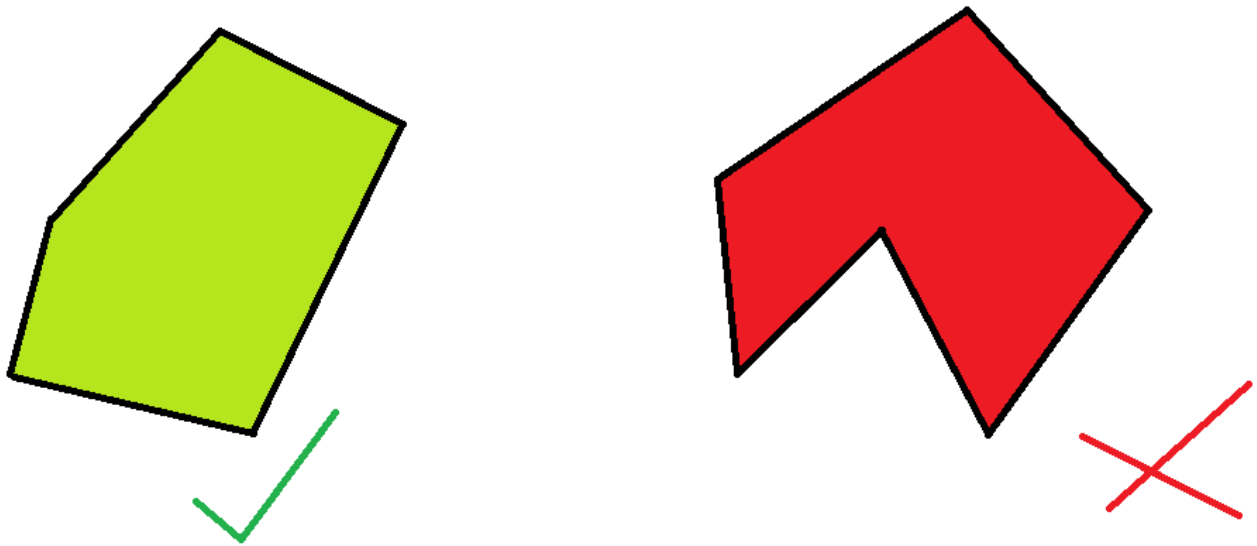
Primjer. Jednakostranični trokut i kvadrat:

```
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 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)`

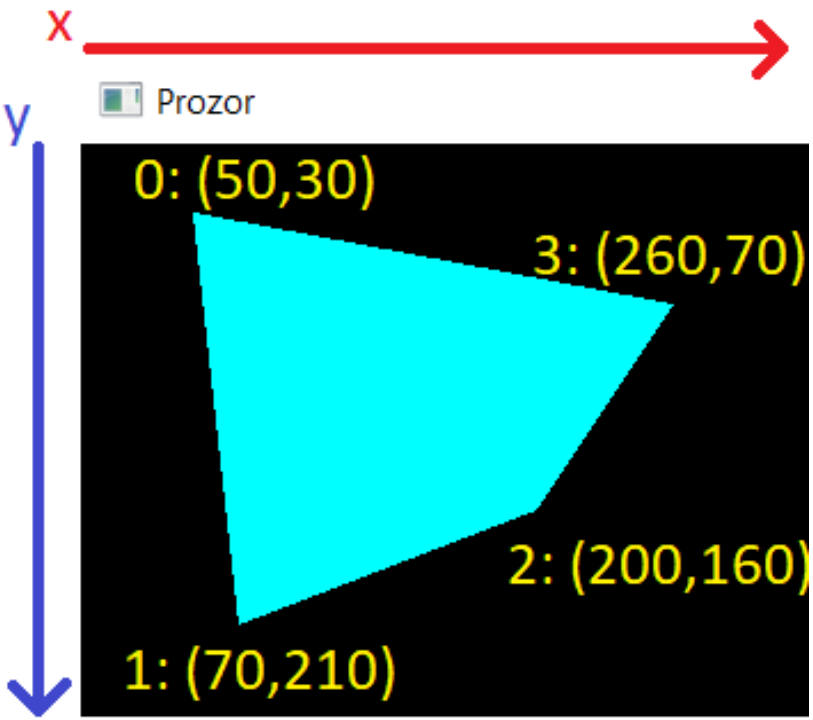
- konstruktor prima broj točaka mnogokuta (za valjani oblik treba biti $\geqslant 2$ točaka)
- funkcija `getPointCount()` vraća postavljeni broj točaka
- točke se nakon toga navode **po redu** (u smjeru kazaljke na satu ili u smjeru obratnom kazaljci na satu)
- **indeks mora biti u rasponu** $[0, \texttt{getPointCount()}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

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:

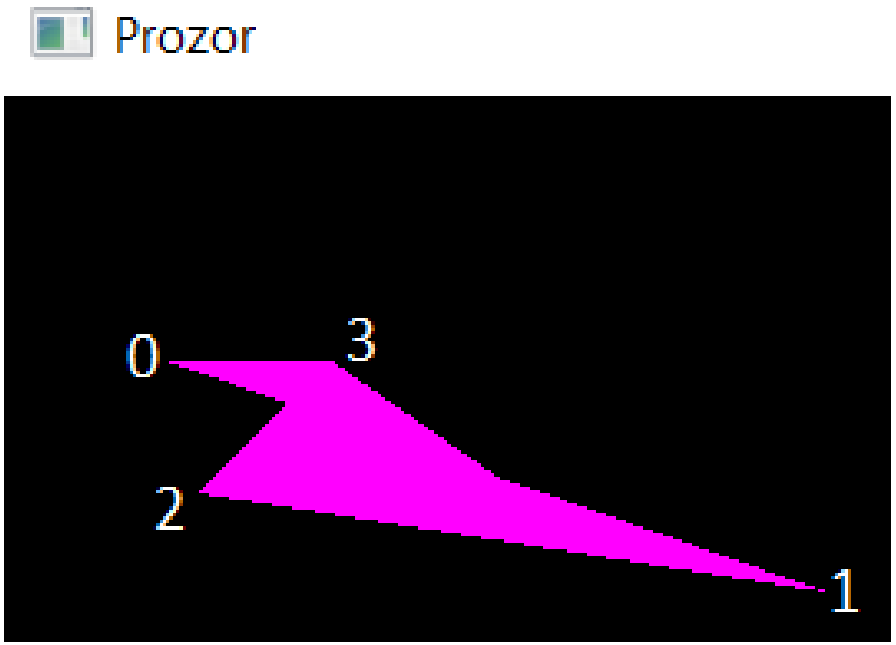
```
sf::ConvexShape p;
p.setPointCount(4);
```



Konveksni oblici - još jedan primjer

Pitanje. Što nije u redu u sljedećem primjeru?

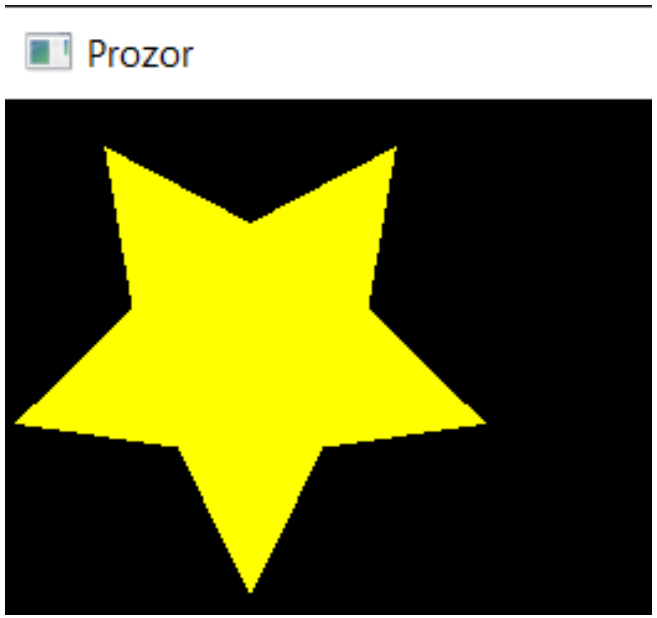
```
sf::ConvexShape p(4);
p.setFillColor(sf::Color::Magenta);
p.setPoint(0,sf::Vector2f(50,80));
p.setPoint(1,sf::Vector2f(250,150));
p.setPoint(2,sf::Vector2f(60,120));
p.setPoint(3,sf::Vector2f(100,80));
```



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. 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>`