

# SFML - Crtanje

## Objektno programiranje - 8. vježbe (3. dio)

Sebastijan Horvat

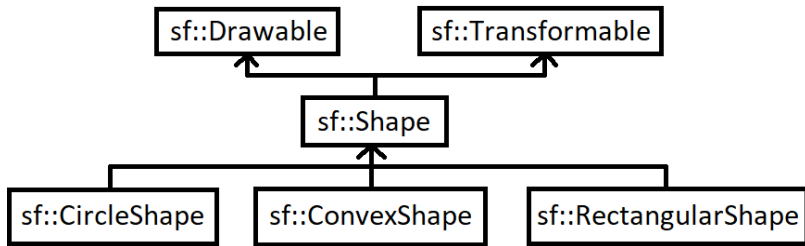
Prirodoslovno-matematički fakultet,  
Sveučilište u Zagrebu

8. svibnja 2024. godine



# Crtanje oblika - podsjetnik na slajd prošle prezentacije

- 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

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

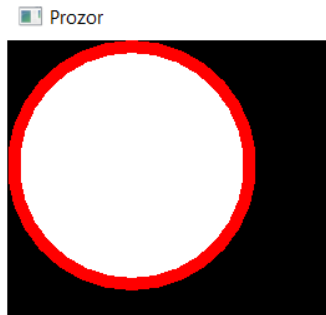
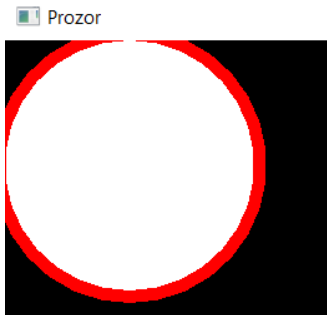
## Primjer.

```
prozor.clear(sf::Color::Black);  
sf::CircleShape s(100.f);  
s.setFill(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)

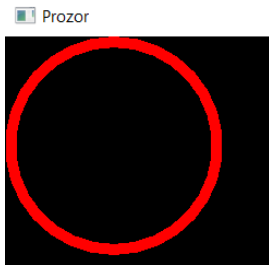


- ako želimo da obrub bude dio oblika, postavimo mu negativnu debljinu (slika desno; ukupan radijus i dalje je 100 piksela):

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

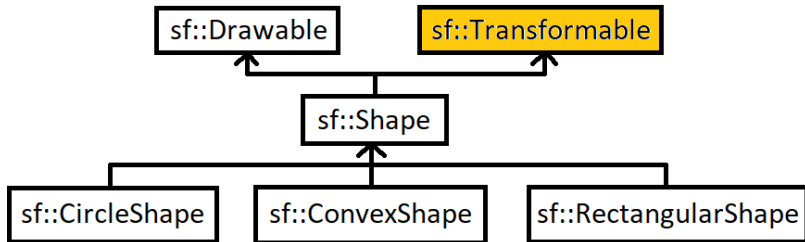
- crtanje samo obruba određene debljine  
→ postavimo transparentnu ispunu oblika:

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



# Transformacija SFML objekata

- `sf::Transformable` daje sučelje za transformacije

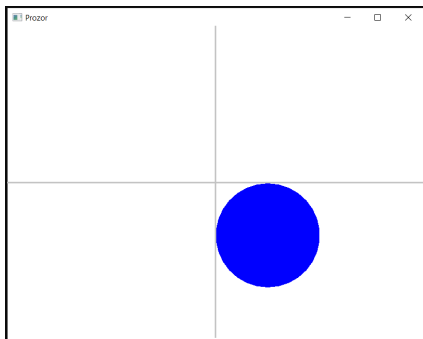


- četiri komponente transformacije (međusobno nezavisne):
  - pozicija
  - rotacija
  - skaliranje
  - ishodiše

# 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);
```

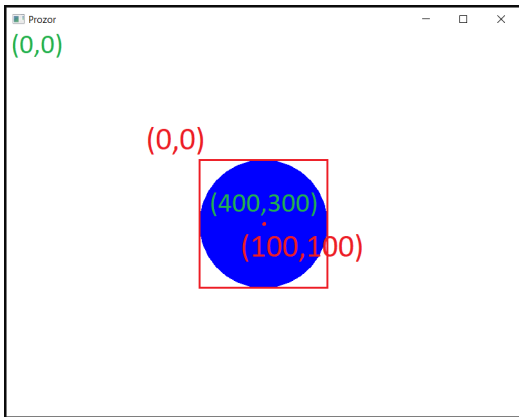


- po *defaultu*, objekti pozicionirani s obzirom na svoj lijevi gornji kut - zbog toga gornji kod nije postavio krug kako smo htjeli

# 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

- ukoliko 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);
```

## 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`

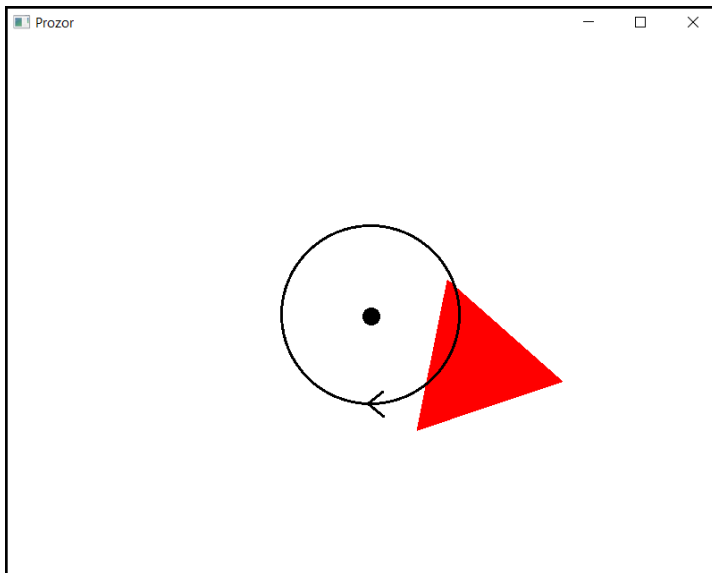
... (nećemo ih navoditi sve i za sve klase)

- 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

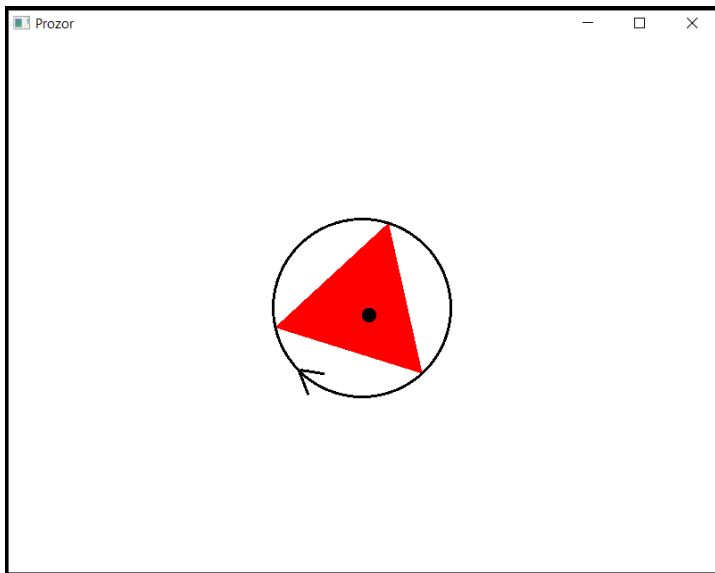
**Primjer.** Isprobati sa i bez zakomentirane linije (kod dostupan [ovdje](#)).

```
sf::CircleShape s(100.f, 3); //jednakostr.  $\Delta$ 
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`)



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

**Primjer.** Iako isto dobivamo pomoću:

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

sljedeće uvijek ispiše 45:

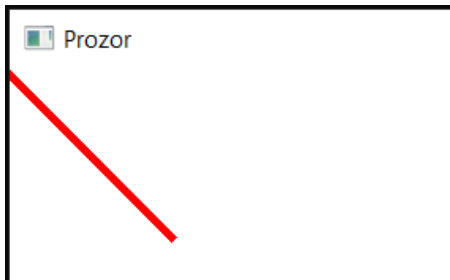
```
cout << s.getRotation() << endl;
```

# Crtanje dužine određene debljine

- pomoću klase `sf::RectangleShape`

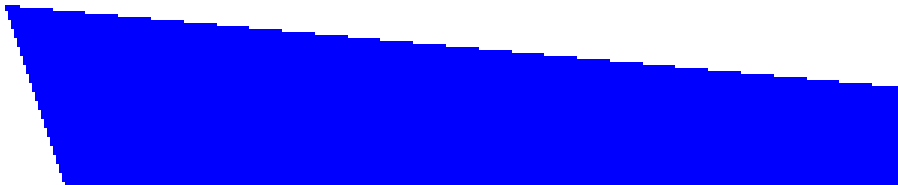
## Primjer.

```
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 поближе 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.

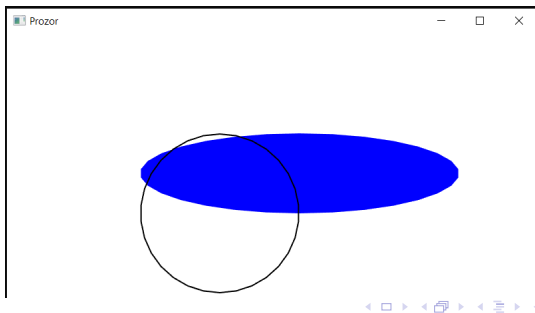
```
sf::RenderWindow prozor;  
sf::ContextSettings postavke;  
postavke.antialiasingLevel = 8;  
prozor.create(sf::VideoMode(800, 600), "Prozor",  
             sf::Style::Default, postavke);
```

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

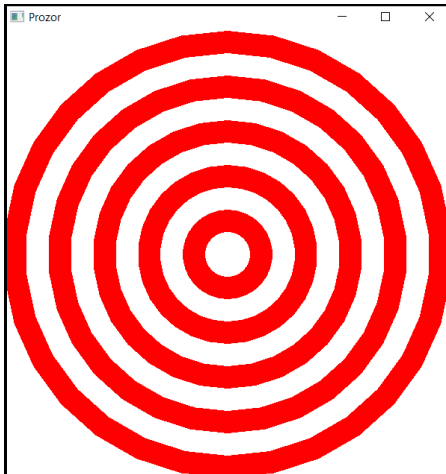
# Skaliranje

- 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.** 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 mogu koje zahtijevaju dva argumenta kao dva broja mogu primiti i jedan argument kao par tih brojeva

**Primjeri.** Sljedeće funkcije imaju takve dvije verzije:

- (1.) 

```
void sf::Transformable::move(float x, float y)
void sf::Transformable::move(const Vector2f& p)
```
- (2.) 

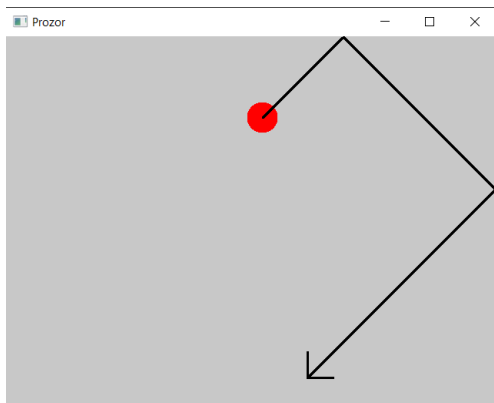
```
void sf::Transformable::scale(float factorX,
                              float factorY)
void sf::Transformable::scale(const Vector2f&
                              factor)
```
- (3.) 

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

itd.

**Zadatak.** Napraviti crveni krug radijusa 20 piksela koji se kreće po prozoru dimenzije  $640 \times 480$  i odbija od njegovih rubova.

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



```
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)

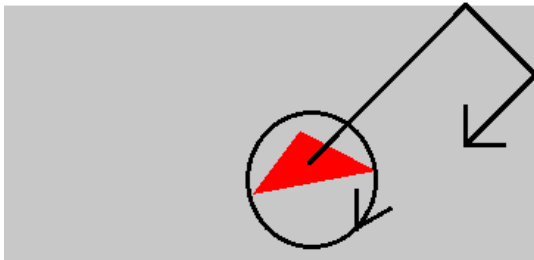
```
if ((krug.getPosition().x + r > prozor.getSize().x
    && pomak.x > 0) ||
    (krug.getPosition().x - r < 0 && pomak.x < 0)) {
    pomak.x = -pomak.x;
}
if ((krug.getPosition().y + r > prozor.getSize().y
    && pomak.y > 0) ||
    (krug.getPosition().y - r < 0 && pomak.y < 0)) {
    pomak.y = -pomak.y;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

**Napomena.** U uokvirenom zbrajamo parove!

# Problem: Što s kompliciranijim oblicima?

**Zadatak.** Kao u prošlom zadatku, ali se sad radi o nekom trokutu koji se usput i rotira. Razmisliti i što ako je riječ o proizvoljnom mnogokutu.

Prozor



# Omeđujući okviri (*Bounding boxes*)

- najjednostavnije, općenito rješenje je naći najmanji pravokutnik koji sadrži naš oblik (tj. sve njegove točke), a čije su stranice paralelne x i y osi



- **općenito** i za upotrebu pri otkrivanju sudara (kolizija) - za konkretno zadani lik možemo naći i bolje načine
- takav omeđujući okvir dobivamo pomoću jedne od sljedećih funkcija:

```
FloatRect sf::Shape::getLocalBounds() const
```

```
FloatRect sf::Shape::getGlobalBounds() const
```

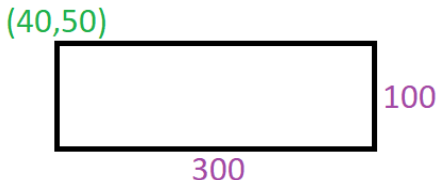


# Omeđujući okviri: Rect predložak klase

- za manipulaciju pravokutnika sa stranicama koje su paralelne osima - takav je definiran **gornjim lijevim kutem** i **veličinom**
- uz *defaultni* konstruktor (`sf::Rect<T>::Rect()`), imamo i konstruktor koji prima 4 podatka koji određuju takav pravokutnik:

```
sf::Rect<T>::Rect(T rectLeft, T rectTop,  
                 T rectWidth, T rectHeight)
```

Primjer. `sf::Rect<int> p(40, 50, 300, 100);`



`sf::Rect<int> p;` isto je kao `sf::Rect<int> p(0, 0, 0, 0);`

# Predložak klase Rect

- zbog jednostavnosti, SFML koristi sljedeće dvije instance (preuzeto iz datoteke Rect.hpp - [link](#)):

```
typedef Rect<int>      IntRect;  
typedef Rect<float>   FloatRect;
```

- Rect predložak klase definiran iz praktičnih razloga pa ima sve dijelove javne  $\Rightarrow$  možemo izravno koristiti **left**, **top**, **width** i **height** varijable članice (nisu funkcije pa ne pišemo () iza!)

## Primjer.

```
sf::IntRect p(20, 30, 10, 50);  
cout << p.left << ", " << p.top << ", "  
      << p.width << ", " << p.height << endl;
```

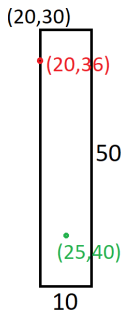
Ispis: 20, 30, 10, 50

# Funkcije članice predložka klase Rect

- zbog jednostavnost nema izravno podržano dobivanje `right`, `bottom` i sl.
- umjesto toga pruža funkcije za provjeru sadrži li pravokutnik neku točku kao i sijeku li se dva pravokutnika
- pravila za granice pravokutnika: lijevi i gornji brid su sadržani u površini pravokutnika, a desni i donji nisu
- provjera pripada li neka točka pravokutniku:
  - `bool sf::Rect<T>::contains(T x, T y)`
  - `bool sf::Rect<T>::contains(const Vector2<T>& point) const`

## Primjer.

```
sf::IntRect p(20, 30, 10, 50);  
cout << p.contains(25, 40) << endl //1  
      << p.contains(20, 36) << endl //1  
      << p.contains(32, 83) << endl; //0
```



(32,83)

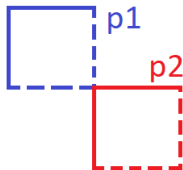
# Funkcije članice za presjek

- prva verzija vraća `true` ako se pravokutnici sijeku, a druga preko drugog parametra vraća presjek (koji je opet pravokutnik!)

```
bool sf::Rect<T>::intersects(const Rect<T>&
                               rectangle) const
bool sf::Rect<T>::intersects(const Rect<T>&
                               rectangle, Rect<T>& intersection) const
```

## Primjer.

```
sf::IntRect p1(0, 0, 1, 1),
             p2(1, 1, 1, 1);
cout << p1.intersects(p2) << endl; //0
```



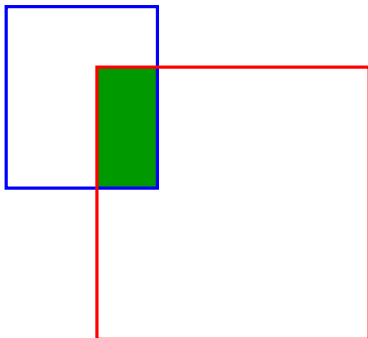
# Funkcije članice za presjek (nastavak)

## Primjer.

```
sf::IntRect p1(0, 0, 5, 6),  
            p2(3, 2, 9, 9), p3;  
cout << p1.intersects(p2,p3) << endl;  
cout << p3.left << "," << p3.top << "," <<  
     << p3.width << "," << p3.height << endl;
```

## Ispis:

```
1  
3,2,2,4
```





# Povratak na temu o omeđujućim okvirima

- **getGlobalBounds** vraća omeđujući okvir u globalnom koordinatnom sustavu (**nakon svih transformacija**)
- **getLocalBounds** - u lokalnim koord. (**ignorira transformacije**)

## Primjer.

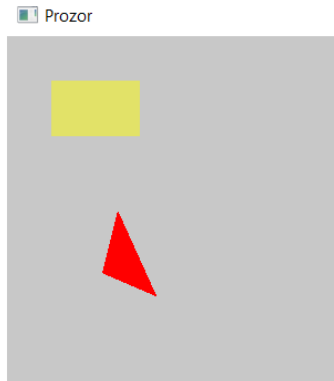
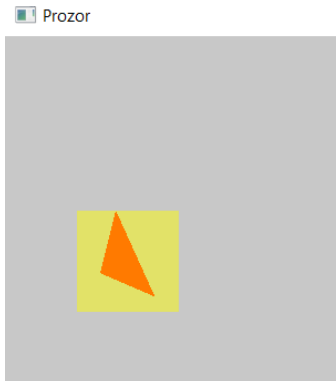
```
sf::ConvexShape t(3);
t.setPoint(0, sf::Vector2f(40,40));
t.setPoint(1, sf::Vector2f(120,70));
t.setPoint(2, sf::Vector2f(70,90));
t.setPosition(100, 100);
t.setRotation(45);
sf::FloatRect okvir = t.getGlobalBounds();
cout << okvir.left << "," << okvir.top << ","
    << okvir.width << "," << okvir.height << endl;
okvir = t.getLocalBounds();
cout << okvir.left << "," << okvir.top << ","
    << okvir.width << "," << okvir.height << endl;
```

# Ispis i prikaz za prethodni primjer

Ispis:

```
64.6447, 156.569, 91.9239, 91.9239  
40, 40, 80, 50
```

Prikaz globalnih vs. lokalnih koordinata:



- **getGlobalBounds** ne vraća nužno najmanji pravokutnik koji omeđuje oblik
- samo jamči da vraćeni pravokutnik pokriva sve vrhove
- to omogućuje **brzu aproksimaciju za provjeru granica** (ako želimo više, treba napraviti detaljnije provjere)