SFML - Prozori

Objektno programiranje - 3. vježbe (2. dio)

Marko Živković

Prirodoslovno-matematički fakultet,
Sveučilište u Zagrebu

20. ožujka 2026. godine

SFML prozor

- Objekt klase **sf::Window** (SFML koristi imenički prostor **sf**)

Primjer 1.

```
#include <iostream>
#include <SFML/Window.hpp>
using namespace std;

int main() {
  sf::Window prozor(sf::VideoMode(800,600),"Prozor");
  while (prozor.isOpen()) { }
  return 0;
}
```

- u gornjem smo stvorili i otvorili jedan prozor
- kako bi nešto vidjeli dodali petlju koja se vrti dok god je taj prozor otvoren (inače program odmah završi)

Komentar vezan uz prethodni `#include`

Uočimo da smo na prethodnom slajdu imali

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

umjesto, primjerice, `#include <Window.hpp>`. Objašnjenje:

- u svojstvima projekta smo pod *Additional Include Directories* naveli *C:\SFML-2.6.1\include*
- u toj mapi je unutar mape **`SFML`** datoteka **`Window.hpp`**

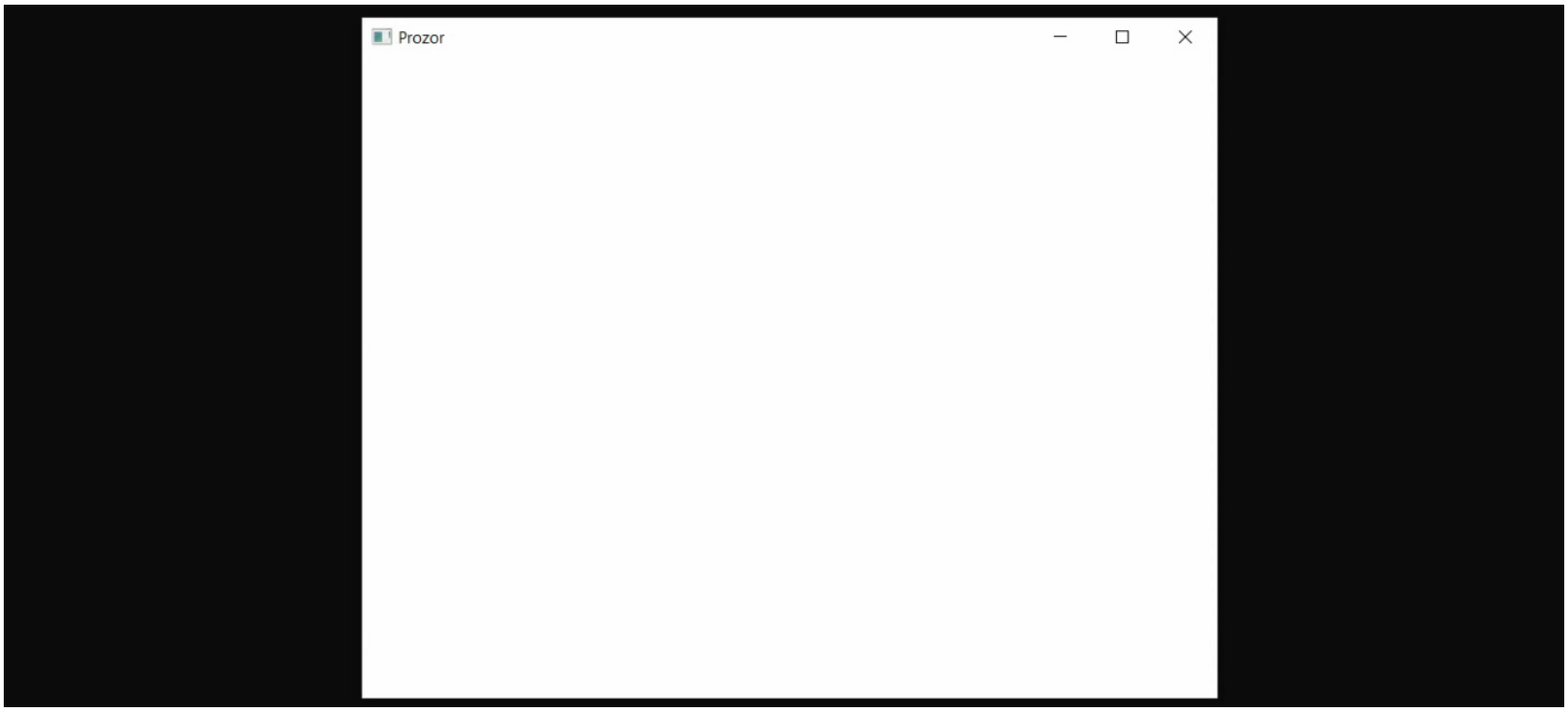
Napomena. Ne možemo samo pod *Additional Include Directories* promijeniti navedeno *C:\SFML-2.6.1\include\SFML* jer u datoteci `Window.hpp` između ostalog piše sljedeće:

```
#include <SFML/System.hpp>
#include <SFML/Window/Clipboard.hpp>
#include <SFML/Window/Context.hpp>
```

`Window` konstruktor

```
sf::Window prozor(sf::VideoMode(800, 600),"Prozor");
```

- prvi argument: *video mode* (klasa `sf::VideoMode`)
- unutarnja veličina prozora (bez obruba i naslovne trake)
- u gornjem primjeru: 800 $\times$ 600 piksela
- drugi argument: naslov prozora



`Window` konstruktor - treći opcionalni argument

- **stil prozora** - bitovna ILI kombinacija `sf::Style` enumeratora:

`sf::Style::None`	bez dekoracija (ne može se kombinirati s ostalima)
`sf::Style::Titlebar`	prozor ima naslovnu traku
`sf::Style::Resize`	prozoru se može mijenjati veličina i ima gumb za maksimiziranje
`sf::Style::Close`	prozor ima gumb za zatvaranje
`sf::Style::Fullscreen`	*fullscreen* način
`sf::Style::Default`	*defaultni* stil (`Titlebar | Resize | Close`)

`Window` - `create` funkcija

- za stvaranje/mijenjanje prozora nakon konstrukcije
- ima iste argumente kao i konstruktor

Primjer. Umjesto konstruktora iz prošlog primjera (s istim efektom):

```
sf::Window prozor;  //defaultni konstruktor
prozor.create(sf::VideoMode(800,600),"Prozor");
```

- dobiveni prozor ne može se pomaknuti, zatvoriti, niti mu se može promijeniti veličina

Događaji (*events*)

Dodamo sljedeći kod u glavnu petlju:

```
while (prozor.isOpen()) {
    sf::Event d;
    while (prozor.pollEvent(d)) {
        if(d.type == sf::Event::Closed)
            prozor.close();
    }
}
```

- Objekt klase `sf::Window` sadrži red događaja koji su se dogodili nad prozorom.
- Funkcija `bool sf::Window::pollEvent(Event &event)`
  - ako red nije prazan u referencu ubacuje događaj s početka reda, uklanja ga iz reda i vraća `true`,
  - ako je rad prazan vraća `false`.
- Dakle to **nije blokirajuća funkcija** - uvijek će vratiti vrijednost i program će se nastaviti
- Unutrašnjom `while` petljom provjeravamo sve događaje u redu.

Događaji - `pollEvent` vs. `waitEvent`

`bool sf::Window::waitEvent(Event &event)`

- kao `pollEvent`, ali **blokirajuća funkcija** - čeka dok se ne dogodi neki događaj

Primjer. Što radi sljedeći kod (i što bi bilo drugačije kad bi stavili `pollEvent` umjesto `waitEvent`)?

```
int i = 0, j = 0;
while (prozor.isOpen()) {
    sf::Event d;
    while (prozor.waitEvent(d)) {
        if (d.type == sf::Event::Closed) {
            prozor.close();
        }
        cout << "i = " << i++ << endl;
    }
    cout << "j = " << j++ << endl;
}
```

Događaji: `sf::Event` tip

`sf::Event` je klasa koja ima varijablu **`type`** tipa `enum EventType` te **uniju** drugih varijabli od kojih se po jedna može koristiti ovisno o `type`.

Unija je skup varijabli koje dijele isti memorijski prostor, pa je samo jedna valjana u danom trenutku. Drugim se varijablama tehnički može pristupiti, ali to treba izbjegavati jer je rezultat nepredvidiv. Uniju koristimo zbog uštede memorije ako znamo da će u danom trenutku najviše jedna varijabla biti potrebna.

- U prethodnom primjeru događaj tipa **`sf::Event::Closed`** predstavlja zahtjev korisnika za zatvaranjem prozora (klik na x).
- Samim time prozor nije zatvoren, nego tek naredbom `prozor.close()`.
- Prije zatvaranja prozora, možemo spremiti stanje aplikacije ili pitati korisnika što učiniti.

Zadatak

Zadatak. Napišite 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 bismo 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),
          "Prozor" + to_string(brojac));
```

Rješenje (2. dio)

- uočimo: početni prozor (`prozor[0]`) je otvoren cijelo vrijeme do kraja programa

$\Rightarrow$ program možemo staviti u glavnu petlju `while (prozori[0].isOpen()).`

- `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].create
        (sf::VideoMode(800,600),
         "Prozor" + to_string(++brojac));
  else for(size_t j = 0; j < brojac; ++j)
        prozori[j].close();
}
```

Funkcije koje primaju prozore

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) {
        stvori_prozor(prozori[brojac], brojac + 1);
        ++brojac;
    }
    ...
```

`++brojac` je u novom redu jer korištenje i mijenjanje varijable u istom izrazu može biti nedefinirano.

Prvi pokušaj takve funkcije i greška

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

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

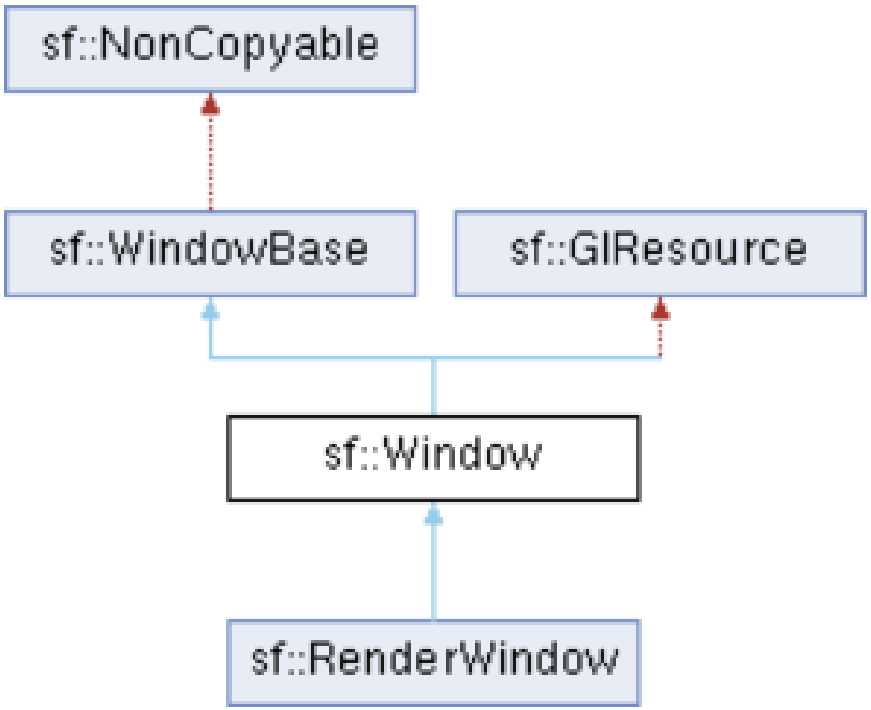
```
stvori_prozor(prozori[0], brojac);
```

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 bismo 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);    ✗
```

`sf::Event::Closed` - ostali tipovi događaja:

`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