


Računarska grafika

JavaFX - interakcija



Uvod

- Interakcija u RG
 - komunikacija između korisnika i prikazanih grafičkih objekata
 - bez interaktivnosti
 - doprinos RG u mnogim oblastima primene – neuporedivo manji
- Interakcija se obavlja preko ulaznih uređaja
 - konvencionalnih, širokog spektra primene
 - tastatura – za unos podataka, sve ređe za unos komandi
 - pokazivački uređaji – za ukazivanje i izbor prikazanog objekta
 - miš (*mouse*), dodirni panel (*touch pad*), dodirni ekran (*touch screen*)
 - konvencionalnih, užeg spektra primene
 - komandna palica (*joystick*), igrački uređaj (*game pad*), grafička ploča (*graphics tablet*), kotrljajuća kugla (*track ball*)
 - specifičnih za virtuelnu realnost i druge primene
 - senzor pokreta (*motion sensor*), pratilac oka (*eye tracker*)
 - HMD (*Head Mounted Display*)

Metafore ulaznih uređaja (1)

- Korišćene metafore određuju način komunikacije sa uređajima,
 - na taj način određuju programsku paradigmu komunikacije
- Pokazivački (lokatorski) uređaji – metafora štapa-pokazivača
 - uređaj služi da bi se dovođenjem kurzora na željeno mesto na ekranu ukazalo na željeni objekat
 - uređaj omogućava i da se ukazani objekat izabere klikom na dugme
 - miš je opremljen sa
 - nekoliko dugmadi (*buttons*) – tipično dva do tri
 - služe za komandu klikom (*click*) – pritiskom i otpuštanjem, ili dvoklikom
 - točkićem (*wheel*) – koji je ujedno i dugme
- Pokazivački uređaji – metafora kuke za povlačenje (*drag*) objekta
 - postiže se povlačenjem miša sa pritisnutim dugmetom
 - izabrani objekat prati kurzor i premešta se na mesto na kojem se otpusti dugme miša (*drag-and-drop, d&d*)

Metafore ulaznih uređaja (2)

- Točkić za proklizavanje - metafora dugmeta “potenciometra”
 - pomeranjem se upravlja nekim klizačem (*scroll*) na ekranu ili virtuelnom kamerom – približavanje/udaljavanje (*zoom in/out*)
 - upravljanje u trećoj dimenziji
- Tastatura – metafora pisaće mašine
 - služi za unos tekstualnih i numeričkih podataka (i komandi)
 - režim editovanja – aplikacija se obaveštava tek kada se pritisne taster koji označava kraj unosa podatka ili komande (*Enter/Return*)
 - eho unošenih znakova preko tastature obezbeđuje grafički sistem, odnosno odgovarajuća komponenta grafičkog korisničkog interfejsa
 - kontrolni znaci nemaju značaja
 - izuzev nekih koji se koriste za editovanje unošenog podatka (*Backspace, Delete, strelice,...*)

Metafore ulaznih uređaja (3)

- Tastatura – metafora komandne table
 - služi za izdavanje komandi jednim tasterom ili kombinacijom tastera
 - sirovi režim (*raw*) – za svaki pritisnuti taster aplikaciji se prenosi podatak
 - može se prenositi kod pritisnutog znaka
 - slova (a, A, Ђ,...), cifre (0-9), specijalni (#,@,...), kontrolni (Ctrl-C,...)
 - može se prenositi kod (identifikacija) tastera
 - na primer: komandni (funkcionalni) tasteri (npr. F1, F2, ...)
- Interakcija preko dodirnog ekrana – više mogućnosti i metafora
 - izbor objekta na ekranu pritiskom prsta ili specijalne olovke (*stylus*)
 - pomeranje sadržaja povlačenjem prsta/olovke u nekom pravcu i smeru
 - simultane akcije pomoću dva ili više prstiju
 - razvlačenje, odnosno sažimanje slike (*zoom in/out*)
 - rotacija slike
 - promena orijentacije, brzina pomeranja

Podaci sa ulaznih uređaja

- Pokazivački uređaji
 - koordinate kurzora u ravni
 - statusi pojedinih dugmadi uređaja (pritisnuto/otpušteno)
 - statusi tastera modifikatora na tastaturi
- Tastatura
 - niska unetih znakova
 - kod znaka ili kod tastera
- Dodirni ekran
 - koordinate jedne, dve ili više tačaka
- Grafički tablet
 - vektor koordinata tačaka

Prihvatanje podataka sa uređaja

- Da bi se ostvarila interakcija potrebno je da:
 - podaci sa ulaznih uređaja stignu do aplikacije
 - aplikacija ih obradi
- Načini prihvatanja podataka sa ulazih uređaja
 - sinhrono – povremenim propitivanjem uređaja (*polling*)
 - drugi naziv – uzorkovanje (*sampling*) uređaja
 - asinhrono - čekanjem na događaj (*event waiting*)
 - danas dominantan način komunikacije sa ulaznim uređajima
- Događaj
 - u opštem slučaju – dešavanje od interesa
 - u programiranju grafičkog korisničkog interfejsa
 - dešavanja izazvana akcijama korisnika preko ulaznih uređaja

Obrada događaja

- Obrada događaja predstavlja i osnovu savremene paradigme za programiranje aplikacija sa grafičkim korisničkim interfejsom (GUI)
 - događaji se dešavaju asinhrono u odnosu na tok GUI-niti aplikacije
 - obrađuju se u posebnim funkcijama – rukovaocima (*event handler*)
 - posao programera grafičke aplikacije – pisanje rukovalaca
 - rukovaoci se automatski pozivaju kao reakcija na događaj
 - izvršavaju se u GUI-niti aplikacije
- Paradigma se naziva programiranjem vođenim događajima (*event driven programming*)
- U aplikacijama sa GUI veliki deo koda se nalazi u
 - rukovaocima događaja
 - metodima koji se direktno ili indirektno pozivaju iz rukovalaca

Mehanizam obrade događaja

- Na nivou operativnog sistema,
 - uređaji generišu prekide (*interrupt*)
 - prekidne procedure (*int. handler*) preuzimaju podatke iz registara uređaja
 - na osnovu ovih podataka formiraju se poruke (*message*) ili događaji
 - poruke/događaji se smeštaju u red (*message/event queue*)
- Aplikacija sa GUI
 - u niti GUI čeka da se u redu pojavi poruka/događaj
 - ne troši se procesorsko vreme na čekanje
 - kada se u redu pojavi događaj, automatski se poziva odgovarajući rukovalac za obradu poruke/događaja (*message/event handler*)
 - čekanje i obrada koja sledi obavljaju se u beskonačnoj petlji za obradu poruka/događaja (*message/event loop*)

Savremena paradigma obrade

- U savremenim tehnologijama za razvoj korisničkih interfejsa
 - red događaja, čekanje na događaj i petlja su u sakriveni od programera
 - programer ima zadatak da
 - obezbedi rukovaoce za odgovarajuće tipove događaja
 - pridruži te rukovaoce odgovarajućim elementima korisničkog interfejsa
- U Java 1.0 – centralizovani model obrade događaja
 - samo jedan rukovalac za obradu svih događaja
 - obrada u razgranatoj strukturi (`switch`, `if-else-if`)
 - nije u duhu OO programiranja

Delegirani model obrade

- Od verzije Java 1.1 – delegirani model obrade događaja
 - interfejsi oslušivača (*listener*) propisuju metode rukovalaca
 - na primer: `ActionListener` – za događaje akcije
 - pritisak na ekranski taster izaziva ovakav događaj
 - taster je izvor i meta događaja
 - zadatak programera je da:
 - u klasama oslušivača implementira propisane metode
 - stvori objekte oslušivača
 - registruje objekte oslušivača kod objekata meta događaja
 - objekat događaja se prosleđuje automatski pozvanom rukovaocu
 - decentralizacija je u duhuOO programiranja
 - paketi `awt` i `swing`
- JavaFX – delegirani model obrade događaja, ali nešto drugačiji

JavaFX – mehanizam obrade

- Dva bitna subjekta: izvor događaja i meta događaja
 - objekat izvora predstavlja poreklo događaja – poziva rukovalac
 - objekat mete – svaki objekat koji može da obradi događaj
- U korisničkim interfejsima
 - izvori i mete su elementi korisničkih interfejsa
 - prozor, scena ili čvor u sceni na proizvoljnoj dubini grafa scene
- Primer: klikne se na dugme miša kada je kurzor nad nekim oblikom (geometrijskom primitivom) na proizvoljnoj dubini u grafu scene
 - pokazani objekat je meta događaja
 - da li korisnik želi da se događaj odnosi (samo) na metu?
 - možda korisnik želi:
 - da samo meta obradi događaj
 - da roditeljski čvor ili čvor nekog pretka u grafu scene obradi događaj
 - da i meta i neko od predaka u grafu scene reaguju na događaj

Dispečerski lanac

- Dispečerski lanac (*event dispatch chain*)
 - čine ga objekti na putanji događaja (*event route*)
 - početak (koren) putanje je pozornica
 - zatim scena na pozornici
 - zatim redom čvorovi u grafu scene počevši od korena prema meti
 - meta je kraj lanca
- Odgovornost za formiranje ovog lanca je na meti događaja
- Element korisničkog interfejsa na putanji
 - predstavlja dispečer događaja (*event dispatcher*)
 - obrađuje i prosleđuje događaj sledećem elementu u lancu
- Proces obrade događaja – obilaskom putanje (*route traversal*)
 - za vreme obilaska meta se ne menja
 - izvor događaja se menja u svakom koraku obilaska
 - prethodnik elementa do kojeg je događaj stigao je izvor događaja

Faze obilaska lanca

- Obilazak je dvosmeran, odnosno ima dve faze
- U prvoj fazi se napreduje od korena putanje prema meti
 - faza se naziva fazom hvatanja (*capture phase*)
- U drugoj fazi se od mete vraća po putanji prema korenu
 - objekti u lancu se obilaze suprotnim redosledom
 - faza se naziva fazom isplivavanja (*bubbling phase*)
- Svakom elementu putanje mogu biti pridružene dve vrste rukovalaca za obradu događaja: filter i obrađivač
- Kada proces obrade događaja stigne do nekog elementa u lancu
 - izvršava se odgovarajuća vrsta rukovaoca za dati smer obilaska

Obrada događaja u lancu

- U fazi hvatanja događaja – izvršavaju se filteri
- U fazi isplivavanja događaja – izvršavaju se obrađivači
- Element u lancu koji je u jednom koraku obilaska putanje filtrirao ili obrađivao događaj – postaje izvor događaja u sledećem koraku
- Metod događaja `consume()`
 - može da se pozove iz rukovaoca da završi obradu tog događaja (da prekine prosleđivanje datog događaja kroz lanac)
- Na primer, za događaj namenjen sceni, a ne čvoru u njenom grafu
 - filter scene može pozvati metod `consume()`
 - sprečava da događaj putuje dalje po putanji u grafu scene

Događaj

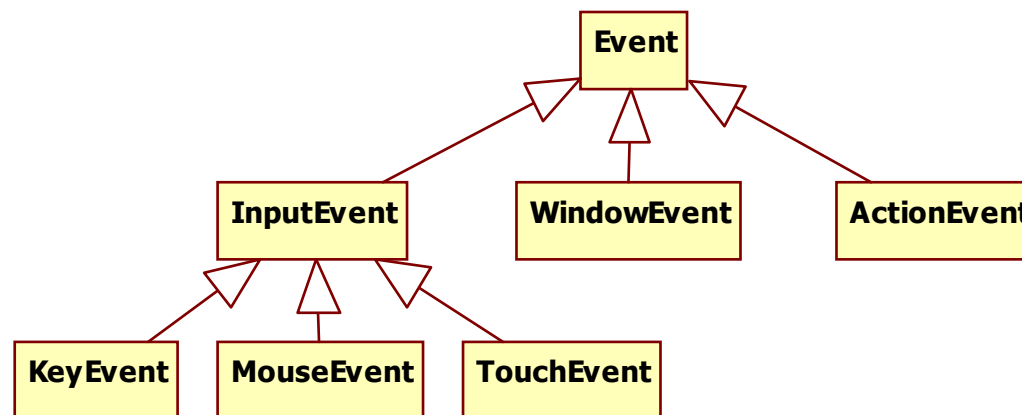
- Koren hijerarhije klasa događaja je klasa `EventObject`
 - iz paketa `java.util`, direktni potomak klase `Object`
 - klasa apstrahuje događaje, bez obzira da li oni potiču iz GUI ili ne
 - sadrži samo informaciju o izvoru događaja
- Događaji koji potiču iz GUI se apstrahuju klasom `Event`
 - iz paketa `javafx.event`
 - sadrži informacije o izvoru (nasleđeno), meti, vrsti događaja i da li je događaj potpuno obrađen, odnosno "konzumiran"
 - konstruktor:

```
Event( Object izvor, EventTarget meta,  
      EventType<? extends Event> vrsta)
```

 - izvor događaja može biti proizvoljan objekat
 - meta događaja – svaki objekat koji implementira `EventTarget` interfejs
 - vrsta događaja je tipa generičke klase `EventType` za dati tip događaja (klase `Event` ili potomka klase)

Tipovi događaja

- Tip događaja je određen klasom događaja
- Iz klase `Event` su izvedene klase:
 - ulaznog događaja (`InputEvent`)
 - prozorskog događaja (`WindowEvent`)
 - događaja akcije (`ActionEvent`)
 - događaja promene transformacije (`TransformChangedEvent`)
 - ...



Vrste događaja (1)

- Tip događaja je određen njegovom klasom iz prikazane hijerarhije
- Vrsta događaja je generički tip:
 - `EventType<T extends Event>`
 - parametar generika je tipa `Event` ili njegovih podtipova
- Klasa `Event` sadrži polje koje određuje vrstu događaja
 - `EventType<? extends Event> vrsta;`
- Vrsta događaja bliže određuje njegovu prirodu
- Na primer, događaji miša su tipa `MouseEvent`
 - oni mogu poticati od različitih akcija korisnika preko miša
 - vrsta događaja je određena konkretnom akcijom:
 - `MouseEvent.MOUSE_PRESSED`, `MouseEvent.MOUSE_RELEASED`, ...
- Specijalna vrednost `EventType` je `ANY`
 - odnosi se na proizvoljnu vrstu događaja datog tipa

Vrste događaja (2)

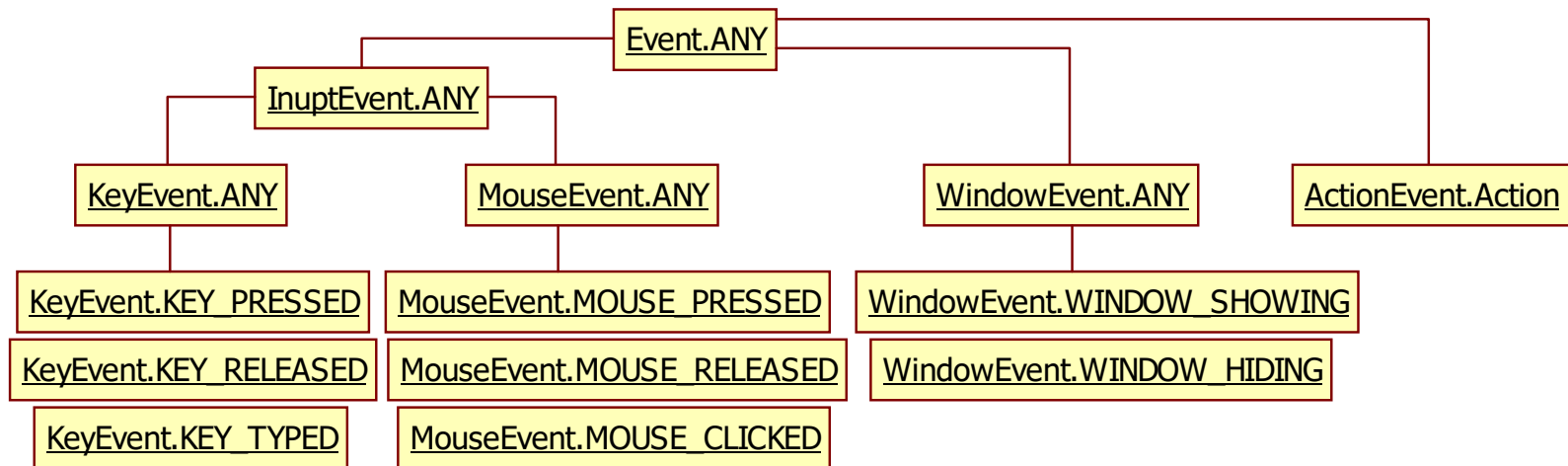
- U odgovarajućim klasama tipova događaja definisani su nepromenljivi objekti koji određuju moguće vrste (reference su `static final`)
 - na primer: `MouseEvent.MOUSE_PRESSED` je definisan na način:

```
public static final EventType<MouseEvent> MOUSE_PRESSED
```
- Konstruktor:

```
public EventType(EventType<? super T> nadVrsta,  
                String ime)
```

 - vrsta događaja je povezana sa svojom nadvrstom
- Vrste događaja formiraju hijerarhiju (stablo)
 - ova hijerarhija nije klasna, već objektna

Hijerarhija vrsta događaja



Programiranje obrade događaja

- Programiranje obrade događaja se svodi na:
 - definisanje objekta rukovaoca
 - pridruživanje tog rukovaoca meti, ili nekom od objekata u dispečerskom lancu
 - objekat se pridružuje kao filter ili obrađivač
- Elementi GUI izvedeni iz klasa `Window`, `Scene` i `Node` mogu biti mete
- Objekti klasa iz biblioteke JavaFX mogu biti mete jer:
 - implementiraju interfejs `EventTarget`
 - imaju implementiran metod za formiranje lanca objekata na putanji događaja
- Ukoliko bi meta trebalo da bude neke klase koja nije potomak navedenih
 - klasa bi trebalo da implementira interfejs `EventTarget`
- Za objekte klasa koje eksplicitno implementiraju interfejs `EventTarget`
 - programer mora da obezbedi metod za formiranje lanca

Definisanje rukovaoca

- Za obe vrste rukovalaca, filtere i obrađivače
 - odgovarajuće klase se definišu na isti način
- Piše se anonimna klasa koja implementira generički funkcijski interfejs (`EventHandler<T extends Event>`)
 - tip događaja `T` može biti neka od potklasa klase `Event`
 - metod interfejsa je: `void handle (T događaj);`
- Primer za događaj miša tipa `MouseEvent`:
 - objekat rukovaoca koji samo ispisuje poruku na standardnom izlazu

```
EventHandler<MouseEvent> ruk =
    new EventHandler<MouseEvent> () {
        void handle(MouseEvent događaj) {
            System.out.println("Miš.");
        }
    }
```
- Elegantnije – isti objekat se može definisati korišćenjem lambda izraza:

```
EventHandler<MouseEvent> ruk =
    događaj -> System.out.println("Miš.");
```

Povezivanje rukovalaca

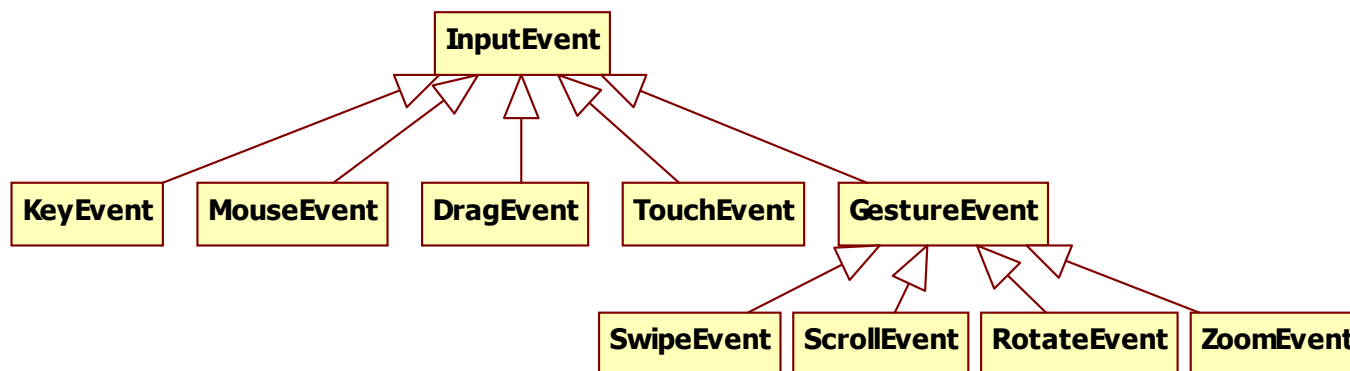
- Rukovaoci filtera i obrađivača se definišu na isti način
- Koriste se različiti metodi za njihovu registraciju
 - pridruživanje elementu dispečerskog lanca
 - registrovanje filtera: `addEventFilter()`
 - registrovanje obrađivača: `addEventHandler()`
- Prilikom povezivanja rukovaoca sa elementom potrebno je navesti i vrstu događaja koji će se filtrirati/ obrađivati navedenim rukovaocem
- Na primer, za prethodno definisani rukovalac
 - filtriranje/obrađivanje događaja klika miša od strane objekta `element`
`element.addEventFilter(MouseEvent.MOUSE_CLICKED, ruk);`
`element.addEventHandler(MouseEvent.MOUSE_CLICKED, ruk);`
- Drugi način povezivanja – samo za obrađivače tipičnih vrsta događaja:
 - pozivanje `setOnXYZ(rukovalac)` metoda (XYZ – vrsta događaja)
 - na primer, povezivanje: `element.setOnMouseClicked(ruk);`
 - razvezivanje: `element.setOnMouseClicked(null);`
 - ne može za filtere i ne može za više obrađivača pridruženih elementu

Redosled izvršenja

- U prvoj fazi "hvatanja" događaja:
 - od pozornice, preko scene i njenog grafa, u smeru od predaka prema potomcima, do čvora mete
- U drugoj fazi "isplivavanja" događaja:
 - u suprotnom smeru od smeru u fazi hvatanja
- Ako se iz nekog rukovaoca pozove metod događaja `consume()`
 - tok obrade događaja se prekida
- Ako je više filtera ili obrađivača vezano za isti element lanca
 - prvo će se izvršiti rukovaoci specifičnijih vrsta događaja
 - na kraju rukovalac vrste događaja `ANY`
- Ako su neki rukovaoci pridruženi čvoru metodom `addEventFilter()` / `addEventHandler()`
 - oni će se izvršiti pre rukovaoca pridruženog metodom `setOnXYZ()` za datu vrstu događaja `XYZ`

Ulazni događaji

- Događaj ulaznog uređaja (ulazni događaj)
 - apstrahovan klasom `InputEvent`
 - klasa `InputEvent` je izvedena iz klase `Event`
 - koren hijerarhije ulaznih događaja
 - cela hijerarhija klasa ulaznih događaja
 - u paketu `javafx.scene.input`



Primer

- Obrada proizvoljnog ulaznog događaja (proizvoljan tip i vrsta):

```
Circle krug = new Circle(110.0, 25.0, 20.0);  
krug.setFocusTraversable(true);
```

```
EventHandler<InputEvent> r = dog -> {  
    System.out.println("Dogadaj.");  
    System.out.println("Tip: " +  
        dog.getEventType().getName());  
    System.out.println("Izvor: " +  
        dog.getSource().getClass().getSimpleName());  
    System.out.println("Cilj:" +  
        dog.getTarget().getClass().getSimpleName());  
};  
krug.addEventHandler(InputEvent.ANY, r);
```

Događaji miša

- Klasa `MouseEvent` apstrahuje događaje miša
 - izvedena iz klase `InputEvent`
- Miš generiše razne vrste događaja kao što su:
 - pomeranje miša
 - kada se mišem pomeri kurzor, a da pri tome nije bilo pritisnuto dugme miša
 - vučenje miša
 - kada se mišem pomeri kurzor uz pritisnuto dugme miša
 - pritisak na dugme miša
 - otpuštanje dugmeta miša
 - klik na dugme miša
 - ulazak miša
 - kada kurzor pomeran mišem naiđe na neki čvor
 - izlazak miša
 - kada kurzor pomeran mišem napušta čvor

Vrste događaja miša (1)

- Vrste su definisane vrednostima tipa `EventType<MouseEvent>`:
- `ANY`: nadvrsta svih vrsta događaja miša
 - njegova nadvrsta je `InputEvent.ANY`
- `MOUSE_PRESSED`: pritisak na dugme miša
 - metod `getButton()` klase `MouseEvent` vraća konstantu tipa `MouseButton`
 - konstanta određuje koje je dugme pritisnuto
 - konstante su: `NONE`, `PRIMARY`, `MIDDLE` i `SECONDARY`
- `MOUSE_RELEASED`: otpuštanje dugmeta miša
 - cilj događaja – čvor nad kojim je bio kurzor u trenutku pritiskanja dugmeta bez obzira nad kojim čvorom se nalazio kurzor u trenutku otpuštanja dugmeta
- `MOUSE_CLICKED`: klik dugmetom miša dok je kurzor iznad nekog čvora
 - pritiskanje i otpuštanje dugmeta mora biti iznad istog čvora
 - ako ne – događaj se dešava, ali će izvor biti roditelj (na primer scena), a ne čvor nad kojim je pritisnuto dugme miša

Vrste događaja miša (2)

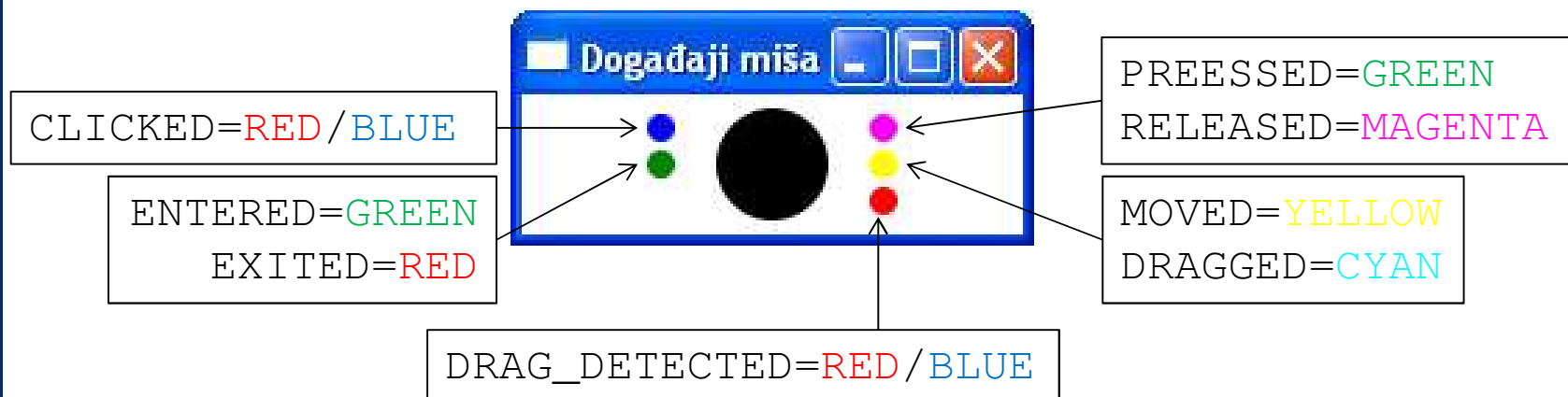
- **MOUSE_MOVED**: pomeranje miša bez pritisnutog dugmeta
- **MOUSE_ENTERED**: kad kurzor vođen mišem ulazi u prostor čvora
 - roditelj čvora se ne obaveštava o događaju ni u fazi hvatanja ni u fazi isplivavanja događaja
- **MOUSE_ENTERED_TARGET**: generiše ga roditelj kad kurzor nailazi na čvor deteta
 - **MOUSE_ENTERED** se transformiše u **MOUSE_ENTERED_TARGET** čiji je izvor roditelj
 - takav događaj prolazi na uobičajen način kroz faze hvatanja i isplivavanja događaja
- **MOUSE_EXITED**: kad kurzor vođen mišem izlazi iz prostora čvora
 - roditelj se ne obaveštava o događaju ni u fazi hvatanja ni u fazi isplivavanja događaja
- **MOUSE_EXITED_TARGET**: generiše ga roditelj kad kurzor napušta čvora deteta
 - **MOUSE_EXITED** se transformiše u **MOUSE_EXITED_TARGET** čiji je izvor roditelj
- **DRAG_DETECTED**: kad je dugme miša pritisnuto i kurzor prevučen preko čvora
 - sa pragom rastojanja od čvora koji je zavisen od platforme
 - događaj se može sprečiti metodom `setDragDetect(boolean)` klase `MouseEvent`
 - metod se može pozvati iz rukovaoca kojim se obrađuje `MOUSE_PRESSED` / `MOUSE_DRAGGED`
- **MOUSE_DRAGGED**: kad se pomera miš sa pritisnutim dugmetom
 - bez obzora na položaj kurzora, događaj se isporučuje čvoru nad kojim je pritisnuto dugme

Primer (1)

```
Circle krug = new Circle(90.0, 25.0, 20.0);
Circle indK = new Circle(50, 12, 5.0); // klik
Circle indUI = new Circle(50, 25, 5.0); // ulaz/izlaz
Circle indPO = new Circle(130, 12, 5.0); // pritisak/puštanje
Circle indPV = new Circle(130, 25, 5.0); // pomeranje/vučenje
Circle indDV = new Circle(130, 38, 5.0); // detekcija vučenja
EventHandler<MouseEvent> r0 = dog -> {
    if (!indK.getFill().equals(Color.RED)) indK.setFill(Color.RED);
    else indK.setFill(Color.BLUE); };
EventHandler<MouseEvent> r1 = dog -> indUI.setFill(Color.GREEN);
EventHandler<MouseEvent> r2 = dog -> indUI.setFill(Color.RED);
EventHandler<MouseEvent> r3 = dog -> indPO.setFill(Color.GREEN);
EventHandler<MouseEvent> r4 = dog -> indPO.setFill(Color.MAGENTA);
EventHandler<MouseEvent> r5 = dog -> indPV.setFill(Color.YELLOW);
EventHandler<MouseEvent> r6 = dog -> indPV.setFill(Color.CYAN);
EventHandler<MouseEvent> r7 = dog -> {
    if (!indDV.getFill().equals(Color.RED)) indDV.setFill(Color.RED);
    else indDV.setFill(Color.BLUE);
};
```

Primer (2)

```
krug.addEventHandler(MouseEvent.MOUSE_CLICKED, r0);  
krug.addEventHandler(MouseEvent.MOUSE_ENTERED, r1);  
krug.addEventHandler(MouseEvent.MOUSE_EXITED, r2);  
krug.addEventHandler(MouseEvent.MOUSE_PRESSED, r3);  
krug.addEventHandler(MouseEvent.MOUSE_RELEASED, r4);  
krug.addEventHandler(MouseEvent.MOUSE_MOVED, r5);  
krug.addEventHandler(MouseEvent.MOUSE_DRAGGED, r6);  
krug.addEventHandler(MouseEvent.DRAG_DETECTED, r7);
```



Kordinate miša

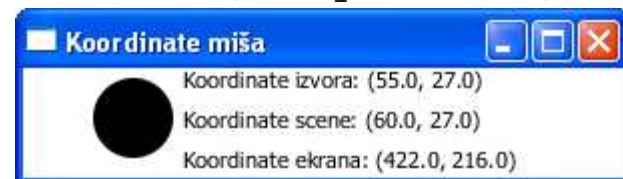
- Miš je 2D pokazivački uređaj kojem se mogu pročitati x i y koordinate
 - koordinate miša se mogu pročitati iz događaja miša
 - podaci predstavljaju koordinate kurzora u odgovarajućem koordinatnom sistemu
- Koordinate mogu biti u koordinatnom sistemu:
 - izvora događaja, scene, ekrana

- **Metodi:**

```
double getX() // x-koordinata u sistemu izvora događaja
double getY() // y-koordinata u sistemu izvora događaja
double getSceneX() // x-koordinata u sistemu scene
double getSceneY() // y-koordinata u sistemu scene
double getScreenX() // x-koordinata u sistemu ekrana
double getScreenY() // y-koordinata u sistemu ekrana
```


Primer

```
Circle krug = new Circle(50.0, 25.0, 20.0);
krug.setTranslateX(5);
Text tekstIzvor = new Text(80, 10, "Koordinate izvora: ");
Text tekstScena = new Text(80, 30, "Koordinate scene: ");
Text tekstEkran = new Text(80, 50, "Koordinate ekrana: ");
EventHandler<MouseEvent> r = dog -> {
    double x0=dog.getX(),y0=dog.getY();
    double x1=dog.getSceneX(),y1=dog.getSceneY();
    double x2=dog.getScreenX(),y2=dog.getScreenY();
    piši(tekstIzvor, "Koordinate izvora: (" +x0+" , "+y0+ " )");
    piši(tekstScena, "Koordinate scene: (" +x1+" , "+y1+ " )");
    piši(tekstEkran, "Koordinate ekrana: (" +x2+" , "+y2+ " )");
};
krug.addEventHandler(
    MouseEvent.MOUSE_MOVED, r);
```



Dugmad miša

- Miš je pokazivački uređaj koji ima nekoliko pridruženih dugmadi
 - najčešće tri dugmeta: primarno, srednje i sekundarno
- Primarno/ sekundarno – podešava se u operativnom sistemu
 - primarno dugme je ono koje se pritiska kažiprstom
 - za dešnjake je primarno levo dugme miša, a za levake desno dugme
- Tip nabiranja `MouseButton` definiše konstante:
 - `PRIMARY`, `SECONDARY`, `MIDDLE`
 - `NONE` ako nije ni jedno od navedenih dugmadi
- Kada se desi događaj miša
 - dohvati se pritisnuto dugme metodom: `MouseButton getButton()` klase `MouseEvent`
 - ispita se dugme poređenjem sa konstantama tipa `MouseButton`
 - drugi način - uslužni metodi klase `MouseEvent`:
 - `boolean isPrimaryButtonDown()`
 - `boolean isMiddleButtonDown()`
 - `boolean isSecondaryButtonDown()`

Metodi klase MouseEvent

- Da li pritisnuto dugme miša predstavlja okidač za iskačući meni?
- Odgovor – metod klase MouseEvent:
`boolean isPopupTrigger()`
- Poseban tip događaja `ContextMenuEvent` – dešava se kada se pritisne
 - odgovarajuće dugme okidača iskačućeg menija
 - kombinacija tastera na datoj platformi koja okida iskačući meni
- Informacija o broju klikova na dugme miša – metod klase MouseEvent:
`int getClickCount()`
- Da li se za vreme događaja kurzor nalazio u "histerezisnoj" okolini mesta gde se dogodio poslednji događaj vrste `MOUSE_PRESSED`?
- Odgovor – metod klase MouseEvent:
`boolean isStillSincePress()`

Modifikatori

- Za vreme događaja miša koji uključuje neko pritisnuto dugme miša, moguće je da je pritisnut i neki specijalni taster na tastaturi
 - takvi tasteri se nazivaju modifikatorima
- Metodi klase `MouseEvent` omogućavaju proveru pritisnutog modifikatora:
 - `boolean isAltDown()`
 - `boolean isControlDown()`
 - `boolean isShiftDown()`
 - `boolean isMetaDown()`
 - `boolean isShortcutDown()`
- Na *Microsoft Windows* platformi
 - Taster [*Ctrl*]: aktivira modifikatore `Shortcut` i `Control`
 - Taster [*Windows*] aktivira modifikator `Meta`
 - Taster [*AltGr*] (desni [*Alt*]) aktivira modifikatore `Alt`, `Control` i `Shortcut`
- Na *Apple Mac* platformi
 - `Shortcut` je isto što i `Meta`