1 Koncept obrade događaja

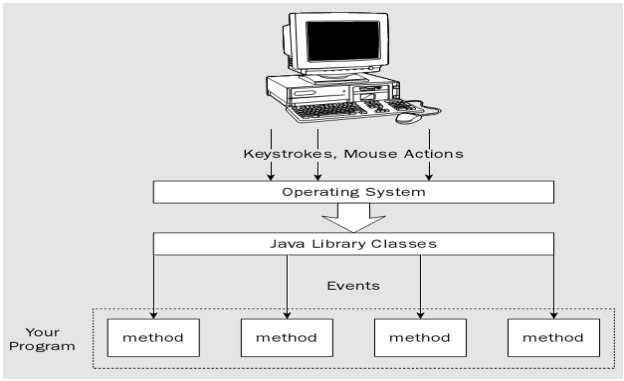
Za razliku od konzolnih programa, gde je tok izvršenja programa predefinisan, kod aplikacija koje koriste grafički korisnički interfejs program je vođen akcijama u okviru interfejsa.
Ovakve akcije najpre prepoznaje operativni sistem.

Primer:

Ako se klikne mišem (pritisak levog ili desnog dugmeta miša), operativni sistem to registruje i beleži poziciju kursora na ekranu. Nakon toga, operativni sistem odlučuje koja aplikacija kontroliše prozor gde se nalazi kursor u trenutku pritiska dugmeta i obaveštava aplikaciju da je pritisnuto dugme miša.

Signali koje program prima od operativnog sistema kao razultat neke korisničke akcije predstavljaju **događaje.**

Svaki događaj koji program prepozna povezan je sa jednim ili više metoda koji se automatski poziva kada se događaj desi i koji vrši njegovu obradu.
Program nije obavezan da reaguje na određeni događaj. Na primer, ako samo pomerimo miša, program ne mora da poziva bilo kakav kôd da reaguje na tu akciju.



2 javafx.event.Event

Osnovna klasa za JavaFX događaje. Korisnik može da definiše novu klasu koja opisuje određenu vrstu događaja, nasleđivanjem klase `Event`.

Svaki FX događaj je povezan sa:

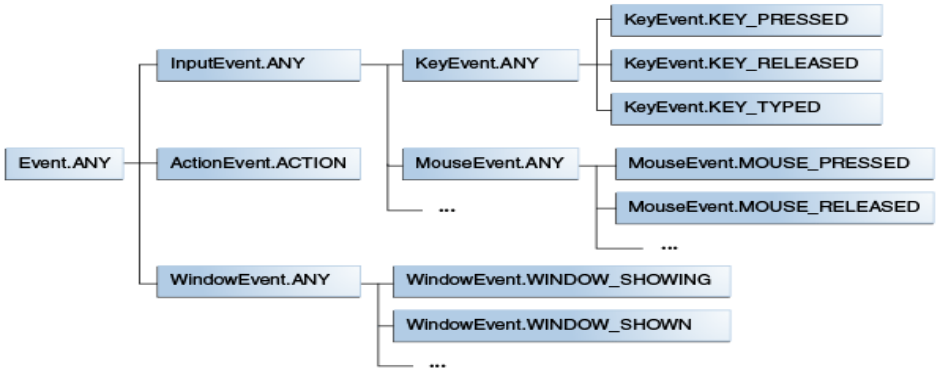
- **izvorom događaja** (*event source*)
- **ciljem događaja** (*event target*) i
- **tipom događaja** (*event type*)

Tip događaja

Tip događaja predstavlja instancu klase `javafx.event.EventType<T>` i njime se klasifikuju događaju u okviru iste klase.
Na primer, klasa događaja `KeyEvent` koji se odnose na dugme tastature sadrži sledeće tipove:

- `KEY_PRESSED, KEY_RELEASED, KEY_TYPED`

Tipovi događaja su opsani klasama koje čine hijerarhiju. Svaki tip događaja ima svoje ime i svoj nadtip. Tip na vrhu hijerarhije (`Event.ROOT` ili `Event.ANY`) nema svoj nadtip (jednak je null).



Tip događaja ANY u bilo kom nivou hijerarhije označava bilo koji tip iz date klase događaja.
Na primer, ako na isti način treba reagovati na bilo koji tip događaja koji se odnosi na dugme sa tastature, tip događaja treba predstaviti klasom `KeyEvent.ANY`. Ako treba reagovati samo na slučaj kada je dugme pritisnuto, koristiti klasu `KeyEvent.KEY_PRESSED`.
Informacija o tipu događaja potrebna je objektu koji vrši njegovu obradu, što može biti *event filter* ili *event handler*.

Cilj događaja

Cilj događaja predstavlja čvor nad kojim se dešava neka akcija. Cilj događaja je instanca bilo koje klase koja implementira interfejs `EventTarget`. Implementacijom metoda `buildEventDispatchChain` kreira se *event dispatch chain* za dati ciljni čvor. Događaj mora da prođe kroz dati lanac da bi stigao do cilja.
Event dispatch lanac čine objekti čije klase implementiraju interfejs `EventDispatcher` , koji mogu da procesiraju događaje za dati ciljni čvor. Ciljni čvor se ne dodaje automatski u lanac, tako da, ako želi da vrši obradu događaja, mora da doda u lanac jedan `EventDispatcher` objekat za sebe.

Klase `Window`, `Scene` i `Node` implementiraju interfejs `EventTarget` i njihove potklase nasleđuju implementaciju metoda `buildEventDispatchChain`. Iz tog razloga, većina elemenata koji su deo GUI-ja ima svoje definisane *event dispatch* lance, tako da korisnik ne treba da se fokusira na njihovo pravljenje, već samo na to kako treba obraditi događaje.

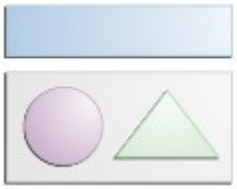
Da bi element GUI-ja mogao da prima događaje (tj. da bude cilj događaja), neophodno je da njegova klasa direktno ili indirektno implementira interfejs `EventTarget`.

Proces isporuke (prenosa) događaja (event delivery)

Obuhvata sledeće procese:
- izbor cilja (*target selection*)
- generisanje putanje (za dostavljanje događaja) (*route construction*)
- hvatanje događaja (*event capturing*)
- event bubbling

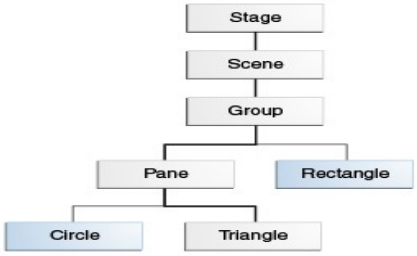
Izbor cilja

Kada dođe do akcije, sistem određuje koji je ciljni čvor na osnovu internih pravila. Na primer, ako je događaj vezan za taster sa tastature, ciljni čvor je onaj koji ima fokus, a ako je vezan za miša, ciljni čvor je onaj koji se nalazi na lokaciji kursora.
Ako je više od jednog čvora locirano na poziciji kursora, poslednji prikazani čvor se smatra ciljem. Na primer, ako korisnik klikne na trougao sa slike, upravo on predstavlja cilj događaja, a ne pravougaonik koji sadrži krug i trougao.



Generisanje putanje

Inicijalna putanja događaja određuje se na osnovu *event dispatch* lanca generisanog metodom `buildEventDispatchChain()` za ciljni čvor. U primeru sa trouglom, odgovarajući lanac čine čvorovi prikazani sivom bojom na sledećoj slici:



Putanja može biti izmenjena tokom procesiranja događaja od strane *event handler* ili *event filter* objekta.

Hvatanje događaja

Događaj se isporučuje (otprema, šalje) od korenog čvora aplikacije do ciljnog čvora kroz odgovarajuću putanju. U prethodnom primeru, događaj „putuje“ od `Stage` čvora do `Triangle` čvora.

Ako bilo koji čvor duž putanje ima registrovan ***event filter*** za dati tip događaja, isti se primenjuje kada događaj stigne do tog čvora. Kada se završi primena filtera, događaj se prosleđuje do sledećeg čvora niz lanac, dok ne stigne do ciljnog čvora koji procesira događaj.

Event bubbling

Nakon dostizanja ciljnog čvora i procesiranja događaja (i eventualne prethodne primene filtera), događaj se prosleđuje nazad kroz lanac, od ciljnog do korenog čvora. U prethodnom primeru, događaj „putuje“ od `Triangle` čvora do `Stage` čvora.

Ako bilo koji čvor duž te putanje ima registrovani ***event handler*** za dati tip događaja, isti se primenjuje kada događaj stigne do tog čvora. Kada se završi primena, događaj se prosleđuje do sledećeg čvora uz lanac, sve do korenog čvora, kada je procesiranje završeno.

Obrada događaja (event handling)

Obrada događaja se realizuje od strane *event filter* ili *event handler* objekata koji implementiraju interfejs `EventHandler`<T extends Event>. Razlika je jedino u kojoj se fazi isporuke događaja primenjuju (*event capturing* ili *event bubbling*).
Za dati *event handler*, izvor događaja (event source) određuje objekat (čvor) za koji je handler registrovan.
Čvor može da registruje više od jednog filter-a ili handler-a. Redosled primene zavisi od hijerarhije tipova događaja (najpre za konkretan tip događaja, a potom za generičke tipove događaja).

Upotreba događaja (consuming)

Event filter ili *event handler* bilo kog čvora duž *event dispatch* lanca mogu da „konzumiraju“ događaj pozivom metoda *consume()*, čime se signalizira da je procesiranje događaja završeno (događaj se ne delegira do ciljnog čvora). Ako čvor ima više registrovanih objekata za obradu događaja, izvršavaju se svi.

http://docs.oracle.com/javafx/2/api/javafx/event/Event.html

3 Potklase klase Event

- **ActionEvent** (događaj koji predstavlja neki vid akcije; koristi se u različite svrhe, na primer događaji vezani za dugme (`Button`) se opisuju ovom klasom)
- CheckBoxTreeItem.TreeModificationEvent
- **InputEvent (**ContextMenuEvent, DragEvent, GestureEvent, InputMethodEvent, KeyEvent, **MouseEvent**, TouchEvent**)**
- ListView.EditEvent
- MediaErrorEvent
- TableColumn.CellEditEvent
- TreeItem.TreeModificationEvent
- TreeView.EditEvent
- WebEvent
- WindowEvent
- WorkerStateEvent anje obrade događaja dugmetu

4 ActionEvent, MouseEvent, WindowEvent, KeyEvent

Akcija korisnika	Tip događaja	Klase koje podržavaju događaje
Pritisnuto je dugme, izabrana je stavka menija, combo box je prikazan ili ne itd.	`ActionEvent`	`ButtonBase, ComboBoxBase, ContextMenu, MenuItem, TextField`
Pomeren je miš ili je pritisnuto dugme miša	`MouseEvent`	`Node, Scene`
Prozor je zatvoren, prikazan ili sakriven	`WindowEvent`	`Window`
Pritisnut je taster sa tastature	`KeyEvent`	`Node, Scene`

5 Pridruživanje obrade događaja dugmetu

Upotrebljava se metod **setOnAction()** iz klase **ButtonBase** da bi se definisalo ponašanje prilikom pritiska na dugme i kreiranja događaja tipa **ActionEvent**.

ActionEvent je tip događaja koji se obrađuje **EventHandler**-om. Interfejs **EventHandler<T extends Event>** vrši obradu događaja različitih tipova. (T – klasa događaja koji se obrađuje).

Objekat **EventHandler** obezbeđuje metod **handle(T event)** za obradu događaja nad dugmetom. Metod se poziva kada se desi određeni događaj za koji je EventHandler registrovan.

http://docs.oracle.com/javafx/2/api/javafx/event/EventHandler.html

6 MouseEvent

Metodi za obradu događaja su:

- setOnMouseEntered (pomeren je miš i kursor je na površini cilja)
- setOnMouseExited (pomeren je miš i kursor je izvan površine cilja)
- setOnMousePressed (pritisnuto je dugme miša kada je kursor pozicioniran na cilju)

7 Event filter

Omogućava obradu događaja tokom *capturing* faze. Čvor može imati jedan ili više filtera za obradu jednog događaja. Isti filter može da se koristi od strane više čvorova i za obradu više tipova događaja.
Namena filtera je da omoguće korenom čvoru da izvrši neki vid zajedničke obrade događaja za čvorove decu ili da uhvati događaj i time spreči čvorove decu da reaguju na događaj.

Filter je implementacija interfejsa `EventHandler` i obezbeđuje metod `handle()` za obradu događaja.

Registracija filtera

Registracija filtera za čvor vrši se metodom `addEventFilter()`. Prvi argument metoda je tip događaja, a drugi filter koji se registruje za dati čvor. Kada čvor „primi“ događaj datog tipa, izvršava se metod `handle()`.

Primer

```
// registracija filtera za cvor i dati tip dogadjaja
node.addEventFilter(MouseEvent.MOUSE_CLICKED,
                    new EventHandler<MouseEvent>() {
                        public void handle(MouseEvent) { ... };
                    });

// definicija filtera
EventHandler filter = new EventHandler(<InputEvent>() {
    public void handle(InputEvent event) {
        System.out.println("Filtering out event " + event.getEventType());
        event.consume();
    }
```

```
// registracija istog filtera za dva razlicita cvora
myNode1.addEventFilter(MouseEvent.MOUSE_PRESSED, filter);
myNode2.addEventFilter(MouseEvent.MOUSE_PRESSED, filter);

// registracija postojeceg filtera za drugi tip dogadjaja
myNode1.addEventFilter(KeyEvent.KEY_PRESSED, filter);
```

Sledeći kod:

```
new EventHandler<MouseEvent>() {
                        public void handle(MouseEvent) { ... };
                    }
```

podrazumeva da se istovremeno definiše nova klasa i pravi njena instanca. Radi se o **anonimnoj** klasi, klasi koja nema ime, ali ono što se zna jeste da ona implementira interfejs `EventHandle`. Iz tog razloga se nakon ključne reči `new` umesto konstruktora klase navodi ime interfejsa i odmah se u nastavku sama klasa definiše. Pošto klasa implementira interfejs `EventHandler` koji ima samo metod `handle`, definicija klase se sastoji u definisanju ponašanja ovog metoda.

Napomena

Filter registrovan za jedan tip događaja, može se koristiti i za bilo koji njegov podtip.

Uklanjanje filtera

Ukoliko se ne želi dalja primena filtera za obradu događaja odgovarajućeg čvora, isti se može ukloniti metodom `removeEventFilter()`.

```
// uklanja se filter za dati čvor i dati tip dogadjaja
myNode1.removeEventFilter(MouseEvent.MOUSE_PRESSED, filter);
```

Isti filter i dalje može da se primeni za obradu događaja čvora `myNode2`, kao i čvora `myNode1` za događaje tipa `KeyEvent.KEY_PRESSED`.

Upotreba filtera

Obično se primenjuju za čvorove-grane u okviru event dispath lanca. Koriste se obično za predefinisanje načina obrade događaja ili blokiranje da događaj stigne do svog cilja.

8 Event handler

Omogućava obradu događaja tokom *bubbling* faze. Čvor može imati jedan ili više handler objekata za obradu jednog događaja. Isti handler može da se koristi od strane više čvorova i za obradu više tipova događaja.
Namena handler objekata je da omoguće korenom čvoru da izvrši neki vid zajedničke obrade događaja za čvorove decu ili da reaguje na događaj ukoliko ga čvor-dete nije obradio.

Handler je implementacija interfejsa `EventHandler` i obezbeđuje metod `handle()` za obradu događaja.

Registracija handler objekta

Registracija za dati čvor vrši se metodom `addEventHandler()`. Prvi argument metoda je tip događaja, a drugi handler koji se registruje za dati čvor. Kada čvor „primi“ događaj datog tipa, izvršava se metod `handle()`.

Primer

```
// registracija event handler-a za dati cvor i dati tip dogadjaja
node.addEventHandler(DragEvent.DRAG_ENTERED,
                     new EventHandler<DragEvent>() {
                         public void handle(DragEvent) { ... };
                     });

// definicija event handler-a
EventHandler handler = new EventHandler(<InputEvent>() {
    public void handle(InputEvent event) {
        System.out.println("Handling event " + event.getEventType());
        event.consume();
    }

// registracija istog handler-a za dva razlicita cvora
myNode1.addEventHandler(DragEvent.DRAG_EXITED, handler);
myNode2.addEventHandler(DragEvent.DRAG_EXITED, handler);

// registracija handler-a za drugi tip dogadjaja
myNode1.addEventHandler(MouseEvent.MOUSE_DRAGGED, handler);
```

Napomena

Handler registrovan za jedan tip događaja, može se koristiti i za bilo koji njegov podtip.

Uklanjanje handler objekta

Ukoliko se ne želi dalja primena handler-a za obradu događaja odgovarajućeg čvora, isti se može ukloniti metodom `removeEventHandler()`.

```
// uklanja se handler za dati čvor i dati tip dogadjaja
myNode1.removeEventHandler(DragEvent.DRAG_EXITED, handler);
```

Isti filter i dalje može da se primeni za obradu događaja čvora `myNode2`, kao i čvora `myNode1` za događaje tipa `KeyEvent.KEY_PRESSED`.

Upotreba handler-a

Obično se primenjuju za čvorove-listove ili čvorove-grane u okviru event dispath lanca kada treba obezbediti isto ponašanje za sve čvorove decu.