GUI u Python – 2. termin

QPushButton

Klasa predstavlja dugme koje se može postaviti na widget.

Metoda	Opis
void **setDefault**(default)	Postavlja dugme kao „default“ dugme na dijalogu. Dugme reaguje na taster Enter.Parametar: **default** - bool
void **setCheckable**(checkable)	Postavlja dugme kao „checkable“ – ponašanje dugmeta da nakon pritiska ostaje utisnuto. Parametar: **checkable** - bool
bool **isChecked**()	Za checkable dugme vraća true ukoliko je utisnuto, false ukoliko nije.
void **setText**(text)	Postavlja tekst na dugmetu. Parametar: **text** - unicode
void **setShortcut**(sequence)	Postavlja prečicu za dugme. Parametar: **sequence** - PySide.QtGui.QKeySequence – sekvenca tastera
void **setIcon**(icon)	Postavlja ikonicu na dugme. Parametar: icon - **QIcon**
SIGNAL **clicked**	Ispaljuje se prilikom klika na dugme koje nije checkable.
SIGNAL **toggled**	Ispaljuje se prilikom promene stanja checkable dugmeta.

Dijalozi

Dijalozi predstavljaju kontejnerske prozore najčešće namenjene prikazu poruka ili unosu sadržaja. PySide poseduje određeni broj predefinisanih dijaloga:

- **QFileDialog** – dijalog namenjen odabiru fajlova/foldera. Metode:
  - getOpenFileName – dijalog za odabir fajla/foldera
  - getSaveFileName – dijalog za odabir fajla za čuvanje
- **QMessageBox** – dijalog za prikaz bilo kakve poruke korisniku. Metode:
  - critical – poruka greške
  - information – prikaz obaveštenja
  - question – prikaz dijaloga sa pitanjem
  - warning – prikaz dijaloga sa upozorenjem
- **QInputDialog –** dijalog za unos jedne vrednosti. Metode:
  - getText – unos teksta
  - getInt, getInteger – unos int vrednosti
  - getDouble – unos double vrednosti
  - getItem – izbor vrednosti iz liste

- QColorDialog – dijalog za odabir boje
- QPrintDialog – dijalog za štampu i postavke štampača
- QPageSetupDialog – podešavanje vezano za stranicu za štampu
- QWizard – dijalog za prikaz sekvence stranica (wizard) npr. prilikom instalacije
- QProgressDialog – za prikaz napretka sporijih procesa (progress bar)

- QPrintPreviewDialog – pregled pred štampu
- QFontDialog – dijalog za odabir fonta

Primer korišćenja dijaloga:
https://www.blog.pythonlibrary.org/2013/04/16/pyside-standard-dialogs-and-message-boxes/

Pored ovih vrsta dijaloga, moguće je kreirati korisničke dijaloge nasleđivanjem klase QDialog (**CustomDialogRetVal**).

Upravljanje rasporedom (Layout)

Raspoređivanje komponenti u GUI prozorima u PySide funkcioniše na dva načina:

- Apsolutno pozicioniranje – zadavanje pozicije i veličine svakom elementu u okviru prozora
- Layout containers – način za upravljanje automatskim pozicioniranjem i promenom veličine korišćenjem Qt layout klasa

Apsolutno pozicioniranje

Najjednostavniji način za upravljanje rasporedom sadržaja GUI prozora. Postiže se dodeljivanjem fiksne pozicije i veličine elemenata u GUI prozoru u pikselima. Pozicija elementa je pozicija gornjeg levog ugla elementa, u odnosu na gornji levi ugao roditeljskog prozora. Apsolutno pozicioniranje ima određene nedostatke:

- Preračunavanje pozicije i veličine za svaki element je veliki posao
- Promena veličine prozora promeniće raspored elemenata
- Elementi ne odgovaraju na promene stila
- Prilikom promene jezika ili fonta moguće je da postojeći sadržaj više ne može da stane u element
- Može izgledati različito za različite rezolucije

Layout containers

Korišćenjem ovih komponenti postižu se sledeća poboljšanja:

- Razumne podrazumevane veličine prozora
- Pozicioniranje child elemenata
- Upravljanje promenom veličine
- Automatska izmena pri izmeni sadržaja

U nastavku će biti predstavljeni Layout container-i u PySide

QBoxLayout (QVBoxLayout, QHBoxLayout)

Kreira red (kolonu) u kojem se nalaze “kutije” u koje se smeštaju elementi. Moguće je definisati smer u kojem će se vršiti popunjavanje:

- QBoxLayout.LeftToRight
- QBoxLayout.RightToLeft
- QBoxLayout.TopToBottom
- QBoxLayout.BottomToTop

Metoda	Opis
void **addWidget**(widget)	Ubacuje widget na kraj.Parametar: **widget** – QWidget ili bilo koja klasa naslednica
void **insertWidget**(index, widget)	Ubacuje widget na odabranu poziciju. Parametri: **index** – int **widget** – QWidget ili bilo koja klasa naslednica
void **addLayout** (layout)	Ubacuje layout na kraj.Parametar: **layout** –bilo koja klasa naslednica klase QLayout
void **insertLayout** (index, layout)	Ubacuje layout na odabranu poziciju. Parametri: **index** – int **layout** –bilo koja klasa naslednica klase QLayout
void **addStretch**(stretch)	Dodaje komponentu koja se ponaša kao opruga. Parametar: **stretch** – int, predstavlja faktor širenja opruge, **OPCIONI**
void **addSpacing** (size)	Dodaje prazninu fiksne veličine. Parametar: **size** - int
void setDirection(direction)	Postavlja pravac popunjavanja layout-a. Parametar: **direction** - PySide.QtGui.QBoxLayout.Direction

Moguće je koristiti i naslednice QBoxLayout (QVBoxLayout i QHBoxLayout) za koje je smer predefinisan

QGridLayout

Deli prozor na redove i kolone,

Metoda	Opis
void **addWidget**(widget, x, y, vspan, hspan)	Dodaje widget u x red, y kolonu, sa proširenjem na vspan redova, hspan kolona
void **addLayout** (layout, x, y, vspan, hspan,)	Dodaje layout komponentu x red, y kolonu, sa proširenjem na vspan redova, hspan kolona

QFormLayout

Raspoređuje komponente u dve kolone – labela i polje za unos – kao na standardnim formama za unos sadržaja.

Metode:

Metoda	Opis
void **addRow**(label, field)	Dodaje par labela-polje u naredni red
void **setWidget**(row, role, widget)	Dodaje widget (moze da se prosiri na obe kolone korišćenjem **role** parametra QFormLayout.SpanningRole)

QStackedLayout

Služi slaganju komponenti, pri čemu je samo jedna vidljiva u jednom momentu.

Metode:

- insertWidget
- removeWidget

MDI

MultipleDocumentInterface (MDI) predstavlja pristup pri izradi editora u kojem u okviru jedne instance aplikacije može postojati više otvorenih dokumenata. U PySide postoje dve komponente namenjene MDI radu – QmdiArea i QTabWidget.

QMdiArea

QMdiArea predstavlja GUI komponentu namenjenu radu sa internim prozorima. Najčešće se nalazi u okviru Central widget prostora QMainWindow instance.

Metode:

- addSubWindow – metoda za dodavanje novog prozora. Pri dinamičkom dodavanju prozora potrebno je pozvati show metodu widgeta koji se dodaje.
- activeSubWindow – vraća trenutno aktivni prozor
- removeSubWindow – zatvara prozor koji sadrži zadati widget
- setViewMode – definiše način na koji će prozori biti prikazani
- signal: subWindowActivated – označava da je određeni prozor dobio fokus

QTabWidget

QTabWidget komponenta predstavlja GUI komponentu namenjenu implementaciji MDI korišćenjem kartica (tab-ova).

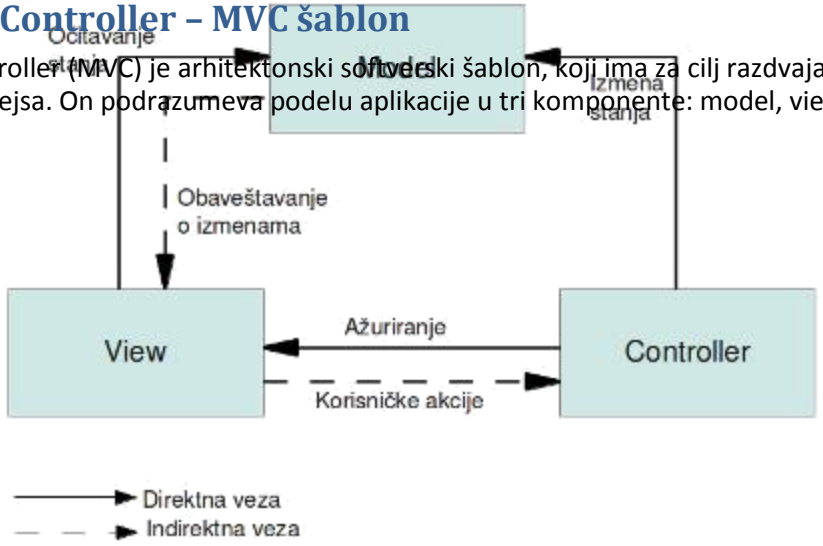
Metode:

- addTab – dodaje novi tab, sa prosleđenim widgetom i naslovom
- removeTab – uklanja tab na zadatoj poziciji
- currentWidget/currentIndex – vraća trenutno aktivni widget/poziciju aktivnog tab-a
- setTabText – postavlja naslov tab-a
- setTabsClosable – omogućava zatvaranje tab-ova (pojavi se dugme za zatvaranje na svakom)

- signal: tabCloseRequested – ispaljuje se prilikom poziva zatvaranja tab-a

Model View Controller – MVC šablon

Model View Controller (MVC) je arhitektonski softverski šablon, koji ima za cilj razdvajanje podataka od korisničkog interfejsa. On podrazumeva podelu aplikacije u tri komponente: model, view i controller.



Model predstavlja centralnu komponentu aplikacije, u kojoj se čuvaju podaci i logika domena (operacije nad podacima koje očuvavaju njihov integritet). Model treba da ima metode koje omogućavaju okruženju da pristupe njegovim podacima, kao i metode koje omogućavaju njihovu izmenu. Cilj MVC-a je da model bude nezavisan od view-a i controller-a. To znači da model ne sme da ima direktne veze, tj. reference na te komponente. Jedan model može istovremeno da učestvuje u više MVC trojki, tj. za jedan model može da bude vezano više view-controller parova. Problem održavanja konzistentnosti modela i korisničkog interfejsa (propagacija izmena) može da se reši na jedan od sledećih načina:

- Pasivni model: model je potpuno nesvestan okruženja, a za obaveštavanje view-a o promeni u modelu zaslužen je controller.
- Aktivni model: model obezbeđuje vezu sa okruženjem pomoću Observer šablona, pri čemu model igra ulogu observable komponente, koja obaveštava okruženje o izmenama kada se one dese.

View je zadužen za predstavljanje podataka modela u okviru korisničkog interfejsa. Zbog toga on najčešće ima direktnu vezu (referencu) na model, s tim što treba da je koristi samo za čitanje podataka, ne i za njihovu izmenu. Ako se koristi aktivni model, view se implementira kao njegov observer.

Controller komponenta šablona zadužena je za reagovanje na korisnički input, te primenu odgovarajućih izmena na modelu ili view-u. Ako se koristi pasivni model, nakon obavljene izmene controller obaveštava view o tome, kako bi on mogao da osveži prikaz.

View i controller zajedno čine korisnički interfejs. Njihova međusobna veza je obično najčvršća i često je u pitanju jedan objekat. Ako su razdvojeni, obično svakoj view komponenti odgovara jedan controller i obrnuto.

Benefiti koji se stiču upotrebom ovog šablona su višestruki:

1. Razdvojenost nadležnosti komponenti šablona omogućava njihovu lakšu i odvojenu implementaciju, održavanje i testiranje. Na primer, izmene u interfejsu koje se tiču načina prikaza ne zahtevaju nikakve izmene u modelu.
2. Jednostavno zamenljiv korisnički interfejs: pošto model ne zavisi od okruženja, za njega se mogu povezati različiti view-controller parovi, bez potrebe za izmenama u modelu. Na primer, MVC aplikacija koja radi u komandnoj liniji lako može da se nadogradi grafičkim korisničkim interfejsom.
3. Isti model može da ima više view-ova, tj. njegovi podaci mogu da se predstave na više različitih načina u okviru iste aplikacije.
4. Svi view-ovi nad istim modelom u isto vreme dobijaju obaveštenje o izmenama, te su njihovi prikazi uvek sinhronizovani.

Rukovanje događajima(slot-signal mehanizam)

Rukovanje dođajima ima važnu ulogu u GUI baziranim aplikacijama. Njihova uloga je da reaguju na određene akcije koje korisnik ili drugi sistem proizvodi (unost/izmena teksta, pritisak dugmeta, potvrdne akcje, zatvaranje aplikacije,…). Sve GUI aplikacije su event-driven aplikacije koje reaguju na razne događaje i u ovom modelu postoje tri aktera:

- event source
- event object
- event target

event source je objekat čije se stanje menja i on proizvodi događaj
event object je objekat koji enkaplusulira promene stanja u izvoru
event target je objekat koji se obaveštava o događaju

U PySide(QT) biblioteci ovaj mehanizam je implementiran kroz signal-slot mehanizam.
NAPOMENA:Ovaj mehanizam se razlikuje u verzijama 2 i 3 Python-a!

Signal se aktivira kada se određeni događaj desi(izazove), a *slot* se poziva kada god se *signal* za koji je povezan emituje. Slot može biti bilo koji Python callable.

Pored već postojećih, korisnik može sam definisati nove signale i slot-ove koje će biti pozvani u različitim slučajevima.

Primer:

```
…

def someFunc():
 print "someFunc has been called!"

button = QtGui.QPushButton("Call someFunc")
button.clicked.connect(someFunc)

…
```

Kreiranje signala

Pored postojećih signala, moguće je kreirati korisničke signale u našim klasama. Signal se kreira instanciranjem Signal klase uz eventualno prosleđivanje tipa parametra. Signal se poziva korišćenjem funkcije emit().

Primer:

```
from PySide.QtCore import QObject, Signal, Slot

class PunchingBag(QObject):
    ''' Represents a punching bag; when you punch it, it
        emits a signal that indicates that it was punched. '''
    punched = Signal()

    def __init__(self):
        # Initialize the PunchingBag as a QObject
        QObject.__init__(self)

    def punch(self):
        ''' Punch the bag '''
        self.punched.emit()
```

Zadatak:

Simulator kotla za grejanje se, između ostalog, sastoji od klase MeracPritiska sa metodama int uzmiPritisak() koja vraća tekući pritisak u kotlu i metodom podesiPritisk(int) sa kojom se postavlja tekući pritisak. Za prikaz pritiska zadužena je GUI klasa PritisakPrikaz. Takođe, sigurnosni ventil, predstavljen klasom SigurnosniVentil treba da automatski reaguje kada pritisak u kotlu pređe unapred zadatu vrednost i da se otvori.Korišćenjem Observer dizajn obrasca (mehanizam signala i slotova) simulirati rad kotla pri čemu će se iz test klase postavljati vrednost pritiska u kotlu dok će se pritisak prikazivati iz metoda klase PritisakPrikaz a otvaranje sigurnosnog ventila će biti prikazano iz metoda klase SigurnosniVentil. Svi prikazi treba da budu na konzoli.