

Objektno orijentirano programiranje

15.2: Grafičko sučelje – Swing

Dugotrajna obrada događaja grafičkog sučelja

Creative Commons

You are free to

- **Share** — copy and redistribute the material in any medium or format
- **Adapt** — remix, transform, and build upon the material

under the following terms

- **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **NonCommercial** — You may not use the material for commercial purposes.
- **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
- <https://creativecommons.org/licenses/by-nc-sa/4.0/>

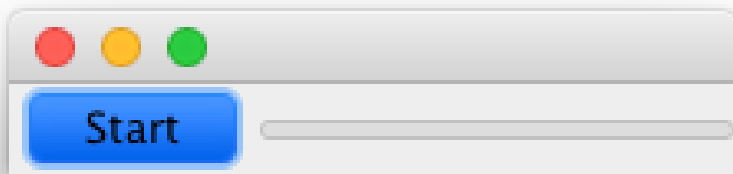


Swing i EDT

- Pristup i mijenjanje podataka grafičkih komponente [obavlja se / mora se obaviti] koristeći zasebnu grafičku dretvu (EDT).
 - Programski kod namijenjen interakciji s grafičkim sučeljem koji nije pokrenut iz EDT-a se prebacuje na EDT metodama *SwingUtilities.invokeLater* ili *SwingUtilities.invokeAndWait*
- Postoji samo jedna EDT!
 - obrada događaja se svodi na to da se *callback* metoda nekog sučelja (npr. *actionPerformed*) stavi na čekanje u red/rep izvođenja
 - obrada događaja ne smije biti dugačka jer se blokira ponovno iscrtavanje ekrana
 - demonstrirano primjerom na sljedećem slajdu

Primjer neispravnog prikaza napredovanja posla (1)

```
public class WrongWayBlocking extends JFrame {  
    JButton btnAction = new JButton();  
    JProgressBar pbProgress = new JProgressBar();  
  
    public WrongWayBlocking() {  
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);  
        pbProgress.setMaximum(10);  
        pbProgress.setMinimum(0);  
        btnAction.setText("Start");  
        btnAction.addActionListener((e) -> {  
            btnAction_actionPerformed(e); //vidi sljedeći slajd  
        });  
        add(btnAction, BorderLayout.WEST);  
        add(pbProgress, BorderLayout.CENTER);  
    }  
}
```

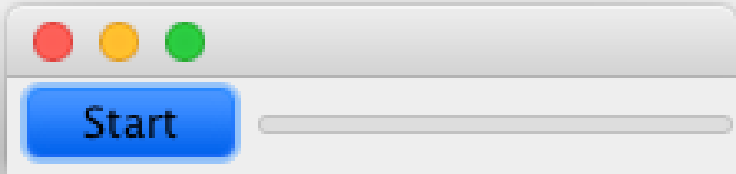


15_Swing/hr.fer.oop.swing2.events.WrongWayBlocking

Primjer neispravnog prikaza napredovanja posla (2)

```
public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
        WrongWayBlocking frame = new WrongWayBlocking();
        frame.pack();
        frame.setVisible(true);
    });
}
15_Swing/hr.fer.oop.swing2.events.WrongWayBlocking

private void btnAction_actionPerformed(ActionEvent e) {
    pbProgress.setValue(0);
    for (int i = 0; i <= 10; i++) {
        pbProgress.setValue(i);
        try { //zaustavi izvođenje na 500 milisekundi
            Thread.sleep(500);
        } catch (InterruptedException ie) {
        }
    }
}
```



Kako riješiti navedeni problem?

- Dugotrajno izvođenje prebaciti u zaseban programski odsječak koji se izvodi paralelno s našim programom
 - naredbu za pojedinačnu promjenu komponente grafičkog sučelja zakazati za izvođenje na EDT-u
 - zaseban paralelni (konkurentni) programski odsječak → nova dretva

zajednička memorija

Legenda:



- proces – aktivna jedinka (npr. pokrenuti program) sa skupom resursa
 - npr. Eclipse, Word, Outlook, video player...
- dretva (eng. thread) – nit izvođenja koja koristi resurse procesa unutar kojeg se izvodi
 - istovremeno tipkanje i provjera pravopisa u Wordu
 - pisanje koda i automatska kompilacija u Eclipseu
 - pisanje nove poruke i skidanje pošte u pozadini
 - slaganje liste uz istovremenu reprodukciju
- ne mora biti nužno vezano za grafičko sučelje

Stvaranje i pokretanje dretvi u Javi

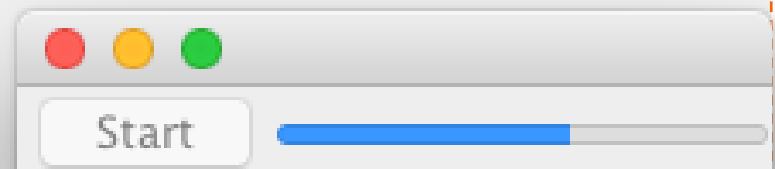
- Da bi se pokrenula dretva potrebno je nešto od sljedećeg:
 - napraviti vlastitu klasu koja nasljeđuje klasu `java.lang.Thread` (i nadjačati metodu `run` koja se poziva prilikom pokretanja dretve)
 - moguće koristiti anonimne klase ili lambda izraze
 - napraviti objekt klase koja implementira sučelje `Runnable` (implementirati metodu `run` koja se poziva kada se pokrene dretva), a zatim stvoriti objekt iz klase `Thread` kojem se u konstruktoru proslijedi `Runnable`)
- Dretva nastaje pozivom metode `start` nad objektom iz klase `Thread`
 - po pokretanju dretve poziva se metoda `run` (u objektu iz klase `Thread` ili o objektu iz klase koja implementira `Runnable`)
- Uočiti: dretva nije objekt instanciran iz klase `Thread`!
 - Metoda `run` nije dretva. Metoda `start` pokreće dretvu koja izvrši kod koji se nalazi u `run`

Primjer pokretanje dviju dretvi implementacijom sučelja Runnable

```
public class ThreadImplements implements Runnable {  
    @Override  
    public void run() {  
        for (int i = 0; i < 5; i++) {  
            System.out.println("broj " + i);  
        }  
        System.out.println("Gotovo brojanje");  
    }  
  
    public static void main(String[] args) {  
        Runnable r1, r2;  
        r1 = new ThreadImplements();  
        r2 = new ThreadImplements();  
        new Thread(r1).start();  
        new Thread(r2).start();  
    }  
}
```


Primjer ispravnog prikaza napredovanja posla

```
public class CorrectWay extends JFrame {  
    ...  
    private void btnAction_actionPerformed(ActionEvent e) {  
        btnAction.setEnabled(false);  
        pbProgress.setValue(0);  
        new Thread(() -> {                                15_Swing/hr.fer.oop.swing2.events.CorrectWay  
            for (int i = 0; i <= 10; i++) {  
                int progressValue = i;  
                SwingUtilities.invokeLater(() ->  
                    pbProgress.setValue(progressValue));  
            }  
            try {  
                Thread.sleep(500); //zaustavi izvođenje dretve na 500ms  
            } catch (InterruptedException ie) { }  
        }  
        SwingUtilities.invokeLater(() ->  
            btnAction.setEnabled(true));  
    }).start();  
}
```



Obrada zahtjevnih poslova u pozadinskoj dretvi

- Moguće rješenje pokazano u prethodnom primjeru s *ProgressBarom*
 - pokrenuti novu pozadinsku dretvu koja će odraditi zahtjevni posao
 - iz pozadinske dretve se ne smije direktno vršiti interakcija s grafičkim komponentama (tj. grafičkim korisničkim sučeljem)!
 - interakcija s grafičkim komponentama se iz pozadinske dretve ostvaruje dodavanjem odgovarajućih zadataka u rep EDT dretve
- Problem: pozadinska dretva ovako ne vraća rezultat, već je samo u interakciji s grafičkim korisničkim sučeljem
 - stoga se obrada zahtjevnih poslova u pozadini obavlja posredstvom apstraktne klase `SwingWorker`
 - u praksi treba naslijediti ovu klasu i napraviti implementaciju odgovarajućih metoda što je objašnjeno u nastavku

Klasa *SwingWorker* (1)

- Java uvodi klasu *SwingWorker* koja olakšava izvođenje poslova u pozadini

```
* @param <T> the result type returned by this {@code SwingWorker's}
*           {@code doInBackground} and {@code get} methods
* @param <V> the type used for carrying out intermediate results by this
*           {@code SwingWorker's} {@code publish} and {@code process} methods
*
* @since 1.6
*/
public abstract class SwingWorker<T, V> implements RunnableFuture<T> {
    protected abstract T doInBackground() throws Exception;
    public final T get() throws InterruptedException, ExecutionException
    {...}

    protected final void publish(V... chunks) {...}
    protected void process(List<V> chunks) {...}
    protected void done() {...}

    ...
}
```

Klasa *SwingWorker* (2)

- Metoda `doInBackground` se jednom izvodi na pozadinskoj dretvi i vraća rezultat obrade
- Rezultat se može dohvatiti i pozivom **blokirajuće** metode `get` (najčešće se to radi u metodi `done`)
- Završetkom pozadinskog posla metoda `done` se izvodi na dretvi event dispatch pa u njoj možemo vršiti interakciju s grafičkim komponentama
- Pozadinski posao može proizvoditi međurezultate
 - međurezultat(i) (*chunk(s)*) se objavljuje(u) pozivom metode `publish` (iz metode `doInBackground`)
 - objava međurezultata inicira buduće (automatsko) izvođenje metode `process` na EDT pa se i u njoj može vršiti interakcija s grafičkim komponentama

Zašto metoda *process* kao parametar ima listu?

- Metoda `publish` se može pozvati i nekoliko puta zaredom prije nego li se pozove metoda `process` na EDT
- Zato se međurezultati (koji čekaju na obradu u metodi `process`) slažu u listu
- Kad se pozove metoda `process` potrebno je obraditi sve međurezultate u listi

Primjer: izračun faktoriijela

- Prije pokretanja posla *CalculateFactorialTask* potrebno je postaviti početne vrijednosti na grafičkom sučelju

```
int number = Integer.parseInt(tfNumber.getText());  
progressBar.setValue(0);  
bCalculate.setEnabled(false);  
lResult.setText("");  
//schedule for execution on one of working threads  
new CalculateFactorialTask(number).execute();
```

~~15_Swing/hr.fer.oop.swing2.workers.FactorialCalculatorFrame~~

- Metoda `doInBackground` računa faktorijel predanog broja i pri tome pozivom metode `publish` dojavljuje napredak izračuna
- Metoda `process` ažurira stanje *ProgressBar*a na osnovu dojavljenog napretka
- Metoda `done` prikazuje rezultat izračuna i omogućava ponovni klik na gumb

Neovisnost *worker* o grafičkom sučelju (1)

- U primjeru s računanjem faktorijela, *worker* implementiran kao ugniježđena klasa koja direktno koristi kontrole grafičkog sučelja
- Što ako želimo isti *worker* koristiti na više grafičkih sučelja ili ga želimo implementirati prije nego stvorimo grafičko sučelje?
 - koristit ćemo već viđeni koncept funkcijskih sučelja (vidi primjer s traženjem najbližnjih automobila iz ranijih predavanja)
- Potrebna sučelja:
 - *Consumer<List<Integer>>* za obradu liste međurezultata u metodi *process*
 - *Runnable* za akciju nakon što *worker* završi
- Bit će opisano na primjeru *workera* koji računa *n* prostih brojeva, proizvodi listu prostih brojeva kao rezultat, dojavljuje pojedini izračunati prosti broj *i* ima mogućnost dojave napretka postupka

Neovisnost *workera* o grafičkom sučelju (2)

- Worker u konačnici proizvodi listu *Integera*, a pojedini međurezultat je *Integer*
 - worker mora znati što na kraju i što s međurezultatima (akumulirani u listu *Integera*)

```
public class PrimeNumbersWorker extends
    SwingWorker<List<Integer>, Integer> {
    private final int numberOfPrimes;
    private Consumer<List<Integer>> chunksProcessor;
    private Runnable onDone;
    public PrimeNumbersWorker(int numberOfPrimes,
        Consumer<List<Integer>> chunksProcessor, Runnable onDone) {

        this.numberOfPrimes = numberOfPrimes;
        this.chunksProcessor = chunksProcessor;
        this.onDone = onDone;
    } ...
}
```

~~15_Swing/hr.fer.oop.swing2.workers.properties.PrimeNumbersWorker~~

Neovisnost *worker*a o grafičkom sučelju (3)

- Kad je gotov ili spreman za procesiranje međurezultata *worker* pozove metodu *accept* iz *Consumer*a, odnosno *run* iz *Runnable*

```
public class PrimeNumbersWorker extends
    SwingWorker<List<Integer>, Integer> {
    private final int numberOfPrimes;
    private Consumer<List<Integer>> chunksProcessor;
    private Runnable onDone;

    @Override
    protected void process(List<Integer> chunks) {
        chunksProcessor.accept(chunks);
    }
    @Override
    protected void done() {
        onDone.run();
    }
}
```

~~15_Swing/hr.fer.oop.swing2.workers.properties.PrimeNumbersWorker~~

Neovisnost *worker*a o grafičkom sučelju (3)

- Glavni prozor određuje što će se konkretno obaviti kad *worker* dojaviti da je završio i kad producira listu međurezultata.

```
Runnable onDone = () -> bCalculate.setEnabled(true);

Consumer<List<Integer>> processChunks = chunks -> {
    for (int primeNumber : chunks) {
        textArea.append(primeNumber + "\n");
    }
};

SwingWorker worker = new PrimeNumbersWorker(number,
                                              processChunks, onDone);

...
worker.execute();
```

Dojavljivanje napretka s obradom međurezultata

- U prethodnom primjeru međurezultati korišteni za dojavu napretka
- Često želimo iskoristiti međurezultate za nešto korisno, a uz to ipak želimo dojavljivati i napredak
 - npr. u sljedećem primjeru međurezultati će biti izračunati prosti brojevi, a napredak predstavlja postotak obavljenog posla
- SwingWorker ima definirana dva svojstva (s *getterima* i *setterima*) koja se mogu „oslušivati” :
 - `int progress`
 - `StateValue state`

Svojstvo koje se može „oslušivati”

- Slično slušaču događaja nad grafičkom komponentom, ali ne mora biti nužno vezano za grafičko sučelje

`15_Swing/hr.fer.oop.swing2.workers.properties.console.*`

- Atributi implementirani tako da se prilikom promjene obavještavaju svi njegovi slušači
 - jedan od mehanizama evidencije slušača je klasa *PropertyChangeSupport* iz paketa *java.beans*
- Slušači promjene svojstava moraju biti objekti klase koja implementira sljedeće sučelje iz paketa *java.beans*:

```
public interface PropertyChangeListener extends  
    java.util.EventListener {  
    void propertyChange (PropertyChangeEvent evt) ;  
}
```

Dojavljivanje napretka s obradom međurezultata

- Potrebno postaviti slušača promjene svojstva `progress`

```
SwingWorker worker = new PrimeNumbersWorker(...
worker.addPropertyChangeListener(
    new PropertyChangeListener() {
        public void propertyChange(PropertyChangeEvent evt) {
            if ("progress".equals(evt.getPropertyName())) {
                progressBar.setValue((Integer) evt.getNewValue());
            }
        }
    }
);
worker.execute();
```

15_Swing/hr.fer.oop.swing2.workers.properties.PrimeNumbersCalculatorFrame

- Prilikom izračuna međurezultata je potrebno promijeniti vrijednost svojstva

```
setProgress(100 * count / numberOfPrimes);
```

Periodičko i/ili odgođeno okidanje različite vrste događaja

- Za *Swing* aplikacije se preporuča `javax.swing.Timer`, a ne općeniti `java.util.Timer`
 - Svi *Swing*ovi brojači vremena (*timers*) koriste posebnu dretvu *timer*
 - Ona ima sortirani rep u koji se slažu događaji po vremenu nailaska
 - Zadaci vezani uz GUI se tada automatski izvode na EDT
 - Ne moramo voditi brigu o sinkronizaciji
- *Swing*ov brojač vremena se može koristiti na dva načina
 - Za (odgođeno) izvođenje događaja jedanput (pozivom metode `setRepeats(false)` nad instancom brojača vremena)
 - Npr. *ToolTipManager* ga koristi za prikaz i sakrivanje tooltipa
 - Za periodičko izvođenje događaja
 - Primjer je ažuriranje napretka u regularnim intervalima

Dojavljivanje napretka u regularnim intervalima

- Potrebno postaviti brojač vremena koji će periodički očitavati svojstvo (umjesto slušača promjene svojstva *progress*)

```
worker = new PrimeNumbersWorker(...  
worker.execute();  
timer = new Timer(DELAY, new ActionListener() {  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        progressBar.setValue((Integer) worker.getProgress());  
        if (worker.getProgress() == 100)  
            timer.stop();  
    }  
});  
15_Swing/hr.fer.oop.swing2.workers.properties.PrimeNumbersCalculatorFrame2  
timer.setInitialDelay(INITIAL_DELAY);  
timer.start();
```

- Prilikom izračuna međurezultata je potrebno promijeniti vrijednost svojstva

```
setProgress(100 * count / numberOfPrimes);
```