

Zvyšok semestra

- riešenia Midtermu sú na gite, vyhodnotenie 20.4.
- Dnes začíname JavaFX – prečo JavaFX
 - končia automatické testy
 - odovzdávate .zip CELÉHO IntelliJ projektu od DU9-CV9
- Ako na JavaFx ?
- Projekty Java, 27.4.
- Quadterm 2, najneskôr 13.5.
- 16.5. prvá skúška, oprava Quad-Midtermov
- DV/CV cez VN nebude

Príprava na cvičenie

Návod, ako získať Fx...

Oracle v Java11 vyhodil JavaFX z distribúcie JDK11. Preto tento návod. Ak ale máte JDK ≥ 8 & JDK < 11 , ste zrejme v pohode. Testuj skôr. Alternatíva je doinštalovať si iné, staršie $8 \leq \text{JDK} < 11$.

V IntelliJ skúste File/New/Project/JavaFX/Project SDK11/Next, ProjectName: PokusnyFX, Finish.
V src/sample/Main.java ak vidíte more červeného a projekt vám nejde skompilovať (a spustiť), čítajte ďalej.
Ak vám projekt ide, alebo máte JDK <11 , alebo máte iné šťastie, nečítajte ďalej.

Inak:

1. Stiahnite si JavaFX SDK, podľa OS, a rozbalte .zip niekde u seba <https://gluonhq.com/products/javafx/>
Príklad: rozbalím do c:\Program Files\Java\javafx-sdk-11.0.2\, ak taký priečinok mám...
2. cez File/Project Structure/Project Settings/Libraries/+/Java nájdite **lib podadresár**, kde si si JavaFX SDK rozbalili (teda Príklad: c:\Program Files\Java\javafx-sdk-11.0.2\lib), OK. Library 'lib' will be added to the selected modules, vidíte tam meno projektu PokusnyFX, ťapnite na OK, ešte raz OK.
3. V src/sample/Main.java vám už javafx nesvieti červeno, skúste Run Main
4. ak dostanete podobnú smršť, pokračujte ďalším bodom

Exception in Application start method java.lang.reflect.InvocationTargetException

Caused by: java.lang.RuntimeException: Exception in Application start method

Caused by: java.lang.IllegalAccessError: class com.sun.javafx.fxml.FXMLLoaderHelper (in unnamed

Príprava na cvičenie

5.---- odtiaľto ďalej už môžete preskočením bodov 4..., a pokračovať, ak zrušíte celý podadresár sample/, a použijte niečo z cvičenia. Problém je FX-Loader, ktorý nebudeme používať.

6.Run/Edit Configurations nastavte v VM Options

7.--module-path **cesta-až-k-javafx-sdk-11/lib** --add modules=javafx.controls,javafx.fxml

8.Príklad: --module-path "c:\Program Files\Java\javafx-sdk-11.0.2\lib" --add-modules=javafx.controls,javafx.fxml

9.Apply, OK, Run.

10.Ak sa vám nezjaví okno s Hello World, skúste si návod prejsť ešte raz,

11.Ak ani potom, skúste <https://stackoverflow.com/questions/52467561/intellij-cant-recognize-javafx-11-with-openjdk-11>

12.Ak ani potom, ozvite sa so screen sharom, skúsime vám pomôcť.

Nezabudnite, táto zostava už neobsahuje junit testy, vaše riešenia budú hodnotené ručne.

Odovzdávajte vždy CELÉ zozipované projekty v IntelliJ tak, aby pri opravovaní stačilo naimportovať vaše riešenie (event. s obrázkami), skompilovať a pustiť. Žiadne ďalšie úpravy vášho projektu (presúvanie, dopĺňanie a hľadanie chýbajúcich častí projektu) od opravovateľa neočakávajú.

Neúplný, neskompilovateľný projekt sa nehodnotí.

Malá JavaFx aplikácia ilustruje, ako sa kreslia objekty (kruh, čiara, námorník, ...) do Panelu/Canvasu, odchyťávajú mouse eventy [HowtoWithFx](#).

<https://github.com/Programovanie4/Kod/tree/main/HowToWithJavaFX>

Vlákná a konkurentné výpočty

dnes bude:

- konkurentné vs. paralelné výpočty,
- vlákna (threads) vs. procesy,
- jednoduché simulácie, úvod do Java Fx

dnes nebudú (ťažké veci o vláknach):

- komunikácia cez rúry (pipes),
- synchronizácia a kritická sekcia (semafóry),
- deadlock

literatúra:

- [Thinking in Java, 3rd Edition](#), 13.kapitola,
- [Concurrency Lesson](#), resp. [Lekcia Súbežnosť](#),
- [Java Threads Tutorial](#),
- [Introduction to Java threads](#)
- [JavaFX 2.0: Introduction by Example](#)
- [Liang : Introduction to Java Programming !!!!Tenth Edition!!!](#) ☺

Cvičenia:

- simulácie konzolové či grafické (ak treba, použiť existujúci kód),
- napr. iné triedenie, iné guľičky, plavecký bazén, lienky na priamke, ...



Procesy a vlákna

- každý program v Jave obsahuje aspoň jedno vlákno (main thread)
- okrem užívateľom definovaných vlákien, runtime spúšťa tiež "neviditeľné" vlákna, napríklad pri čistení pamäte (multi-thread GC)
- pri aplikáciach, ktoré budú obsahovať GUI sa nezaobídeme bez vlákien
- každý bežný operačný systém podporuje vlákna aj procesy
- v prípade jedno/dvoj-procesorového systému OS musí zabezpečiť [**preemptívne**] prerozdelenie času medzi vlákna a procesy
- nepreemptívne plánovanie vymrelo s Win 3.0 a Win98/16bit
- na preemptívnom princípe *`každý chvíľku ťahá pítku'* vzniká konkukrentný výpočet miesto skutočne paralelného výpočtu,
- vlákna môžeme chápať ako jednoduchšie procesy (java má aj procesy)
- správu procesov riadi Task scheduler OS, kým vlákna riadi JVM

Čo nás čaká o vláknach

- vlákno je objekt nejakej podtriedy triedy Thread (`package java.lang.Thread`),
- vlákno vieme vytvoriť (`new Thread()` , `new SubTread()`),
- vlákno vieme spustiť (`metóda Thread.start()`),
- vláknu vieme povedať, čo má robiť (`v metóde run() {...}`),
- vlákno vieme pozastaviť (`Thread.yield()`) a dať šancu iným vláknam,
- vláknam vieme rozdať priority (`Thread.setPriority()`), akými bojujú o čas,
- vlákno vieme uspať na dobu určitú (`Thread.sleep()`),

Prvé pokusy o synchronizáciu:

- na vlákno vieme počkať, kým dobehne (`Thread.join()`),
- na vlákno vieme prerušiť (`Thread.interrupt()`).

Praktický pohľad na vlákna:

- programy s vláknami sa ťažšie ľadia,
- pri dvoch behoch rovnakého programu nemáme zaručené, že dôjde k rovnakej interakcii vlákien, ak však interagujú,
- ladenie chybnnej synchronizácie vlákien je náročné, lebo v debuggeri ťažko nasimulujete reálnu situáciu, pri ktorej vám program padá,
 - ak sa vám to podarí, že padne, máte šťastie, lebo aspoň viete, čo ladiť,
- vo všeobecnosti, na konkurentné výpočty nie sme veľmi trénovaní,
 - a celá pravda je, že sa to skoro vôbec neučí...

Vlákná na príkladoch



Krok za krokom:

- nasledujúci príklad vytvorí a spustí **15** vlákien,
- všetky vlákna sú podtriedou Thread,
- konštruktor **SimpleThread** volá konštruktor triedy Thread s menom vlákna,
- metóda getName() vráti meno vlákna,
- každé vlákno si v cykle počíta v premennej countdown od 5 po 0 (metóda run()),
- pri prechode cyklu vypíše svoj stav (metóda toString()),
- keď countdown = 0 metóda run() dobehne, život vlákna končí,
- aj keď si to priamo neuvedomujeme, vlákna zdieľajú výstupný stream System.out tým, že do neho „súčasne“ píšú.

Vytvorenie vlákna

```
public class SimpleThread extends Thread {
    private int countDown = 5;
    private static int threadCount = 0;
    public SimpleThread() {
        super("" + (++threadCount)); // meno vlákna je threadCount
        start();                     // naštartovanie vlákna run()
    }
    public String toString() {       // stav vlákna
        return "#" + getName() + ": " + countDown;
    }
    public void run() {              // toto vlákno robí, ak je spustené
        while(true) {
            System.out.println(this); // odpočítava od countDown
            if(--countDown == 0) return;
        }
    }
    public static void main(String[] args) {
        for(int i = 0; i < 15; i++)
            new SimpleThread();      // vytvorenie vlákna, ešte nebeží
    }
}
```

```
.....
#11: 5
#11: 4
#11: 3
#11: 2
#11: 1
#10: 5
#10: 4
#10: 3
#10: 2
#10: 1
#8: 5
#5: 5
#8: 4
#8: 3
#8: 2
#8: 1
#6: 3
#6: 2
#6: 1
#12: 4
#12: 3
#12: 2
#12: 1
#15: 5
#15: 4
#15: 3
#15: 2
#15: 1
```


Zat'azenie vlákna



- v predchádzajúcom príklade sme nemali pocit, že by vlákna bežali súbežne,
- lebo čas pridelený plánovačom k ich behu im postačoval, aby vlákno prebehlo a (skor) skončilo metódu run(),
- preto pridajme im viac roboty, príklad je umelý ale ilustratívny

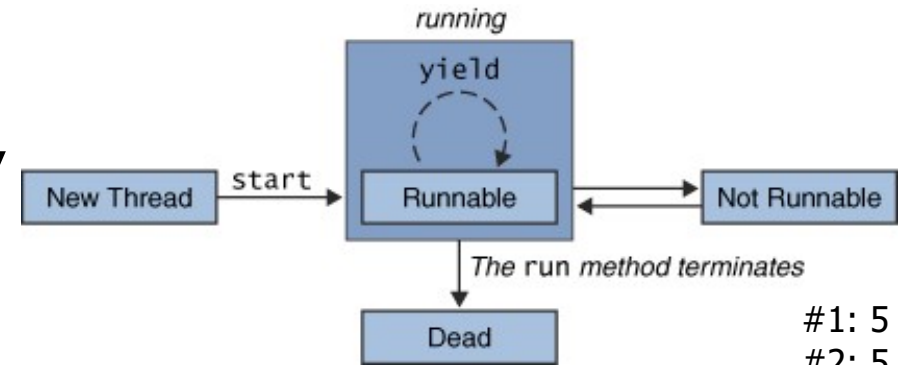
```
public void run() {  
    while(true) {  
        System.out.println(this);  
        for(int j=0; j<50000000; j++) {           // kým toto zráta  
            double gg = 0-Math.PI+j+j-j+Math.PI; // zapotí sa ...  
        }  
        if(--countDown == 0) return;  
    }  
}
```

#1: 5
#3: 5
#2: 5
#5: 5
#6: 5
#9: 5
#8: 5
#7: 5
#4: 5
#11: 5
#10: 5
#14: 5
#15: 5
#12: 5
#13: 5
#8: 4
#1: 4
#2: 4
#4: 4
#7: 4
#8: 3
#13: 4
#1: 3
#9: 4
#12: 4
#5: 4

- toto je jedna možnosť, ako pozdržať/spomaliť výpočet vlákna, ktorá však vyčerpáva procesor (pozrite si CPU load),
- ak chceme, aby sa počas náročného výpočtu vlákna dostali k slovu aj iné vlákna, použijeme metódu **yield()** – „daj šancu iným“, resp. nastavíme rôzne priority vlákien, vid' nasledujúce príklady

Pozastavenie/uvoľnenie vlákna

- metóda `yield()` zistí, či nie sú iné vlákna v stave pripravenom na beh (Runnable),
- ak sú, dá im prednosť.



```
public void run() {
    while(true) {
        System.out.println(this);
        for(int j=0; j<50000000; j++) {

            double gg = 0-Math.PI+j+j-j+Math.PI;
        }
        yield();
        if(--countDown == 0) return;
    }
}
```

// kým toto zráta
// zapotí sa ...

// daj šancu iným

Súbor: YieldingThread.java

- iná možnosť spočíva v nastavení priorít vlákien,
- pripomeňme si, že vlákna nie sú procesy na úrovni OS,
- plánovač vlákien pozná 10 úrovní priorít z intervalu `MAX_PRIORITY(10)`, `MIN_PRIORITY(1)`, ktoré nastavíme pomocou `setPriority(int newPriority)`

Súbor: [YieldingThread.java](#)

#1: 5
#2: 5
#3: 5
#4: 5
#5: 5
#8: 5
#11: 5
#6: 5
#10: 5
#13: 5
#9: 5
#14: 5
#15: 5
#12: 5
#7: 5
#3: 4
#11: 4
#8: 4
#4: 4
#2: 4
#10: 4
#9: 4
#3: 3
#12: 4
#5: 4
#15: 4

Priority vlákna

```
public class PriorityThread extends Thread {
    private int countDown = 5;
    private volatile double d = 0;    // d je bez optimalizácie
    public PriorityThread (int priority) {
        setPriority(priority);        // nastavenie priority
        start();                      // spustenie behu vlákna
    }
    public void run() {
        while(true) {
            for(int i = 1; i < 100000; i++)
                d = d + (Math.PI + Math.E) / (double)i;
            System.out.println(this);
            if(--countDown == 0) return;
        }
    }
    public static void main(String[] args) {
        new PriorityThread (Thread.MAX_PRIORITY); // #0=10
        for(int i = 0; i < 5; i++)
            new PriorityThread (Thread.MIN_PRIORITY); // #i=1
    }
}
```

```
#1: 5, priority: 1
#10: 5, priority: 10
#6: 5, priority: 6
#7: 5, priority: 7
#9: 5, priority: 9
#3: 5, priority: 3
#4: 5, priority: 4
#8: 5, priority: 8
#1: 4, priority: 1
#6: 4, priority: 6
#10: 4, priority: 10
. . . . .
#5: 4, priority: 5
#3: 2, priority: 3
#8: 2, priority: 8
#4: 2, priority: 4
#10: 1, priority: 10
done
#6: 1, priority: 6
done
#9: 1, priority: 9
done
#1: 3, priority: 1
#3: 1, priority: 3
done
#7: 1, priority: 7
done
#5: 3, priority: 5
#8: 1, priority: 8
done
#4: 1, priority: 4
done
#2: 5, priority: 2
#1: 2, priority: 1
```

Pozastavenie/uspanie vlákna

- zaťaženie vlákna (nezmyselným výpočtom) vyčerpáva procesor, potrebujeme jemnejšiu techniku,
- nasledujúci príklad ukáže, ako uspíme vlákno bez toho aby sme zaťažovali procesor nepotrebným výpočtom,
- vlákno uspíme na čas v milisekundách metódou `Thread.sleep(long millis)` throws `InterruptedException`,
- spánok vlákna môže byť prerušený metódou `Thread.interrupt()`, preto pre sleep musíme ošetriť výnimku `InterruptedException`,
- ak chceme počkať, kým výpočet vlákna prirodzene dobehne (umrie), použijeme metódu `Thread.join()`
- ak chceme testovať, či život vlákna bol prerušený, použijeme metódu `boolean isInterrupted()`, resp. `Thread.interrupted()`.

Uspatie vlákna

```
public class SleepingThread extends Thread {
    private int countDown = 5;
    private static int threadCount = 0;
    public SleepingThread() { ... .start(); }
    public void run() {
        while(true) {
            System.out.println(this);
            if(--countDown == 0) return;
            try {
                sleep(100); // uspi na 0.1 sek.
            } catch (InterruptedException e) { // výnimku musíme ochytiť
                throw new RuntimeException(e); // spánok bol prerušený
            }
        }
    }
    public static void main(String[] args) throws InterruptedException {
        for(int i = 0; i < 5; i++) {
            new SleepingThread().join(); // počkaj kým dobehne
            System.out.println("--");
        }
    }
}
```

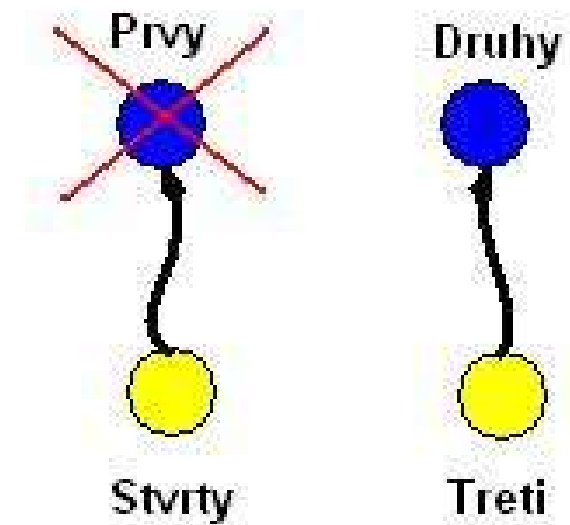
```
#1: 5
#1: 4
#1: 3
#1: 2
#1: 1
--
#2: 5
#2: 4
#2: 3
#2: 2
#2: 1
--
#3: 5
#3: 4
#3: 3
#3: 2
#3: 1
--
#4: 5
#4: 4
#4: 3
#4: 2
#4: 1
--
#5: 5
#5: 4
#5: 3
#5: 2
#5: 1
--
```

Čakanie na vlákno

- nasledujúci príklad vytvorí 4 vlákna,
- dva modré (Prvy, Druhy) triedy **Sleeper**, ktorý zaspia na 1.5 sek.
- ďalšie dva žlté (Treti, Stvrty) triedy **Joiner**, ktoré sa metódou **join()** pripoja na sleeperov a čakajú, kým dobehnú,
- aby vedelo vlákno triedy **Joiner**, na koho má čakať, konštruktor triedy **Joiner** dostane odkaz na vlákno (sleepera), na ktorého má čakať,
- medzičasom, výpočet vlákna Prvy násilne zastavíme v hlavnom vlákne metódou **interrupt()**.

// hlavný thread:

```
Sleeper prvy = new Sleeper("Prvy", 6000);  
Sleeper druhy = new Sleeper("Druhy", 6000),  
Joiner treti = new Joiner("Treti", druhy),  
Joiner stvrty = new Joiner("Stvrty", prvy);  
Thread.sleep(3000);  
prvy.interrupt();
```

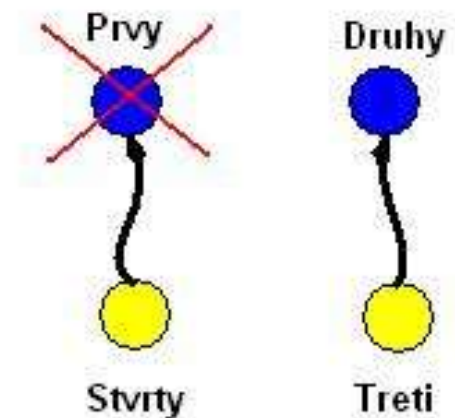


Čakanie na vlákno - Sleeper

```
class Sleeper extends Thread {  
    private int duration;  
    public Sleeper( String name,  
                    int sleepTime) {  
        super(name);  
        duration = sleepTime;  
        start();  
    }  
    public void run() {  
        try {  
            sleep(duration);  
        } catch (InterruptedException e) {  
            System.out.println(getName() + " preruseny");  
            return;  
        }  
        System.out.println(getName() + " vyspaty");  
    }  
}
```

Súbor: **Sleeper.java**

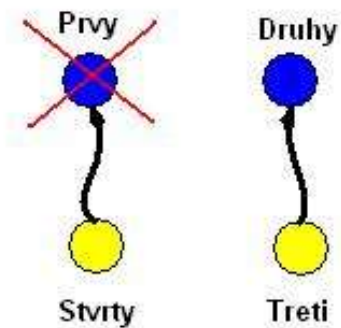
```
class Joiner extends Thread {  
    private Sleeper sleeper;  
    public Joiner(String name, Sleeper sleeper) {  
        super(name);  
        this.sleeper = sleeper;  
        start();  
    }  
    public void run() {  
        try {  
            sleeper.join();  
        } catch (InterruptedException e) {  
            throw new RuntimeException(e);  
        }  
        System.out.println(getName() + " dobehol");  
    }  
}
```



Súbor: [Sleeper.java](#)

Čakanie na vlákno - Joiner

```
class Sleeper extends Thread {  
    private int duration;  
    public Sleeper(String name, int sleepTime) {  
        super(name);  
        duration = sleepTime;  
        start();  
    }  
    public void run() {  
        try {  
            sleep(duration);  
        } catch (InterruptedException e) {  
            System.out.println(getName() + " preruseny");  
            return;  
        }  
        System.out.println(getName() + " vyspaty");  
    }  
}
```

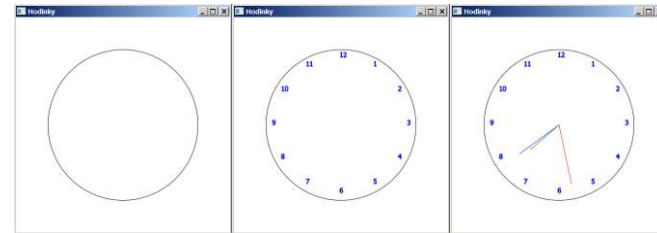


Prvy preruseny
Stvrtý dobehol
Druhý vyspaty
Tretí dobehol

```
class Joiner extends Thread {  
    private Sleeper sleeper;  
    public Joiner(String name,  
        Sleeper sleeper) {  
        super(name);  
        this.sleeper = sleeper;  
        start();  
    }  
    public void run() {  
        try {  
            sleeper.join();  
        } catch (InterruptedException e) {  
            throw new RuntimeException(e);  
        }  
        System.out.println(getName() + "  
        dobehol");  
    }  
}
```


Simulácie

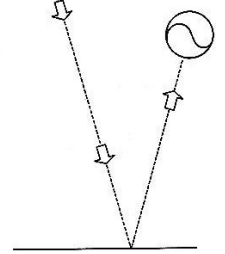
- simulovať konkurentné vlákna pomocou konzolovej aplikácie je prinajmenej málo farbisté a lákavé (aj napriek tomu, 1/2 takéhoto cvičenia si zajtra urobíme),
- preto potrebujeme nejaké grafické rozhranie,
- používame JavaFx,
- dnes JavaFx použijeme na zobrazenie simulácií, bez detailného úvodu, to príde
- JavaFx cez príklady – zo začiatku budete dopĺňať len chýbajúce kúsky kódu do pred-pripraveného projektu.



Kde začať:

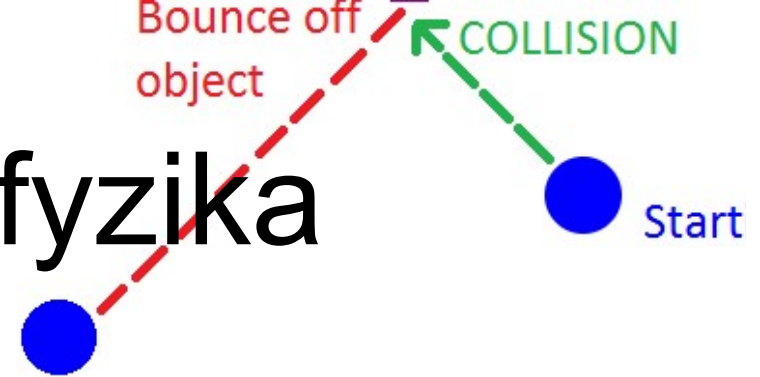
- [What Is JavaFX](https://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm)
<https://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm>
- [JavaFX 2.0: Introduction by Example](http://it-ebooks.info/book/399/)
<http://it-ebooks.info/book/399/>
- [Programování v JavaFX: úvod, příprava systému a prostředí](http://www.root.cz/clanky/programovani-v-javafx-uvod-priprava-systemu-a-prostredi/)
<http://www.root.cz/clanky/programovani-v-javafx-uvod-priprava-systemu-a-prostredi/>
- Liang : [Introduction to Java Programming, !!!!Tenth Edition](#)

Guličky v krabici



- nasledujúci príklad ilustruje simuláciu dvoch jednoduchých „procesov“,
- v krabici lietajú dve rôznofarebné guličky,
- každá z guľčiek je simulovaná jedným vláknom,
- toto vlákno si udržiava lokálny stav simulovaného objektu, t.j.
 - polohu, súradnice $[x, y]$,
 - smer, vektor rýchlosti $[dx, dy]$,
 - farbu, event. rýchlosť, ...
- metóda `run()` počíta nasledujúci stav (polohu, smer) objektu (guličky),
- treba k tomu trochu „fyziky“ (lebo uhol dopadu sa rovná uhlu odrazu),
- keďže strany krabice sú rovnobežne so súradnicami, stačí si uvedomiť, že
 - ak guľčička nenarazí, jej nová poloha je $[x+dx, y+dy]$,
 - ak guľčička narazí, zmení sa jej smerový vektor na $[-dx, -dy]$,
- po každom kroku simulácie si vlákno vynúti prekreslenie panelu, t.j. vlákno má odkaz na panel `Balls`,
- hlavný program len:
 - vytvorí obe vlákna a naštartuje ich,
 - vykreslí polohu/stav guľčiek (to musí vidieť ich súradnice, ktoré sú vo vláknach)

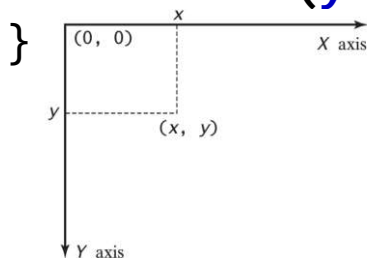
Vlákno guľičky - fyzika



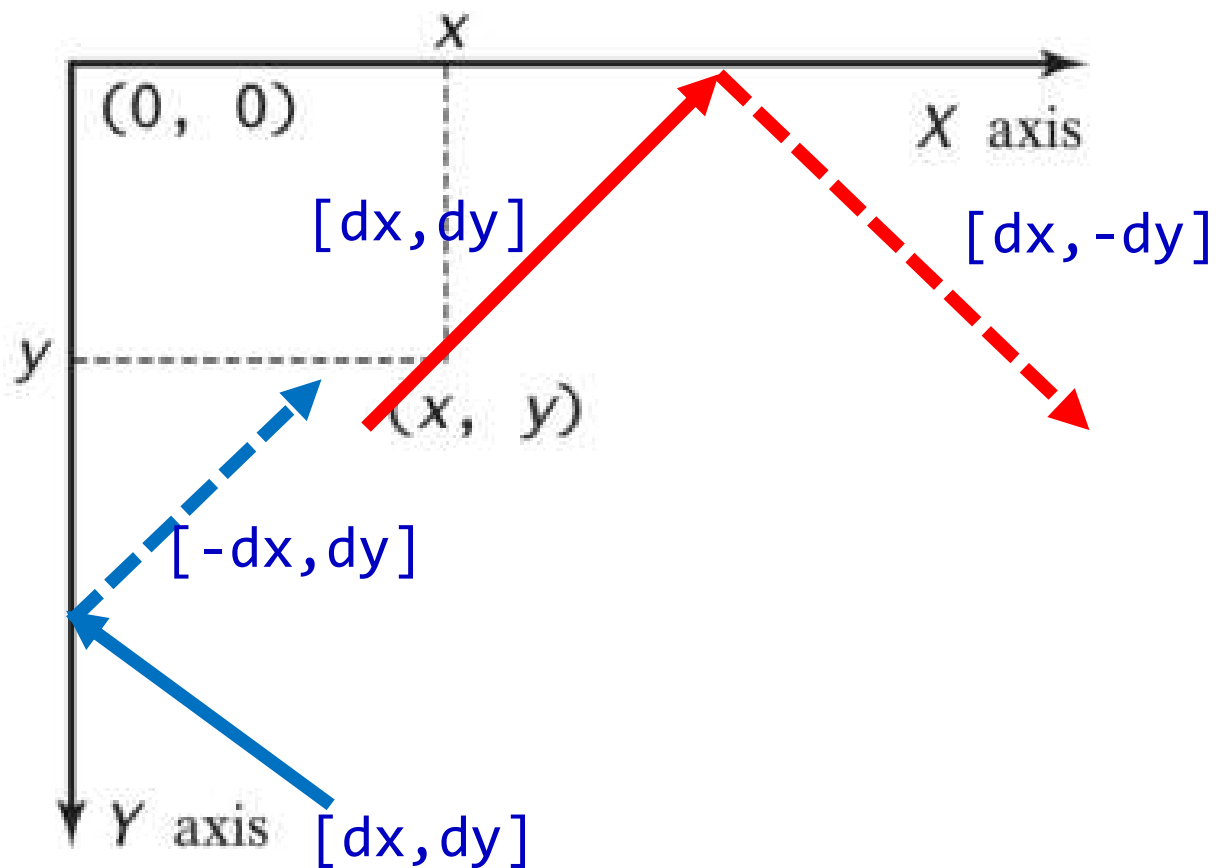
```
class BallThread extends Thread { // stav guľičky
    int x, y; // súradnice guľičky
    int dx, dy; // smerový vektor
    int size; // polomer guľičky
    int w, h; // veľkosť krabice, to potrebujem kvôli odrážaniu
    BallThreadPanel bp; // pane zodpovedný za vykresľovanie plochy s guľičkami

    public BallThread(BallThreadPanel bp, int x, int y, // konštruktor uloží všetko
        int dx, int dy, int size, int w, int h) { . . . }

    public void update(int w, int h) { // simulácia pohybu guľičky
        x += dx; // urob krok
        y += dy;
        if (x < size) dx = -dx; // odrážanie od stien ľavá
        if (y < size) dy = -dy; // horná
        if (x > w - size) dx = -dx; // pravá
        if (y > h - size) dy = -dy; // dolná
    } // simulácia má svoje rezervy v rohoch...
```



“fyzika”



Vlákno guľičky - prekresľovanie

Hlavný cyklus vlákna guľičky v nekonečnom cykle volá update, prekreslí plochu a pozastaví sa. Problém je, že GUI aplikácie beží v jednom vlákne, do ktorého iné vlákna **nesmú** zasahovať.

@Override

```
public void run() { // run pre Thread
    while (true) { // nekonečná simulácia
        update(w, h); // vypočítame novú polohu jednej guľičky
        try { // try-catch kvôli Thread.sleep
            Thread.sleep(10); // lebo aj sleep môže zlyhať, ??
            Platform.runLater(new Runnable() { // jedine takto
                @Override // môžeme meniť GUI aplikácie
                public void run() { // malý/krátky kúsok kódu
                    bp.paintBallPane(); // neblokuje GUI
                }
            });
            Platform.runLater(() -> bp.paintBallPane()); // JDK 8
        } catch (InterruptedException e) { // try-catch kvôli sleep
            e.printStackTrace();
        }
    }
}
```

Panel guľičiek – vytvorenie a spustenie vlákien

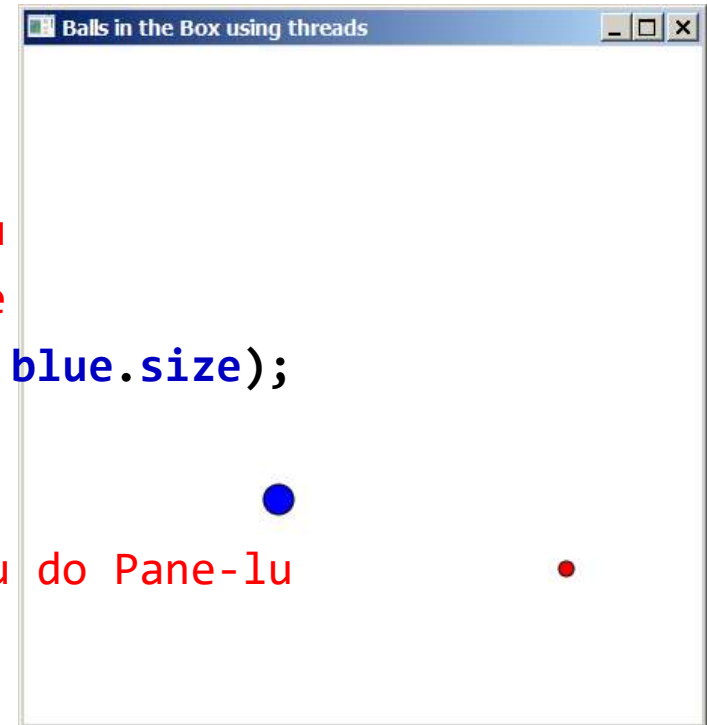
```
class BallThreadPanel extends Pane {           // Pane-1 je základný Fx komponent
    private int w = 450, h = 450;              // veľkosť panelu
    private BallThread red;                    // červená guľička
    private BallThread blue;                   // modrá guľička

    public BallThreadPanel() {                  // konštruktor Pane-1u
        Random rnd = new Random(); // náhodne x=[0,w],y=[0,h],dx,dy=[-1,0,1]
        red = new BallThread(this, rnd.nextInt(w), rnd.nextInt(h),
                                rnd.nextInt(3) - 1, rnd.nextInt(3) - 1, 5, w, h);
        blue = new BallThread(this, rnd.nextInt(w), rnd.nextInt(h),
                                rnd.nextInt(3) - 1, rnd.nextInt(3) - 1, 10, w, h);
        red.start(); // naštartovanie simulácie, de-facto sa
        blue.start(); // vytvorí vlákno a v ňom sa spustí metóda run()
    }

    // tragédia a občasná chyba, ak miesto .start() zavoláte .run()
    // syntakticky správne, ale NEvytvorí vlákno a spustí sa metóda run.
```

Panel guľičiek – kreslenie do panelu

```
class BallThreadPanel extends Pane {  
    ...  
    protected void paintBallPane() {  
        getChildren().clear(); // kreslenie do Pane-lu  
        if (blue != null) { // ak modrá už existuje  
            Circle blueR = new Circle(blue.x, blue.y, blue.size);  
            blueR.setFill(Color.BLUE); // plnka  
            blueR.setStroke(Color.BLACK); // čiara  
            getChildren().add(blueR); // pridanie Nodu do Pane-lu  
        }  
        if (red != null) {  
            Circle redR = new Circle(red.x, red.y, red.size);  
            redR.setFill(Color.RED);  
            redR.setStroke(Color.BLACK);  
            getChildren().addAll(redR);  
        }  
    }  
}
```



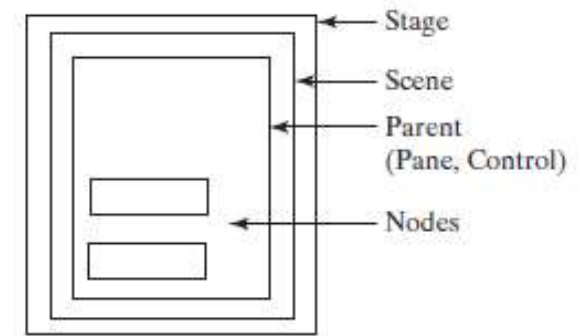
Hlavná scéna

```
import javafx.application.Application;
import javafx.application.Platform;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;

public class BallThreadFx extends Application {
    @Override
    public void start(Stage primaryStage) {
        BallThreadPanel balls = new BallThreadPanel();
        Scene scene = new Scene(balls, 450, 450);

        primaryStage.setTitle("Balls in the Box using threads");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

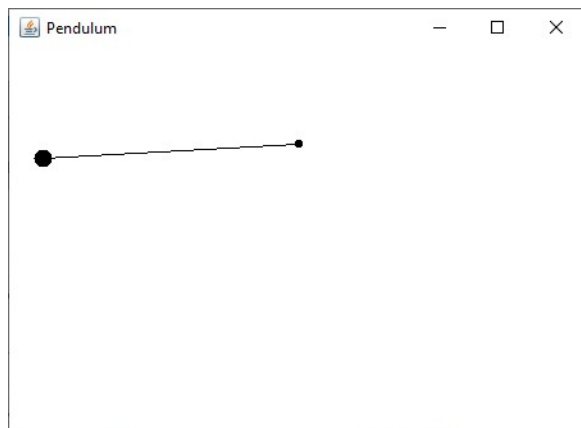
    public static void main(String[] args) {
        Launch(args);          // zavolá metódu start, sem nič nedopisujte
    }
}
```



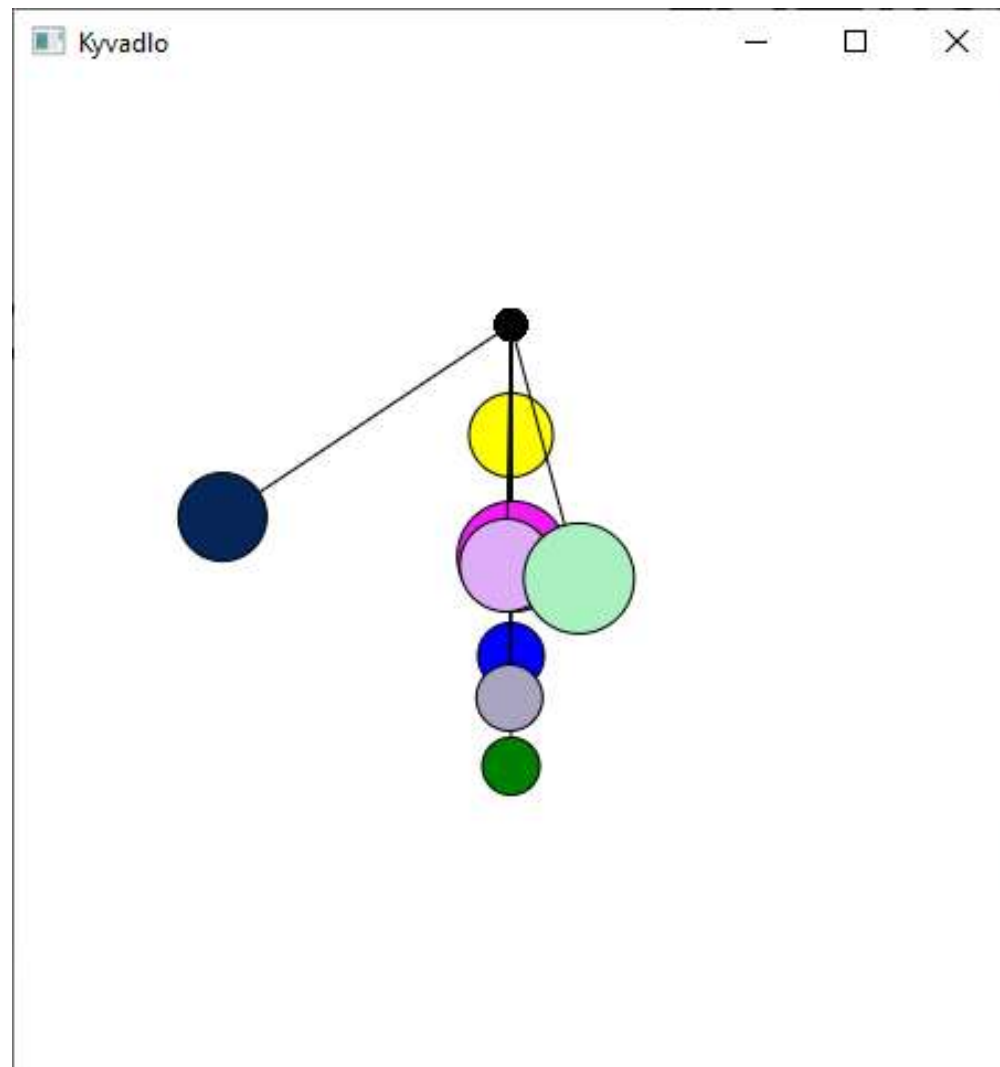
“fyzika”



Súbor: Pendulum.java

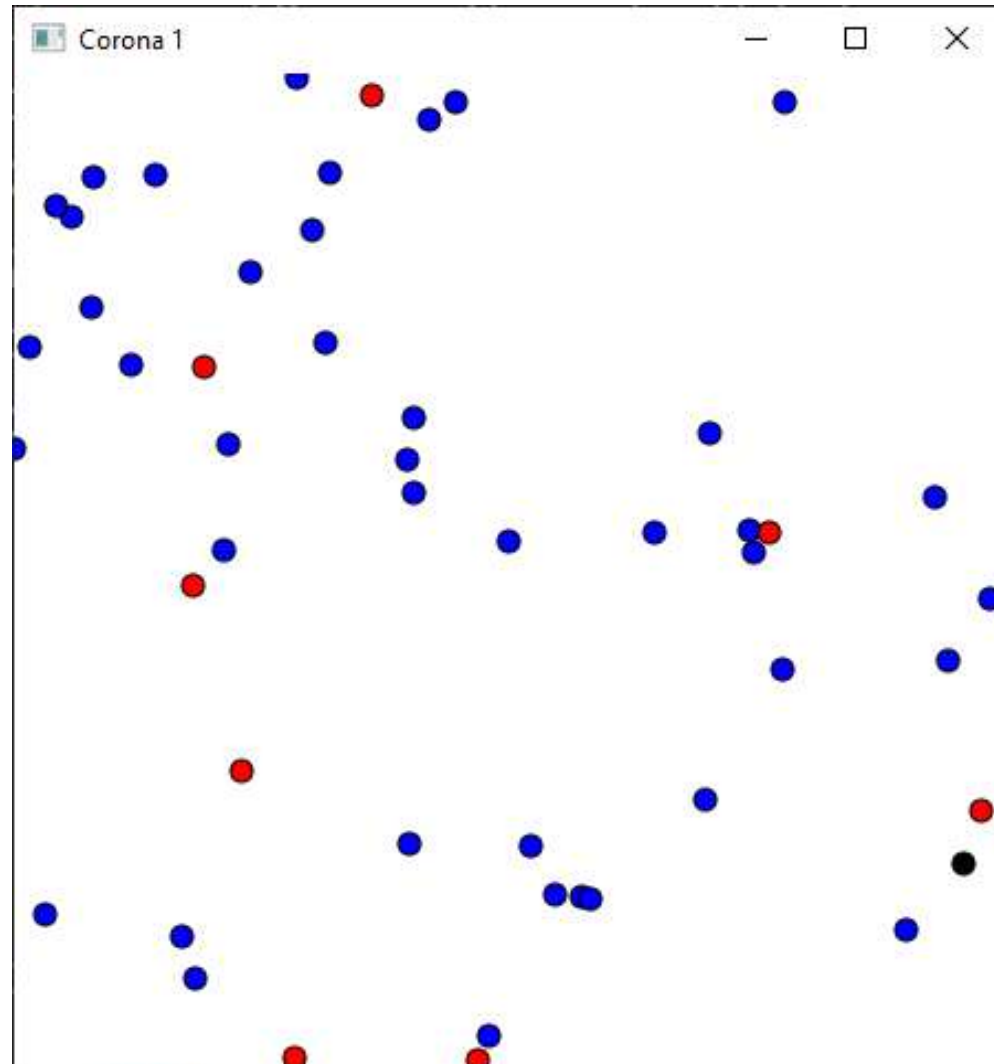


Súbor: PendulumSwing.java



Súbor: PendulumFx.java

Synchronizácia a GUI update



Animácia pomocou Timeline

(iná možnosť)

- Teraz `IdealBall` nie je vlákno, vlákno skrýva objekt triedy `Timeline`

```
class IdealBall {  
    int x, y, dx, dy, size;  
    public IdealBall(int x, int y, int dx, int dy, int size) { ... }  
    public void update(int w, int h) { ... } // analogicky ako predtým
```

- do `BallPane` pridáme `update`

```
class BallPane extends Pane {  
    public void update() { // guličky nie sú viac dva úplne nezávislé  
        red.update(w, h); // vlákna, ale jedno vlákno bude v každom kroku  
        blue.update(w, h); // updatovať krok červenej a krok modrej guličky  
    } // de-facto, to nie je to isté, aj keď vizuálny zážitok bude podobný
```

- Animácia v `start(Stage primaryStage):`

```
Timeline animation = new Timeline(new KeyFrame(Duration.millis(10), // 10ms.
```

```
    e -> { // λ funkcia – ide len v Java 1.8
```

```
        balls.update(); // každých 10ms. sa toto vykoná
```

```
        balls.paintBallPane();
```

```
    }));
```

```
Timeline animation = new Timeline(new KeyFrame
```

'_' should not be used as an identifier, since it is a reserved keyword from source level 1.8 on

```
animation.setCycleCount(Timeline.INDEFINITE); // a to do nekonečna
```

```
animation.play(); // štart animácie
```

Súbor: [BallFx.java](#)

λ-funkcia podrobnejšie

```
EventHandler<ActionEvent> evHandler = e -> {           // λ funkcia – ide len v Java 1.8
    balls.update();                                     // v tomto jednoduchom príklade hodnotu parametra e
    balls.paintBallPane();                             // nikde vo funkcii nepotrebuje...
};
Timeline animation = new Timeline(new KeyFrame(Duration.millis(10),evHandler));
```

```
//-----
```

```
EventHandler<ActionEvent> evHandler = new EventHandler<ActionEvent>() {
    @Override                                           // ide aj v < Java 1.8
    public void handle(ActionEvent e) {               // nešikovnejší ale rovnocenný zápis
        balls.update();
        balls.paintBallPane();
    }
};
Timeline animation = new Timeline(new KeyFrame(Duration.millis(10),evHandler));
//-----
animation.setCycleCount(Timeline.INDEFINITE);
animation.play();                                     // štart animácie
```

AnimationTimer

```
AnimationTimer at = new AnimationTimer() {
    @Override
    public void handle(long now) { // v nanosekundach, 10^9, mili,micro,nano
        if (now > LasttimeNano + 1_000_000_000) { // ak uplynie sekunda,
            System.out.println(frameCnt + " fps"); // tak vypis fps
            frameCnt = 0;
            LasttimeNano = now;
        }
        balls.update();
        balls.paintBallPane();
        frameCnt++;
    }
};
at.start();
```

60 fps
61 fps
61 fps
61 fps
61 fps
60 fps
61 fps
61 fps

AnimationTimer

```
AnimationTimer at = (now) → { // cas v nanosekundach, 10^9, mili,  
                             // micro, nanoseconds  
    if (now > lasttimeNano + 1_000_000_000) { // ak uplynie sekunda,  
                                                // tak vypis fps  
        System.out.println(frameCnt + " fps");  
        frameCnt = 0;  
        lasttimeNano = now;  
    }  
    balls.update();  
    balls.paintBallPane();  
    frameCnt++;  
};  
at.start();
```

60 fps

61 fps

61 fps

61 fps

61 fps

60 fps

61 fps

61 fps

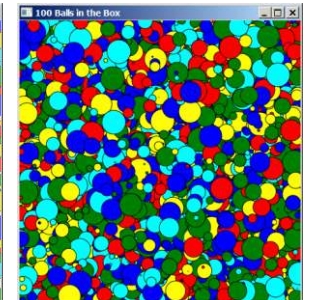
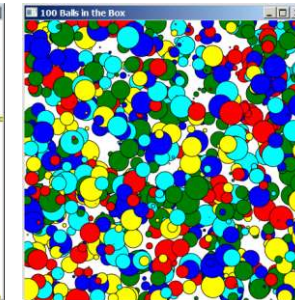
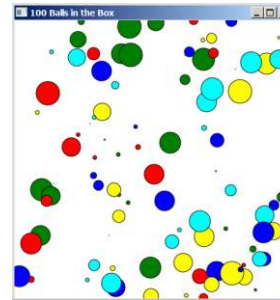
100, 1000, 10000 Balls

```

class BallPane2 extends Pane {
private ArrayList<IdealBall2> balls = new ArrayList<IdealBall2>();
final int SIZE = 100; // SIZE = 1000; SIZE = 10000;
Color[] cols = { Color.RED, Color.BLUE, Color.GREEN, Color.CYAN, Color.YELLOW };
public BallPane2() {
    Random rnd = new Random();
    for (int i = 0; i < SIZE; i++)
        balls.add(new IdealBall2(rnd.nextInt(w), rnd.nextInt(h),
            rnd.nextInt(3) - 1, rnd.nextInt(3) - 1,
            rnd.nextInt(20),
            cols[rnd.nextInt(cols.length)]));
}
public void update() {
    for (IdealBall2 b : balls) b.update(w, h);
}
protected void paintBallPane() {
    getChildren().clear();
    for (IdealBall2 b : balls) {
        Circle ci = new Circle(b.x, b.y, b.size);
        ci.setFill(b.c);
        ci.setStroke(Color.BLACK);
        getChildren().add(ci);
    }
}
}

```

// x,y
 // dx, dy
 // size
 // color



59 fps
61 fps
61 fps
60 fps
61 fps
61 fps

52 fps
61 fps
61 fps
61 fps
61 fps
61 fps

9 fps
18 fps
20 fps
20 fps
20 fps

Hra Bomba-Štít

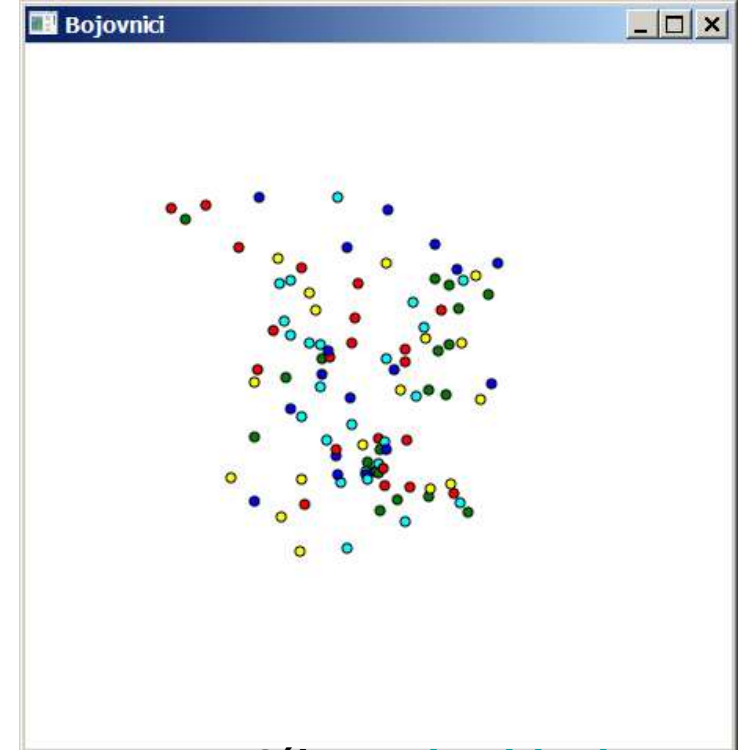
Hraje N ľudí, každý má určený jedného hráča ako **štít**, jedného ako **bombu**, pričom sa snaží postaviť tak, aby ho štít chránil pred bombou (t.j. boli v priamke)

```
class Playground extends Pane { // hlavný zobrazovaný pane-1
    final static int N = 100;
    BojovnikFx[] bojovnik; // pole všetkých bojovníkov
    Color[] cols = {Color.RED, Color.BLUE, Color.GREEN, Color.CYAN, Color.YELLOW};

    public Playground() {
        Random rnd = new Random();
        bojovnik = new BojovnikFx[N];
        for(int i=0; i<N; i++) // vytvorenie bojovníkov
            bojovnik[i] = new BojovnikFx(this, // čo bojovník to vlákno
                rnd.nextInt(w), rnd.nextInt(h), // náhodná pozícia na začiatok
                cols[rnd.nextInt(cols.length)]); // farba bojovníka pre efekt
        for(int i=0; i<N; i++) { // priradenie zabijáka a štítu
            bojovnik[i].zabijak(bojovnik[(i+1)%N]); // nasledujúci je bomba-killer
            bojovnik[i].stit(bojovnik[(i>0)?i-1:N-1]); // predchádzajúci je štít-
        } // -defender
        for(int i=0; i<N; i++) // spustenie všetkých vlákien
            bojovnik[i].start();
    }
}
```


Vykreslenie bojovníkov

```
class Playground extends Pane {  
    ...  
    protected void paintPlayground() {           // kreslenie  
        getChildren().clear();                   // zmaž všetky nody  
        for(int i=0; i<N; i++) {  
            Circle c = new Circle((int)Math.round(bojovnik[i].x),  
                                   (int)Math.round(bojovnik[i].y),3);  
            c.setFill(bojovnik[i].col);  
            c.setStroke(Color.BLACK);  
            getChildren().add(c);                 // pridaj  
        }  
    }  
}
```



Súbor: [BojovniciFx.java](#)

Správanie bojovníka

```
class BojovnikFx extends Thread { // lokálny stav bojovníka
    public double x,y; // jeho súradnice
    public Color col; // jeho farba
    BojovnikFx killer, defender; // kto je jeho bomba a štít
    Playground ap; // pointer na nadradený panel
    public BojovnikFx(Playground ap, int x, int y, Color col) {...} // konštruktor
    public void zabijak(BojovnikFx killer) { this.killer = killer; } // set killer
    public void stit(BojovnikFx defender) { this.defender=defender; } // set defender
    public void run() { // súradnice bodu, kam sa treba teoreticky postaviť, aby
        while (true) { // defender bol v strede medzi mnou a killerom
            double xx = 2*defender.x - killer.x;
            double yy = 2*defender.y - killer.y;
            double laziness = 0.1; // rovnica priamky, nič viac...
            x = (xx-x)*laziness+x; y = (yy-y)*laziness+y; // parameter lenivosti (0-1)
            Platform.runLater(new Runnable() { // ako rýchlo smerujem do xx,yy
                @Override // nové súradnice bojovníka
                public void run() { ap.paintPlayground(); } // prekreslenie
            });
            try { sleep(100); } catch(Exception e) {} // pozastavenie
        }
    }
}
```

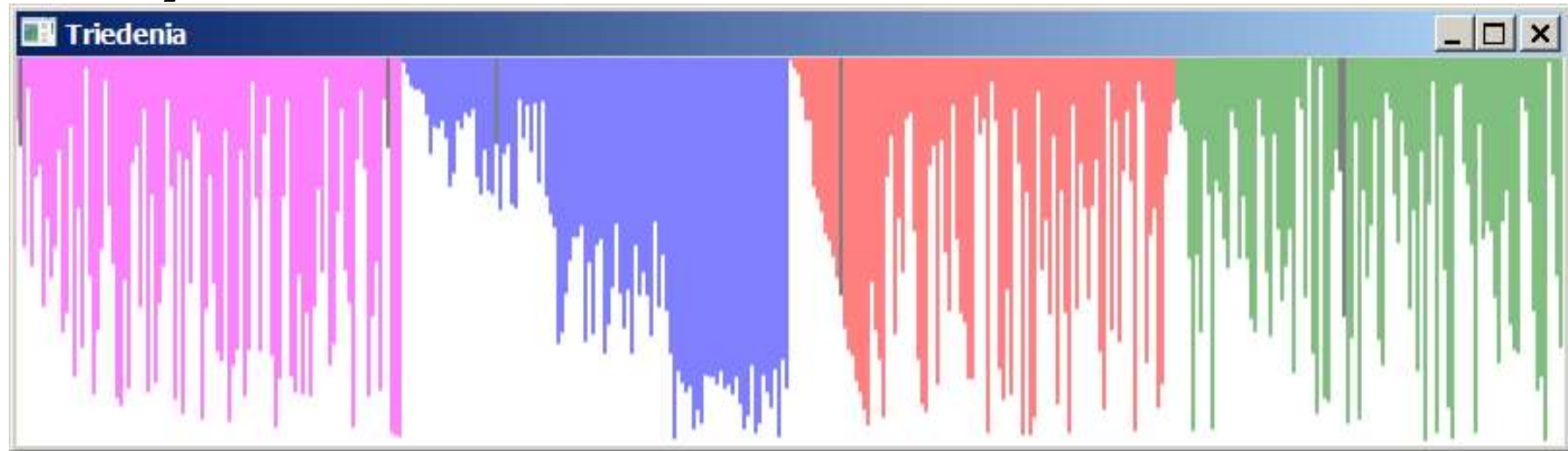
Preteky v triedení

- ďalší príklad je pretekom 4 triediacích algoritmov v java,
- hlavný panel je rozdelený na 4 panely (SortPanelFx extends Pane),
- každý SortPanelFx
 - náhodne vygeneruje (iné) pole na triedenie,
 - vytvorí vlákno triedy SortThreadFx a spustí,
 - poskytuje pre vlákno SortThreadFx metódu swap(i,j) – prvky i, j sa vymenili
 - vymenené paličky (hi, lo) znázorní čierno,
 - zabezpečuje vykresľovanie paličiek,
- SortThreadFx triedi vygenerované pole daným algoritmom (parameter "buble"),
- Random sort je jediný (len mne) známy algo triedenia horši ako bubblesort ☺

```
i = random(); j = random();
```

```
if (i < j && a[i] > a[j]) { int pom = a[i]; a[i] = a[j]; a[j] = pom; }
```

Sorty



```
public void start(Stage primaryStage) {  
    buble = new SortPanelFx("Buble", Color.MAGENTA);  
    quick = new SortPanelFx("Quick", Color.BLUE);  
    merge = new SortPanelFx("Merge", Color.RED);  
    random = new SortPanelFx("Random", Color.GREEN);  
    FlowPane flowpane = new FlowPane(buble, quick, merge, random); // vedľa seba  
  
    Scene scene = new Scene(flowpane, 800, 200); // vytvor scenu 4x200x200
```

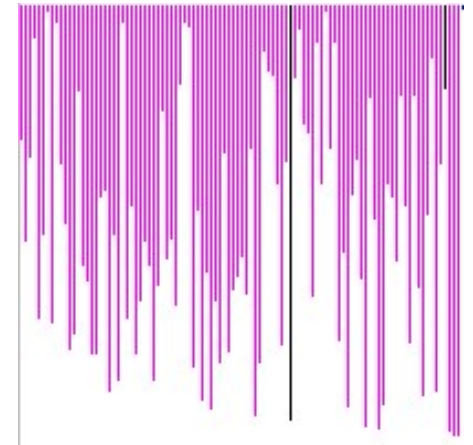
SortPanel

```
class SortPanelFx extends Pane {  
    private int[] a;    // triedené pole  
    private int lo, hi; // vymieňané prvky  
    private Color c;    // farba algo
```

```
    public SortPanelFx(String algo, Color col) { // konštruktor  
        this.c = col;    // zapamätá farbu  
        setPrefSize(200, 200); // nastaví veľkosť  
        a = new int[100];    // generuje pole  
        for (int i = 0; i < a.length; i++)  
            a[i] = (int) (200 * Math.random());  
        SortThreadFx thread = // vytvorí vlákno  
            new SortThreadFx(this, algo, a);  
        thread.start();    // naštartuje ho  
    }
```

```
    // public, poskytuje pre triediace algo  
    public void swap(int i, int j) {  
        lo = i; // zapamätá, ktoré paličky sme  
        hi = j; // práve vymieňali  
    }
```

```
    // kreslenie paličiek  
    public void paintSortPanel() {  
        getChildren().clear();  
        for (int i=0;i<a.length;i++) {  
            Line li =  
                new Line(2*i,a[i],2*i,0);  
            li.setStroke(  
                (i==lo || i==hi) ?  
                    Color.BLACK : c);  
            getChildren().add(li);  
        }  
    }
```



```

class SortThreadFx extends Thread {
SortPanelFx sPane;           // kto vie prekresliť Pane-l
String algo;                  // meno algo
int[] a;
public void run() {           // toto spustí .start()
    if (algo.equals("Buble")) bubbleSort(a);
    . . .
    else randomSort(a);
}
void swap(int i, int j) { // ak vymieňame paličky, tak treba prekresliť Pane-l
    sPane.swap(i, j);
    Platform.runLater(new Runnable() {           // prístup do GUI vlákna
        @Override
        public void run() { sPane.paintSortPanel(); } });
    try { sleep(10); } catch (Exception e) { }    // spomalenie animácie
}
void randomSort(int a[]) {           // samotný triediaci algoritmus
    while (true) {
        int i = (int) ((a.length - 1) * Math.random());
        int j = i + 1;
        swap(i, j);                  // tu znázorňujeme, ktoré prvky porovnávame
        if (i < j && a[i] > a[j]) {
            int pom = a[i];
            a[i] = a[j];
            a[j] = pom;
        }
    }
}
}

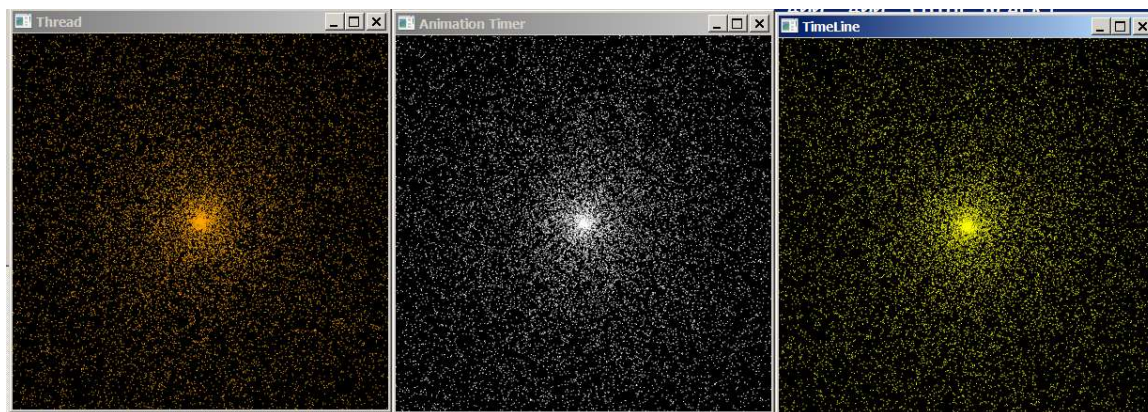
```

RandomSmileSort

SortThreadFx

MultiStage aplikácia

```
public void start(final Stage primaryStage) {  
    final Scene scene = new Scene(getAnimationPanel(), 400, 400, Color.BLACK);  
    primaryStage.setTitle("Animation Timer");  
    primaryStage.setScene(scene);  
    primaryStage.show();
```



```
    Stage stage = new Stage();  
    stage.setTitle("TimeLine");  
    stage.setScene(new Scene(getTimeLinePanel(), 400, 400, Color.BLACK));  
    stage.show();
```

```
    Stage thstage = new Stage();  
    thstage.setTitle("Thread");  
    thstage.setScene(new Scene(getThreadPanel(), 400, 400, Color.BLACK));  
    thstage.show();
```

```
}
```


ThreadPanel

runLater

// vyrobí a vráti Pane-1

```
public Pane getThreadPanel() {
```

```
    Rectangle[] nodes = new Rectangle[STAR_COUNT];
    double[] angles = new double[STAR_COUNT];
    long[] start = new long[STAR_COUNT];
    Pane p = new Pane();
    p.setPrefSize(w, h);
    for (int i = 0; i < STAR_COUNT; i++) {
        nodes[i] = new Rectangle(1, 1, Color.ORANGE);
        angles[i] = 2.0 * Math.PI * random.nextDouble();
        start[i] = random.nextInt(2000000000);
        p.getChildren().add(i, nodes[i]);
    }
```

```
    Thread th = new Thread() {
```

```
        public void run() {
```

```
            while (true) {
```

```
                final double centerW = 0.5 * w;
                final double centerH = 0.5 * h;
                final double radius = Math.sqrt(2) * Math.max(centerW, centerH);
```

```
                Platform.runLater(() -> { // ktorá, ak chce niečo do GUI
```

```
                    // tak musí zaradiť „malú“ rutinku Runnable
```

```
                    for (int i = 0; i < STAR_COUNT; i++) { // do event-dispatch fronty pomocou
```

```
                        final Node node = nodes[i]; // Platform.runLater()
```

```
                        final double angle = angles[i];
```

```
                        SandBoxFx.thnow -= 400;
```

```
                        final long t = (thnow - start[i]) % 2000000000;
```

```
                        final double d = t * radius / 2000000000.0;
```

```
                        node.setTranslateX(Math.cos(angle) * d + centerW);
```

```
                        node.setTranslateY(Math.sin(angle) * d + centerH);
```

```
                    });
```

```
                try { Thread.sleep(10); } catch (InterruptedException e) { ... }
```

```
            }
```

```
        };
```

```
        th.start();
```

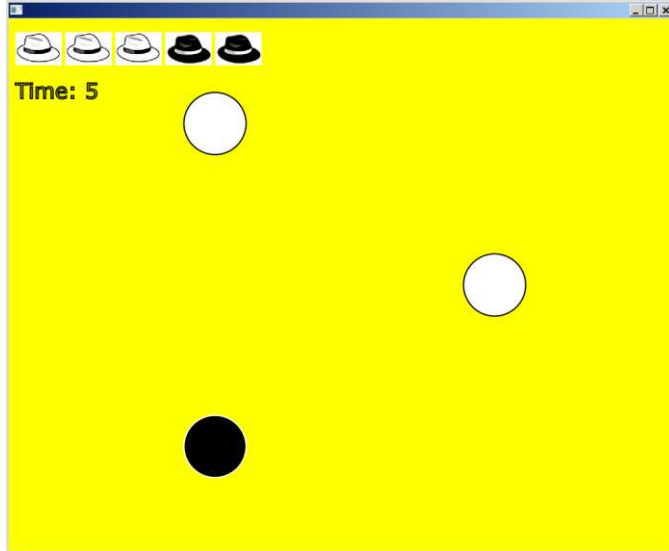
```
    return p;
```


Timeline

```
public Pane getTimeLinePanel() {                                // vyrobí a vráti Pane-1
    Rectangle[] nodes = new Rectangle[STAR_COUNT];
    double[] angles = new double[STAR_COUNT];
    long[] start = new long[STAR_COUNT];
    Pane p = new Pane();
    p.setPrefSize(w, h);
    for (int i = 0; i < STAR_COUNT; i++) {
        nodes[i] = new Rectangle(1, 1, Color.YELLOW);
        angles[i] = 2.0 * Math.PI * random.nextDouble();
        start[i] = random.nextInt(2000000000);
        p.getChildren().add(i, nodes[i]);
    }                                                            // ktorý vytvorí objekt Timeline
    Timeline tl = new Timeline(new KeyFrame(Duration.millis(40), e -> {
        final double centerW = 0.5 * w;                        // naprogramujeme EventHandler, napríklad ako λ funkciu
        final double centerH = 0.5 * h;
        final double radius = Math.sqrt(2) * Math.max(centerW, centerH);
        for (int i = 0; i < STAR_COUNT; i++) {
            final Node node = nodes[i];
            final double angle = angles[i];
            SandBoxFx.now -= 400;
            final long t = (now - start[i]) % 2000000000;
            final double d = t * radius / 2000000000.0;
            node.setTranslateX(Math.cos(angle) * d + centerW);
            node.setTranslateY(Math.sin(angle) * d + centerH);
        }
    } ));
    tl.setCycleCount(Timeline.INDEFINITE);
    tl.play();                                                  // timeline nezabudneme pustiť
    return p;
}
```

AnimationTimer

```
public Pane getAnimationPanel() {                                // vyrobí a vráti Pane-1
    Rectangle[] nodes = new Rectangle[STAR_COUNT];
    double[] angles = new double[STAR_COUNT];
    long[] start = new long[STAR_COUNT];
    Pane p = new Pane();
    p.setPrefSize(w, h);
    for (int i = 0; i < STAR_COUNT; i++) {
        nodes[i] = new Rectangle(1, 1, Color.WHITE);
        angles[i] = 2.0 * Math.PI * random.nextDouble();
        start[i] = random.nextInt(2000000000);
        p.getChildren().add(i, nodes[i]);
    }
    AnimationTimer at = new AnimationTimer() {                  // ktorý vytvorí objekt AnimationTimer
        @Override
        public void handle(long now) {                          // naprogramujeme metódu handle
            final double centerW = 0.5 * w;
            final double centerH = 0.5 * h;
            final double radius = Math.sqrt(2) * Math.max(centerW, centerH);
            for (int i = 0; i < STAR_COUNT; i++) {
                final Node node = nodes[i];
                final double angle = angles[i];
                final long t = (now - start[i]) % 2000000000;
                final double d = t * radius / 2000000000.0;
                node.setTranslateX(Math.cos(angle) * d + centerW);
                node.setTranslateY(Math.sin(angle) * d + centerH);
            }
        }
    };
    at.start();                                                  // a tiež ho treba pustiť
    return p;
}
```



Klobúky

(čo to má s programovaním)



- 3 biele a 2 čierne
- A, B, C si navzájom vidia farby klobúkov
- nesmú komunikovať, ale (aj tak) sú inteligentní 😊
- vyhrávajú, ak **všetci** uhádnu farbu svojho klobúka
- resp. ak sa jeden pomýli, prehrali všetci.

Hint: A,B,C sú spoluhráči, preto predpokladaj, že sú chytrí a myslí aj za nich

Hint: úloha nie je o šťastí=hádaní správneho riešenia

Späť ku concurrency

hodnotí sa najrýchlejšia odpoveď v chate

- jedna matka porodí dieťa za 9 mesiacov, za koľko dieťa porodí 9 matiek
- vojak vykrváca za 2 hodiny, za koľko hodín vykrváca čata 30 tich vojakov
- 3 mačky zjedia 3 myši za 3 hodiny, za koľko hodín zje 100 mačiek 100 myší