Laboration 7 (Datakom F5) – Returvärde från tråd

Syftet med laborationen är att du ska träna på att låta en tråd returnera värde. Jobba med laborationen tillsammans med någon eller några andra. Om du tar hjälp av facit så kontrollera noga hur lösningen fungerar.

Uppgift 1a – Callback

Använd filerna i mappen *l5alarm*. Klassen *AlarmThread* har en tråd vilken skriver ut ”Nu är det dags för alarm!” efter en viss tid. *DemoAlarm* skapar en instans av *AlarmThread* och startar tråden genom anrop till metoden *startAlarm()*. Se till att *AlarmThread*-objektet rapporterar att det är alarm genom ***Callback***:

1. Interfacet *AlarmListener* ska användas (är färdigt). Interfacet innehåller endast en metod: `public void alarm();`
2. Klassen *AlarmThread* ska ha en privat instansvariabel av typen *AlarmListener.*
3. Klassen *AlarmThread* ska ha en metod *addAlarmListener(AlarmListener listener)*. Genom anrop till denna metod kan en annan klass registrera en lyssnare.
4. Klassen *DemoAlarm* ska ha en inre klass, *AlarmPrinter*, vilken vid anrop till *alarm*-metoden ska skriva ut ”ALARM!” i console-fönstret. En instans av *AlarmPrinter* ska registeras i *AlarmThread* genom anrop till *addAlarmListener*-metoden.
5. Ersätt utskriften av ”Nu är det dags för alarm!” med anrop till *AlarmListener*-implementeringens *alarm*-metod.

Kör main-metoden i *DemoAlarm.* Efter 4 sekunder ska ”ALARM!” skrivas ut på konsolen.

Uppgift 1b – Callback

Skriv ytterligaren en inre klass i *DemoAlarm* vilken ska implementera *AlarmListener*. Ge den inre klassen namnet *WakeUpPrinter*. Då *alarm*-metoden anropas ska ”WAKE UP!” skrivas ut på konsolen. Lägg till en rad i *DemoAlarm* vilken registrerar en instans av *WakeUpPrinter* i *AlarmThread*.

Vad får du för utskrifter då du kör main-metoden i *DemoAlarm*?

Det blir endast en utskrift trots att två *AlarmListener*-implementeringar är registrerade (en *AlarmPrinter* och en *WakeUpPrinter*). Att det endast blir en utskrift beror på att instansvariabeln *listener* endast kan hålla referens till en lyssnare åt gången.

Ändra instansvariabeln till en LinkedList: `LinkedList<AlarmListener>`

Metoden *addAlarmListener* ska lägga till lyssnaren i *list*.

När det är dags för notifieringar ska *alarm*-metoden anropas för samtliga lyssnare i *list*.

Uppgift 1c – Callback

Skriv ytterligaren en inre klass, *ConsolePrinter*, i *DemoThread*. ConsolePrinter ska implementera *AlarmListener*.

Vid konstruktion av ett *ConsolePrinter*-objekt ska en sträng bifogas till konstruktorn. Det är denna sträng som ska skrivas ut vid alarm.

Registrera en instans av ConsolePrinter och kör programmet på nytt. Nu bör du få tre olika utskrifter vid alarm.

Uppgift 2 – PropertyChangeListener

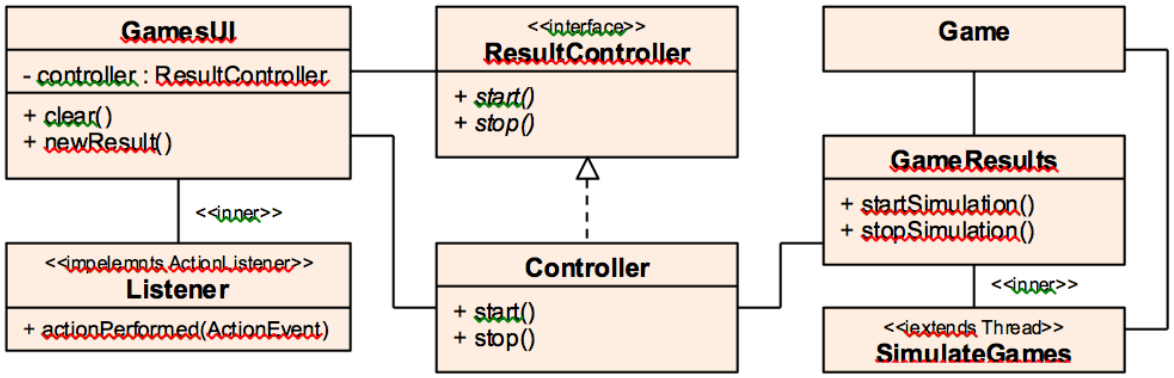
I den här uppgiften ska samma funktionalitet som i uppgift 1 implementeras men istället för **Callback** ska du använda **PropertyChangeListener** som finns i *java.beans* paketet.

Använd filerna i mappen *l5alarm_pcl*. Klassen *AlarmThread* har en tråd vilken skriver ut ”Nu är det dags för alarm!” efter en viss tid. *DemoAlarm* skapar en instans av *AlarmThread* och startar tråden genom anrop till metoden *startAlarm()*. Se till att *AlarmThread*-objektet rapporterar att det är alarm genom en ***PropertyChangeListener***:

1. I den här uppgiften ska interfacet *PropertyChangeListener* användas (från *java.beans*). Interfacet innehåller metoden:
   ```
   public void propertyChange(PropertyChangeEvent evt);
   ```
2. Klassen *AlarmThread* ska ha en privat instansvariabel av typ *PropertyChangeSupport* som används för att förvalta alla lyssnare.
3. Klassen *AlarmThread* ska ha en metod *addAlarmListener(PropertyChangeListener listener)* och en metod *removeAlarmListener(PropertyChangeListener listener)*. Genom anrop till dessa metoder kan en annan klass registrera eller ta bort en lyssnare. Klassen använder respektive metoder från *PropertyChangeSupport* (*addPropertyChangeListsner* och *removePropertyChangeListener*)
4. Klassen *DemoAlarm* ska ha en inre klass, *AlarmPrinter*, som implementerar *PropertyChangeListener* interfacet och vilken vid anrop till sin *propertyChange*-metod ska skriva ut ”ALARM!” på konsolen. En instans av *AlarmPrinter* ska registeras i *AlarmThread* genom anrop till *addAlarmListener*-metoden.
5. Ersätt utskriften av ”Nu är det dags för alarm!” med anrop till *PropertyChangeSupport*-implementeringens *firePropertyChange*-metod.

Kör main-metoden i *DemoAlarm.* Efter 4 sekunder ska ”ALARM!” skrivas ut på konsolen.

Uppgift 3a – PropertyChangeListener



Använd filerna i mappen *l5games*. Filen games.txt ska placeras i katalogen *files* i projektet. Klassen *GameResults* innehåller en tråd vilken rapporterar resultat i ett antal fotbollsmatcher.

Om du kör main-metoden i klassen Controller visar sig ett fönster. Med Start-knappen kan du starta resultatrapporteringen (utskrifter i console-fönstret) och med Stop-knappen stoppa resultatrapportering.

Din uppgift är att *GamesResults* rapporterar resultaten genom ***PropertyChangeListener***:

1. Klassen *GamesResult* ska implementera ett objekt av typ *PropertyChangeSupport* samt respektive metod *addPropertyChangeListener(PropertyChangeListener listener)* för att kunna registrera *PropertyChangeListener*. Varje gång det ska rapporteras ett resultat (nu utskrifter i tråden) ska metoden *firePropertyChange()* anropas. Välj ett lämpligt *propertyName* (t.ex. ”game”) och *game*-objektet som *newValue*. Som *oldValue* kan du skicka *null*.
2. Klassen *Controller* ska ha en inre klass som implementerar *PropertyChangeListener*. Implementationen ska publicera information från *Game*-objektet i *GamesUI*-objektet. Det är lämpligt att *GamesUI*-objektet anropas med UI-tråden genom metoden *SwingUtilities.invokeLater*:

```
SwingUtilities.invokeLater(new Runnable() {
    public void run() {
        resultUI.newResult(game.toString());
    }
});
```

   Klassen *Controller* måste se till att en instans av den inre klassen registreras som *Listener* i *GamesResults*.

När du är färdig och kör programmet ska resultaten visas i ”Game results”-fönstret.

Uppgift 3b – PropertyChangeListener

Lägg till en till inre klass i *Controller* som implementerar *PropertyChangeListener* och som skriver ut resultaten på konsolen. Kör programmet så att resultaten rapporteras både i game results-fönstret och i console-fönstret.

Uppgift 4 – ProductGenerator

Använd filerna i mappen *l5product*. Vid en multiplikation multipliceras två faktorer. Resultatet är produkten: *product = factor * factor*

Tre klasser är givna i uppgiften:

ProductGenerator
Innehåller en tråd vilken slumpmässigt söker efter två tal vilka multiplicerade med varandra blir en viss produkt. När ett sådant resultat hittas skrivs det ut i output-fönstret. Sökandet startas genom anrop till ***start***-metoden. Argumentet ska vara ett positivt heltal (produkten som söks).

Exempel: ***start(1025)*** kan ge utskrifter som

```
5*205=1025
205*5=1025
25*41=1025
1025*1=1025
1025*1=1025
205*5=1025
1*1025=1025
1025*1=1025
1025*1=1025
205*5=1025
```

ProductUI
Låter användaren starta och stoppa produkt-genereringen. Användaren ska ange sökt produkt innan Start-knappen används.

Controller
Instansierar en *ProductGenerator* och en *ProductUI*. Sköter kommunikationen mellan objekten i programmet.

Placera filerna i paketet *laboration5* och testkör programmet (*main* är i *Controller*).

Uppgift 4a – Callback

ProductGenerator-objectet ska inte skriva ut resultaten i Output-fönstret utan meddela via callback

Skapa ett lämpligt interface, t.ex. *ProductListener*. Låt *ProductGenerator* ha en instansvariabel av typen *ProductListener* och en metod där det går att registrera lyssnare.

Låt Controller-klassen registrera en *ProductListener*- implementering i *ProductGenerator*. Implementering ska visa resultaten i *ProductUI*-fönstret.

Uppgift 4b – Callback

Låt *Controller*-klassen registrera ytterligare en *ProductListener*- implementering i *ProductGenerator*. Implementering ska skriva resultaten, rad för rad till en textfil. Dock ska max 10 rader skrivas.

Vad krävs för att det ska gå att registrera ett antal *ProductListener*-implementeringar? Naturligtvis ska samtliga lyssnare meddelas så nya resultat kommer.

Lösningar

Uppgift 1a

```
public class AlarmThread {
    private Thread thread;
    private AlarmListener listener;
    private long ms;

    public AlarmThread(long ms) {
        this.ms = ms;
    }

    public void addListener(AlarmListener listener) {
        this.listener = listener;
    }

    public void startAlarm() {
        if(thread==null) {
            thread = new AT();
            thread.start();
        }
    }

    private class AT extends Thread {
        public void run() {
            try {
                Thread.sleep(ms);
            }catch(InterruptedException e) {

            }
            listener.alarm();
            thread = null;
        }
    }
}

-------------------------

public class DemoAlarm {
    public DemoAlarm(int ms) {
        AlarmThread bt = new AlarmThread(ms);
        bt.addListener(new AlarmPrinter());
        bt.startAlarm();
    }

    private class AlarmPrinter implements AlarmListener {
        public void alarm() {
            System.out.println("ALARM!");
        }
    }

    public static void main(String[] args) {
        DemoAlarm da = new DemoAlarm(4000);
    }
}
```

Uppgift 1bc

```
public class AlarmThread {
    private Thread thread;
    private LinkedList<AlarmListener> list = new LinkedList<AlarmListener>();
    private long ms;

    public AlarmThread(long ms) {
        this.ms = ms;
    }

    public void addListener(AlarmListener listener) {
        if(listener!=null)
            list.add(listener);
    }

    public void startAlarm() {
        if(thread==null) {
            thread = new AT();
            thread.start();
        }
    }

    private class AT extends Thread {
        public void run() {
            try {
                Thread.sleep(ms);
            }catch(InterruptedException e) {

            }
            for(AlarmListener al : list) {
                al.alarm();
            }
            thread = null;
        }
    }
}
-----------------------
public class DemoAlarm {
    public DemoAlarm(int ms) {
        AlarmThreadB bt = new AlarmThreadB(ms);
        bt.addListener(new AlarmPrinter());
        bt.addListener(new WakeUpPrinter());
        bt.addListener(new ConsolePrinter("Kilroy was here"));
        bt.startAlarm();
    }

    private class AlarmPrinter implements AlarmListener {
        public void alarm() {
            System.out.println("ALARM!");
        }
    }

    private class WakeUpPrinter implements AlarmListener {
        public void alarm() {
            System.out.println("WAKE UP!");
        }
    }

    private class ConsolePrinter implements AlarmListener {
        private String message;

        public ConsolePrinter(String message) {
            this.message = message;
        }

        public void alarm() {
            System.out.println(message);
        }
    }

    public static void main(String[] args) {
        DemoAlarm da = new DemoAlarm(4000);
    }
}
```

Uppgift 2

```
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeSupport;

public class AlarmThread {
   private Thread thread;
   private long ms;

   private PropertyChangeSupport change = new PropertyChangeSupport( this );

   public AlarmThread(long ms) {
      this.ms = ms;
   }

   public void addAlarmListener(PropertyChangeListener listener){
      change.addPropertyChangeListener(listener);
   }

   public void removeAlarmListener(PropertyChangeListener listener){
      change.removePropertyChangeListener(listener);
   }

   public void startAlarm() {
      if(thread==null) {
         thread = new AT();
         thread.start();
      }
   }

   private class AT extends Thread {
      public void run() {
         try {
            Thread.sleep(ms);
         }catch(InterruptedException e) {

         }
         System.out.println("Nu är det dags för alarm!");
         change.firePropertyChange("alarm", false, true);
         thread = null;
      }
   }
}

----------------

import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;

public class DemoAlarm {
   public DemoAlarm(int ms) {
      AlarmThread at = new AlarmThread(ms);
      at.addAlarmListener(new AlarmPrinter());
      at.startAlarm();
   }

   public static void main(String[] args) {
      DemoAlarm da = new DemoAlarm(4000);
   }

   private class AlarmPrinter implements PropertyChangeListener {
      @Override
      public void propertyChange(PropertyChangeEvent evt) {
         if( evt.getPropertyName().equals("alarm") )
            System.out.println("ALARM!");
      }
   }
}
```

Uppgift 3

```
public class GameResults {
    private ArrayList<Game> games = new ArrayList<Game>();
    private SimulateGames thread;

    private final PropertyChangeSupport changes = new PropertyChangeSupport(this);

    public void addPropertyChangeListener( PropertyChangeListener listener ){
            changes.addPropertyChangeListener(listener);
    }

    public GameResults(String filename) throws IOException {
        try (BufferedReader bw = new BufferedReader(new InputStreamReader(new
FileInputStream(filename),"UTF-8")) ) {
            String line = bw.readLine();
            String[] teams;
            while(line!=null) {
                teams = line.split(",");
                games.add(new Game(teams[0],teams[1]));
                line = bw.readLine();
            }
        }
    }

    public void startSimulation() {
        if(thread==null) {
            thread = new SimulateGames();
            thread.start();
        }
    }

    public void stopSimulation() {
        if(thread!=null) {
            thread.interrupt();
            thread = null;
        }
    }

    private class SimulateGames extends Thread {
        public void run() {
            int gameIndex, team;
            Random rand = new Random();
            Game game;
            while(thread!=null) {
                try {
                    Thread.sleep(1000);
                    team = rand.nextInt(2);
                    gameIndex = rand.nextInt(games.size());
                    game = games.get(gameIndex);
                    switch(team) {
                        case 0: game.increaseGoal1(); break;
                        case 1: game.increaseGoal2(); break;
                    }

                    changes.firePropertyChange("game", null, game);

                } catch(InterruptedException e) {
                    break;
                }
            }
        }
    }
}
```

Controller är på nästa sida

```
public class Controller implements ResultController {
    private GamesUI resultUI;
    private GameResults result;

    public Controller() {
        try {
            result = new GameResults("files/games.txt");
            resultUI = new GamesUI(this);
            showFrame(resultUI);
            result.addPropertyChangeListener(new WindowUpdater());
            result.addPropertyChangeListener(new ConsolePrinter());
            } catch(IOException e) {}
    }

    private void showFrame(JPanel panel) {
        JFrame frame = new JFrame("Game results");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(panel);
        frame.pack();
        frame.setVisible(true);
    }

    @Override
    public void start() {
        if(result!=null) {
            result.startSimulation();
        }
    }

    @Override
    public void stop() {
        if(result!=null) {
            result.stopSimulation();
        }
    }
    // Uppgift 3a
    private class WindowUpdater implements PropertyChangeListener{
      @Override
      public void propertyChange(PropertyChangeEvent evt) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                if(evt.getPropertyName().equals("game") &&
                    evt.getNewValue() instanceof Game) {

                    Game game = (Game)evt.getNewValue();
                    resultUI.newResult(game.toString());
                }
            }
        });
      }
    }
    // Uppgift 3b
    private class ConsolePrinter implements PropertyChangeListener{
      @Override
      public void propertyChange(PropertyChangeEvent evt) {
        if(evt.getPropertyName().equals("game") &&
                 evt.getNewValue() instanceof Game) {

          Game game = (Game)evt.getNewValue();
          System.out.println(game.toString());
        }
      }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                Controller controller = new Controller();
            }
        });
    }
}
```

Uppgift 4a

```
public interface ResultListener {
    public void newResult(Game game);
}

-----------------------------------------------------------------
public class ProductGenerator {
    private Multiplication thread;
    private int maxProduct;
    private ProductListener listener;

    public ProductGenerator(int maxProduct) {
        this.maxProduct = Math.abs(maxProduct);
    }

    public void addProductListener(ProductListener listener) {
        this.listener = listener;
    }

    public void start(int product) {
        if(thread==null) {
            if(product<0)
                product = (-product) % maxProduct;
            thread = new Multiplication(product);
            thread.start();
        }
    }

    public void stop() {
        if(thread!=null) {
            thread.interrupt();
        }
    }

    private class Multiplication extends Thread {
        private int product;

        public Multiplication(int product) {
            this.product = product;
        }

        public void run() {
            int factor1, factor2, res;
            Random rand = new Random();
            while(!Thread.interrupted()) {
                factor1 = rand.nextInt(product)+1;
                factor2 = rand.nextInt(product)+1;
                res = factor1*factor2;
                if(res==product && listener!=null) {
                    listener.newProduct(factor1+"*"+factor2+"="+product);
                }
            }
            thread=null;
        }
    }
}

-----------------------------------------------------------------
public class Controller {
    private ProductUI productUI=new ProductUI(this);
    private ProductGenerator productGenerator = new ProductGenerator(10000);

    public Controller() {
        JFrame frame = new JFrame("Game results");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(productUI);
        frame.pack();
        frame.setVisible(true);
        productGenerator.addProductListener(new PL());
    }
```

```
    // div metoder

    private class PL implements ProductListener {
        public void newProduct(final String str) {
            SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                    productUI.append(str);
                }
            });
        }
    }
}
```

Uppgift 4b

```
public class ProductGenerator {
    private Multiplication thread;
    private int maxProduct;
    private LinkedList<ProductListener> list = new LinkedList<ProductListener>();

    public ProductGenerator(int maxProduct) {
        this.maxProduct = Math.abs(maxProduct);
    }

    public void addProductListener(ProductListener listener) {
        if(listener!=null) {
            list.add(listener);
        }
    }

    public void removeListener(ProductListener listener) {
        list.remove(listener);
    }

    public void start(int product) {
        if(thread==null) {
            if(product<0)
                product = (-product) % maxProduct;
            thread = new Multiplication(product);
            thread.start();
        }
    }

    public void stop() {
        if(thread!=null) {
            thread.interrupt();
        }
    }

    private class Multiplication extends Thread {
        private int product;

        public Multiplication(int product) {
            this.product = product;
        }

        public void run() {
            int factor1, factor2, res;
            Random rand = new Random();
            while(!Thread.interrupted()) {
                factor1 = rand.nextInt(product)+1;
                factor2 = rand.nextInt(product)+1;
                res = factor1*factor2;
                if(res==product) {
                    for(ProductListener l : list)
                        l.newProduct(factor1+"*"+factor2+"="+product);
                }
            }
            thread=null;
        }
    }
}
```

```
public class Controller {
    private ProductUI productUI=new ProductUI(this);
    private ProductGenerator productGenerator = new ProductGenerator(10000);

    public Controller() {
        JFrame frame = new JFrame("Game results");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(productUI);
        frame.pack();
        frame.setVisible(true);
        productGenerator.addProductListener(new PL());
        productGenerator.addProductListener(new
FileLogger("files/product_log.txt"));
    }

    // div metoder och inre klasser

    private class FileLogger implements ProductListener {
        private BufferedWriter bw;
        private int rows = 0;
        private boolean ok = true;

        public FileLogger(String filename) {
            try {
                bw = new BufferedWriter(new FileWriter(filename));
            } catch(IOException e) {
                ok = false;
            }
        }

        public void newProduct(String str) {
            if(rows<10) {
                try {
                    rows++;
                    bw.write(str);
                    bw.newLine();
                    bw.flush();
                }catch(IOException e) {}
            } else {
                productGenerator.removeListener(this);
                try {
                    if(bw!=null) {
                        bw.close();
                    }
                }catch(IOException e) {}
            }
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                Controller controller = new Controller();
            }
        });
    }
}
```