# Laboration 5 – Returvärde från tråd

Syftet med labborationen 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 – Observer

Använd filerna i mappen *l5alarm*. Klassen *AlarmThreadA* har en tråd vilken skriver ut ”Nu är det dags för alarm!” efter en viss tid. *DemoAlarmA* skapar en instans av *AlarmThreadA* och startar tråden genom anrop till metoden *startAlarm()*.

Se till att *AlarmThreadA*-objektet rapporterar att det är alarm genom ***Observer***-mönstret:

5. Klassen *AlarmThreadA* ska ärva *Observable*. Varje gång det ska rapporteras ett resultat (nu utskrifter i tråden) ska metoderna *setChanged()* och *notifyObservers()* anropas.
5. Klassen *DemoAlarmA* ska ha en inre klass, AlarmPrinter, vilken ska implementera *Observer*. *update*-metoden ska skriva ut ”ALARM” i console-fönstret då den anropas. En instans av *AlarmPrinter* ska registreras i *AlarmThreadA*.
5. Ersätt utskriften av ”Nu är det dags för alarm!” med notifiering av *Observers*, dvs. genom anrop av setChanged() och notifyObservers().

Kör main-metoden i *DemoThreadA*. Efter ca 4 sekunder ska ALARM skrivas ut i console-fönstret.

## Uppgift 1b – Observer

Skriv ytterligaren en inre klass i *DemoAlarmA* vilken ska implementera *Observer*. Ge den inre klassen namnet *WakeUpPrinter*. Då *update*-metoden anropas ska ”WAKE UP” skrivas ut i console-fönstret.
Lägg till en rad i *DemoAlarmA* vilken registrera en instans av *WakeUpPrinter* i *AlarmThreadA*. Nu ska det vara två Observer-instanser registrerade att lyssna efter alarm.

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

## Uppgift 1c – Observer

Skriv ytterligaren en inre klass, *ConsolePrinter*, i *DemoThreadA*. *ConsolePrinter* ska implementera *Observer*.

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 2a – Callback

Använd filerna i mappen *l5alarm*. Klassen *AlarmThreadB* har en tråd vilken skriver ut ”Nu är det dags för alarm!” efter en viss tid. *DemoAlarmB* skapar en instans av *AlarmThreadB* och startar tråden genom anrop till metoden *startAlarm()*. Detta är samma som i Uppgift 1. Men nu ska du implementera samma funktionalitet som i Uppgift 1 men med hjälp av ***Callback***.

1. Interfacet *AlarmListener* ska användas (är färdigt). Interfacet innehåller endast en metod,
   `public void alarm();`
2. Klassen *AlarmThreadB* ska ha en instansvariabel av typen *AlarmListener,*
   `private AlarmListener listener;`
3. Klassen *AlarmThreadB* ska ha metoden *addAlarmListener(ActionListener listener)*. Genom anrop till denna metod kan en annan klass registrera en lyssnare.
4. Klassen *DemoAlarmB* 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 *AlarmThreadB* 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 *DemoAlarmB*. Efter ca 4 sekunder ska ALARM skrivas ut i console-fönstret.

## Uppgift 2b – Callback

Skriv ytterligaren en inre klass i *DemoAlarmB* vilken ska implementera *AlarmListener*. Ge den inre klassen namnet *WakeUpPrinter*. Då *alarm*-metoden anropas ska ”WAKE UP” skrivas ut i console-fönstret.
Lägg till en rad i *DemoAlarmB* vilken registrerar en instans av *WakeUpPrinter* i *AlarmThreadB*.

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

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:
```
private LinkedList<AlarmListener> list = new 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*.

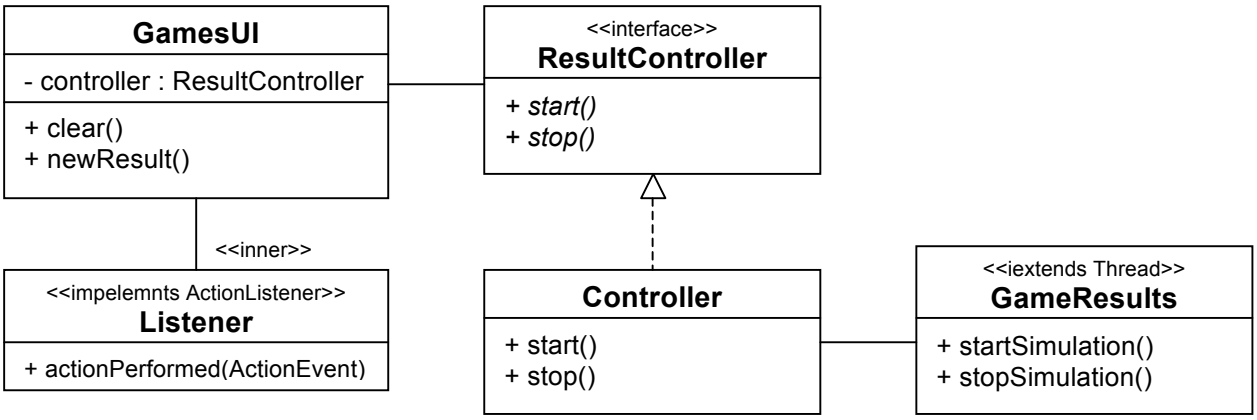
## Uppgift 2c – Callback

Skriv ytterligaren en inre klass, *ConsolePrinter*, i *DemoThreadB*. 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 3a – Observer




Använd filerna i mappen *l5games*. Filen games.txt ska placeras i katalogen *files* i projektet. Klassen *GameResult* är en tråd vilken rapporterar resultat i ett antal forbollsmatcher.

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 se till att *GamesResult*-objektet rapporterar resultat genom ***Observer***-mönstret till registrerade *Observers*:

1. Klassen *GamesResult* ska ärva *Observable*. Varje gång det ska rapporteras ett resultat (nu utskrifter i tråden) ska metoderna *setChanged()* och *notifyObservers(Game-objekt)* anropas.
2. Klassen *Controller* ska ha en inre klass vilken implementerar *Observer*. *Observer*-implementationen ska publicera information från *Game*-objektet i *GamesUI*-objektet. Det är lämpligt att GamesUI-objektet anropas med UI-tråden genom att anropa 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 *Observer* i *GamesResult*.

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

## Uppgift 3b – Observer

Lägg till en *Observer*-implementering vilken skriver ut resultat i Console-fönstret. Kör programmet så att resultat 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 muliplicerade 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 1

```
public class AlarmThreadA extends Observable {
    private Thread thread;
    private long ms;

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

    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) {

            }
            setChanged();
            notifyObservers();
            thread = null;
        }
    }
}

public class DemoAlarmA {
    public DemoAlarmA(int ms) {
        AlarmThreadA at = new AlarmThreadA(ms);
        at.addObserver(new AlarmPrinter());
        at.addObserver(new WakeUpPrinter());
        at.addObserver(new ConsolePrinter("Kilroy was here"));
        at.startAlarm();
    }
    // Uppgoft 1a
    private class AlarmPrinter implements Observer {
        public void update(Observable o, Object arg) {
            System.out.println("ALARM");
        }
    }
    // Uppgift 1b
    private class WakeUpPrinter implements Observer {
        public void update(Observable o, Object arg) {
            System.out.println("WAKE UP");
        }
    }
    // Uppgift 1c
    private class ConsolePrinter implements Observer {
        private String message;

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

        public void update(Observable o, Object arg) {
            System.out.println(message);
        }
    }

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

## Uppgift 2a

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

    public AlarmThreadB(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 DemoAlarmB {
    public DemoAlarmB(int ms) {
        AlarmThreadB bt = new AlarmThreadB(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) {
        DemoAlarmB da = new DemoAlarmB(4000);
    }
}
```

## Uppgift 2bc

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

    public AlarmThreadB(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 DemoAlarmB {
    public DemoAlarmB(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) {
        DemoAlarmB da = new DemoAlarmB(4000);
    }
}
```

## Uppgift 3

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

    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;
                    }
                    setChanged();
                    notifyObservers(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.addObserver(new GamesObserver());
            result.addObserver(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 GamesObserver implements Observer {
        public void update(Observable o, Object arg) {
            Game game = (Game)arg;
            SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                    resultUI.newResult(game.toString());
                }
            });
        }
    }
    // Uppgift 3b
    private class ConsolePrinter implements Observer {
        public void update(Observable o, Object arg) {
            Game game = (Game)arg;
            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();
            }
        });
    }
}
```