Laboration 6 – Synkronisering

Syftet med labborationen är att du ska träna på att använda och skriva synkroniserade kodsekvenser. Till laborationen bifogas följande filer:

1. Paketet *laboration6*
   Uppgift2: Concurrency.java
   Uppgift4: ZipArchive.java, Controller.java, TextWindow.java
2. Paketet *l6games*
   Uppgift 3: GameResults.java, Controller.java, Buffer.java , Game.java, TextWindow.java, games.txt

Uppgift 1

Uppgiften är att testa några av föreläsningsfilerna (F6).

Kör ***CounterThread1***. Testa olika varianter av synkronisering:

- Använd den översta run-metoden i CounterThread1. Kontrollera körresultatet då metoden incCounter inte är synchronized respektive när metoden är synchronized.
- Använd den mittersta run-metoden. Hur blir körresulatet med synchronized-sats.
- Använd den tredje run-metoden. Hur påverkas körresultatet om även println-satsen tas med i synchronized-bolocket?

Kör ***CounterThread2***. Hur förändras körresultatet om synchronized(counter) ändras till synchronized(this)? Hur kan du förklara detta?

Uppgift 2

Programmet ***Concurrency*** innehåller instansvariabeln *c* av typen *Color* och instansvariabeln *lbl* av typen *JLabel*,

Metoden ***changeColor()*** slumpar fram ett färgobjekt vilket lagras i c. Sedan sätts bakgrundsfärgen i JLabel-komponenten till samma färg. Det innebär att *c* och *JLabel*-komponentens bakgrundsfärg alltid ska vara samma efter anrop till metoden.

Metoden ***getColor1()*** returnerar färgobjektet som *c* lagrar.
Metoden ***getColor2()*** returnerar färgobjektet som är *JLabel*-komponentens bakgrundsfärg.

I programmet används två trådar. En tråd som anropar *changeColor()*-metoden. Och en tråd vilken anropar *getColor1()*-metoden och *getColor2()*-metoden för att därefter jämföra färgvärdena. Är de inte samma så skrivs färgvärdena ut i Console-fönstret (de ska vara samma).

Kör *Concurrency.java*. Varje gång det skriv ut något liknande:

```
count=79: C1=java.awt.Color[r=255,g=255,b=255]  C2=java.awt.Color[r=255,g=0,b=0]
```

i Console-fönstret har det blivit ett problem, då är det inte samma färgobjekt som lagras i c och som används som bakgrundsfärg i *JLabel*-komponenten.

Din uppgift är att synkronisera kod i klassen ***Concurrency*** (även i inre klass/klasser) så att färgvärdet i *c* och i *JLabel*-komponenten alltid är samma då den andra tråden kontrollerar detta.

Hjälp:

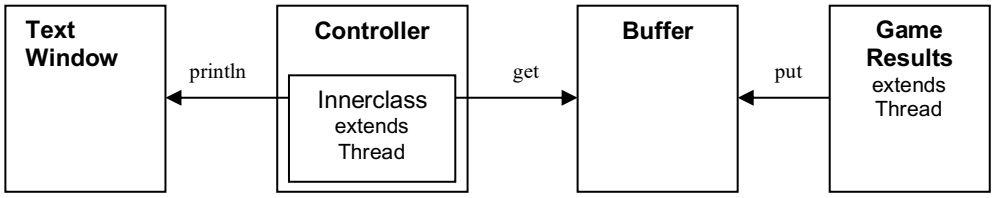
Det gäller att se till att trådarna inte exekverar viss kod samtidigt:
* då färgerna sätts (changeColor())
* då färgerna avläses (anrop till getColor1() och getColor2())
Det gäller alltså att synkronisera dessa delar av koden (en metod respektive en sekvens med satser)

Uppgift 3 – Lite lik L5

Filerna 1-3 ovan liknar filer som användes i Laboration 5. Testkör main-metoden i Controller-klassen. Som du ser skrivs matchresultat endast i Output-fönstret. Det är GameResults-objektet som simulerar och skriver ut resultaten.

Controller klassen vill ha matchresultat (*update*-metoden) att publicera i TextWindow. Men några sådana kommer inte, det är inte någon kontakt mellan GameResults och Controller. Och det är din uppgift att skapa en sådan kontakt med hjälp av en buffert (Buffer.java).

Struktur



Att göra:

1. Se till att ***GamesResult***-objektet placerar nya resultat i bufferten (i stället för att skriva ut dem). Det innebär att *GamesResult*-objektet måste ha en instansvariabel av typen *Buffer*<*Game*>. Konstruktorn måste ändras så den tar två argument, ett filnamn och en *Buffer*<*Game*>-referens. Testa *GamesResult* med nedanstående main-metod (ska ge utskriften: Buffer, size=5):

```
public static void main(String[] args) throws IOException {
    Buffer<Game> buffer = new Buffer<Game>();
    GameResults gr = new GameResults("files/games.txt",buffer);
    gr.startSimulation();
    try {
        Thread.sleep(5500);
    } catch(InterruptedException e) {}
    gr.stopSimulation();
    System.out.println("Buffer, size=" + buffer.size());
}
```

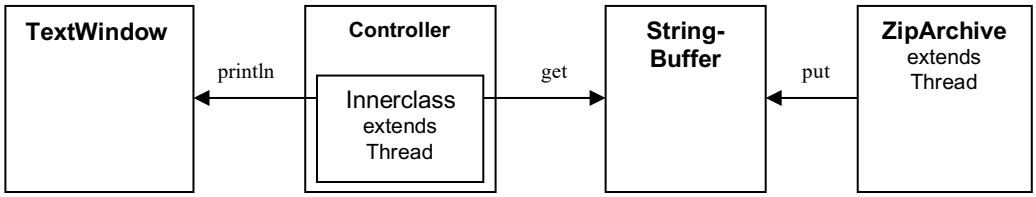
2. Se till att ***Controller*** har en inre klass som ärver *Thread* och som hämtar *Game*-objekt ur bufferten. Sedan ska *TextWindow* uppdateras med erhållna *Game*-objekt.

```
public static void main(String[] args) throws IOException {
    Buffer<Game> buffer = new Buffer<Game>();
    Controller controller = new Controller(buffer);
    GameResults gr = new GameResults("files/games.txt",buffer);
    gr.startSimulation();
    try {
        Thread.sleep(20500);
    } catch(InterruptedException e) {}
    gr.stopSimulation();
}
```

Uppgift 4 – ZipArchive - Controller

I föreläsning 5 presenterades klassen ***ZipArchive***. Meddelanden från *ZipArchive* till *Controller* skedde del via *Observer*-mönstret, dels via *callback*. I denna uppgift ska dessa meddelanden gå via en buffert (se figur nedan), liknande meddelandesystemet i Uppgift 3.
Utgå från de bifogade filerna ZipArchive.java och Controller.java när du ska göra din lösning.

Nu är det ***ZipArchive*** som ska placera meddelanden i en buffert. ***Controller*** ska i sin tur hämta meddelanden från bufferten och visa dessa i ett ***TextWindow***-fönster. Gör så här:



1. Skriv klassen ***StringBuffer*** vilken är en buffert för *String*-objekt. Testa bufferten när du är färdig genom att skapa en tråd (*StringProducer*) vilken placerar String-objekt i bufferten och en tråd (*StringConsumer*) vilken hämtar String-objekt ur bufferten och skriver ut dessa i Console-fönstret.
```
StringBuffer buffer = new StringBuffer();
StringProducer prod = new StringProducer(buffer);
StringConsumer cons = new StringConsumer(buffer);
prod.start();
cons.start();
```
2. Ändra i klassen ***ZipArchive*** så att meddelanden läggs i en StringBuffer (i stället för att skrivas ut i Console-fönstret). Testa *ZipArchive* genom att skapa en instans av *ZipArchive* och en instans av *StringConsumer* (samma som ovan) och kontrollera att meddelandena når Console-fönstret.
```
StringBuffer buffer = new StringBuffer();
File file = new File("files/Material 6");
ZipArchive archive = new ZipArchive(file, buffer);
StringConsumer cons = new StringConsumer(buffer);
archive.zip();
cons.start();
```
3. Lägg till en tråd i ***Controller*** som hämtar *String*-objekt ur bufferten och visar dessa i ***TextWindow***. Använd println-metoden i TextWindow-objektet. Testa hela systemet med:
```
StringBuffer buffer = new StringBuffer();
File file = new File("files/Material 6");
ZipArchive archive = new ZipArchive(file, buffer);
Controller controller = new Controller(buffer); // tråd startas i konstruktor
archive.zip();
```

Uppgift 5 – RunOnThread med flera trådar, EXTRA

I föreläsning 6 presenterades klassen ***RunOnThread*** vilken innehåller en buffert och en tråd. Tråden exekverar successivt Runnable-objekt som placeras i bufferten.

Skriv klassen RunOnThreadN vilken använder n st trådar för att exekvera Runnable-objekt i bufferten.

Klassen bör innehålla konstruktorn:

```
public RunOnThreadN( int n )
```

där n anger antalet trådar som ska användas.

Testa din lösning med DemoRunOnThread men se till att RunOnThreadN används i testet.

Lösningar

Uppgift 2

```
public class Concurrency {
    // instansvariabler och konstruktor
    public Color getColor1() {
        return c;
    }

    public Color getColor2() {
        return lbl.getBackground();
    }

    public synchronized void changeColor() {
        c = colors[random.nextInt(colors.length)];
        lbl.setBackground(c);
    }

    private class Compare implements Runnable {
        public void run() {
            int count = 0, notSame=0;
            Color color1;
            Color color2;
            while(notSame<10 && count<100000) {
                count++;
                synchronized(Concurrency.this) {
                    color1 = getColor1();
                    color2 = getColor2();
                }
                if(!color1.equals(color2)) {
                    notSame++;
                    System.out.println("count=" + count +": C1=" + color1 + "  C2="
+ color2);
                }
            }
            System.out.println(this + " stopped, count="+count);
        }
    }

    private class Change implements Runnable {
        public void run() {
            int count = 0;
            while(t2.isAlive()) {
                count++;
                changeColor();
            }
            System.out.println(this+" stopped, count="+count);
        }
    }
}
```

Uppgift 3

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

    public GameResults(String filename, Buffer<Game> buffer) throws IOException {
        this.buffer = buffer;
        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;
                    }
                    buffer.put(game);
                } catch(InterruptedException e) {
                    break;
                }
            }
        }
    }
}


public class Controller {
    private Buffer<Game> buffer;

    public Controller(Buffer<Game> buffer) {
        this.buffer = buffer;
        new Results().start();
    }

    private class Results extends Thread {
        public void run() {
            Game game;
            while(!Thread.interrupted()) {
                try {
                    game = buffer.get();
                    TextWindow.println(game);
                } catch (InterruptedException e) {
                    break;
                }
            }
        }
    }
}
```

Uppgift 4_1

```
public class StringBuffer {
    private LinkedList<String> buffer = new LinkedList<String>();

    public synchronized void put(String str) {
        buffer.addLast(str);
        notifyAll();
```

```
    }

    public synchronized String get() throws InterruptedException {
        while(buffer.size()==0) {
            wait();
        }
        return buffer.removeFirst();
    }
}

public class StringProducer extends Thread {
    private StringBuffer buffer;

    public StringProducer(StringBuffer buffer) {
        this.buffer = buffer;
    }

    public void run() {
        String[] arr = {"Hej","du","glade","tag","en","spade","..."};
        System.out.println("StringConsumer runs");
        for(String s : arr) {
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {}
            buffer.put(s);
        }
    }
}

public class StringConsumer extends Thread {
    private StringBuffer buffer;

    public StringConsumer(StringBuffer buffer) {
        this.buffer = buffer;
    }

    public void run() {
        String str;
        System.out.println("StringConsumer runs");
        while(!Thread.interrupted()) {
            try {
                str = buffer.get();
                System.out.println(str);
            } catch (InterruptedException e) {
                break;
            }
        }
    }

}
```

Uppgift 4_2

```
public class ZipArchive {
    private File file;
    private String archive;
    private StringBuffer buffer;

    public ZipArchive(File file, StringBuffer buffer) {
        this.file = file;
        this.buffer = buffer;
    }


    public void zip() {
        new Zip().start();
    }

    public void unzip() {
        new Unzip().start();
    }
```

```
    private class Zip extends Thread {
        public void run() {
            buffer.put("TO ZIP: " + file.getAbsolutePath());
            if(file.isDirectory()) {
                archive = file.getAbsolutePath()+".zip";
            } else if(file.isFile()) {
                archive = getFileName(file.getAbsolutePath())+".zip";
            } else {
                buffer.put("NOT directory or file: " + file.getAbsolutePath());
                return;
            }

            try (ZipOutputStream zos = new ZipOutputStream(new
BufferedOutputStream(new FileOutputStream(archive)))){
                zip(file, zos, "");
            }catch(Exception e) {
                buffer.put("EXEPTION: " + e.getMessage());
                return;
            }
            buffer.put("ZIP-FILE: " + archive);
        }

        private String getFileName(String filename) {
            int index = filename.indexOf('.');
            if(index>=0)
                return filename.substring(0,index);
            else
                return filename;
        }

        private void zip(File file, ZipOutputStream zos, String directories) throws
IOException {
            if(file.isFile()) {
                buffer.put("ZIP: " + file.getAbsolutePath());
                zos.putNextEntry(new ZipEntry(directories+file.getName()));
                try (BufferedInputStream bis = new BufferedInputStream(new
FileInputStream(file))) {
                    int b = bis.read();
                    while(b!=-1) {
                        zos.write(b);
                        b = bis.read();
                    }
                    zos.flush();
                }
                zos.closeEntry();
            } else if(file.isDirectory()) {
                for(File f : file.listFiles()) {
                    zip(f,zos,directories+file.getName()+"/");
                }
            }
        }
    }


    private class Unzip extends Thread {
        private File directory;
        private HashMap<String,File> directories = new HashMap<String,File>();

        public void run() {
            directory = file.getParentFile();
            buffer.put("TO UNZIP: " + file.getAbsolutePath() + " to " +
directory.toString());
            try (ZipInputStream zis = new ZipInputStream(new
BufferedInputStream(new FileInputStream(file)))){
                unzip(file, zis, "");
            }catch(Exception e) {
                buffer.put("EXEPTION: " + e.getMessage());
                return;
            }
        }
```

```
        private void unzip(File file, ZipInputStream zis, String directories)
throws IOException {
            ZipEntry entry;
            while((entry=zis.getNextEntry())!=null) {
                buffer.put("UNZIP: " + entry.getName() +" to " + directory + "/");
                checkDirectories(entry.getName());
                try (BufferedOutputStream bos = new BufferedOutputStream(new
FileOutputStream(directory+"/"+entry.getName()))) {
                    int b = zis.read();
                    while(b!=-1) {
                        bos.write(b);
                        b = zis.read();
                    }
                    bos.flush();
                }
            }
        }


        private void checkDirectories(String file) {
            int index = file.indexOf("/");
            while(index>=0) {
                String path = file.substring(0,index);
                if(directories.get(path)==null) {
                    new File(directory+"/"+path).mkdir();
                }
                index = file.indexOf("/",index+1);
            }
        }
    }
}
```

Uppgift 4_3

```
public class Controller {
    private StringBuffer buffer;

    public Controller(StringBuffer buffer) {
        this.buffer = buffer;
        new Messages().start();
    }

    private class Messages extends Thread {
        public void run() {
            String str;
            while(!Thread.interrupted()) {
                try {
                    str = buffer.get();
                    TextWindow.println(str);
                } catch (InterruptedException e) {
                    break;
                }
            }
        }
    }
}
```

Uppgift 5

```
public class RunOnThreadN {
    private Buffer<Runnable> buffer = new Buffer<Runnable>();
    private Worker[] workers;
    private boolean started = false;

    public RunOnThreadN(int n) {
        if(n<=0)
            n = 1;
        workers = new Worker[n];
        while(--n>=0) {
            workers[n] = new Worker();
        }
    }

    public synchronized void start() {
        for(int i=0; !started && i<workers.length; i++) {
            workers[i].start();
        }
    }

    public synchronized void execute(Runnable runnable) {
        buffer.put(runnable);
    }

    public synchronized void stop() {
        buffer.clear();
        for(int i=0; started && workers!=null && i<workers.length; i++) {
            workers[i].interrupt();
            workers[i] = null;
        }
        workers = null;
    }

    private class Worker extends Thread {
        public void run() {
            while(!Thread.interrupted() && workers!=null) {
                try {
                    buffer.get().run();
                } catch(InterruptedException e) {
                    System.out.println(e);
                    break;
                }
            }
            System.out.println(Thread.currentThread().getName() + " avslutas");
        }
    }

    private class Buffer<T> {
        private LinkedList<T> buffer = new LinkedList<T>();

        public synchronized void put(T obj) {
            buffer.addLast(obj);
            notifyAll();
        }

        public synchronized T get() throws InterruptedException {
            while(buffer.isEmpty()) {
                System.out.println(Thread.currentThread() + " is waiting");
                wait();
            }
            return buffer.removeFirst();
        }

        public synchronized void clear() {
            buffer.clear();
        }

        public int size() {
            return buffer.size();
        }
    }
}
```