Föreläsning 5

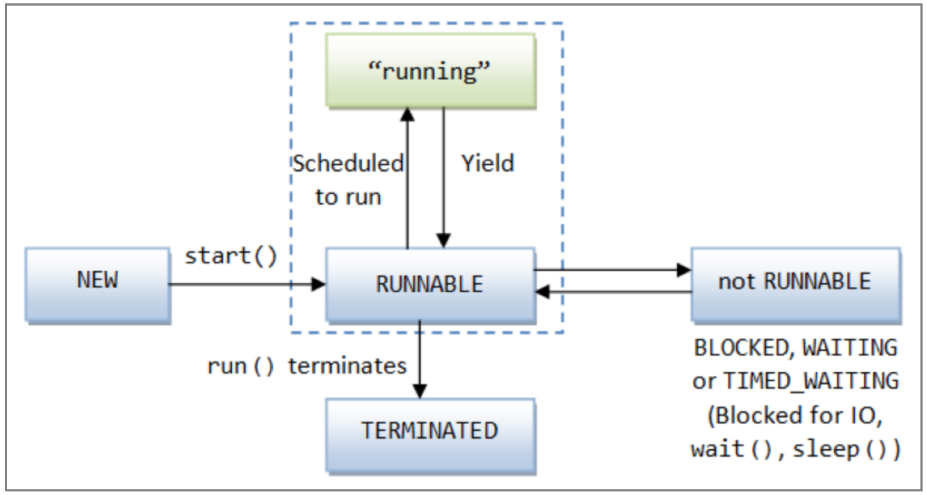
- Returvärde från tråd
- Observer, Observable
- Callback

JNP: s 53-70

Trådens livscykel

Tråden kan befinna sig i olika skeenden, states:

- NEW
  Tråden är skapad men start() har inte anropats.
- RUNNABLE
  start() har anropats och tråden kan vara aktiv.
- BLOCKED, WAITING, TIMED_WAITING
  T.ex. då någon av metoderna *Object – wait()* eller *Thread.sleep()* har anropats. Tråden väntar ofta på en resurs eller en timeout.
- TERMINATED
  Trådens run-metod har exekverat klart



Tråd gör en uppgift

En vanlig användning av en tråd är att tråden har en uppgift och rapporterar resultat under arbetets gång och/eller när tråden är färdig med uppgiften.

Klassen ***ZipArchive*** kan:

* starta en tråd vilken skapar en komprimerar fil , en zip-fil, av innehållet i en mapp eller av en enstaka fil (inre klassen Zip).
* starta en tråd som packar upp en zip-fil (inre klassen Unzip).

Detta sker:

1 Klassens konstruktor tar emot ett File-objekt som argument. Ett File-objekt kan beskriva en fil eller en katalog.

2 Tråden startas och komprimeringen / uppackning sätter igång. Under processens gång skrivs meddelanden ut i Output-fönstret.

3 Resultatet placeras i katalogen som innehåller katalogen/filen som ska komprimeras / innehåller zip-filen som ska packas upp

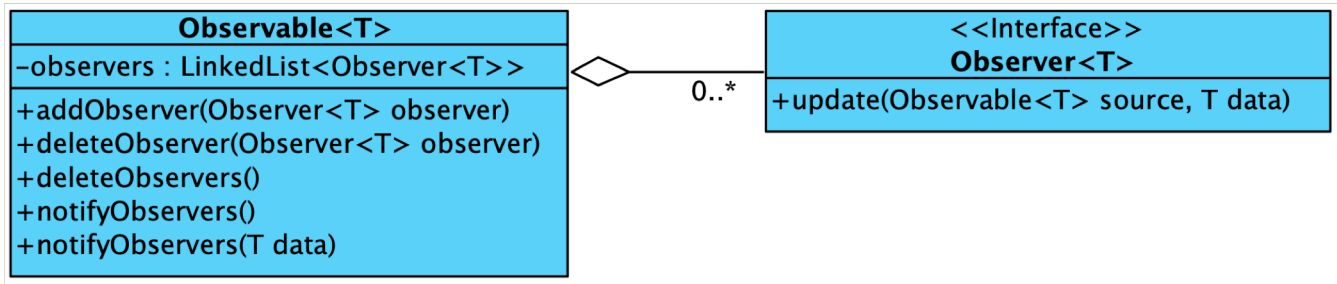
För att skapa zip-filen används en **ZipOutputStream**
För att packa upp en zip-fil används **ZipInputStream**

ZipArchive.java

Tråd returnerar värde, Observer

Men hur gör man för att få utskrifterna till t.ex. en JTextArea i ett UI eller för loggning i en fil?

Designmönstret ***Observer*** ger möjlighet att lyssna på meddelanden från ett objekt.



Tanken är att *klassen som ska observeras*, dvs skicka meddelande till lyssnande (observerande) klasser, *ska ärva Observable*.

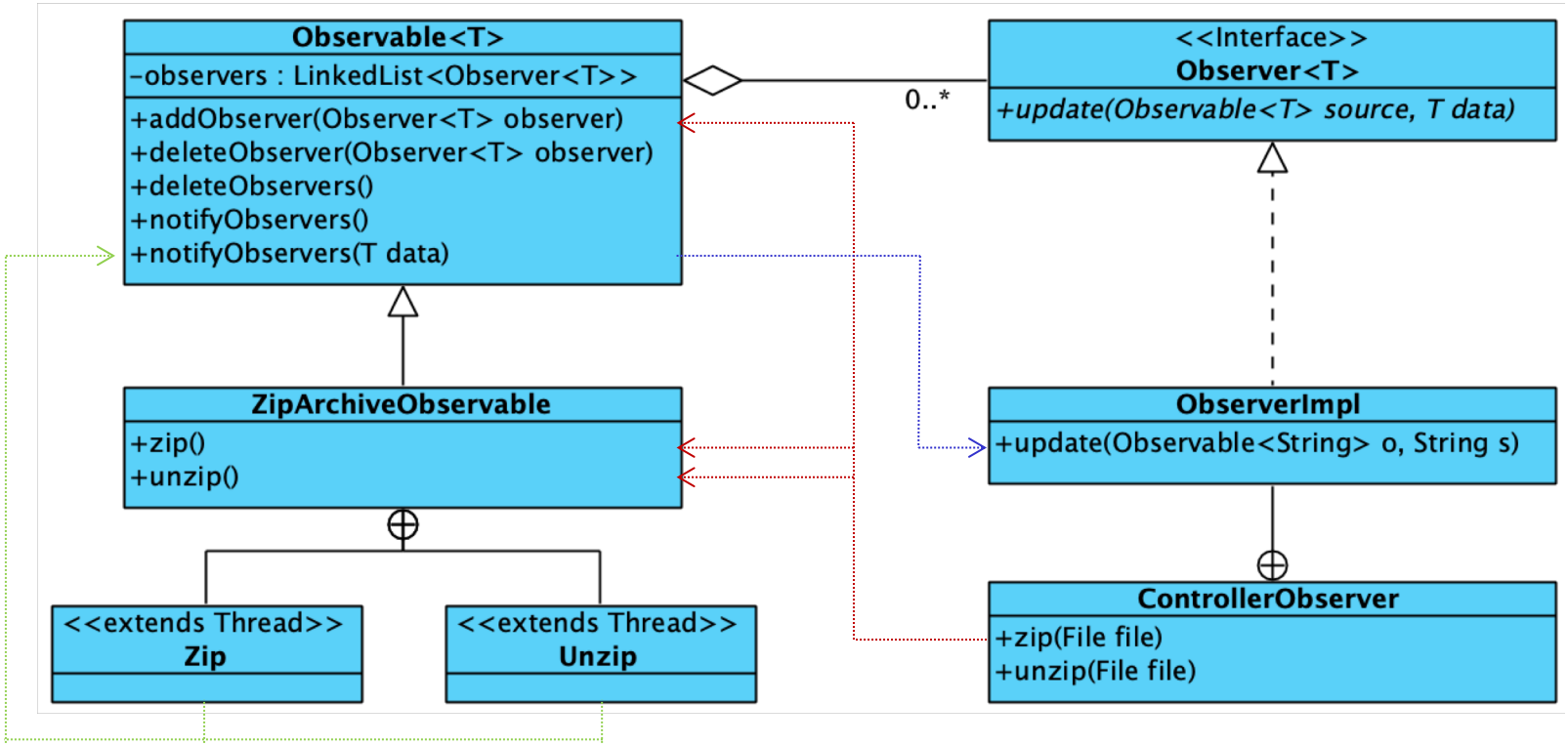
Klasser som ska lyssna (observera) *ska implementera Observer*.

Designmönstret är implementerat i java men depricated, ska ej användas.

I jar-filen *da343a.jar* finns Observer-mönstret.

Tråd returnerar värde, Observer

ZipArchiveObservable ska observeras (ärva *Observable*) och ett observerande objekt behövs (implementera *Observer*)



Först registreras Observer-implementeringen och därefter startas en av trådarna.
Då någon av trådarna exekverar anropas *notifyObservers*
Vid varje anrop till notifyObservers så anropas *update* i sin tur.

Tråd returnerar värde, Observable

ZipArchive (ska observeras) ska alltså ***ärva Observable***.

1. I denna version får filen ZipArchive heta ***ZipArchiveObservable*** och ärver *Observable*.

   public class ZipArchiveObservable extends Observable<String> {

   Typen String anger att notifieringar till observerande instanser sker med String-objekt.

2. Alla anrop till *System.out.println(str)* ersätts med anropet:

   notifyObservers(str); // Alla registrerade Observers meddelas

ZipArchiveObservable

Tråd returnerar värde, Observer

1. ***ZipArchiveObservable*** ska observeras av en klass vilken ***implementerar Observer***.
   Klassen ***ObserverImpl*** implementerar *Observer*:

```
private class ObserverImpl implements Observer<String> {
    public void update(Observable<String> source, String s) {
        gui.append(s);
    }
}
```

2. Registrering av det observerande objektet sker genom anrop till *addObserver*-metoden i *ZipArchiveObservable*:

```
archive.addObserver(new ObserverImpl());
```

   Det krävs alltså en referens till *ZipArchiveObservable-instansen* för att kunna registrera sig som observer.

ControllerObserver

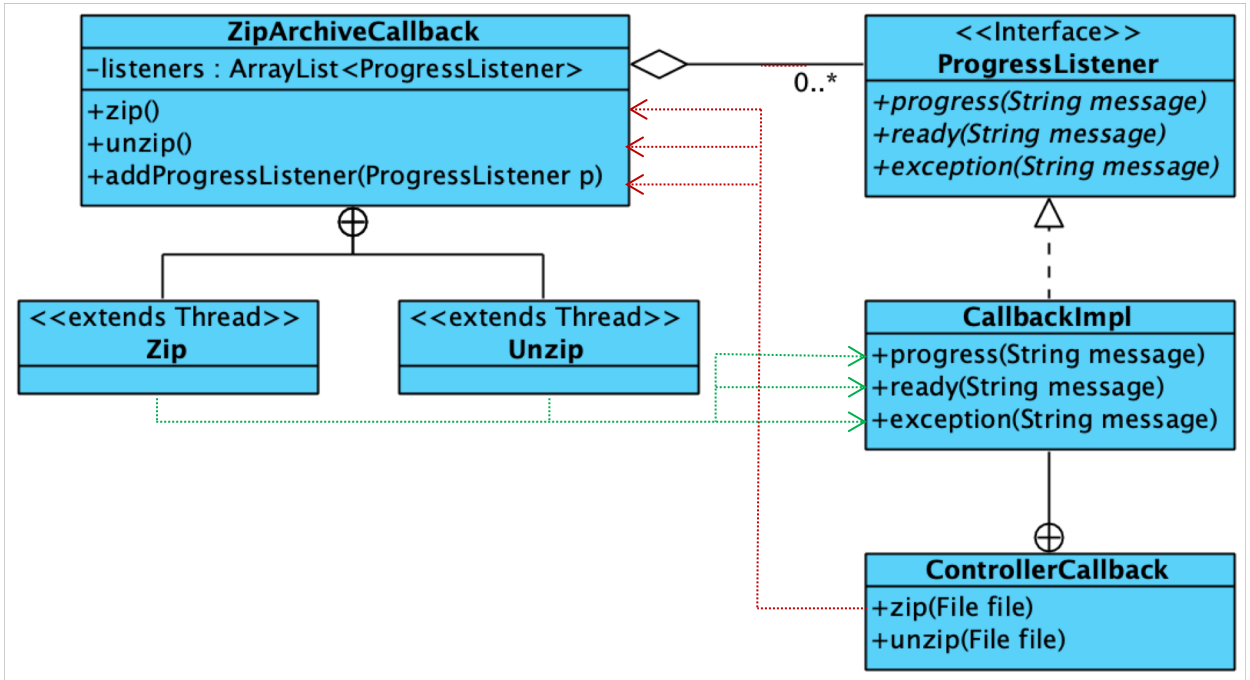
ZipController <<interface>>

ZipUI

Tråd returnerar värde, Callback

Ett alternativ till Observer-mönstret är ett ***Callback***.

Klassen som ska observeras (***ZipArchiveCallback***) tar emot en eller flera instanser av klasser som implementerar ***ProgressListener*** via en metod. Det är klassen som observerar som implementerar interfacet *ProgressListener*



Först registreras ProgressListener-implementeringen och därefter startas en av trådarna.

Då någon av trådarna exekverar anropas *progress* / *ready* / *exception.*

Tråd returnerar värde, Callback

Klassen som ska observeras, ***ZipArchiveCallback***, tar emot implementeringar av ***ProgressListener*** via en metod. Det är klassen som observerar *ZipArchiveCallback* som implementerar interfacet.

```
public interface ProgressListener {
   public void progress(String filename);
   public void ready(String archive);
   public void exception(String message);
}

public class ZipArchiveCallback {
   private ArrayList<ProgressListener> listeners;
   :
   public void addProgressListener( ProgressListener listener ) {
      if( listener!=null )
         this.listeners.add( listener );
   }
}

public class ControllerCallback {
   :
   private class CallbackImpl implements ProgressListener {
      public void progress(String filename) {…}
      public void ready(String archive) {…}
      public void exception(String message) {…}
   }
}
```

Tråd returnerar värde, Callback

Callback, steg för steg:

1 Definiera ett interface med lämpliga metoder. Det är dessa metoder tråden ska anropa.

```
public interface  ProgressListener {
   public void processing(String filename);
   public void ready(String archive):
   public void exception(String message);
}
```

2 Implementera interfacet i en controller-klass. Registrera ProgressListener.

```
public class ControllerCallback implements ZipController {
   :
   public void zip(File file) {
      ZipArchiveCallback archive = new ZipArchiveCallback(file);
      archive.addProgressListener(new CallbackImpl());
      archive.zip();
   }

   private class CallbackImpl implements ProgressListener {
      public void progress(String message) {
         gui.append(message);
      }

      public void ready(String archive) {
         gui.append(”ZIP-FILE: " + archive);
      }

      public void exception(String message) {
         gui.append("EXCEPTION: " + message);
      }
   }
}
```

ProgressListener

ControllerCallback

Tråd returnerar värde, Callback

3 Låter *ZipArchiveCallback* ta emot en eller flera instanser av klasser som implementerar *ProgressListener* (t.ex. via en add-metod). Sedan anropas metoderna från run-metoden i tråden som exekverar. Viktigt är att metoderna utförs snabbt eftersom det är den processande tråden som används.

```
public class ZipArchiveCallback {
   private ArrayList<ProgressListener> listeners;
   private String archive;

   public ZipArchive(File file) {..}

   public void addProgressListener(ProgressListener listener) {
      if( listener!=null )
         this.listeners.add( listener );
   }

   private class Zip extends Thread {
      public void run() {
         :
         try (…) {
            progress( filename );
            :
         }
         ready( archive );
      }
   }
}
```

ZipArchiveCallback