Programmeringsuppgift 1 – Strömmar och trådar

1 Inledning

Programmeringsuppgiften ska bidra till en grundläggande förståelse för:

- Användning av trådar
- Användning av synkroniserad buffert
- Användning av strömmar
- Implementering av Observer eller Callback

1.1 Klasser, interface och övriga filer som bifogas

Klasser: Buffer, ArrayProducer, MainP1, MainDS1, TestDS1, MainDS2, TestDS2, ViewerWindow, FileProducer, Viewer

Interface: IconProducer

Övriga filer: *new.txt* vilken ska placeras i katalogen *files* i projektet
new1.jpg, *new2.jpg*, ..., *new10.jpg* vilka ska placeras i katalogen *images* i projektet

1.2 Redovisning

Din lösning av uppgiften lämnas in via It's learning senast kl 09.00 onsdagen den 7/2. Inlämningen ska innehålla samtliga klasser som används i lösningen. Klasserna Producer och FileProducer ska vara javadoc-kommenterade och javadoc ska vara genererad.

Vid redovisningen torsdagen den 8/2 kommer din lösning att köras med programmet MainP1. Kontrollera därför noga funktion innan inlämningen.

Zip-filen ska du ge namnet AAABBBP1.zip där AAA är de tre första bokstäverna i ditt efternamn och BBB är de tre första bokstäverna i ditt förnamn. Använd endast tecknen a-z när du namnger filen.

- Om Rolf Axelsson ska lämna in sina lösningar ska filen heta AxeRolP1.zip.
- Om Örjan Märla ska lämna in sina lösningar ska filen heta MarOrjP1.zip.
- Är ditt förnamn eller efternamn kortare än tre bokstäver så ta med de bokstäver som är i namnet: Janet Ek lämnar in filen EkJanP1.zip

1.3 Granskning

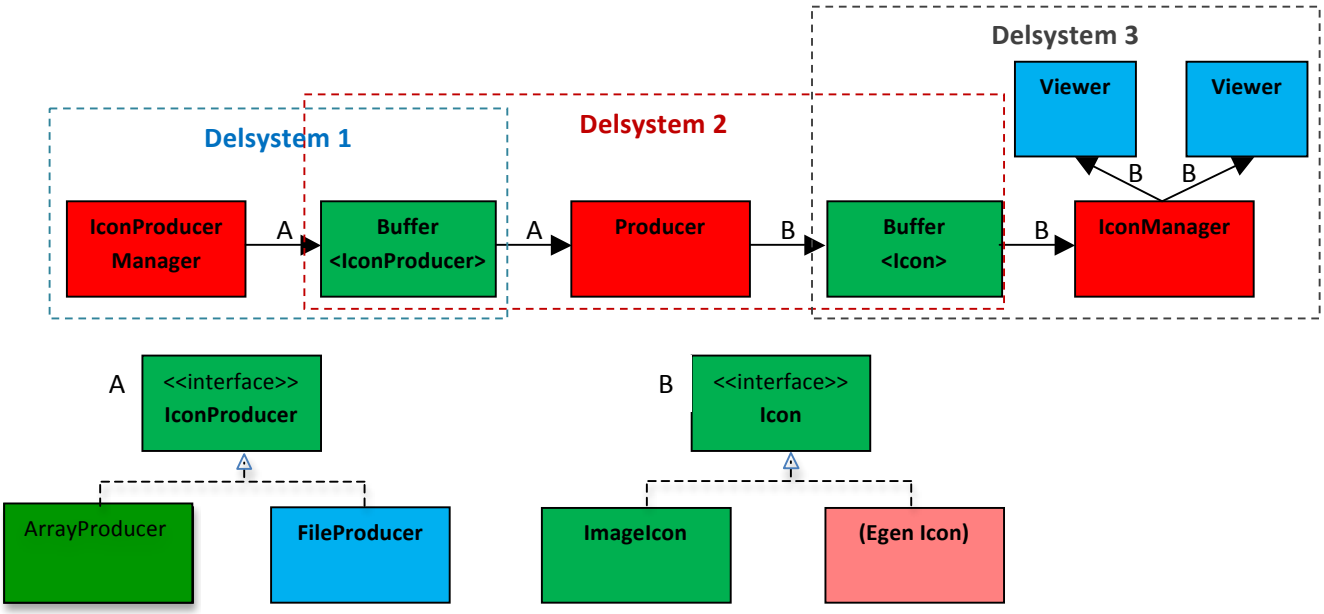
Ca kl 14.00 den 7/2 kommer en kamrats lösning finnas i din inlämning på It's learning. Din uppgift är att granska kamratens lösningar på uppgifterna avseende:

- funktion – hur väl uppfyller lösningen kraven i uppgiften? Fungerar klasserna på avsett sätt?
- kan du tänka dig något alternativt sätt att lösa uppgiften?
- javadockommentarer – är klasserna kommenterade enligt instruktion? Och är kommentarena vettiga?
- genererad javadoc – är javadoc-dokument genererade? Är dokumenten vettiga? Fungerar länkar?

Resultatet av din granskning, 1-2 A4-sidor, ska du lämna in via It's learning senast 8.00 den 8/2.

2 Beskrivning av uppgiften

Figuren visar olika klasser och interface som ska ingå i din lösning. De som är gröna är färdiga. De som är blå måste modifieras / ärvas för att få full funktion. De som är markerade med rött ska du skriva. Bokstaven A visar var objekt vilka implementerar IconProducer hanteras och bokstaven B var objekt vilka implementerar Icon hanteras.



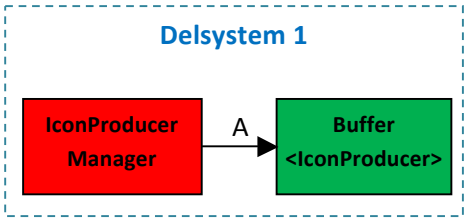
Figur 1, Klasser och interface i Inlämningsuppgift 1

Meningen med systemet är följande (från vänster till höger i figuren)

Till vänster genereras serier av bilder (IconProducer-implementeringar). Dessa bildserier delas upp i Icon-implementeringar (Producer) vilka slutligen visas, en åt gången, längst till höger (i Viewer-komponenter). Man kan ange hur lång tid bilderna i bildserien ska visas och hur många gånger bildserien ska visas.

På följande sidor följer beskrivning av de olika delsystemen i figuren.

2.1 Delsystem 1



IconProducerManager-objektet får vid anrop till metoden *addIconProducer* tillgång till en instans av en klass vilken implementerar **IconProducer**. Detta objekt ska placeras i en buffert av typen **Buffer<IconProducer>**.

Som du ser i figuren ska du implementera klassen **IconProducerManager**.
I klassdiagrammet till höger ser du instansvariabler, konstruktorer och metoder som **IconProducerManager** behöver innehålla. Konstruktorn och add…-metoden måste finnas i klassen, annars är inte *MainP1* körbar.

IconProducerManager
-buffer : Buffer<IconProducer>
konstruktorer +IconProducerManager(Buffer<…>)
+addIconProducer(IconProducer)

IconProducer är ett interface som definierar funktionalitet för att

* hantera en sekvens av Icon-objekt (*size()*, *nextIcon()*)
* ange hur lång tid varje Icon-objekt ska visas (*delay()*, samma tid för samtliga Icon-objekt)
* ange hur många gånger sekvensen ska visas (*times()*)

På så sätt går det bra att skapa ett bildspel (delay ett antal sekunder) eller en animation (delay delar av sekund).
I Figur 1 ser du att du ska modifiera klassen **FileProducer**.

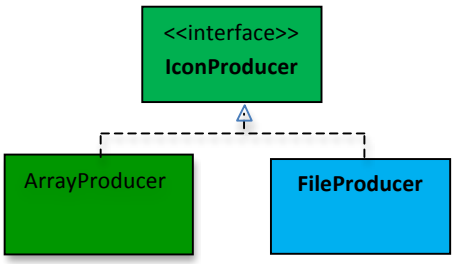
<<interface>> **IconProducer**
+public int delay() +public int times() +public int size() +public Icon nextIcon()

Klassen **ArrayProducer** ger exempel på en *IconProducer*-implementering.

Med programmet ***MainDS1*** kan du testa *IconProducerManager* och att IconProducer-implementeringar placeras i bufferten. ***TestDS1*** ger exempel på hur man kan använda en *IconProducer*-implementering.
Om du placerar filerna *new1.jpg-new10.jpg* i mappen *images* i projektmappen så kommer *TestDS1* skriva ut information om de ImageIcon-objekt som levereras av ArrayProducer-instansen. Utskrifterna sker en gång i sekunder eftersom andra argumentet är 1000 då *ArrayProducer*-objektet skapas. Samtliga utskrifter sker två gånger eftersom sekvensen ska visas två gånger (argumentet 2 då *ArrayProducer*-objektet skapas).

FileProducer

Du ska komplettera klassen ***FileProducer*** vilken implementerar IconProducer. Till din hjälp har du klassen ArrayProducer.



FileProducer ska ha konstruktorn:

```
public FileProducer(String filename)
```

Filen som är argument vid konstruktionen ska vara en textfil formatterad på följande sätt:

```
2
200
images/new1.jpg
images/new2.jpg
images/new3.jpg
images/new4.jpg
images/new5.jpg
images/new6.jpg
images/new7.jpg
images/new8.jpg
images/new9.jpg
images/new10.jpg
```

Det första talet (2 ovan) är antalet gånger Icon-sekvensen ska upprepas, det andra är tiden varje bild ska visas i millisekunder (200 ovan). Därefter följer ett antal filnamn (bildfiler), ett per rad. Antalet rader med filnamn avgör hur många Icon-implementeringar det är i sekvensen (i exemplet ovan blir det tio Icon-implementeringar).

Läs textfilen med en *BufferedReader*. Se till att ange teckenkodningen till "UTF-8". Det kan du göra om du använder in *InputStreamReader*.

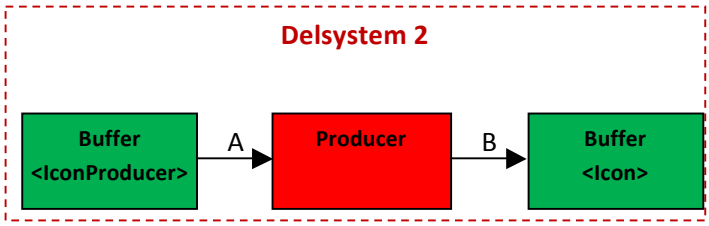
Använder du ovanstående textfil (new.txt), och nedanstående version av **MainDS1** (dvs FileProducer i stället för ArrayProducer) ska utskrifterna bli samma som vid tidigare körningar, men ske betydligt snabbare.

```
public class MainDS1 {
    public static void main(String[] args) {
        Buffer<IconProducer> producerBuffer = new Buffer<IconProducer>();

        IconProducerManager ipManager = new IconProducerManager(producerBuffer);
        ipManager.addIconProducer(new FileProducer("files/new.txt"));

        TestDS1 testDs1 = new TestDS1(producerBuffer);
        testDs1.start();
    }
}
```

2.2 Delsystem 2



Producer-objektet ska fungera så här:

A. Hämta en **IconProducer**-implementering ur bufferten till vänster i figuren.

B. Använd IconProducer-implementeringen för att placera **Icon**-implementeringar i bufferten till höger i figuren. Men de sak inte placeras i bufferten så fort det går utan med en viss paus mellan varje objekt. Det är metoden *delay* i **IconProducer**-implementeringen som ger pausen. Metoden *times* anger hur många gånger **Icon**-sekvensen ska placeras i bufferten. Metoden *size* anger hur många **Icon**-objekt det är i sekvensen. Och slutligen returnerar metoden *nextIcon* **Icon**-implementeringar, en i taget. När sekvensen är slut så returneras det första elementet på nytt.

Därefter upprepas A och B på nytt.

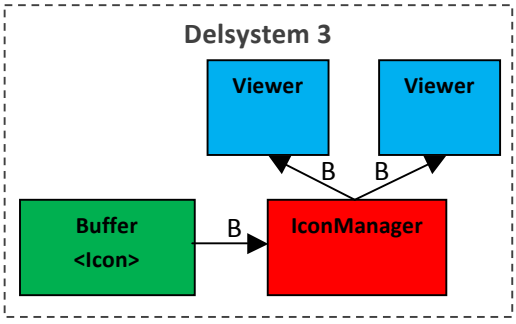
Som du ser i uppgiften ska du implementera klassen **Producer**. Klassen ska använda en tråd och konstruktorn

```
public Producer(Buffer<IconProducer prodBuffer, Buffer<Icon> iconBuffer)
```

Slutligen ska man kunna starta tråden genom att anropa metoden *start*.

Programmet ***MainDS2*** (förutsätter att **MainDS1** fungerar) ska som resultat ge samma utskrift som **MainDS1**.

2.3 Delsystem 3



I det tredje delsystemet ska ***IconManager***-objektet hämta **Icon**-implementeringar ur bufferten och skicka dessa till **Viewer**-objekten. Denna överföring ska du ordna genom att antingen använda klassen **Observable** och interfacet **Observer** eller genom att definiera ett interface för *callback*. Lägg märke till att **IconManager**-objektet ska kunna meddela många **Viewer**-objekt.

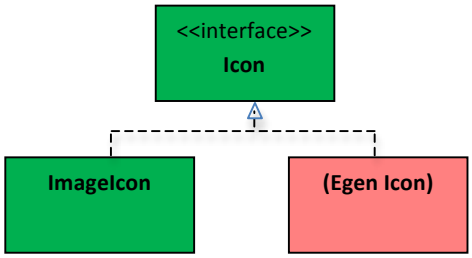
Som du ser i ***MainP1*** så känner inte **IconManager**-objektet till **Viewer**-objekten efter instansiering. Däremot känner **Viewer**-objekten till **IconManager**-instansen:

```
Viewer viewer = new Viewer(iconManager,640,480);  // Används i MainP1
```

Och kan därför registrera någon form av lyssnare.
Du måste komplettera **Viewer** med en matchande konstruktor för att ovanstående instansiering ska fungera.

Detta delsystem får du testa på egen hand. Till det levereras inget program.

2.5 Extra



Det kan vara trevligt att implementera en egen **Icon**. Sedan kan man låta t.ex. ***PixelArrayProducer*** använda **PixelIcon** (se nedan). **PixelArrayProducer** kan t.ex. *ärva ArrayProducer* och sedan *överskugga nextIcon* på ett listigt sätt.

- **BlurIcon** vilken har konstruktorn
  public BlurIcon(Icon)
  och som ser till att Icon-implementeringen blir suddig (se laboration 23 från hösten - ResizeIcon)

- **PixelIcon** vilken gör bilden ”pixlig”. Det går att använda ett trick: Rita först Icon-implementeringen i en BufferedImage med korrekt storlek, t.ex. 600x480. Skapa sedan en ny BufferedImage, t.ex. 120x96, och överför sedan den första BufferedImage till den nya (newG är Graphics från den nya - newG.drawImage(oldBI,0,0,120,96)). Slutligen används den sista BufferedImage då Icon-objektet ska ritas i JLabel-komponenten. Tänk på att preferredSize inte ska ändras.