



# 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, TestMessageProducer, TestMessageProducerInput inkl MPConsumer, TestProducer inkl MessageConsumer, Viewer

Interface: MessageProducer

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 Canvas senast kl 09.00 onsdagen den 13/2. Inlämningen ska innehålla samtliga klasser som används i lösningen. Klasserna Producer och TextfileProducer ska vara javadoc-kommenterade och javadoc ska vara genererad.

Vid redovisningen torsdagen den 14/2 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 Märta 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

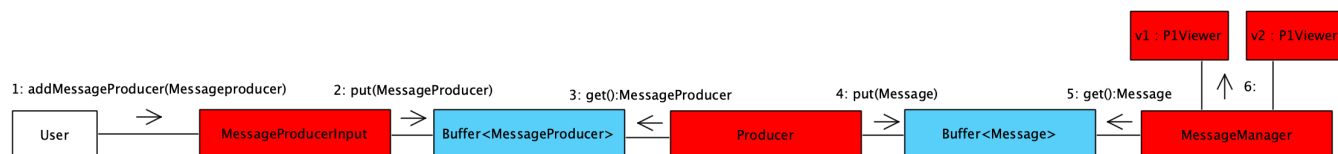
Ca kl 14.00 den 13/2 kommer en kamrats lösning finnas i din inlämning på Canvas. Din uppgift är att granska kamratens lösningar på uppgifterna avseende:

- 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 Canvas senast 13.00 den 14/2.

## 2 Beskrivning av uppgiften

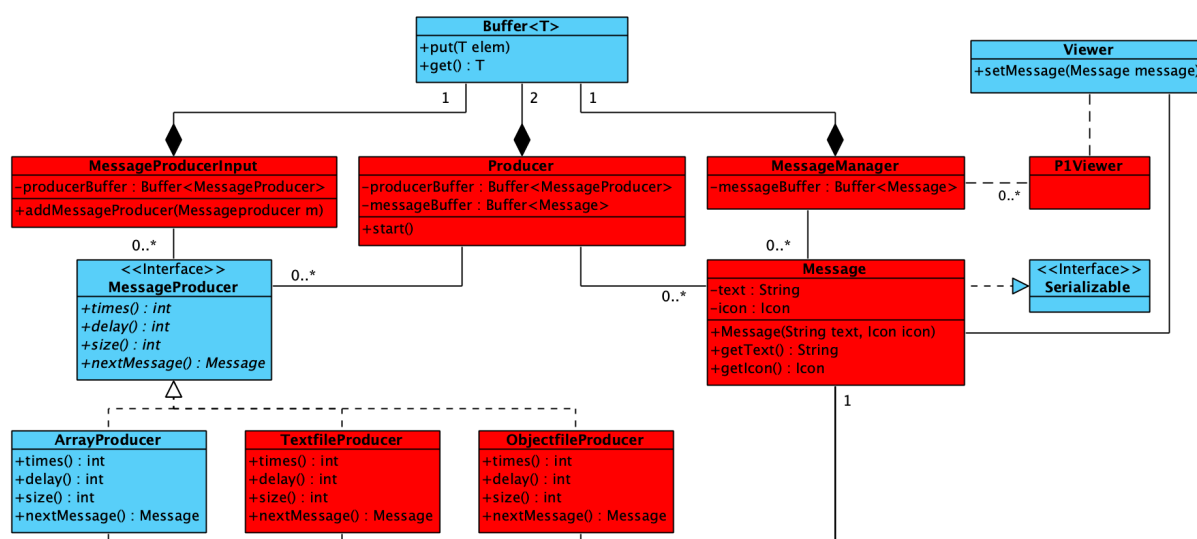
Figuren nedan visar några väsentliga klasser som ska ingå i din lösning. De rödmarkerade är klasser som du ska skriva. Det tillkommer några klasser att skriva i klassdiagrammet längre ner på sidan. I bilaga 1 finner du figuren i större storlek.



Figuren ovan visar hur kommunikationen i systemet ska ske.

1. Användare av systemet anropar metoden *addMessageProducer* i en *MessageProducerInput*-instans. Argument vid anropet är en *MessageProducer*-implementering, dvs en instans av en klass som implementerar *MessageProducer*.
2. *MessageProducerManager* placerar *MessageProducer*-implementeringen i en *Buffer<MessageProducer>*-instans, genom anrop till metoden *put*.
3. *Producer*, vilken använder en egen tråd, hämtar *MessageProducer*-implementering ur *Buffer* genom anrop till metoden *get*.
4. Varje hämtad *MessageProducer*-implementering ger ett antal *Message*-placeras i *Buffer<Message>*-instansen (anrop till metoden *put*)
5. *MessageManager* använder en egen tråd och hämtar *Message*-objekt ur *Buffer<Message>*-instansen.
6. Varje hämtat *Message*-objekt levereras till ett antal *P1Viewer*-objekt varvid *Message*-objektet visas i ett fönster.

Klassdiagrammet visar samtliga nödvändiga klasser i systemet. Steg för steg ska du skriva de rödfärgade klasserna.

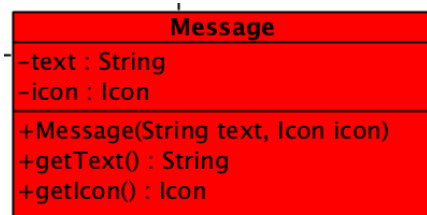




## 2.1 Message

**Message** är en klass vilken innehåller data och Message-objekt flödar genom systemet. Klassen *Message* har två instansvariabler, en lämplig konstruktör och get-metoder. Klassen ska implementera *Serializable* eftersom den används i strömmar (t.ex. i *ObjectfileProducer*).

Skriv klassen.

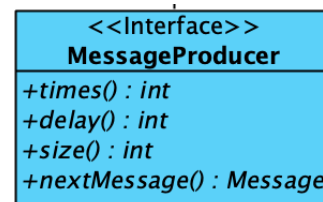


## 2.2 TextfileProducer

**MessageProducer** är ett interface som definierar funktionalitet för att

- \* hantera en sekvens av Message-objekt (*size()*, *nextMessage()*)
- \* ange hur lång tid varje Message-objekt ska visas (*delay()*, samma tid för samtliga Message-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 en sekund).



Klassen **ArrayProducer** ger exempel på en *MessageProducer*-implementering. Programmet *TestMessageProducer* visar ett exempel på vad man kan göra med en *MessageProducer*-implementering. Om du kör programmet kommer viewer visa en sekvens med text+bild.

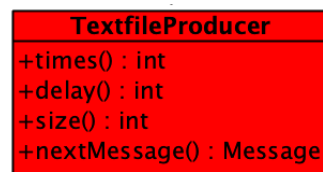
Du ska skriva klassen **TextfileProducer**, vilken ska implementera *MessageProducer*. Studera klassen *ArrayProducer* innan du börjar!

Klassen ska ha konstruktorn:

```
public TextfileProducer(String filename)
```

Filen som är argument vid konstruktionen ska vara en textfil formaterad på följande sätt (se *new.txt*):

```
4
200
10
UP and
images/new1.jpg
and...
images/new2.jpg
and...
images/new3.jpg
and...
images/new4.jpg
and...
images/new5.jpg
and...
images/new6.jpg
DOWN and
images/new7.jpg
and...
images/new8.jpg
and...
images/new9.jpg
and...
images/new10.jpg
```



Det första talet (4 ovan) är antalet gånger Message-sekvensen ska upprepas, det andra är tiden varje bild ska visas i millisekunder (200 ovan), och det tredje talet är antalet par av text och bildfil som följer (10 par ovan). Av paren skapar du Message-objekt vilka lagras på lämplig sätt i klassen.



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

I programmet *TestMessageproducer* kan du ersätta raden

```
MessageProducer mp = getArrayProducer(4, 500);
```

med

```
MessageProducer mp = new TextfileProducer("files/new.txt");
```

Om du kör programmet får du ett körresultat liknande det tidigare men med texterna i *new.txt*.

Bilderna är samma som i *ArrayProducer* som användes.

## 2.3 ObjectfileProducer

**ObjectfileProducer** ska implementera *MessageProducer*. Klassen läser, precis som *TextfileProducer*, från en fil. Denna fil ska vara formaterad på följande sätt:

5, 10000, 3, Message-objekt, Message-objekt, Message-objekt

Dvs:

times, delay, size, size st Message-objekt

När du är färdig med klassen kan du, i programmet *TestMessageProducer*, ersätta raden

```
MessageProducer mp = new TextfileProducer("files/new.txt");
```

med raderna

```
writeToOutputStream("files/new.dat", getArrayProducer(4, 500));
```

```
MessageProducer mp = new ObjectfileProducer("files/new.dat");
```

Om du kör programmet får du ett körresultat identiskt med det då *ArrayProducer* användes.

| ObjectfileProducer       |
|--------------------------|
| +times() : int           |
| +delay() : int           |
| +size() : int            |
| +nextMessage() : Message |

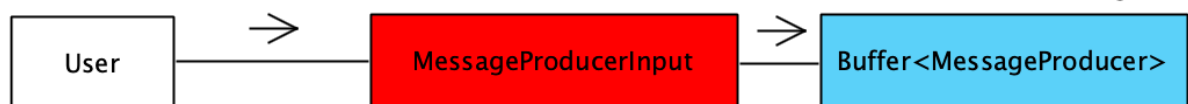
## 2.4 MessageProducerInput

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

1: addMessageProducer(Messageproducer)

2: put(MessageProducer)

3: get():M



Skriv klassen **MessageProducerInput**. I klassdiagrammet

ser du instansvariabler och metoder som

*MessageProducerInput* måste innehålla. Klassen ska

dessutom innehålla konstruktorn

```
public MessageProducerInput(Buffer<Messageproducer>)
```

| MessageProducerInput                      |
|---|
| -producerBuffer : Buffer<MessageProducer> |
| +addMessageProducer(Messageproducer m)    |

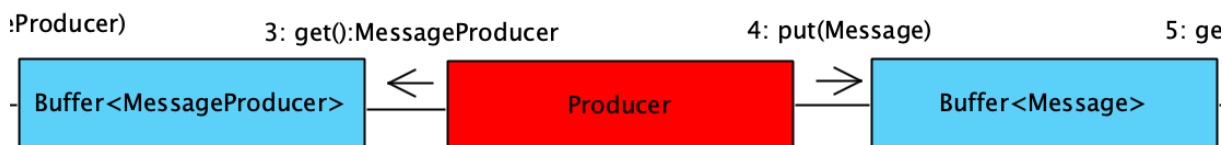
Med programmet **TestMessageProducerInput** kan du testa *MessageProducerInput* och att

*MessageProducer*-implementeringar placeras i bufferten. **MPCConsumer** ger exempel på hur en tråd

hämtar *MessageProducer*-implementeringar ur *Buffer*-instansen. *MPCConsumer*-klassen är i filen

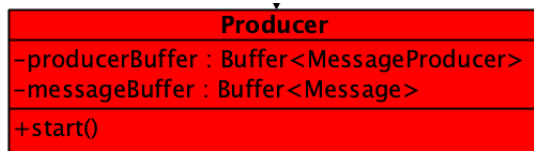
*TestMessageProducerInput.java*.

## 2.5 Producer



**Producer**-instansen ska fungera så här:

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



Därefter upprepas 3 och 4 på nytt.

Din uppgift är att skriva klassen **Producer**. Klassen ska använda en tråd och konstruktorn

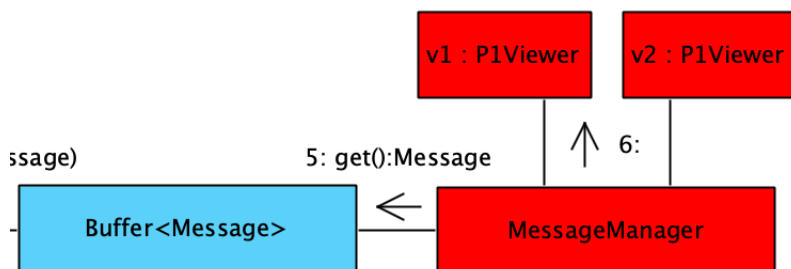
```
public Producer(Buffer<MessageProducer> prodBuffer, Buffer<Message> messageBuffer)
```

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

Programmet **TestProducer** (vilket förutsätter att *TestMessageProducerInput* fungerar) visar resultatet i en *Viewer*. Klassen **MessageConsumer** ger exempel på hur man hämtar *Message*-objekt ur *Buffer<Message>*-instansen. *MessageConsumer* är i filen *TestProducer.java*.

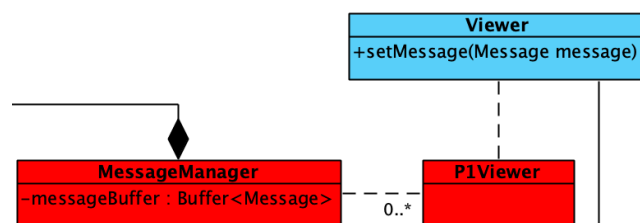
**MPConsumer** ger exempel på hur en tråd hämtar *MessageProducer*-implementeringar ur *Buffer*-instansen. *MPConsumer*-klassen är i filen *TestMessageProducerInput.java*.

## 2.6 MessageManager och P1Viewer



I denna del av systemet ska **MessageManager**-objektet hämta *Message*-instanser ur bufferten och se till att dessa visas i ett antal **P1Viewer**-instanser. Denna överföring ska du ordna genom att antingen använda klassen **Observable** och interfacet **Observer** eller genom att definiera ett interface för **Callback**.

Som du ser i **MainP1** så känner inte **MessageManager**-objektet till **P1Viewer**-objekten efter instansiering. Däremot känner **P1Viewer**-objekten till **MessageManager**-instansen:



```
P1Viewer viewer = new P1Viewer(messageManager, 640, 480);
```



Och kan därför registrera någon form av lyssnare.

När du skriver *P1Viewer* ska du använda dig av *Viewer*-klassen. *P1Viewer* ska antingen ha en *Viewer* eller ärva en *Viewer*. Avgör själv vilket som är lämpligast. Oavsett vilket du väljer måste *P1Viewer* ha en konstruktor som tar tre argument: en *MessageManager*-instans, *width* och *height*.

Denna del av system får du testa på egen hand. Till det levereras inget program.

OBS! Om du låter *P1Viewer* ha en *Viewer* som attribut så kan klassen innehålla en *getViewer*-metod. När viewern ska visas så används metoden (i *MainP1*):

```
Viewer.showPanelInFrame(v1.getViewer(), "Viewer 1", 100, 50);
```

## Bilaga 1

