Programmering Fortsättning

Inför Laboration 2

Joachim von Hacht

Innehåll

1 Strömmar

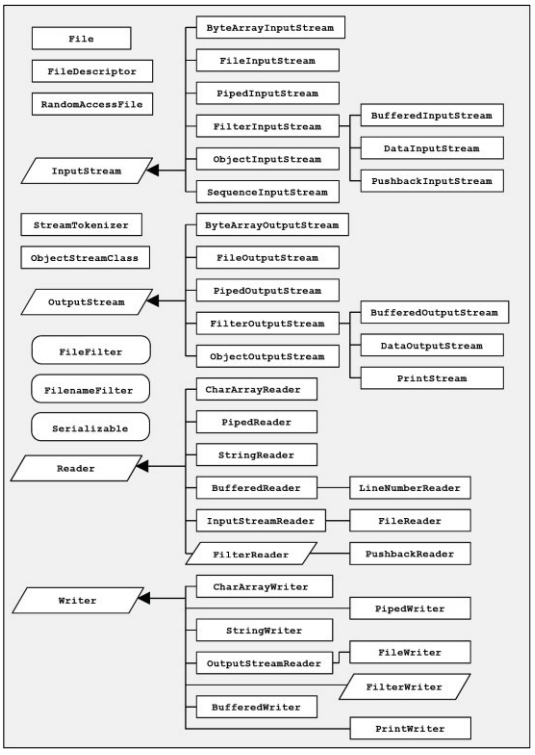
(Bara en översikt) En ström är en abstraktion för datautbyte. En ström är en sekvens av (någon slags) data som kommer från en källa (source) och flödar till en destination (destination).

- Man stoppar in data i ena änden och plockar ut i andra (typ sladd, slang..).

Tänkbara kombinationer; Program -> Fil, Fil-> Program, Program -> Nätverk, Databas -> Program, e.t.c.

1.1 Strömmar i Java

Strömmer representera i Java med klasser ut IO biblioteket. Klasshierarkierna ser ut som;



(Parallellogram = klasser som inte kan instansieras, Ovaler = interface, Rektanglar = klasser)

- Superklasser är Input/OutputStream och Reader/Writer.
- Allt med output/Writer är utströmmar (från prg. till annat).
- Allt med input/Reader är inströmmar (från annat till prg.).
- Allt med "stream" i namnet representerar byte-strömmar (rå binär data, t.ex. en bild).
- Allt med Reader/Writer i namnet representerar strömmar av tecken (Unicode).
- Det är effektivare att skicka /skriva/läsa/ större mängder data på en gång (allt går via operativsystemanrop, varje anrop tar viss tid). Buffered innebär att data samlas ihop i en buffer innan den skickas till/läses från strömmen. (kan behöv flush:a en buffrad ström för att t.ex. skicka det som finn i bufferten).

```
// s is some stream
s.flush();
```

- För att skapa specialicerade strömmar "wrappas" klasser "i varann" (m.h.a. konstruktor[1]). Exempel;

```
// Create buffered stream
BufferedReader in = new
 BufferedReader(new
  InputStreamReader(inStream));

// Later
String s = in.readLine();
```

[1] Egentligen designmönstret Decorator.

- Vissa strömmar kan man stoppa tillbaks data i (efter att man läst)

1.2 Att använda strömmar

- Alla Java-program har som default tre öppna strömmar System.in, System.out och System.err.

Alternativt man skapar något strömobjekt.

- I och med detta har man en öppnad ström man kan läsa från/skriva till m.h.a. olika metoder i strömobjektet.
- När man öppnar strömmen anger man var den andra änden skall kopplas (den första änden finns i vårt program).

Viktigt är att;

- En ström kräver resurser. Då man är klar med strömmen skall den stängas (även vid undantag, återkommer...). Måste göras explicit med en close-metod. Annars resource-leak.

1.3 Felhantering

Att arbete med strömmar innebär att programmet blir beroende av sin omgivning. Många fel kan uppstå t.ex;

- Problem med filer/filsystem
- Problem med rättigheter.
- Problem med nätverk.

Mycket vanligt att metoder kastar IOException, se vidare undantagshantering.

KOD streams.*

Produced with Lyx, the open source wordprocessor

2 Random Access Filer

Random access: Kan "hoppa" runt i filen och läsa (java.oi.RandomAccessFile). Se vidare på nätet eller bok,...

3 Serialisering

Klasserna ObjectOutputStream och ObjectOutputStream kan skriva/läsa hela objekt (= hela grafer av objekt, superobjekt...) till/från en ström. Kräver att klassen implementerar gränssnittet Serializable.

- Det är objektets tillstånd som serialiseras (metoderna tillhör ju klassen, delas av alla objekt).

Serializable Är ett s.k. märk (mark) interface d.v.s. det finns inga metoder. Specificerar inte ett explicit kontrakt utan används bara för att "märka" en klass (inte helt lyckat)[2].

Det implicita kontraktet för en klass som implementerar Serializable är;

- klassen skall identifiera vad som skall serialiseras. Anges inget serialiseras allt (som går) med en default-metod.
- Vill man undviks att data serialiseras kan man ange "transient" vid deklarationen m.m.
- Klassen måste ha tillgång till den första icke serialiserbara superklassens default-kontruktor!!
- Valfritt är att:
  - Implementer metoderna writeObject och readObject. Om dessa finns ersätter de defaultbeteendet.
  - OBS! Att readObject fungerar som en slags konstruktor (vi läser in data från strömmen och skapar ett objekt). Måste se till att objektet får ett giltigt starttillstånd. Kan vara mycket komplicerat.

Förutom felen med strömmer dyker ett antal andra tänkbara fel upp;

- Något i klassen är inte serialiserbart (eller i superklassen...).
- Man har serialiserat ett objekt, när man senare deserialiserar det har klassen en ny implementation (version). alt. klassen saknas.
- Säkerhetsfrågor; Deserialiserar vi ett "hackad" objekt? m.m.

Många (de flesta) standardklasser är serialiserbara, arrayer är serialiserbara[3].

- Skriver man till fil brukar man använda .ser som suffix.

Många klasser implementerar Serializable t.ex. JFrame. En seriös implementation av Serializable är icke-trivial bl.a. bör man ange serialVersionUID, så att objektet vi läser in matchar den klass vi har.

> "The serialization runtime associates with each serializable class a version number, called a serialVersionUID, which is used during deserialization to verify that the sender and receiver of a serialized object have loaded classes for that object that are compatible with respect to serialization".

- Vi "klämmer bara dit" ett serialVersionUID för att slippa varningen.

[2] Kallas ofta lite vagt "the general contract"

[3] Ett alternativ är att skriva ut objekten som XML, finns klasser som sköter detta.

Produced with Lyx, the open source wordprocessor

KOD serialization

3.1 Serialisering och arv

Om superklassen implementerar Serializable kommer alla subklasser att ärva detta!

- Man brukar undvika att göra klasser högt upp i arvshierakier Serializable för att minska kraven på implementationen av subklasserna (enl. ovan).

4 Java Remote Invocation (RMI)

Vi utökar våra applikationers möjligheter genom att tillåta metodanrop på objekt i andra JVM:er (över nätverk)[4].

BILD

4.1 RMI

Är ett subsystem[5] i JVM:en[6]. Systemet sköter allt på låg nivå (sockets, m.m). Vi skall (förhoppningsvis) slippa en massa detaljer på den nivån (utom några, se nedan). Vi kallar i fortsättningen:

- Server: Objektet som kan hantera anrop från andra JVMer.
- Klient: Objektet som anropar servern.

4.1.1 Nätverket

Några punkter;

- Alla datorer som skall köra applikationen (använda RMI) måste ha ett s.k. IP-nummer (IP-adress). Vi kommer att köra systemet på en enda dator (men med många JVM:er)[7]. För att simulera ett nätverk på en enda dator finns en speciell adress (kallas ofta loop back) med IP-numret 127.0.0.1. Använd denna.
- Det räcker inte med att veta adressen till datorn. Datorn kan ju köra många program. Hur når vi rätt program? Detta löser man genom att använda portar d.v.s. en koppling till ett speciellt program. T.ex kan man ange 127.0.0.1:9966 d.v.s. maskinen är 127.0.0.1 och porten 9966. Vi anger vilken port vårt program skall använda i koden (argument till programmet eller config-filer). Man kan få konflikter mellan portar, för oss gäller en port/applikation. Det finns gott om portar, omkring 65000 st. Portar mellan 0-1024 skall inte användas (de används av andra program t.ex. port 80 är den port web-servrar använder).
- IP-nummret ges som argument till JVM:en
- Exempel: http://java.sun.com/j2se/1.5.0/docs/guide/rmi/hello/hello-world.html

BILD

KOD se utdelad kod för labben

4.2 Remote objects

För att kunna ta emot metodanrop från andra JVM:er måste ett objekt "exporteras"

[4] Javas version av remote procedure call (RPC).
[5] Det finns inga klasser som representerar själva RMI-systemet. Vi kan inte anropa metoder på systemet.
[6] Finns annat också.
[7] Att få det att fungera på flera datorer kan vara knepigt men är intressant och lärorikt, prova...!

Produced with Lyx, the open source wordprocessor

till RMI-subsystemet. Då det är exporterat är det redo att ta emot anrop.

BILD

- För att kunna exporteras måste (server)objektet implementera gränssnittet Remote som är ett märkgränssnitt, kallas att objektet är ett "remote" objekt.

- De metoder som skall kunna anropas (remote methods, fjärrmetoder?) deklareras i ett "server"-gränssnitt som ärver Remote (extends Rremote). Samtliga metoder måste kasta RemoteException (throws RemoteException).

- Därefter låter man en serverklass implementera gränssnittet (och därmed Remote).

BILD UML

- Slutligen instansierar man och exporterar serverobjektet till RMI-systemet. För att exportera objektet används klassen UnicastRemoteObject.exportObject. Då man exporterar objektet får man en s.k. "stub" som resultat. Stubben kommer att fungera som en "standin" (proxy[8]) för remoteobjektet (servern).

- Man kan bara exportera ett objekt en gång. Försöker man exportera samma objekt igen får man en exception. Använd metoden UnicastRemoteObject.unexportObject.

[8] Finns ett designmönster som heter (remote) proxy.

Klienten använder en instans av stub:en. Alla anrop i klienten görs på stub:en som i sin tur vidarebefordrar dessa till det verkliga remoteobjektet över nätet, stub:en vet vart anropet skall vidarebefordras. D.v.s. det gäller för en klient att få tag i ett stubobjekt (som alltså skall köras i klientens JVM).

- OBS! Stub:en implementerar samma gränssnitt som servern men "lever" på klienten (i en annan JVM) d.v.s. servergränssnittet måste vara känt för klienten.

4.3 RMI registry

Servern sparar "stub" objektet (som man får då man exporterar objektet) under ett "känt" namn i en tabellapplikation (ett eget program som kan startas programmatiskt (i koden)), kallat RMI registret. Registret måste ligga på en känd nätadress, se nedan.

Klienter som vill anropa servern går först till registret och hämtar en kopia av stub:en (klienten måste alltså veta adress till registret och "namn" på stub:en).

BILD

- Obs! att det bara är den första stubb:en klienten måste få från registret. Efter detta kan servern vid behov returnera referenser till remote objects (stubbar).

4.4 Metodanrop

Anropen görs "at most once" d.v.s. anropet sker en enda gång. Anropet kan antingen lyckas eller misslyckas (RemoteException). Följande gäller;

- Parametrar och returvärden kan vara primitiva typer, lokala objekt eller fjärrobjekt (objektgrafer).

Produced with Lyx, the open source wordprocessor

- Primitiva typer skickas alltid som kopior (by value).
- Lokala objekt (sådana som inte implementerar Remote) skicka som kopior. Klientens ändringar av objektet påverkar inte objektet i servern (och tvärt om). Lokala objekt måste implementera Serializable.
- Fjärrobjekt (alla som implementerar Remote) skickas och returneras alltid som referenser (stubbar, by reference). Klienten kan i detta fall ändra ett objekt på servern. Det finns alltså bara ett objekt, det på servern.
- Om två parametrar i ett anrop refererar till samma objekt i anropande JVM kommer de att referera till samma objekt (kopia) också i anropad JVM.

4.5 Parallellism

Inkommade anrop exekversa normalt i en egen tråd. OBS! Om anropet skall uppdatera något Swing GUI måste man "lämna över" till Swings EDT (använd invokeLater()).

4.6 Säkerhet

Java har ett mycket omfattande system för säkerhet (som vi inte tar upp).

RMI innebär att körbar kod laddas ner till vår applikation, en uppenbar säkerhetsrisk. RMI kräver därför att man använder en SecurityManager. Rättigheter kan sättas mycket “finmaskigt” m.h.a. policy-filer (som måste finnas).

För att det skall fungera måste man dessutom ge JVM:en policyfilen som argument.

Produced with Lyx, the open source wordprocessor