Exercises for Package-Import

Package-Import-01

Bakgrund

Hittills har vi varit lata och skrivit klasser i Java-filer utan `package`-deklaration. Vi har kompilerat och kört klasserna (om de haft en main-metod) från samma katalog som klasserna ligger i. Vi har använt klasser som ligger i samma katalog utan att behöva importera något paket. Klasser som liger i `java.lang`-paketet, som t ex `Math` eller `String` har vi också kunnat använda utan att ha någon `import`-sats i början av filen. Hur har då detta fungerat och varför behöver vi inte importera klasser från samma katalog eller `java.lang`?

Svaret är att alla klasser ligger i ett paket, vare sig vi vill det eller inte. Skriver vi en klass utan en `package`-deklaration (så som vi gjort hittills) så hamnar klassen ändå i ett paket utan namn. När java och javac letar efter filer i namnlöst paket, så letar java och javac i aktuell katalog. Det är därför som vi måst stå i samma katalog som klasserna vi vill kompilera eller köra (om de har en main-metod kan vi köra dem med java). När det gäller klasserna i `java.lang` så är de så fundamentala att man beslöt sig för att alla dessa klasser skall vara åtkomliga direkt utan att vi behöver skriva `import java.lang.*` (även om det går att skriva så, så är det meningslöst och behövs inte).

Det är ju smidigt att kunna skriva t ex `String name = "Henrik";` utan att antingen behöva importera `java.lang.String` eller skriva det långa namnet: `java.lang.String name = "Henrik";`.

Men vad är då ett paket, *egentligen*? Ni kan tänka på paket ur två perspektiv. Dels syntaktiskt (vilka skrivregler finns för hur vi skapar och använder paket i språket Java) och dels ur ett användarperspektiv (hur använder jag klasser som ligger i ett paket?).

Om vi börjar med att skapa ett paket så är syntaxen att en klass som ska ingå i ett paket har en deklaration `package paketnamn;` som sin första instruktion i Java-filen för klassen. Och paketnamn består av namnet på en katalog, eller flera kataloger som sökväg avskilda med "." (en punkt), där klassen ligger i filsystemet. Ett exempel är att om vi deklarerar att en klass ska tillhöra paketet exercises så skriver vi som första instruktion i Java-filen `package exercises;` och lägger Java-filen i en katalog med samma namn.

Ofta så har man nivåer av paket och det kan vara t ex

`package exercises.arrays;` eller `package exercises.loops;`

Man tänker att klasser som är övningsuppgifter ligger i någon katalog under katalogen `exercises`. Eftersom vi övar på olika områden så har vi underkataloger för olika övningar, vi lägger övningar om arrayer i katalogen `exercises/arrays/` och övningar om loopar i katalogen `exercises/loops/`. De klasser som är

övningsuppgifter på arrayer kommer då deklarera att de tillhör paketet som heter `exercises.arrays` och de klasser som är övningsuppgifter om loopar att de tillhör paketet `exercises.loops`. Respektive `package`-deklaration blir då:

`package exercises.arrays;` respektive `package exersices.loops;`

Man kan ha många fler nivåer och regeln är då att varje katalog kommer vara med i paketnamnet och mellan katalogerna lägger vi en punkt. Exempel: Filer i paketet `exersices.loops.forloops` ligger i katalogen `exercises/loops/forloops/` och filer i paketet `exercises.loops.while` ligger i katalogen `exercises/loops/while/`. Man kan alltså tänka sig att paket i Java motsvarar sökvägar i en dators filsystem. Vi skulle kunna kalla det paketsökväg eller liknande.

När vi vill använda klasser i ett paket så använder vi som vi sett hittills `import`-satsen. Man kan importera alla klasser i ett paket genom att skriva t ex `import exercises.arrays.*;`. Med "importera" menas bara att man vill göra klassnamnen tillgänliga i en annan klass, utan att behöva skriva hela långa namnet `paketnamn.Klassnamn` varje gång man vill använda en klass. Man kan också importera bara en enstaka klass från ett paket, t ex `import java.util.ArrayList;`.

När en klass deklarerat att den tillhör ett paket, så kan man bara använda klassen genom sitt långa namn, `paketnamn.Klassnamn` (eller importera den). Vill vi i en annan klass referera till `ExerciseArray1` måste vi använda det långa namnet `exercises.arrays.ExerciseArray1`. Samma gäller när vi vill kompilera en klass i ett paket på vår dator. Tillhör klassen `ExerciseArray1` paketet `exercises.arrays;` så kan vi bara kompilera klassen genom hela dess "paketsökväg":

```
javac exercises/arrays/ExerciseArray1.java
```

En sökväg som inte börjar med "/" (UNIX/ GNU/Linux) eller "\" (Windows) är en relativ sökväg som utgår från"aktuell katalog", dvs där vi"står".

Detta betyder att vi måste stå i katalogen "ovanför" katalogen `exercises` när vi kompilerar, så att katalogsökvägen verkligen är en relativ sökväg från där vi står till Java-filen `ExerciseArray1.java`. Har klassen en main-metod och vi vill köra den, gäller samma princip. Nu vill vi köra en klass och klassen ligger i ett paket. Då måste vi tala om för kommandot `java` klassens fullständiga namn, dvs:

```
java exercises.arrays.ExerciseArray1
```

Så, när vi vill kompilera, då befinner vi oss i datorvärlden och måste ge sökvägen till filen på samma sätt som paketet är uppbygt. Varje "subpaket" motsvarar en katalog med exakt samma namn och vi avskiljer ju kataloger med "/" (på min dator, på windowsdatorer använder man "\"). På samma sätt som när vi i kommandotolken vill lista innehållet i katalogen `exercises/arrays/` för att se vilka filer som ligger där.

När vi vill köra en klass med main-metod och klassen tillhör ett paket däremot, då befinner vi oss i Java-världen, kan man säga. Kommandot `java` bryr sig inte om filer utan om namn på klasser. Så då måste vi ange det långa namnet på klassen enligt ovan.

Sammanfattning: Kompilera: "relativ sökväg till Java-filen". Köra: "Det långa klassnamnet med paket, punkter och själva klassnamnet". I bägge fallen måste vi "stå" i katalogen ovanför första paket-katalogen.

Sluta babbla och ge oss övningarna någon gång!

OK.

Look at the attached Java file (Hello.java).

1. Compile and run it
2. Add a package name to the class. Put the following line before the class definition.

   package something;

Compile and run the program. Why doesn't it work?

3. Create a folder (directory) called something. Move the Java file to that directory. And move into the something directoy. Typically you can do the following in a termnial window (at least on GNU/Linux and Mac):

   mkdir something mv Hello.java something/

4. Compile the program, like this

   javac something/Hello.java

5. Run, or rather try to run, the program like this:

   java Hello

Huahhh... it didn't work did it? Why oh why?

6. Move to the something directory - where the class file (compiled Java file) is and try to run the program.

   cd something java hello

Arggrgr, still no success. Ok, let's see what Java tells us:

```
Exception in thread "main" java.lang.NoClassDefFoundError: Hello (wrong name: something/Hel
```

Ok, wrong name. It seems as if the class is called something/Hello and we're invoking it with the name Hello. How about invoking it in another way.

7. Move back to the previous folder
   cd ..

And invoke the program with the complete package name:

```
java something.hello
```

Yes!!! Finally....

8. But let's not give up on (6), so lets move to something directory again
   cd something

and invoke Java this way

```
java -cp .. something.Hello
```

Nice, but why?

Package-Import-02

Övningar Package-Import

Syfte

2.1 Katalogstruktur

Skapa en katalog som ni kallar för "exercises":

```
mkdir exercises
```

Skapa en underkatalog till exercises som heter "main". Skapa en annan underkatalog till exercises som heter "domain".

Ni ska ha följande katalogstruktur när ni är klara:

```
.
‘-- exercises
    |-- domain
    ‘-- main
```

Ni befinner er i "." dvs "current directory/aktuell katalog" är ovanför exercises.

2.2 Paket och klasser

Skriv en klass Message i filen `exercises/domain/Message.java`

Klassen skall deklarera (på första raden i filen) att den tillhör paketet `exercises.domain` .

Klassen skall ha en instansmetod (dvs inte static!) som heter `public void talk()` och som i kroppen skriver ut strängen "Hello package world!" med `System.out.println`.

Skriv en klass PackageMain i filen `exercises/main/PackageMain.java`

Klassen skall deklarera att den tillhör `exercises.main` och den ska ha en vanlig main-metod.

I main-metoden skall det skapas en referensvariabel till ett objekt av klassen `exercises.domain.Message` och sedan skall metoden `talk()` anropas via referensvariabeln.

Ni får välja om ni vill använda det långa namnet `exercises.domain.Message` eller om ni i stället vill använda `import` för att få tillgång till det korta namnet `Message` inne i main. Tänk på att om ni använder `import` så måste det stå under paketdeklarationen eftersom regeln var att eventuella paketdeklarationer måste vara första instruktionen.

Nu ska katalogstrukturen se ut så här:

```
.
‘-- exercises
    |-- domain
    |   ‘-- Message.java
    ‘-- main
        ‘-- PackageMain.java
```

2.3 Kompilera

Kompilera filen `exercises/main/PackageMain.java`

Tänk på att du måste stå i katalogen ovanför exercises för att kunna kompilera med den relativa sökvägen ovan.

Om ni får kompileringsfel, åtgärda dem tills ni kan kompilera. Läs noga eventuella felmeddelanden innan ni försöker åtgärda.

2.4 Verifiera att alla nödvändiga klasser är kompilerade.

Om ni lyckats kompilera `PackageMain.java` och i den lyckats använda klassen `Message`, så har faktiskt klassen `Message` automatiskt blivit kompilerad också.

Verifiera detta genom att lista innehållet i katalogerna där Java-filerna ligger genom att ge de relativa sökvägarna till katalogerna:

Unix/Mac/GNU/Linux:

```
ls exercises/main/
ls exercises/domain/
```

Windows/DOS:

```
dir exercises\main\
dir exercises\domain\
```

Det skall ligga såväl Java-filer som class-filer i katalogerna.

2.5 Kör PackageMain

Kör javaprogrammet PackageMain med kommandot `java`.

Tänk på att du nu måste använda "Java-namnet" på klassen, dvs det långa namnet inklusive paketet.

Tänk på att du måste stå i katalogen ovanför exercises för att detta ska fungera.

Om texten `Hello package world!` skrivs ut på skärmen har ni lyckats skapa två paket, använda klasser i ena paketet från en klass i det andra paketet, kompilerat alla klasser som behövs och slutligen lyckats köra en main-metod i en klass som ligger i ett paket. Bra jobbat!

02 (del 6-11 frivilligt) Lite svårare men inte mycket

Det är god sed att inte blanda källkodsfilerna (`.java`-filerna) med klassfilerna (`.class`-filerna) i samma katalog. Man brukar ha en katalog för källkodsfilerna som heter `src`.

Er uppgift är att lära er att kompiera filer men tala om för `javac` att ni vill att resultatet, dvs klassfilerna, skall hamna i en annan katalog.

För att klassfilerna från en kompilering skall hamna någon annan stans än i samma katalog som respektive Java-fil, så använder man en så kallad flagga till `javac`.

Flaggan för att tala om att man vill ha en annan destination för klassfilerna är `-d` och ett exempel är:

```
javac -d ../bin exercises/main/PackageMain.java
```

Katalogen `../bin` måste finnas för att det ska fungera, naturligt nog.

Vi använder här en relativ sökväg till katalogen `bin` genom att använda `..` som betyder “katalogen ovanför där jag är”.

Uppgiften här är att separera klassfilerna till katalogen bin och källkodsfilerna till katalogen src.

2.6 Radera gamla klassfiler

Ta bort alla klassfiler från uppgift 02. Det vill säga, radera filerna:

`exercises/main/PackageMain.class`

`exercises/domain/Message.class`

2.7 Skapa katalogerna src och bin

Skapa två nya kataloger:

`mkdir bin` `mkdir src`

2.8 Flytta källkoden till src

Flytta hela katalogträdet exercises till src:

UNIX/MAC/GNU/LINUX:

```
mv exercises src/
```

Windows/DOS:

```
move exercises src\
```

(eller i värsta fall, använd filutforskaren men ni bör kunna grundläggande filhantering från kommandoraden!)

Nu ska katalogstrukturen vara som följer:

```
.
|-- bin
‘-- src
    ‘-- exercises
        |-- domain
        |   ‘-- Message.java
        ‘-- main
            ‘-- PackageMain.java
```

2.9 Kompilera PackageMain.java men låt klassfilerna hamna i bin!

Kompilera filen PackageMain.java med flaggan `-d ../bin` men kom ihåg reglerna för att kompilera filer i paket. Du måste alltså gå ned i katalogen src för att kompilera.

```
cd src
```

UNIX/MAC/GNU/Linux:

```
javac -d ../bin exercises/main/PackageMain.java
```

Windows/DOS:

```
javac -d ..\bin exercises\main\PackageMain.java
```

Vad du nu gjort är att be `javac` kompilera `exercises/main/PackageMain.java` som vanligt med tillägget att `javac` ska lägga klassfilerna i katalogen `bin` som ligger ovanför aktuell katalog. Kom ihåg att du gjorde `cd` katalogen `src`. Katalogen `bin` låg ju på samma nivå som `src` (bredvid skulle man kunna säga), så att om du står i `src` så är den relativa sökvägen till `bin` exakt `../bin` (eller om du är en Windows-användare: `..\bin`.

2.10 Undersök resultatet.

Om du gjort rätt så har inget ändrats i `src/exercises/main/` eller `src/exercises/domain/`. Däremot har det hänt grejer i katalogen `bin`!

Gå upp ett steg (`cd ..`) så att du står ovanför såväl `src` som `bin`.

Lista filerna i `bin`. Det ska ligga en katalog `exercises` i `bin` nu. Lista `bin/exercises`. Det ska ligga två kataloger i `bin/exercises` nu: `bin/exercises/domain` och `bin/exercises/main`. Dessa kataloger ska innehålla klassfilerna som motsvarar Java-filerna i `src`. Så här ska hela katalogträdet se ut om du lyckats:

```
.
|-- bin
|   `-- exercises
|       |-- domain
|       |   `-- Message.class
|       `-- main
|           `-- PackageMain.class
`-- src
    `-- exercises
        |-- domain
        |   `-- Message.java
        `-- main
            `-- PackageMain.java
```

2.11 Kör PackageMain

Du kommer ihåg reglerna för hur man kör en klass som ligger i ett paket? Precis som förut så får vi ge `java` det långa namnet, inklusive paketnamn. Men vad är det `java` egentligen kör? Det är den kompilerade klassfilen. Så för att köra klassen `exercises.main.PackageMain` så måste vi gå ned i `bin`.

```
cd bin
```

Kör `klassen`exercises.main.PackageMain`med kommandot`java‘.

Om texten `Hello package world!` skrivs ut på skärmen har ni lyckats skapa två paket, använda klasser i ena paketet från en klass i det andra paketet, separerat källkodsfilerna från klassfilerna, kompilerat alla klasser som behövs (och sätt till att klassfilerna hamnar i katalogen `bin`) och slutligen lyckats köra en main-metod i en klass som ligger i ett paket, och detta i en katalog som bara har kataloger med klassfiler. Bra jobbat!

Problemshooting

Se filmen om paket. Flera gånger om det behövs. Läs texten först i detta dokument. Flera gånger om det behövs. Var noga med att följa instruktionerna i varje steg noggrannt. Om det fortfarande inte fungerar, be en klasskamrat som lyckats om hjälp. Om inte det hjälper, be en handledare eller lärare om hjälp. Läs alla kompileringsfel noggrannt.

Om ni har problem att kompilera, skilj på syntaxfel som har med Java i allmänhet att göra (glömt semikolon osv) och sådana fel som har med paket att göra. Vanligtvis så brukar fel kring paket säga något i stil med “Class not found” “Class X must be defined in a file called X.java” osv.

Titta på filmen och de exempel som ges på kompileringsfel och körningsfel när vi står i fel katalog.

En strategi i denna laboration är att strunta i paketen först och skapa bägge filerna i samma katalog och se till att de kompilerar utan att vara medlemmar i paket. Sedan skapa katalogerna för paketen, flytta ned filerna i katalogerna och lägga till `package`-deklarationer och `import`-satser som behövs.

Kom ihåg huvudregeln: Varje del av ett paketnamn som avskiljs med “.” (punkt) motsvarar en katalog med exakt samma namn (utom Klassnamnet som motsvarar filen Klassnamn.java). Paket är ett slags sökväg och sökvägen är relativ till det bibliotek du står i när du försöker kompilera en fil eller köra en klass (som har en main-metod).

Lycka till!