Paket i Java

Joachim von Hacht

The Java Language Specification (JLS)

Beskriver formellt språket Java (senaste Java SE 7 edition)
http://docs.oracle.com/javase/specs/

- Ofta tung läsning (15.12.1) med det finns enklare avsnitt (4.1, 4.3.1)

Hur man får skriva program (syntax)

```
// Bad
4 = x;
```

Betydelsen av olika uttryck (semantik)

```
// Syntactically correct. What's the meaning?
x++;    // See, JLS 15.14.2, 15.15,1
++x;
```

Kompilatorn kontrollerar att alla regler följs

Identifierare

"An **identifier** is an unlimited-length sequence of Java letters and Java digits, the first of which must be a Java letter."**// JLS 3.8**

A "Java letter" is a character for which the method Character. isJavaIdentifierStart(int) returns true.

A "Java letter-or-digit" is a character for which the method Character. isJavaIdentifierPart(int) returns true.

An identifier cannot have the same spelling (Unicode character sequence) as a keyword (§3.9), boolean literal (§3.10.3), or the null literal (§3.10.7), or a compile-time error occurs.

Namn och Identifierare

"A name is used to refer to an entity declared in a program."

"There are two forms of names: simple names and qualified names."

- A simple name is a **single identifier**.
- A qualified name consists of a name, a "." token, and an identifier. **// JLS 6.2**

Namn och betydelse

"The meaning of a name depends on the **context** [sammanhang, omgivning] in which it is used."
// JLS 6.5

The determination of the meaning of a name requires three steps:

First, context causes a name syntactically to fall into one of six categories: PackageName, TypeName, ExpressionName, MethodName, PackageOrTypeName, or AmbiguousName.

Second, a name that is initially classified by its context as an AmbiguousName or as a PackageOrTypeName is then reclassified to be a PackageName, TypeName, or ExpressionName.

Third, the resulting category then dictates the final determination of the meaning of the name (or a compile-time error if the name has no meaning).

Deklarationer

"A declaration **introduces** an entity into a program and includes an identifier that can be used in a **name** [as a name] to refer to this entity" // JLS 6.1

Example: A *class body* may contain declarations of members of the class, that is, fields (§8.3), classes (§8.5), interfaces (§8.5) and methods (§8.4)
// JLS 8.1.6

[Not all identifiers in programs are names ... In declarations, where an identifier may occur to **specify** the name by which the declared entity will be known]

Synlighetsområde

"The **scope [synlighetsområde]** of a declaration is the region of the program within which the entity declared by the declaration can be referred to using a **simple name**" // JLS 6.3

- Frånsett context får ingenting i ett program heta samma sak men...
- ...synlighetsområden gör att vi kan använda samma namn (slipper hitta på nya)
- I Java finns bl.a. synlighetsområdet **block scope** (mellan { ... })
- "The package java is always in scope"
- Best practices: Use minimal scopes

Synlighetsområde för Klasser

"The scope of a top level type [class, enum or interface] is all type declarations in the **package** in which the top level type is declared" //JLS 6.3

Paket

"[Java] Programs are organized as sets of **packages [paket]**. Each package has its own set of names for types" // JLS 7

"The members of a package are its **subpackages** and all the top level class types [class, enum] and top level interface types declared in of the package." //JLS 7.1

"A package may **not** contain two members of the same name, or a compile-time error results."//JLS 7.1

Paketdeklaration

"A package declaration ... specifies the name of the package to which the compilation unit [class, enum or interface] belongs" //JLS 7.4.1

```
package mypackage;  //Declaration first in file
import ...
// Qualified name of class is mypackage.MyClass
public class myClass {
    ...
}
```

Paketnamn med enbart små bokstäver (**inte** camelCase)

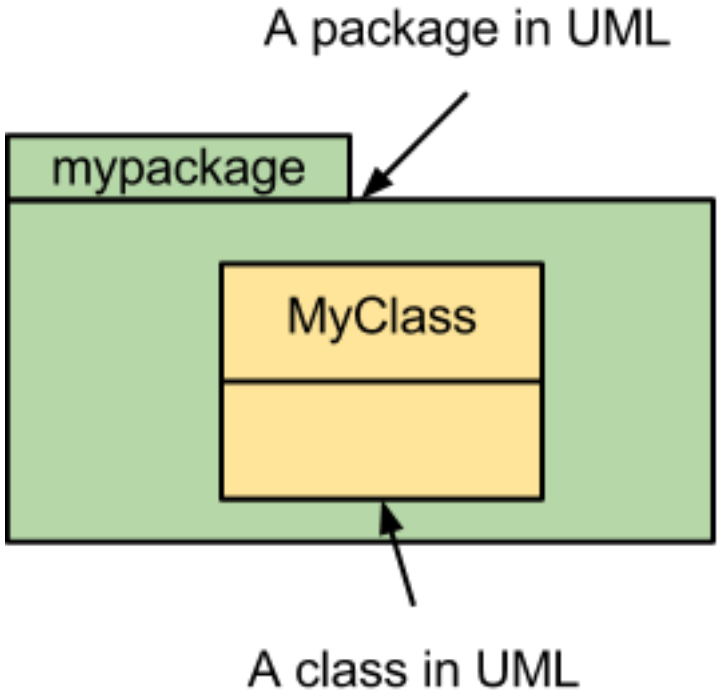
Saknas deklaration: compilation unit in "unnamed package"

- Kan inte ha subpaket!

Paket i UML

UML = Unified Modeling Language

```
package mypackage;
import ...
public class myClass {
    ...
}
```

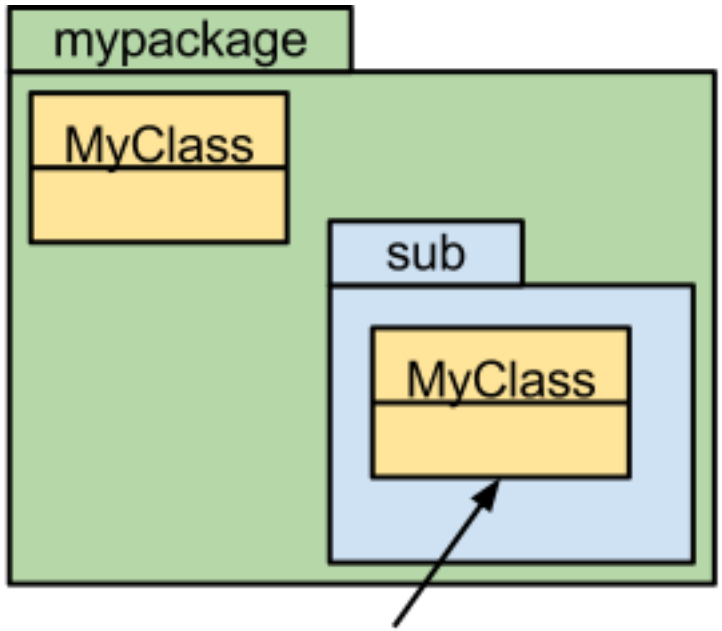


Paket Hierarkier

Paketnamn har en hierakisk namnstruktur

```
package mypackage;
import ...
public class myClass {
    ...
}

package mypackage.sub;
public class myClass {
    ...
}
```



Qualified name: mypackage.sub.MyClass

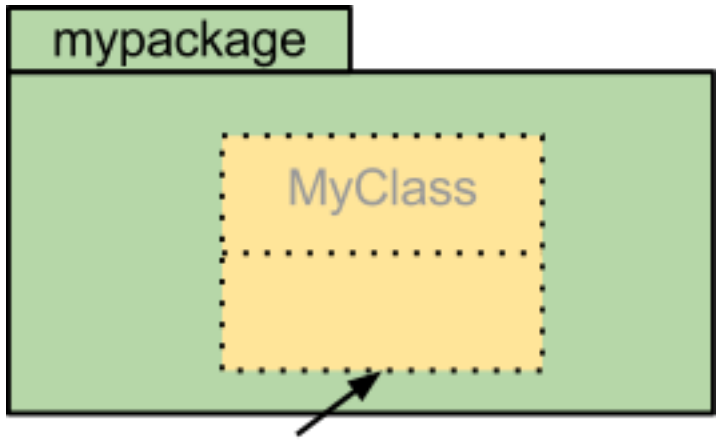
Sub-paket har ingen speciell relation (access), till omslutande paket, bara ett sätt att organisera

Paket och Synlighet

A top level type [class, enum, interface] is accessible outside the package that declares it **only** if the type is declared public. // JLS 7

En ny teknik för "information hiding", **designmöjligheter!**

```
package mypackage;
// No public before class
class myClass {
    ...
}
```



Paket-intern access

Obs! Att om man utelämnar private, protected eller public i klasser så gäller "**default access**"

Innebär att alla typer i samma paket har åtkomst

```
package mypackage;
class myClass {
    // Oh, oh, all classes can use! BAD!
    int secret = ..
}
```

Ange alltid accesspecifikation!

Import

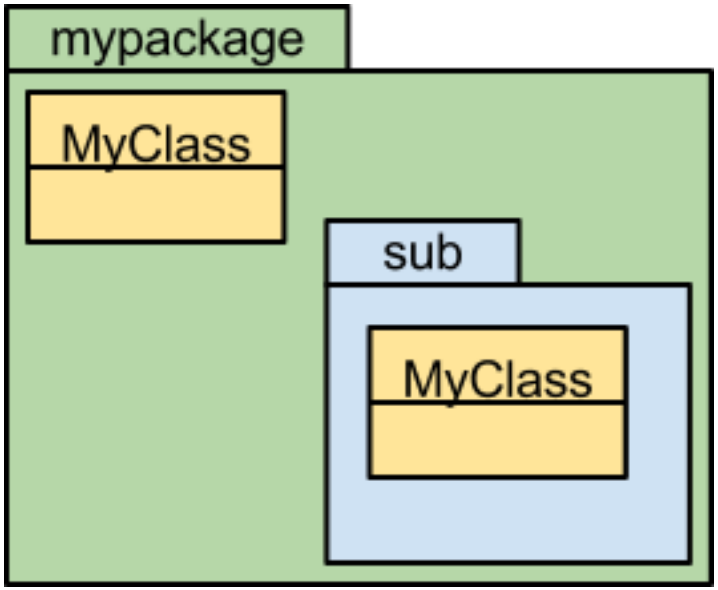
"An import declaration allows ... a named type to be referred to by a simple name ... Without the use of an appropriate import declaration, the only way to refer to a type declared in another package ... is to use a fully qualified name" // JLS 7.5

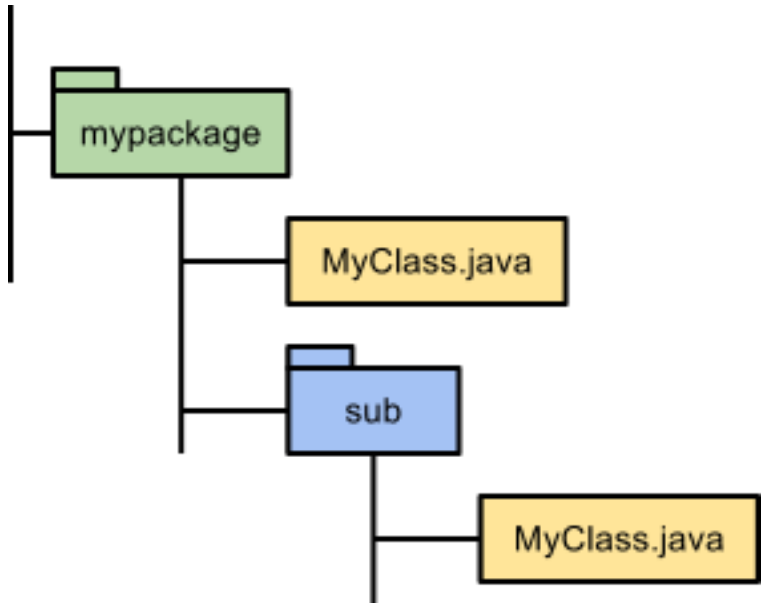
```
import java.util.Vector;
...
// Possible to use simple name
Vector<String> v = new ...
```

Allt i java.lang.*; **importeras automatiskt**

Paketrepresentation

Paket representeras i vårt fall som mappar i filsystemet





Filsystem

Unika Paketnamn

Modern programutveckling innebär att kod som producerats av andra (tredjepart) används (bibliotek). Kan leda till namnkollisioner

"Developers should take steps to avoid the possibility of two published packages having the same name by choosing *unique package names* for packages that are widely distributed"
//JLS 7.7

Använd "den omvända" internetdomänen som paket-prefix (inledning) tillsammans med Chalmers id, tillsammans med applikationens namn. Därefter subpaket enlig nedan

```
// This should be unique (no types in prefix packages)
se.chalmers.cid.myapp.mypackage.MyClass
```

Exekvering med Paket

JVM:en måste veta var alla klasser finns (för att kunna ladda in dessa)

- Alla standard Java klasser hittas automatiskt
- Klasser i aktuell katalog är kända (om de inte ingår i något paket)

För att ange var paketerade klasser finns används **classpath (-cp)**

- Ange namn på katalog som innehåller toppaketet (se)
- Därefter kvalificerat namnet på klass med main-metod
- Därefter eventuell kommandorads argument (String args[])

```
$ java -cp ./bin se.chalmers.hajo.myapp.Main Pelle
```

Paket och Design

Hur placera klasser i paket?

- Har med modulär design att göra (kommer mer i senare kurser)

"Modular design, or "modularity in design", is an approach that subdivides a system into smaller parts (modules) that can be independently created and then used in different systems to drive multiple functionalities. A modular system can be characterized by the following:

(1) Functional partitioning into discrete scalable, reusable modules consisting of isolated, self-contained functional elements
(2) Rigorous use of well-defined modular interfaces, including object-oriented descriptions of module functionality
(3) Ease of change to achieve technology transparency and, to the extent possible, make use of industry standards for key interfaces." **//Wikipedia**

Paket och Design i Denna Kurs

Enkel uppdelning

- Klassen med main-metoden (Main.java) placeras som enda klass i paketet som ger applicationens namn (myapp)
- Klasser som ingår i OO-modellen placeras i ett paket (core eller model)
- Subsystem, klasser som används till att utföra en speciell service i eget paket (med utåt sammanhåller gränssnitt)
- GUI-klasser för sig
- Hjälpklasser (utilities)

Dölj klasser som inte används av andra paket