Programmering Fortsättning

Föreläsning 5

Joachim von Hacht

Innehåll

1 Fortsättning: Arv


(BJ 10, JLS överallt...) Arv (inheritance) är mycket komplicerat, så tveksamt att man rekommenderar komposition i stället för arv!

1.1 Subtyp och subklass

Är inte samma sak....!

- Subtypen måste kunna hantera alla operationer och värden i supertypen. Exempel; Det finns ingen operation på rationella tal som inte kan göras på ett heltal. Rationella tal kan ses som en subtyp till heltalen.
- Subklass säger bara att kod delas. Det är lätt att åstadkomma en subklass som inte är utbytbar mot sin superklass, *trots att de har exakt samma publika gränssnitt* (metoder). Subklassen är alltså inte en riktig subtyp (måste undvikas, se nedan)!

För att en subklass (extends) skall fungera som en subtyp måste den uppfylla/respektera superklassens kontrakt (kan vara svårt att avgöra vilket kontraktet är).

1.1.1 Liskov substitution principle[1]

Säger vad som krävs för att en subklass skall fungera som en subtyp.

[1] Barbara Liskov, berömd mjukvaruexpert på MIT.

- subklassens @pre får inte vara starkare än superklassens. Får inte kräva mer än superklassen.
- subklassens @post får inte vara svagare än superklassens. Får inte lova mindre.
- subklassen måste upprätthåll superklassens invarianter.

KOD inherit.liskov

1.2 Metoder

En subklass kan;

- deklarera en ny metod (som alltså inte finns i superklassen). Superklassobjektet kan aldrig exekvera denna!
- ärva (inherit) en metod från superklassen. Metoden syns inte i subklassens kod. Superklassens kod körs vid anrop på subklassobjekt.
- override en metod i superklassen. Objektets runtimetyp avgör vad som körs (som gränssnitt).
- dölja (hide) en *klassmetod* i superklassen om samma signtur. Anrop i subklassen kommer att använda den "döljande" metoden (den i subklassen). En statisk metod får inte dölja en instansmetod (som sagt).
- Dessutom kan overload uppkomma om en klass i super och subklass har samma namn men olika parametrar (kan uppstå av misstag).
  - Påminner om: Ange alltid @Override för metoden som skall overridas.

BILD

KOD inherit.method, inherit.overload

1.2.1 Varians

Ko- och kontravarians (co- contravariance) gäller bl.a. parametrar och returtyper i samband med overriding (referenser inte primitiva)[2].

BILD

- Vid overriding i subklasser accepterar Java kovarianta returtyper (returtyperna har samma arvsförhållande (samma riktning, co-) som klasserna. Antag A :> B och C :> D . I A finns metod; public C get() och i B overridden till public D get() .

```
A a1 = new A();
A a2 = new B();
C c1 = a1.get();
//c2 runtime är D
C c2 = a2.get(); //Säkert
```

- Parametertyper i Java är invarianta (måste vara exakt samma typ, inte sub eller superklass)[3].

KOD inherit.covariance

[2] En typ regel eller typ konverteringsoperator är kovariant om typförhållandena bibehålls (:>). Om de vänds blir det kontravarians i övrigt invarians.

[3] Tänkbart med kontravarianta parametrar d.v.s. parameter i subklass är basklass till parameter i superklass, d.v.s. kräver mindre, ofarligt.

1.3 Instans- och klassvariabler

En subklass kan;

- deklarera en ny instansvariabel (som alltså inte finns i superklassen).
- ärva superklassens instansvariabler. Finns osynliga variabler i subklassens kod.
- dölja superklassens instansvariabler genom att ange samma namn för en variabel. Inte bra! Finns nu två variabler med samma namn! Anges inget används den i subklassen.
  - Metoder i superklassen använder superklassens variabler! Undvik!
- döja (hide) superklassens *klassvariabler* (static) genom att ange samma namn. Inget ärvs.

BILD

KOD inheritance.fields

> "In this respect, hiding of methods differs from hiding of fields (§8.3), for it is permissible for a static variable to hide an instance variable. Hiding is also distinct from shadowing (§6.3.1) and obscuring (§6.3.2)."//JLS 8.4.8.2

OBS! Variabler är inte polymorfa, deklarerad typ gäller!

> "The following distinction between invoking instance methods on an object and accessing fields of an object must be noted. When an instance method is invoked on an object using a reference, it is the class of the current object denoted by the reference, not the type of the reference, that determines which method implementation will be executed. When a field of an object is accessed using a reference, it is the type of the reference, not the class of the current object denoted by the reference, that determines which field will actually be accessed."

1.4 this

Om vi anropar en metod på ett object så syftar this alltid på runtime typen (polymofism!), även om kod från superklassen exekveras, *this byter aldrig typ*! Superobjektet är en del av subobjektet.

BILD

KOD inherit._this

1.5 Arrayer

- För arrayer av primitiv typer gäller invarians d.v.s. int[] är inte en subtyp till long[].
- För arrayer med referenstyper gäller om S :> T så S[] :> T[]. Exempel Object[] :> String[].
  - OBS! Detta bryter mot typsäkerheten (the array loophole)! Beror på kovarians. Eventuellt ett fel vid konstruktionen av Java[4].
    Tvingas införa runtime kontroll för ArrayStoreException, d.v.s. varje gång man stoppar in något i en array kostar det.

[4] Generiska typer löser problemet, återkommer...

> "...an assignment to an element of an array whose type is A[], where A is a reference type, is checked at run-time to ensure that the value assigned can be assigned to the actual element type of the array, where the actual element type may be any reference type that is assignable to A."//JLS 10.10

- Förutom att ha Object som supertyp har alla arrayer med primitiva element två supertyper till (i form av två interface) Cloneable och Serializble
  - Om p primitiv typ, then: Object :> p[], Cloneable :> p[], Serializable :> p[]

  Vad interfacen står för återkommer vi till.

KOD inherit.array

1.5.1 Bizzare

Följande går inte (trots att String[] är typkompatible med Object);

```
class A extends String[]{
  :
}
```

1.6 Multipelt arv

1.6.1 Variabler

En klass kan ärva flera variabler med samma namn. Om så måste det kvalificerade namnet eller super användas (d.v.s. klass.variabel) annars compile time error, Undvik!

1.6.2 Metoder

Java stöder som sagt bara enkelt implementationsarv för metoder d.v.s.

```
// Error
class A extends B, C {
:
}
```

BILD Diamond

1.7 Exponering via subklasser

Inget hindrar en subklass att skapa en public-metod som exponerar en protected variabel i superklassen.

- En icke-muterbar klass kan alltså bli muterbar om man slarvar med implementationen

KOD inherit.expose

1.8 Final

- En final metod kan inte omdeklareras i subklassen.
- En final static metod kan inte döljas i subklassen.
- En final klass kan inte användas som superklass. Använd vid design för att förhindra arv och exponering.

2 Fragil base class problem

Har att göra med det starka beroendet mellan super och subklass vi implementationsarv.

- Att byta representation i basklassen skall inte kunna knäcka subklasser,

man säger att representationsbyte är en tillåten operation (legal operation)...

- ...samma sak med implementation av metoder, ändrar man en metod skall inte subklasser knäckas, men...
- ...tyvärr visar det sig ofta att detta ändå sker!
- Subklassen är ofta beroende av basklassens implementation;
  - subklassen känner till implementationen (bryter inkapsling, ökar komplexitet).
  - ändras implementaion i basklass kan subklassen bryta ihop (sluta fungera).
  - subklasser kan även knäcka basklassen!
- *"The fundamental problem of inheritance"*

KOD inherit.fbc

2.1 Design

- Igen..., föredra komposition före arv.
- Designa för arv eller förbjud (final)
  - Vissa klasser skall aldrig ärvas t.ex. behållare.
- "A class can take these general policies with respect to subclassing, in order of increasing liberality :
  - disallow it completely , declare the class as final
  - allow it, but disallow all overrides, declare all methods as final
  - allow it, and permit some overrides, declare some methods as non-final
  - require it, declare some methods as abstract" (återkommer med abstrakta klasser).
- Design för arv är mycket komplicerat. Några punkter...
  - variabler skall vara private (man kommer åt med set/get).
  - Metoder som kan overrida:s utgör en speciell och känslig del i koden, kommer de att fungera vad än subklasser/superklasser hittar på?
  - låt icke-set/get-metoder i basklassen använda set/get (inte variabeln direkt). Ge möjligheter till modifikation (aldrig visa upp den konkreta representationen).
  - Problem med Objects generella kontrakt. Måste ev. hantera metoder så som, hashCode, equals och clone, återkommer...
  - Även andra "specialla" gränssnitt måste ev. hanteras t.ex. Serializable, återkommer...

3 Implementation av egna ADT:er

Normalt skall detta inte behövas men om så...kan man utnyttja Javas Collection framework. Sker genom att subklassa någon av;

- AbstractCollection, AbstractSet, AbstractList, AbstractSequentialList, AbstractQueue eller AbstractMap (se vidare abstrakta klasser)
- Man skall aldrig subklassa List, Map, Set,...

- Man behöver då bara implementera visa metoder resten sköts av basklassen.

4 Sammanfattning

- Implementationsarv är komplicerat, tveksamt. Designa för arv (kräver stor noggranhet) eller förbjud (final klass).
- Subtyp och subklass är inte samma sak.
- Java tillåter kovariante returtyper. Parametrar är invarianta (det blir overload).
- Arv kan exponera representationer.