Gränssnitt och implementation
- Javas förtrollade djungel

Erika Jansson

6 december 2015

Gränssnitt. Vad är det egentligen för någonting? Min första tanke när jag hör ordet gränssnitt handlar om vad jag som användare av någonting ser för något och vad jag kan göra med det jag ser. Till exempel så kan jag sätta på tv'n genom att trycka på av/på knappen. Vad som egentligen sker rent tekniskt bryr jag mig inte så mycket om utan huvudsaken är att tv'n startar. Ett exempel på något som skulle påverka mig som användare är om ikonerna som visas på datorskärmen var så små att jag knappt kunde nudda dem med datormusen eller om volymknappen satt gömd bakom tv'n.
Allt det här handlar om hur man som designer har valt att implementera sitt gränssnitt. Alla gränssnittsval en designer gör kommer på ett eller annat sätt att påverka användaren av den färdiga produkten samt designern själv. Ovan nämnda exempel passar bättre att kategorisera in som en typ av grafiskt gränssnitt snarare än att de får representera hela begreppet gränssnitt.

Det finns olika typer av gränssnitt, däribland det grafiska, vi skall dock inte fördjupa oss något nämnvärt i detta utan fokuset ligger på hur gränssnitt kan separeras från implementationen i ett java program. Vad är då ett interface egentligen? Ett interface kan liknas vid ett kontrakt. Tar vi exemplet med tv'n igen så kan vi säga att det finns ett kontrakt mellan mig och knappen på tv, kontraktet säger att om jag trycker på knappen så ska tv'n starta, återigen så struntar jag i hur knappen får tv'n att starta bara den gör det.

Gränssnitt i Java

Ett gränssnitt i Java deklarerar vad olika klasser ska innehålla. Med andra ord så upprättas ett kontrakt där gränssnittet lovar att klassen som nyttjar gränssnittet kommer innehålla de metoder som nämns, kontraktet säger dock ingenting om hur metoderna kommer vara implementerade. Ett gränssnitt är en abstrakt typ som endast innehåller metodsignaturer och variabel deklarationer som är både static och final. Ett gränssnitt kan inte instansieras.[1]
Gränssnittet talarom vilka metoder klassen innehåller och hur de är definierade, alltså vad metoderna heter, vad de har för inparametrar, vad de returnerar, samt dess tillgänglighetstyp (static, private osv.). För alla Java versioner lägre än 8 gäller att ett gränssnitt inte innehåller några implementationer av metoder, gör det det så är det inte längre ett gränssnitt. Från och med Java 8 kan default och static metoder ha sin implementation i gränssnittet.
För att implementera ett gränssnitt använder man sig av nyckelordet "interface". En implementation av ett interface kan se ut som följer;

```
interface Lamp {

void litLamp(boolean true);

void TurnLightOff(boolean true);

}
```

För att sedan koppla samman detta interface med klassen där metoderna definieras skriver man klassen på följande vis (nyckelordet är "implements");

```
class klassensNamn implements Lamp {

void litLamp(boolean true) {
// Funktions Implementation
}


void TurnLightOff(boolean true) {
// Funktions Implementation
}
}
```

Genom att skriva klasser på följande sätt har vi skiljt mellan gränssnitt och implementation av klasser. Implementationen av själva klassen ligger i klassfilen medans all information någon annan kan tänkas behöva om klassen ligger i gränssnittet.
Precis som klasser så kan även flera interfaces ärva av ett annat interface, detta görs genom att använda nyckelordet ”extends”.[2] Exempelvis;

```
public interface YellowLamp extends Lamp
{
//MetodDeklarationer
}.
```

Vidare gäller att till skillnad från klasser så kan ett interface ärva från flera andra interfaces, detta skrivs då som ovan men efter ”extends Lamp” läggs ett kommatecken till och den/det andra interfacets namn skrivs ut.

Användning av interfaces i Java

Då kommer vi till den stora frågan, varför ska vi egentligen implementera interfaces i Java? Räcker det inte med att implementera klasserna som behövs för att få den produkt man önskar? Gränssnitt fyller stor funktion av flera olika anledningar, här följer några exempel.

Vi börjar med att tänka oss scenariot där en grupp människor på ett företag arbetar med att implementera en viss del av ett program. Låt oss säga att den här gruppen ska implementera själva ”tänkandet” i en elektronisk robotapa. Parallellt med detta har vi en annan grupp som ska implementera robotapans rörelsemönster, för att de båda grupperna ska kunna arbeta överhuvudtaget måste de veta vad den andra gruppen har för tankar och implementations upplägg. Genom att skapa interfaces kan de olika grupperna förmedla sina implementations tankar till varandra så att de båda grupperna kan jobba parallellt med varandra istället för emot varandra.
Rent konkret kan vi ta som exempel att en grupp implementerar en metod som en annan grupp vill använda sig av, ponera att den första gruppen ännu ej skrivit klart metoden och kanske inte heller vet exakt hur den ska implementeras, då kan gruppen genom att deklarera den här metoden i interfacet iallafall förmedla hur metoden är uppbyggd i form utav in resp retur parametrar samt tillgänglighet och typ av metod. På så sätt kan då gruppen som ska nyttja metoden göra det utan några problem. Detta beror på att gruppen då vet vad funktionen ställer för ”krav” och hur den kan användas.

Vi ska nu tänka oss scenario nummer 2. Vi fortsätter på exemplet med robotapan och antar att gruppen som implementerat robotens tänkande efter mycket slit äntligen är klara med den biten. Roboten kan nu tänka och allting fungerar som det ska, den andra gruppen arbetar vidare på sin bit med att implementera robotens rörelsemönster. Plötsligt inser en person i gruppen att väldigt mycket tid spenderas i en av metoderna som implementerats, helt i onödan. Detta anses vara en sån stor sak att det är värt att optimera.
Eftersom gruppen använt sig utav ett interface är detta inget som helst problem. Så länge metoden fortfarande har samma struktur är det fritt fram för gruppen att ändra runt så mycket de vill i metoden/metoderna så länge det som är lovat i gränssnittet fortfarande uppfylls. Hade inte ett interface funnits här så hade gruppen behövt vara mycket mer restriktiv i sina implementationsändringar då någon eventuellt kunnat nyttja funktionen på ett sådant sätt som inte ”var tänkt”. Om vi då ändrar runt i koden kommer den personens användning av koden krascha vilket i värsta fall kan leda till stora förödelser.

I ett interface behöver vi inte heller deklarera alla klassens metoder och egenskaper utan endast det som vi vill att andra ska ha tillgång till. Ett exempel på detta är en hjälpmetod, alltså en metod som endast skapats för att ”huvudmetoden” ska kunna utför det vi vill. Denna metod har inte någon annan någonting med att göra och vi kan därför hålla den ”hemlig” genom att inte deklarera den i interfacet.

En annan sak som är viktig att tänka på när det gäller interfaces är att när vi väl publicerat dem och låtit andra användare ta del utav dem så kan vi inte lägga till/ta bort saker ur dom hur som helst. Detta då den andra användarnas kod kommer brytas vilket leder till kraschade program. [3]

Avslutning

Vi har nu kommit fram till att gränssnitt ändå är rätt bra trots allt. Vi har även konstaterat att gränssnitt handlar om mer saker än av/på knappen på tv apparaten. Att skriva gränssnitt till triviala program kan jag förstå känns rätt så meningslöst men som vi sett ovan så är det väldigt bra att implementera gränssnitt när programmen börjar bli lite mer komplexa eller då flera personer ska vara inblandade. Det är inte bara bra för att andra ska veta hur dina klasser fungerar utan även för att du själv ska ha ryggen fri och känna dig trygg till att kunna göra ändringar i koden du skrivit.
Nackdelen är dock att du inte kan göra precis vad du vill med din kod längre för så fort du publicerat ditt gränssnitt så har ett kontrakt upprättats som inte bör brytas och du har därmed förlorat ”rätten” att ändra i dina metoddefinitioner. Hur som helst är det en relativt lätt sak att skapa ett gränssnitt och på så sätt skilja det från den fullständiga implementationen av klassen vilket som ovan nämnt underlättar flera situationer.

Referenser

[1] Oracle. ”InterfacesTillgänglig via: `https://docs.oracle.com/javase/tutorial/java/IandI/createinterface.html` (3/12-2015)

[2] Tutorialspoint. ”Java - InterfacesTillgänglig via: `http://www.tutorialspoint.com/java/java_interfaces.htm` (5/12-2015)

[3] ToggleOn. Grundläggande objektorientering i Java: Gränssnitt och abstrakta klasser”21/5-2010 Tillgänglig via: `https://toggleon.wordpress.com/2010/05/21/grundlaggande-objektorientering-i-java-granssnitt-och-abstrakta-klasser/` (5/12-2015)