Java Klasse nedarvninger

Et let lille overblik i hvordan klasse nedarvning virker i java

Skrevet den **07. dec 2011** af **mochners** I kategorien **Programmering / Java** | ⭐⭐⭐⭐☆

Hver gang jeg starter på et nyt sprog, er mine første spørgsmål altid hvordan sproget behandler sine objekter og variabler, da det er meget forskelligt for hvert sprog, derfor satte jeg mig ned og skrev lidt testkode i java, for at få en lille forståelse for hvor dynamisk java er med sin forståelse af data.

Jeg har stillet følgende spørgsmål.

Kan en subclass genkendes som sin superclass.
Kan subclasses af en subclas stadig genkendes som en superclass.
Kan man genkalde data fra en subclass, der gemt som en Superclass variabel.

disse spørgsmål virker lidt specielle, men meget simple at gå til, så jeg ville dele hvad jeg selv fandt ud af, da der sikkert er andre der spørger om det samme.

Så jeg startede med at finde på et let koncept at gå ud fra, så det var let at visualisere.. i denne situation blev det Containers... så jeg kaldte min superclass for Container.

```
public class Container {
    private double volume;
    private String content;
    private String name;
    private double weight;

    public Container(){
        this.setVolume(1.0);
        this.setWeight(1.0);
        this.setContent("Generic elements");
        this.setName("Default");
    }

    public void setName(String name){
        this.name = name;
    }
    public String getName(){
        return this.name;
    }
    public void setContent(String content){
        this.content = content;
    }
    public String getContent(){
        return this.content;
    }
    public void setVolume(double volume){
        this.volume = volume;
```

```
    }
    public double getVolume(){
        return this.volume;
    }
    public void setWeight(double weight){
        this.weight = weight;
    }
    public double getWeight(){
        return this.weight;
    }
    public double getDensity(){
        return this.weight/this.volume;
    }
    public String toString(){
        return "This "+this.getName()+" Container, contains "+this.getContent()+" with a Density of "+
this.getDensity()+"kg/m3";
    }
}
```

Den skulle gerne være meget generisk, da den skal bruges til forskellige typer Containers.. lige fra Flasker til Køleskaber.
Jeg lavede dog kun en subclass ud fra Container, som jeg syntes lige så godt kunne være en Flaske, så min næste class blev bottle.

```
public class Bottle extends Container{
    private String bottlecap;

    public Bottle(){
        super();
        this.setName("Default");
        this.setBottlecap("Default");
        this.setContent("Some wierd liquid");
    }

    public void setBottlecap(String bottlecap){
        this.bottlecap = bottlecap;
    }

    public String getBottlecap(){
        return this.bottlecap;
    }

    public String toString(){
        return "This "+this.getName()+" bottle, with a "+this.getBottlecap()+" bottlecap, contains
"+this.getContent()+" with a Density of "+ this.getDensity()+"kg/m3";
    }
}
```

Derfra har jeg en god base til at lave forskellige typer flasker.. og til forskellige ting, meget god ting..
Bottle er selvfølgelig en subclass af Container, da hele experimentet handler om nedarvninger.

Jeg tænkte jeg lige så godt også kunne lave en subclass af Bottle, for at få en generation mere på. da dette ville give mig et bedre billede af nedarvnings processen, så jeg lavede nu en Soda subclass af Bottle.

```
public class Soda extends Bottle {
    private double sugar;

    public Soda(){
        super();
        this.setSugar(12);
        this.setName("Default Soda");
    }

    public void setSugar(double sugar){
        this.sugar = sugar;
    }
    public double getSugar(){
        return this.sugar;
    }

    public String toString(){
        return "This "+this.getName()+" bottle, with a "+this.getBottlecap()+" bottlecap, contains
"+this.getContent()+" it containt "+this.getSugar()+"% of sugar with a Density of "+
this.getDensity()+"kg/m3";
    }
}
```

Jeg har nu en nogenlunde Flexibel base (Container) der kan forme sig ud til det jeg har brug for, som eksemplet en flaske, der har brug for en flaske Kapsel, og en underliggende Sodavand, der har brug for sukkerindhold.

nu mangler jeg bare at teste hvordan de reagere under mine forhold jeg nævnte i toppen.. dette kan gøres på mange måder, men jeg valgte at gøre det hurtigt, da dette kun var en lille test.

```
public class ClassTest {

    /**
    * @param args
    */
    public static void main(String[] args) {
        ClassTest ct = new ClassTest();
        ct.run();
    }

    public int run(){

        Container c = new Container();
        Bottle b = new Bottle();
        Soda s = new Soda();
        Container b2 = new Bottle();
        this.PrintContainer(c);
        this.PrintContainer(b);
```

```
        this.PrintContainer(s);
        this.PrintContainer(b2);
        return 0;
    }

    public void PrintContainer(Container container){
        if(container.getClass() == Bottle.class){
            System.out.println("This is a Bottle");
        }
        else if(container.getClass() == Soda.class){
            System.out.println("This is a Soda");
        }
        else if(container.getClass() == Container.class){
            System.out.println("This is a Container");
        }
        else
        {
            System.out.println("I don't know what this is");
        }
        System.out.println(container.toString());
    }
}
```

Det burde være ligetil at se hvad der sker, jeg har en methode som kun modtager "Container" objekter, dette kræver at Java kan tjekke tilbage i de classes som er på et objekt. da de ikke alle er Containers i run-mode.
Så er der en logik der tjekker hvad type class objektet er i run-mode.. dvs dens Class den er blevet oprettet som.
Jeg prøvede at snyde den ved at lave en Bottle i en Container variabel.. men den genkender stadig Bottle som en Bottle.. jeg har dog ikke adgang til Bottles Methoder, da jeg har sat den som Container, men man kan så videre kaste den til en Bottle, også får du adgang til methoderne, Container.
variablen b2 udskriver stadig sine specielle variabler korrekt, da den har kørt Bottle Konstruktoren i starten, og ikke kun Container konstukrtoren.

Jeg håber denne Journal/guide var til hjælp for nogen derude, da jeg selv følte den gav mig en bedre sikkerhed i hvordan Java reagere til sine nedarvninger :)

Kommentar af arne_v d. 07. dec 2011 | 1

Det er jo ganske rigtigt.

Der er dog nogle ting som jeg synes med fordel kunne tilfoejes.

Container burde nok vaere en abstrakt klasse. Man kan ikke have noget som kun er en container uden at vaere en bestemt form for container. Det er ret sjaeldent at ikke abstrakte super klasser er godt design.

Det burde nok understreges at:

o.getClass()==X.class

er et meget tvivlsomt test som aldrig boer bruges i rigtig kode.

Som oftest er den rigtige loesning at have an abstrakt metode i super klassen som man saa bare kan

kalde.

Til denne lille demo kunne du evt. bare have udskrevet o.getClass().getName().

Maaske var det ogsaa relevant at komme ind paa at ikke statiske metoder i Java er virtual og at toString faktisk er en override.

Og naar du var igang med == paa class var instanceof operatoren maaske vaerd at naevne.

Kommentar af mochners d. 08. dec 2011 | 2

mange tak for dine tilføjelser arne_v, jeg havde helt skudt de ting ud af hovedet, men jeg gir dig helt ret i at mine logiske teste ikke er helt sikre, men hovedsageligt kun for at teste genkendelse af klasserne.

af hensyn til den Abstrakte classes, så undlod jeg den, for at teste javas reaktion til at læse Første generation, ud til 3. generation klasser, men du har dog helt ret.

mange take igen for du gad læse den, og tak for dit feedback.

Kommentar af superanden d. 16. jun 2013 | 3

Jeg ville nok også knytte en forklaring til brugen af super() i dine childs. Det ligger sig jo direkte op af det du vil vise. Det var i hvertfald ikke så selvdokumenterende for mig som resten.