Samlingar, Gränssitt och Programkonstruktion

Förelasning 11

TDA540 – Objektorienterad Programmering

CHALMERS

Samlingar

Vi kommer att behöva hantera samlingar av objekt

- Har oftast använd Array (fält)
- Bra om fix storlek och fixa index, exempel: Holder (Scrabble)
- Inte så bra om vi vill att samlingen skall kunna minska eller växa under körning,
- Exempel TileBag (Scrabble) skall ha en samling av Tiles. Samlingen skall krympa då man tar brickor. När samling tom så är påsen tom = spelet slut

Holder

TileBag

The Collections Framework

Java har en stor mängd färdiga klasser för samlingar*)

- Benämns The Collections Framework (JCF)
- Samlingarna växer och krymper efter behov
- Samlingarna kan lagra objekt av vilken referenstyp som helst, de är **generiska.** Vilken typ som skall lagras anges vid deklaration
- Några exempel på samlingar är ...
- **… Mängd** (HashSet, TreeSet), samma idé som en matematisk mängd, inga dubbletter, ingen ordning
- **… Lista** (ArrayList, LinkedList), … en ordnad sekvens, första, n:te, sista, före, efter, … liknar array
- **… Tabell** (HashMap), givet en nyckel kan man slå upp ett värde … (namn/telefonnummer, svenskt ord/engelsk översättning)
- För att använda samlingar måste vi ange import java.util.* i våra klassfiler (= importera alla klasser ur **paketet** java.util)

*) Alla (moderna) språk har liknande

Listklasser

Vill ha en krympande samling i TileBag

- HashSet, TreeSet tveksamt, vi har dubbletter, eller… ? Har vi … (mer senare)?
- HashMap behövs inte, vi har inget att slå upp
- ArrayList eller LinkedList tänkbara … men vilken?
- ArrayList långsam att lägga till/ta bort, snabb att läsa av!
- LinkedList snabb att lägga till/ta bort, långsam att läsa av!

Anta att vi inte vet just nu, vad göra … ?

Implementation

ArrayList och LinkedList är två olika sätt att bygga en lista

- Två olika **implementationer**
- Men metoderna är samma för båda…?
- Om vi bara är intresserade av att metoderna finns, inte hur de är implementerade … kan vi ange detta på något sätt??
- … JA! … (next slide)!

Gränssnittet List

Om vi inte vet (eller spelar ingen roll) vilken typ av lista vi vill ha kan vi ange enbart List

- List är en specifikation (abstraktion) av en lista (vilken typ som helst)
- Specifikationen ges av ett **gränssnitt**
- List är ett gränssnitt
- Ett gränssnitt i Java påminner om en klass men deklareras med **interface** i stället för class
- Gränsnittsdeklarationen innehåller bara metodhuvuden
- Både ArrayList och LinkedList implementerar gränssnittet List, d.v.s de har metoderna som anges i List
- Att en klass implementerar ett gränssnitt anges i koden med **implements** (vid klasshuvudet efter namnet)

Gränsnitt och Typ

Ett gränssnitt introducerar en ny typ!

- List och ArrayList/LinkedList är typkompatibla med List, de har samma metoder. Vi kan tilldela ett ArrayList objekt till en List variabel!
- Genom att deklarera alla parametrar och returtyper som interface-typer kan vi bortse från vilken lista vi använder, kan byta vid behov!
- Någonstans skapas ett ArrayList eller LinkedList objekt men det skickas alltid runt som typen List
- Exampel: TileBag, CommandLineScrabble.getTiles()

Instansiering och initiering

Nya listobjekt skapas med new (som vanligt)

- Inget enkelt sätt att initiera (sätta värden)

```
// li is of type list (compatible with ArrayList)
List<Integer> li = new ArrayList<>();   // or LinkedList

// Note: li emtpty from start, no elements and *no*
// indexes (there are no positions)
// Normally have to add "element by element" to initialize
li.add(2);
```

Grundläggande List Metoder

Lägga till/ta bort element m.m.

```
List<Integer> li = ArrayList<>();
li.add(4);        // Put last
li.set(0, 5)      // Overwrite index 0
li.remove(2)      // Remove index (or object)
li.get(0);        // Get index 0
li.size();        // Number of elements
li.isEmpty();
li.clear();
li.indexOf( … )   // Index for object (-1) if not found
```

Hur fungerar i List?

List objekt är icke-triviala. Vad händer om vi ...

- anger index utanför listan?
- tar bort ett element i mitten (blir det ett hål)?
- lägger till en dubblett? Skrivs gamla värdet över?
- har dubbletter, vilket värde returneras?
- lägger till null?
- anropar contains(...), vad jämför för att hitta elementet?

(kan inte komma ihåg allt, får gå till dokumentationen,
tyvärr inte säkert att svaret finns där, om ej testa .. .)

Paus

15 min

Hur skapa ett Program?

Hittills har vi tittat på en hel del tekniska aspekter på programmering, men hur skapar vi ett program?

- Finns inget recept! En fri, kreativ, skapande verksamhet!
- I det följande visas en tänkbar väg
- Vi vill så snabbt som möjligt ha “något” körbart (fördjupa vår förståelse)

Fakta

- Vi behöver en objektmodell
- Vi behöver något sätt att kommunicera med modellen, användaren skicka indata, skriva utdata från modell och/eller rita upp (delar av) modellen (**rendera**)
- Vi måste ha en main-metod (krav i Java)

Omgivning till Modellen

Modellens omgivning sköter kommunikation, instansiering och rendering (utritning)

- Vi samlar hela omgivningen i en enda klass (namn: CommandLineNNN)
- I klassen finns en kommandoradsmeny i metoden run()
- I klassen finns main-metoden, i denna skapas kommandoradsobjektet och på detta anropas run() (d.v.s. kommandoraden startar)
- Rendering av modellen sköta av en metod render(), ev. i kombination med toString() i resp. klasser (framför allt under utveckling)
- I samma klass lägger vi en metod för att instansiera modellen (namn: buildNNN). Metoden returnerar hela objektmodellen, alla kopplingar via konstruktor görs i metoden (hjälpmetoder kan behövas)
- Exempel: CommandLineScrabble, Pig

Top down och Bottom up

Det finns (minst) två olika angreppssätt för att bryta ner ett problem och kombinera ihop dellösningar

- Top-down, börja övergripande, från helhet till delar
- Bottom-up, börja med de minsta delarna (atomerna), kombinera dessa till större delar, o.s.v.
- Vi behöver båda …
- Modellen tas fram bottom-up (objekten är våra atomer)
- Metoder tas fram top-down (om de inte helt uppenbart hör ihop med något objekt)

Objektmodell Bottom-up

Hitta objekt borde vara lätt… *)

- Det är därför vi använder OO-programmering!
- Datan i objekten borde också hittas ganska lätt

Hur hitta metoder, i vilka klasser skall de ligga?

- Vissa bör vara givna utifrån problemställning och funna objekt, övriga avvaktar vi med (se metoder top-down senare)

*) Vanligast är att man missar objekt, ofta mer abstrakta, objekt, t.ex. “spelrunda”

Koppling av Modell

Hur koppla ihop objekten?

- Vissa kopplingar borde ges av modellen (tyvärr finns ofta flera möjligheter)
- Vilka objekt behöver andra? Försök koppla så!
- Vi har alltid ett objekt som representerar “hela problemet”, ett **top-objekt**. Top-objektet kopplar ihop modellen så att den blir sammanhängande
- Top-objektet har kopplingar till många andra modellobjekt, övriga objekt har ingen koppling till top-objektet.
- Skapa konstruktorer som kopplar ihop
- Exempel: Scrabble, Pig

Metoder Top-Down, Steg 1

Här gäller abstraktion.

- Utgå från kommandoraden och top-objektet (t.ex. Scrabble)
- Alla kommandon användaren skriver in skall resultera i direkta anrop på top-objektet
- Vilka metoder skulle vi vilja ha? Tänk på vad du vill ha, inte hur det skall kodas!
- Vad händer i verkligheten? Ibland hjälper att tänka indata-> utdata, vad har jag? Vad vill jag ha?!
- Skriv “tomma” metoder med parametrar och returtyper i klassen för top-objektet (sätt return 0 eller return null så länge, så att det går att kompilera)
- Exempel: Scrabble

Metoder Top-Down, Steg 2

Vi har nu flera tomma metoder i klassen för top-objektet

1. Börja med en metod och gör den helt klar (innan andra påbörjas)
2. Inspektera de objekt som är kopplade till klassen, finns några metoder att använda från dessa (saknas objekt eller verkar det vara fel objekt, se över kopplingarna)?
3. Om ej, skapar vi metoder som naturlig hör hemma i de olika objekten (har den data som behövs för att utföra det vi vill). Går inte detta skapa en ny tom metod i någon klass och gå till 2 eller skapa en privat hjälpmetod i top-level klassen
4. Fortsätt tills du kan kombinera ihop metoder till att utföra det som den ursprungliga metoden i top-level objektet skulle göra
5. Om metoden för stor (många rader) dela upp i flera metoder
6. Kontrollera att varje metod gör “en sak”, bättre att varje gör en sak och sedan kombinera ihop vid behov

Gris

Då är vi klara att skapa ett helt program!

- Vi har det tekniska
- Vi har ett sätt att arbeta
- Det blir inte rätt på en gång, så detta blir v. 0.1 …

Vi skall skapa ett program för spelet Gris!