ArrayList

Klasser för samlingar av objekt

I många applikationer behöver man lagra samlingar av objekt. Om man använder sig av arrayer måste man själv skriva kod för att hålla reda på hur många objekt som är lagrade i arrayen, öka storleken när arrayen är full, packa arrayen när objekt tas bort m m.
Eftersom detta är ett återkommande problem finns naturligtvis en uppsättning klasser som löser dessa problem åt oss. Vi ska här studera en av dessa klasser, java.util.ArrayList[1].

java.util.ArrayList

Ett objekt av klassen ArrayList innehåller internt en privat array för att lagra objekten. När man skapar ett ArrayList-objekt anger man med en s k typparameter vilken typ av array man vill ha, d v s vilken typ av objekt man vill lagra.
För att skapa en lista för att lagra Stringobjekt och lägga till objekt skriver man t ex

```
ArrayList<String> band = new ArrayList<String>();
listan.add(new String(“Anders”));
listan.add(new String(“Arina”));
```

…
När den interna arrayen är full utökas listens storlek automatiskt. Vill man komma åt ett objekt med ett visst index skriver använder man metoden get:

```
String s = listan.get(1); // I detta fall ”Arina”
```

Listan storlek får man med metoden size:

```
int antal = listan.size();
```

Loopa igenom ett listobjekt

Vill man loopa igenom alla objekt i listan, t ex för att skriva ut dessa, kan man naturligtvis skriva

```
for(int i = 0; i < listan.size(); i++) {
  System.out.println(listan.get(i).toString();
}
```

Det finns dock ett enklare sätt att loopa igenom objekt av samlingsklasser som ArrayList.

```
for(String s : listan) {
  System.out.println(s.toSring());
}
```

Referensen s, som måste vara av samma typ som objekten i listan, kommer i tur och ordning att referera till varje objekt i listan. Loopen kan när som helst avbrytas med `break`.

[1] Det finns en hel uppsättning klasser för att lagra objekt på olika sätt i det som kallas Java Collections Framework.

Några metoder i klassen ArrayList

Typ står för den typ man vill lagra i listan.

- **ArrayList**<*Typ*>() – konstruktor som skapar en lista initialt med plats för 10 objekt
- **add**(*Typ* obj) – lägger till ett objekt sist i listan
- int **size**() – returnerar antalet objekt i samlingen
- *Typ* **get**(int index) – returnerar objektet i angiven position
- Typ **remove**(int index) – tar bort objektet på angivet index. Listan packas automatiskt.
- boolean **remove**(*Typ* obj) – tar bort objektet
- **clear**() – tömmer listan
- *Typ*[] **toArray**() – returnerar en array med referenser till objekten i listan

Hjälpklassen Collections

I klassen java.util.Collections finns ett antal static-deklarerade hjälpmetoder för samlingsklasser. Bl a finns en metod för att sortera objekten i listan. För att sorteringen ska fungera måste de lagrade objektens typ implementera interfacet Comparable, se föreläsningen om interface.
Klassen String implementerar interfacet Comparable så vi kan för listan ovan skriva

```
Collections.sort(listan);
```

Vips så är objekten sorterade i alfabetisk ordning.

Kodexempel

- TestaArrayList visar hur man skapar en lista, lägger till objekt och loopar igenom dessa.
- KontoLista är ett mer avancerat exempel som läses av den intresserade. Det visar hur man kan skriva den tidigare klassen KontoLista, för att lagra konton, med hjälp av ett ArrayListobjekt istället för en array. Många av metoderna blir enklare att skriva, jämför med den tidigare varianten på klassen (från ”Relationer mellan klasser”).