Stockholms universitet

Programmering 2

Föreläsning 5: Söka och sortera

Isak Samsten, VT22

Stockholms universitet

Exempelklass

```
class Persnr {
    private int fdat, nr;
    private Persnr(int fdat, int nr){
        this.fdat = fdat;
        this.nr = nr;
    }
    public int getÅr(){ return fdat / 10000; }
    public int getMånad(){ return fdat / 100 % 100; }
    public int getDag(){ return fdat % 100; }
    public int getNr(){ return nr; }

    public String toString(){
        return fdat + "-" + nr;
    }

    public static Persnr parsePersnr(String str){
        int pos = str.indexOf("-");
        int fd = Integer.parseInt(str.substring(0,pos));
        int nr = Integer.parseInt(str.substring(pos+1));
        return new Persnr(fd, nr);
    }
}
```

Stockholms universitet

Exempelklass

```
class Person {
    private Persnr pnr;
    private String namn;
    private int vikt;

    public Person(Persnr pnr, String namn, int vikt) {
        this.pnr = pnr;
        this.namn = namn;
        this.vikt = vikt;
    }
    public Persnr getPnr() {return pnr; }
    public String getNamn() { return namn; }
    public int getVikt() { return vikt; }
    public void äter(int kg) { vikt += kg; }
    public String toString() {
        return pnr + " " + namn + " " + vikt;
    }
}
```

Sekventiell sökning

Objektsök

```
public static Person forName(List<Person> lista, String namn) {
    for (Person p : lista) {
        if (p.getNamn().equals(namn)) {
            return p;
        }
    }
    return null;
}
```

```
Person p = Search.forName(lista, "Eva");
if (p != null) {
    System.out.println("hittad!");
}
```

Indexsök

```
public static int indexOfName(List<Person> lista, String namn) {
    for (int i = 0; i < lista.size(); i++) {
        if (lista.get(i).getNamn().equals(namn)) {
            return i;
        }
    }
    return -1;
}
```

```
int index = Search.indexOfName(lista, "Petter");
if (index >= 0) {
    System.out.println("hittade " + lista.get(index));
}
```

Stockholms universitet

Generalisera sökning

- Kan man göra sådan sökning för godtyckliga objekt?
  - Inte utan vidare: hur vet man vad som identifierar objekt?
- Man kan dock söka efter ”hela” objekt

Stockholms universitet

Sökmetoder

- `Collection`-klasser har en sökmetod:
  - `boolean contains(Object object)`
- `List`-klasser har dessutom:
  - `int indexOf(T t)`
  - `int lastIndexOf(T t)`

Stockholms universitet

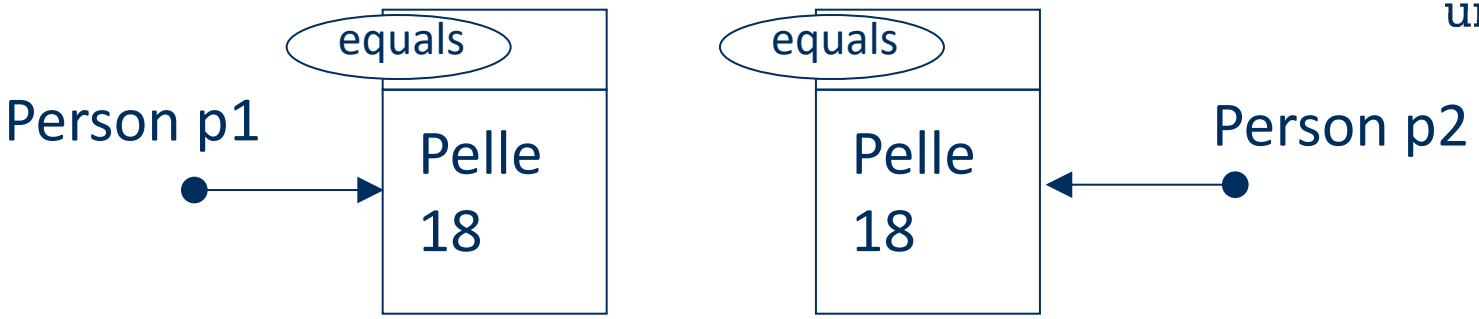
Sökning

```
ArrayList<Persnr> l = new ArrayList<>();
Persnr p = Persnr.parsePersnr("19870410-0101");
l.add(p);
l.contains(p); // return true
l.indexOf(p); // return 0

l.contains(Persnr.parsePersnr("19870410-0101")); // return false
```

- Dessa metoder använder:
  - `Object#equals(Object other)`-metoden
  - Däremot funkar det för strängar…

Vad betyder ”lika”?




```
p1 == p2 ger false
p1.equals(p2) ger false
```

- Strängar och andra klasser vilkas objekt ska bete sig som värden:
  - vi vill att två objekt ska betraktas som lika om de innehåller lika värden.
- Ordnas med överskuggning av `equals`.


```
s1 == s2 ger false
s1.equals(s2) ger true
```

Stockholms universitet

Identitet och likhet

- Vi testar om två objekt har samma identitet dvs om två referenser pekar på **samma** objekt med `a == b`
  - Det är endast sant om a och b pekar på samma objekt
- Vi testar om två objekt är lika, dvs om de har samma värde, med `a.equals(b)`
  - Det är sant om a och b är lika enligt equals-metoden

```
class Object{
    public boolean equals(Object other){
        return this == other;
    }
}
```

Stockholms universitet

Object#equals(Object)

- Object-klassen ger endast sant när det är samma objekt – dvs samma referens
- Vill man att två objekt som innehåller lika värden ska betraktas som lika så måste man överskugga `equals` i den klassen.
- Då SKALL man även överskugga metoden `hashCode()`.

Personnummer-klassen

Stockholms universitet

```
class Persnr {
    ...
    @Override
    public boolean equals(Object o) {
        if (this == o){
            return true;
        }
        if (o instanceof Persnr) {
            Persnr persnr = (Persnr) o;
            return fdat == persnr.fdat &&
                    nr == persnr.nr;
        } else {
            return false;
        }
    }

    @Override
    public int hashCode() {
        return fdat * 10000 + nr;
    }

    public String toString(){
        return fdat + "-" + nr;
    }
}
```

Stockholms universitet

Kontrakt

- hashCode måste konsekvent returnera samma hashCode givet att ingen information som används för equals ändras
- Två objekt som är lika enligt equals måste ha samma hashCode
- Det krävs inte att två objekt som är olika enligt equals har olika hashCode

Stockholms universitet

Problematiskt med muterbara fält

- Om vi kan ändra fält som är del i en hashCode så kommer det ju hamna i olika buckets!

Stockholms universitet

Hur designar man en bra hashCode?

- Maximal prestanda:

```
public int hashCode(){
    return 0;
}
```

- Bra?
- Snabb! Två objekt med samma equals har samma hashCode!
- Men kolliderar alltid!

Stockholms universitet

Hur designar man en bra hashCode?

- För varje fält `f` som används i equals, beräkna `int c=0`
  - Om boolean: `c += (f ? 0 : 1)`
  - Om byte, char, short, int: `c += (int) f`
  - Om long: `c += (int)(f ^ (f >>> 32))`
  - Om double: `c += Double.doubleToLongBits(f)` och konvertera som long
  - Om object: `c += f == null ? 0 : f.hashCode()`
- `return 37 * result + c`

Stockholms universitet

Bra distribution av hash-värden

- Använd `Objects.hash(Object… args)`
- Ungefär samma algoritm som förgående slide

Stockholms universitet

Sortering

- Sorteras med: `Collections.sort(lista);`
- Från och med Java 8, kan man skriva `lista.sort(null);` (null förklaras senare)
- Sortering av array med `Arrays.sort(array);`
- Förutsätter att objekten i samlingen (eller arrayen) implementerar interfacet `Comparable<E>`

```
class Persnr implements Comparable<Persnr>{
    private int fdat, nr;

    public Persnr(int fdat, int nr){
        this.fdat = fdat;
        this.nr = nr;
    }

    public int compareTo(Persnr other){
        if (fdat < other.fdat)
            return -1;
        else if (fdat > other.fdat)
            return 1;
        else if (nr < other.nr)
            return -1;
        else if (nr > other.nr)
            return 1;
        else
            return 0;
    }
}
```

Stockholms universitet

Stockholms universitet

Sortering av icke-Comparable

- Hur gör man om man vill sortera klasser som inte implementerar Comparable?
- Tillhandahålla en extern sorteringsalgoritm
  - Frikoppla implementationen av sortering från klassen – separation of concern

Stockholms universitet

Kodexempel

2020-03-23 /Isak Samsten

Stockholms universitet

Två sätt att ange hur jämförelse ska ske

- Klassen implementerar Comparable<T> och har metoden int

  compareTo(T other);
  - Returnera negativt tal om egna objektet är mindre än other
  - Returnera noll om båda objekten är lika
  - Returnera positivt tal om egna objektet är större än other
- Extern sorterare som implementerar Comparator<T> och int

  compare(T a, T b)
  - Returnerar negativt tal om a < b
  - Returnerar noll om a == b
  - Returnerar positivt tal om a > b

Stockholms universitet

Kodexempel

- Externa komparatorer
- Lambda-uttryck

Stockholms universitet

Kodexempel

- Sortera strängar efter längd

Stockholms universitet

Binärsökning

- Om listan är sorterad kan man söka mer effektivt!
- Hitta mitten (längd // 2)
- Om vi hittar det vi söker – avbryt
- Annars
  - Mindre? Gör samma sak i den mindre hälften
  - Större? Gör samma sak i den större hälften
  - Om det inte finns några större eller mindre? Avbryt utan att objektet finns

Stockholms universitet

Binärsökning i Java

- Använd Collections.binarySearch(lista)
- Måste såklart vara sorterad (med Collections.sort)