Stackar, köer, iteratorer och paket

Programmering för språkteknologer 2

Sara Stymne

2013-09-18

Idag

- Paket
- Stackar och köer
  - Array resp länkad struktur
- Iteratorer
- Javadoc

Kommentarer lab 1

- Bra att de flesta lämnade in i tid, och kompletterade snabbt!
- Vanliga svårighet
  - Dela upp i klasser och metoder
  - Tumregel
    - Varje klass ska ha en tydlig uppgift/ansvar, det ska gå lätt att förklara vad klassen gör
    - Varje metod ska ha en tydlig uppgift
    - Det ska gå att ge namn som relativt tydligt berättar vad en klass/metod gör
    - Övning behövs!
- Andra observationer
  - Använd inkapsling (privata instansvariabler) om ni inte har mycket goda själ att låta bli!
  - Undvik att upprepa liknande kod och köra många loopar över samma data om det ej är nödvändigt
    - I statistkuppgiften räckte det egentligen med en loop över alla ord
  - Det såg bra ut med era reguljära uttryck och automater!

Paket

- En samling klasser som har något gemensamt
  - java.util
  - java.util.regex
  - javax.io
  - javax.swing
  - org.xml.sax
  - ...
- Kan importeras och användas av andra javaprogram

Paket som filstruktur

- Paketnamnen bildar en filstruktur
- java
  - util
    - regex

Egna paket

- Skapa en egen filstruktur!
  - Byt ut alla "." mot "/"
- Placera in klasserna där
- Deklarera att klasserna är medlemmar i paketet
  - `package packageName;`
- När du använder en klass i paketet utanför paketet:
  - `import packageName.className;`
  - `import packageName.*`

Namngivning av paket

- Använd unika namn
- Använd namn som beskriver innehållet
- Se till att namnen matchar filstrukturen

Paket – kompilering

- Från terminalen
  - Gå till paketkatalogen
  - Kompilera filerna där
- Använd verktyg
  - ant – verktyg för att kompilera och skapa jar-filer
  - IDE – integrated development environment
    - Eclipse
    - NetBeans
    - ...

Fördelar med paket

- Saker som hör ihop klumpas ihop
- Namnkonflikter kan potentiellt undvikas
- Lättare att flytta och distribuera

Paket

- Vad händer om man inte själv skapar paket?
  - Klasser som ligger i samma katalog hamnar i ett ”default”-paket utan namn
- God programmeringsstil
  - Använd alltid namngivna paket om du kodar lite större paket

Jar-filer

- Hur gör man om man ska dela med sig av ett Javaprogram?
- Skapa en jar-fil
- Javaprogram kan köras direkt från jar-filer
  - `java -jar filnamn.jar`

Abstrakta datatyper

- Abstrakt datatyp (ADT)
- Ett teoretiskt begrepp
- Definieras som
  - En typ
  - En lista med operationer som ska kunna utföras
- Den abstrakta typen säger ingenting om hur typen ska implementeras, bara om vad man ska kunna göra med den

Abstrakta datatyper – stack

- En listliknande struktur, men där element enbart kan sättas in och tas bort i en ända
- ”LIFO” – last in, first out
- Element tas bort i omvänd ordning från vad de kom in
- ”Stapel” snarare än stack på svenska

Stack – grundläggande metoder

- void push(E element)
- E pop()
- E peek()

Stack – grundläggande metoder

- void push(E element)
- E pop()
- E peek()
- boolean isEmpty()
- int size()
- void clear()

Stack – implementationer

- Hur en stack implementeras är ej specificerat
- Två vanliga implementationer:
  - Med hjälp av en array
  - Som en länkad struktur

Stack – implementationer

- Hur en stack implementeras är ej specificerat
- Två vanliga implementationer:
  - Med hjälp av en array
  - Som en länkad struktur

RITA BILDER!!

Stack – implementationer

- Hur en stack implementeras är ej specificerat
- Två vanliga implementationer:
  - Med hjälp av en array
  - Som en länkad struktur

RITA BILDER!!
VISA KODEXEMPEL

Komplexitet för olika stackimplementationer

	Array	Länkad lista
pop	$O(1)$	$O(1)$
push	$O(1)$	$O(1)$
peek	$O(1)$	$O(1)$

Komplexitet för olika stackimplementationer

	Array	Länkad lista
pop	$O(1)$	$O(1)$
push	$O(1)$	$O(1)$
peek	$O(1)$	$O(1)$

- Eftersom stack är så specialiserad går den att implementera effektivt!

Stack med hjälp av LinkedList

```
public class <T> myArrayStack {

   private LinkedList<T> list;

   ...

   void push(T element) {
      list.addFirst(element);
   }

   E pop() {
      return list.removeFirst();
   }
}
```

Stack med hjälp av LinkedList

```
public class <T> myArrayStack {

   private LinkedList<T> list;

   ...

   void push(T element) {
      list.addFirst(element);
   }

   E pop() {
      return list.removeFirst();
   }
}
```

- Bra lösning?

Stack med hjälp av LinkedList

```
public class <T> myArrayStack {

   private LinkedList<T> list;

   ...

   void push(T element) {
      list.addFirst(element);
   }

   E pop() {
      return list.removeFirst();
   }
}
```

- Bra lösning?
- Helt OK, men något ineffektiv, eftersom LinkedList tillåter fler operationer än stack

Stack med hjälp av LinkedList 2

```
public class <T> myArrayStack extends LinkedList<T> {

    ...

    void push(T element) {
        addFirst(element);
    }
    E pop() {
        removeFirst();
    }
}
```

Stack med hjälp av LinkedList 2

```
public class <T> myArrayStack extends LinkedList<T> {

    ...

    void push(T element) {
        addFirst(element);
    }
    E pop() {
        removeFirst();
    }
}
```

- Bra lösning?

Stack med hjälp av LinkedList 2

```
public class <T> myArrayStack extends LinkedList<T> {

    ...

    void push(T element) {
        addFirst(element);
    }
    E pop() {
        removeFirst();
    }
}
```

- Bra lösning?
- Ger stackoperationerna, precis som förra lösningen, men även alla andra LinkedList-operationer

Abstrakta datatyper – kö

- Queue
- En listliknande struktur, men där element sätts in i en ända, och tas ut i andra änden
- ”FIFO” – first in, first out
- Element tas bort i samma ordning som de sätts in
- Som en kö i en affär

Kö – grundläggande metoder

- void enqueue(E element)
- E dequeue()

Kö – grundläggande metoder

- void enqueue(E element)
- E dequeue()
- boolean isEmpty()
- int size()
- void clear()

Kö – implementationer

- Hur en kö implementeras är ej specificerat
- Två vanliga implementationer:
  - Med hjälp av en array
  - Som en länkad struktur

Kö – implementationer

- Hur en kö implementeras är ej specificerat
- Två vanliga implementationer:
  - Med hjälp av en array
  - Som en länkad struktur

RITA BILDER!!

Komplexitet för olika köimplementationer

	Array	Länkad lista
enqueue	$O(1)$	$O(1)$
dequeue	$O(1)$	$O(1)$

Komplexitet för olika köimplementationer

	Array	Länkad lista
enqueue	$O(1)$	$O(1)$
dequeue	$O(1)$	$O(1)$

- Eftersom kö är så specialiserad går den att implementera effektivt!

Gå igenom en lista/stack/kö

- Hur gör man för att gå igenom alla element i en lista/stack/kö?

Gå igenom en lista/stack/kö

- Hur gör man för att gå igenom alla element i en lista/stack/kö?
- Vanligtvis en loop:

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

Gå igenom en lista/stack/kö

- Hur gör man för att gå igenom alla element i en lista/stack/kö?
- Vanligtvis en loop:

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

- Implicit används en **iterator** i loopen:

```
for(Iterator<String> it = names.iterator();
    it.hasNext();) {
  String s = it.next();
  System.out.println(s);
}
```

Gå igenom en lista/stack/kö

- Hur gör man för att gå igenom alla element i en lista/stack/kö?
- Vanligtvis en loop:

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

- Implicit används en **iterator** i loopen:

```
for(Iterator<String> it = names.iterator();
    it.hasNext();) {
  String s = it.next();
  System.out.println(s);
}
```

- en iterator är ett objekt som används för att stega igenom en samling

Listklass med iterator

- MyList<T> måste implementera Iterable<T>
- MyList<T> måste tillhandahålla metoden `Iterator<T> iterator()`
- Vi behöver en iteratorklass, som implementerar Iterator<T>

Gränssnittet Iterable<T>

- Iterator<T> iterator()
  Returns an iterator over a set of elements of type T
- Implementeras av ”samlings”-klassen

Gränssnittet Iterator<T>

- boolean hasNext()
  Returns true if the iteration has more elements
- E next()
  Returns the next element in the iteration
- void remove() Removes from the underlying collection the last element returned by this iterator (optional operation)

Gränssnittet Iterator<T>

- boolean hasNext()
  Returns true if the iteration has more elements
- E next()
  Returns the next element in the iteration
- void remove() Removes from the underlying collection the last element returned by this iterator (optional operation)
- Implementeras av iteratorklassen

Gränssnittet Iterator<T>

- boolean hasNext()
  Returns true if the iteration has more elements
- E next()
  Returns the next element in the iteration
- void remove() Removes from the underlying collection the last element returned by this iterator (optional operation)
- Implementeras av iteratorklassen
- Det finns även ett gränssnitt ListIterator, som innehåller fler metoder för listor

Iterator för arraybaserad stack

- Vad behöver vi känna till?
  - Vilket index vi är på
  - Vilka värden som finns på vilken plats i stacken
  - Hur många värden som finns i stacken

Iterator för arraybaserad stack

- Vad behöver vi känna till?
  - Vilket index vi är på
  - Vilka värden som finns på vilken plats i stacken
  - Hur många värden som finns i stacken
  - Det vill säga: vi behöver helt enkelt känna till stacken – Gör en intern klass!

Iterator för arraybaserad stack

- Vad behöver vi känna till?
  - Vilket index vi är på
  - Vilka värden som finns på vilken plats i stacken
  - Hur många värden som finns i stacken
  - Det vill säga: vi behöver helt enkelt känna till stacken – Gör en intern klass!
- VISA KODEXEMPEL!

Iterator för länkad stack

- Vad behöver vi känna till?
  - Vilken nod vi är på
  - `next` samt `value` för noder

Iterator för länkad stack

- Vad behöver vi känna till?
  - Vilken nod vi är på
  - `next` samt `value` för noder
  - Det vill säga vi måste känna till stackklassens struktur, men annars räcker det med att känna till första noden

Iteratorer – fördelar

- Iteration fungerar likadant för alla samlingsklasser
  - ArrayList, LinkedList, TreeSet, HashSet, ArrayDeque, TreeMap, HashMap, LinkedHashMap, ...
- Foreach-loopen kan användas, vilken implicit använder iterator

Vad är javadoc?

- javadoc är ett verktyg för att skapa javadokumentation
- Javas API är skapat med hjälp av javadoc
- Man kan skapa egen dokumentation med hjälp av javadoc
  - Kräver en särskild typ av kommentarer
  - Så kallade javadoc-kommentarer

Javadoc-kommentarer – exempel

```
/**
 * dumb example class
 * @author Sara Stymne
 */
public class MyClass {

  /**
   * This subroutine gives the max number if two integers,
   * if they are bigger than 0
   * @param i1 the first integer
   * @param i2 the second integer
   * @return the largest of the two integers, larger than 0
   * @throws MyException if neither integer is larger than 0
   */
   public int method maxPositive(int i1, int i2) {
      ...
   }

}
```

Hur skriver man javadoc-kommentarer?

- Kan finnas för
  - Klasser
  - Metoder
  - Variabler (instans- och klass-)
- Skrivs inom kommentarsblock som startar med: /**
- Kan innehålla
  - Text, som kan vara html-formatterad
  - Taggar, startar med @, har speciell betydelse

Vanliga taggar

- Klass
  - @author – författaren till koden
  - @version – kodversion
- Metod
  - @param – parameter
  - @return – returvärde
  - @throws – exception som kan kastas
  - @deprecated – utdaterad metod
- Klass och metod
  - @see referens till annan symbol

Hur skapar man ett eget Java API?

- Kör kommandot `javadoc` för din kod!
- Eclipse har också kommandon för att köra javadoc

Hur skapar man ett eget Java API?

- Kör kommandot `javadoc` för din kod!
- Eclipse har också kommandon för att köra javadoc
- VISA EXEMPEL!

Lab 3

- Fokus på länkade strukturer
- Simulering av en stormarknad
- Labstruktur:
  - Given kod för stormarknadssimuleringen
  - Ni ska lägga till ett paket med följande klasser
    - Länkad lista
    - Iterator för listan
    - Stack (baserad på länkad struktur)
    - Kö (baserad på länkad struktur)
    - Nodklass (gemensam för ovanstående klasser)
  - Koden ska kasta undantag där det behövs
  - Koden ska vara kommenterad med javadoc-kommentarer, och ni ska skapa ett API för er kod

Kommande veckor

- Lab 2: deadline fredag 27/9
- Lab 3
  - Handledning 30/9, 9/10
  - Deadline 14/10
- Nästa föreläsning: 9/10
  - Hashtabeller
  - Mer om objektorientering + övning
  - Hör av er om ni vill att jag ska ta upp något särskilt!
- Sista föreläsningen: 4/11
  - Repetition
  - Genomgång av exempeltenta
- Det flesta föreläsningarna har varit
  - Mycket eget arbete!
  - Hör av er om ni behöver hjälp/undrar något!
  - Kom på labtillfällena för att få hjälp

Jobba själv

- Gör labbar
- Gör progammeringsövningar
  - Från boken
  - Labbar från tidigare kursomgångar
- Läs/tentaplugga om det vi gått igenom på föreläsningarna och som står i kursmålen
- **Glöm inte att anmäla er till tentan i portalen!**