Технически университет – София
Факултет по Компютърни системи и управление

ПИК 3 - Java

Семинарни упражнения

гл. ас. д-р Антония Ташева

Списъци,
динамични масиви,
стекове и опашки,
изключения дефинирани от потребителя

ТЕМА 5. ДИНАМИЧНИ СТРУКТУРИ ОТ ДАННИ

Масиви

- наредени елементи от един и същи тип данни (може да бъде както примитивен тип, така и данни от определен клас).
- Те са с константна дължина (т.е. точно определен брой елементи).

Масиви - проблеми

- Често се заделя повече памет, отколкото се използва
- Разширение става само чрез създаване на нов масив и копиране на данните в него
- Не можем да изтрием елементи в масива, а само да им зададем стойност “null”

Списъци

- LinkedList или в превод „свързан списък“
- Това е динамична структура от данни (без ограничение на максимален брой елементи, стига да има достатъчно памет)
- „двусвързан“ - един първи, един последен елемент, а всички останали имат свой „предходен“ и „следващ“.

Списъци

- Дефиниция

  java.util.LinkedList<тип> list = new java.util.LinkedList<тип>();

а) Добавяне на елемент в края на списъка:

list.add(new Integer(1));

list.addLast(new Integer(2));

б) Добавяне на елемент в началото на списъка:

list.addFirst(new Integer(3));

в) Размер (брой елементи на списък):

list.size();

Списъци

г) Отпечатване на съдържанието на списъка чрез итератор:

```
java.util.Iterator it = list.iterator();
while(it.hasNext()){
  System.out.print(it.next()+" ");
}
```

Списъци

д) Добавяне на елемент на определена позиция:

```
list.add(2, new Integer(25));
```

// първи аргумент е позиция, а втори стойност

е) Прочитане на елемент от определена позиция:

```
list.get(2);
```

ж) Прочитане на първи и последен елементи:

```
list.getFirst();
list.getLast();
```

Списъци

з) Отпечатване на съдържанието на списъка БЕЗ итератор:

```
for (int i=0; i<list1.size(); i++){
 System.out.print(list1.get(i)+" ");
}
```

Списъци

и) Изтриване на елемент. Освен че се извършва изтриването се връща и стойността:

```
int first = list.removeFirst();
System.out.println("First element: "+first+" removed...");
int last = list.removeLast();
System.out.println("Last element: "+last+" removed...");
int x = list.remove(1);
System.out.println("Element with index 1: "+x+" removed...");
System.out.println("Now list contains:");
it = list.iterator();
while(it.hasNext()){
  System.out.print(it.next()+" ");
}
```

Списъци

й) Изтриване на всички елементи в списък:

list.clear();

к) Проверка дали списък е празен или не:

```
if (list.isEmpty()) System.out.println("List is empty");
```

л) „Клониране“ на списък:

```
java.util.LinkedList list = new java.util.LinkedList();
list.add(new Integer(1));
list.addLast(new Integer(2));
list.addFirst(new Integer(3));
java.util.LinkedList listClone = list.clone();
```

Списъци

м) Търсене на елемент в списък:

```
if(list.contains(3))
System.out.println("List has element with val 3");
```

н) Промяна на елемент с нова стойност:

```
list.set(1, new Integer(25));
// първи аргумент индекс, а втори стойност
```

Списъци

о) Добавяне на един списък в края на друг или вмъкването му в даден индекс:

```
java.util.LinkedList list1 = new java.util.LinkedList();
list1.add(new Integer(1));
list1.addLast(new Integer(1));
list1.addFirst(new Integer(1));
java.util.LinkedList list2 = new java.util.LinkedList();
list2.add(new Integer(2));
list2.addLast(new Integer(2));
list2.addFirst(new Integer(2));
list1.addAll(list2);
list1.addAll(1, list2);
```

LinkedList

+

- Спестява памет
- Лесен за манипулация

-

- Няма реална индексация, което може да го направи бавен

Динамични масиви

ArrayList – съчетава функционалността на списъците и статичните масиви.

- Има индекси, полетата са поредни клетки в паметта
- Капацитетът се разширява автоматично

ArrayList - методи

- премахнати - ~~addFirst, addLast, removeLast, removeLast~~
- add(елемент) – добавя елемент в края на списъка;
- add(индекс, елемент) – добавя елемент в даден индекс (и премества следващите напред);
- clear() – изтрива всички елементи;
- contains(елемент) – boolean;
- get(индекс) – връща стойността на елемент от даден индекс;
- isEmpty() – дали списъка е празен;
- remove(индекс) – изтрива елемент на даден индекс;
- set(индекс, елемент) – променя стойността на елемент на даден индекс с нова стойност);
- size() – връща броя елементи на масива;

ArrayList – нови методи

а) Подсигуряване за минимален капацитет:

```
java.util.ArrayList<Integer> arrlist = new java.util.ArrayList<Integer>();
arrlist.ensureCapacity(10);
```

Но по-добре:

```
java.util.ArrayList<Integer> arrlist = new java.util.ArrayList<Integer>(10);
```

б) indexOf(елемент) – връща индекса на първото срещане на елемента вътре в списъка (не се поддържа търсене от „елемент нататък“, както беше при клас String);

в) lastIndexOf(елемент) – индекса на последното срещане на елемента вътре в списъка;

ArrayList – нови методи

г) removeRange(индекс „от“, индекс „до“)

– изтрива елементите от първия до втория индекс подадени като аргумент. Например list.removeRange(2,4); изтрива елементи от 2 до 4 включително;

д) trimToSize()

– „свива“ капацитета на масива до броя на елементите му. Обикновено го използваме, за да освободим памет в случаи когато знаем, че няма да има повече добавяне на елементи.

Опашка и стек

- Queue (опашка) и Stack (стек) са частни случаи на двусвързания списък (LinkedList)
- Опашка - добавяте елементи само в края на списъка и премахване на елемент само от началото. Няма операции „търсене“ и „извличане по индекс“. (FIFO)
- Стек – добавяне (push) и изтриване (pop) само от края на списъка. Има операция само за прочитане (peek) на последния елемент без да се изтрива. (LIFO)

ПОТРЕБИТЕЛСКИ ИЗКЛЮЧЕНИЯ

extends Exception

```
class Car{
  private int speed;
  private String model;
  public Car(int speed, String model) throws SpeedException{
    this.setSpeed(speed);
    this.model = model;
  }
  public void showInfo(){
    System.out.println("The car "+this.model+" has maxspeed of "+this.speed);
  }
  public int getSpeed(){
    return this.speed;
  }
  public void setSpeed(int speed) throws SpeedException{
    if (speed < 0) throw new SpeedException(speed);
    else this.speed = speed;
}}
```

extends Exception

```
class SpeedException extends Exception{
  private int speed;
  public SpeedException(int speed){
    this.speed = speed;
  }
  public String toString(){
    return "Speed of Car must not be negative: " + this.speed;
}}
```

```
public class CarsExample{
  public static void main(String args[]) throws Exception {
    try{
      Car c = new Car(-150, "BMW X5");
    }
    catch (SpeedException e){
      System.out.println(e.toString());
}}}
```

extends Exception

- Предефиниране на метод „toString()“
- printStackTrace() - е наследен
- Подаване на съобщение към super класа за да работи и getMessage():

```
class SpeedException extends Exception{
  private int speed;
  public SpeedException(int speed){
    super("Speed of car must not be negative: "+speed);
    this.speed = speed;
  }
  public SpeedException(int speed, String msg){
    super(msg);
    this.speed = speed;
  }
  public String toString(){
    return "Speed of Car must not be negative: "+this.speed;
  }}
```

Нишки,
синхронизация на нишки

ТЕМА 6. НИШКИ

Какво е нишка?

- „разклонение на една програма на подпрограми, които работят едновременно“
- обслужваме на всички с равно количество процесорно време, независимо от реда

„многонишковост“ и „многозадачност“

- „многозадачност“, то се има предвид изпълнението на множество „**процеси**“ (програми) от операционната система едновременно. Контролира се от ОС
- „многонишковост“ говорим за контрол над изпълнението на „подпрограми“ от самата програма
- стартирането на едно и също приложение два пъти НЕ създава „две нишки“, а създава „два процеса“

„пускане на нишка“

- наследяване на системния клас „Thread“

```
public class ThreadExample extends Thread{
  public void run(){
    ... оператори ...
}}
```

- имплементирането на интерфейс Runnable

```
public class ThreadExample implements Runnable{
  public void run(){
    ... оператори ...
}}
```

```
public class myfirstprogram{
 public static void main(String[] args){
  // Създаване на нишка с подаден обект имплементиращ Runnable:
  Thread T1 = new Thread(new RunnableExample());
  T1.start();
  // Създаване на нишка, чрез наследник на Thread:
  ThreadExtendExample T2 = new ThreadExtendExample();
  T2.start();
 }}
 class RunnableExample implements Runnable {
  public void run() {
   System.out.println("Аз съм нишка!");
  }}
class ThreadExtendExample extends Thread {
   public void run() {
      System.out.println("Аз също съм нишка!");
}}
```

Пример

- ThreadsExample1
- ThreadsSleep.java
- ThreadsInterrupt.java
- ThreadsInterrupt2.java
- ThreadsInterrupt3.java

Синхронизация

- Работа със споделени ресурси
- Една нишка не трябва да „бърка“ в данните на друга
- Достъпваните данни се „заключват“, променят се и се „отключват“ за достъп от други

Синхронизация на методи

- Пример: ThreadsSynchro
- Когато една нишка извика „синхронизиран“ метод от един обект, то всички други нишки, които в същия момент извикат същия или друг синхронизиран метод от същия обект „заспиват“ и изчакват изпълнението си в опашка.
- Конструктори НЕ могат да се синхронизират

Синхронизация на фрагмент

```
Оператори 1;
synchronized(<име на обект>){
  Оператори 2;
}
Оператори 3;
```

- изпълнението на групата „Оператори 2” обектът дефиниран в блока ще бъде заключен за другите нишки.

Синхронизация на фрагмент

```
public void changeArray(){
 synchronized(this.arr){
    for(int i=0; i<this.arr.length; i++){
      this.arr[i] = (int)Math.round(Math.random()*100);
    }
    System.out.println("Change finished");
 }
public void sortArray(){
 synchronized(this.arr){
    java.util.Arrays.sort(this.arr);
 }
 System.out.println("Sort finished");
}
```

Метод wait()

- нестатичен метод Object.wait()
- предизвиква същия ефект както Thread.sleep() – прехвърля текущата нишка (тази, която е извикала метода) в „Not Runnable“ статус за определено време
- Такъв обект се нарича „заключващ обект“ за нишката
- Разлики:
  - Object.wait() може да бъде извикан само в синхронизиран метод
  - Object.wait() може да приспи нишката за неопределено време, докато Thread.sleep() е с фиксирано
  - ако имаме Thread T, то T.sleep() ще „приспи“ нишката T, докато T.wait() ще приспи текущата нишка (тази, която е извикала метод T.wait()), докато някой друг не извика T.notify()

wait() и notify()

- final методи за клас Object
- всеки един клас в Java ги притежава

```
class ThreadExample extends Thread{

  public static int sum=0;

  public void run(){
    synchronized (this){
      this.sum();
      System.out.println("Finished summing. Now the program can continue...");
      this.notify();
    }
  }

  public void sum(){
    System.out.println("Suming the first 500 integers...");
    for(int i=1; i<=500; i++){
      this.sum = this.sum+i;
    }
  }
}
```

```
public class waitNotifyExample{
  public static void main(String[]args){
    ThreadExample T = new ThreadExample();
    T.start();
    synchronized(T){
      System.out.println("Waiting until calculations finish...");
      try{
        T.wait();
      }
      catch (InterruptedException e){}
    }
    System.out.println("Result: "+T.sum);
  }
}
```

Пример

- ThreadsWaitNotify2
- главната нишка стартира нова нишка и новата нишка изчаква докато главната не я уведоми
- ThreadsWaitNotifyEx Money example

notifyAll()

- „събужда“ всички нишки, които са попаднали в wait() статус
- Обикновено използваме notifyAll() когато нишките имат споделен ресурс, по който те се заключват
- Пример: ThreadsWaitNotifyAll

Домашни задания №3

  - Варианти за работа в екип от 2-ма души
  - Самостоятелни задачи
- Предаване и защита 10-та седмица!

- Четвъртък – лабораторно 15:30, зала 1409
- Контролно:
  - Зала 1151 от 09:30 - 11:30: групи 40 и 41