Тема 10 - Колекции

18 юни 2018 г. 20:14

Wrapper(Обвивка) класове

В Java се използват примитивни типове от данни като например: int, double, float, boolean за да се съхранява проста информация. Тези примитивни типове се използват вместо обекти с цел бързодействие. Но въпреки това понякога имаме нужда от някакво представяне на тези типове като обекти. Java предоставя едни класове наречени Type Wrappers, които обвиват примитивните типове и ги представят като обекти. Класовете, които са type wrappers са: Double, Float, Long, Integer, Short, Byte, Character, Boolean. Те съдържат в тях много полезни методи, които може да използвате, но главната причина да ги използваме е понеже Java Collections API работи с тях. Това, което е удобно за тези Wrapper класове ,е че те правят нещо наречено autoboxing/unboxing.

```
//Файл:Wrap.java
class Wrap {
    public static void main(String args[]) {
        Integer iOb = new Integer(100);
        int i = iOb.intValue();
        System.out.println(i + " " + iOb);  // тука се взима стойността по два начина,
        когато се извиква i се взема стойността чрез метода intValue(), а когато се
        извиква обекта iOb се взема стойността от метода toString().
    }
}
```

Механизма Autoboxing/unboxing

Това е процеса ,в който примитивен тип е автоматично капсулиран в wrapper обект (Autoboxing), и когато стойността на wrapper обект е автоматично взета от него (Autounboxing).

```
//Файл:WrapBox.java
class WrapBox {
    public static void main(String args[]) {
        Integer iOb = 100; // autobox
        int i = iOb; // auto-unbox
        System.out.println(i + " " + iOb);
        System.out.println( getInt(50) );// auto-box
    }
    private static int getInt(Integer someInt) {
        return someInt; //auto-unbox
    }
}
```

Този механизъм е полезен, но е важно да помним, че когато използваме примитивни типове вместо техните wrapper класове програмата работи по ефикасно и бързо. Няма нужда да се използват wrapper класове ако просто ще правим сметки.

```
//Пример за ненужно използване на wrapper класове
Double a, b, c;
a = 10.0;
b = 4.0;
c = Math.sqrt(a*a + b*b);
System.out.println("Hypotenuse is " + c);
```

Java Collections(Колекции) API

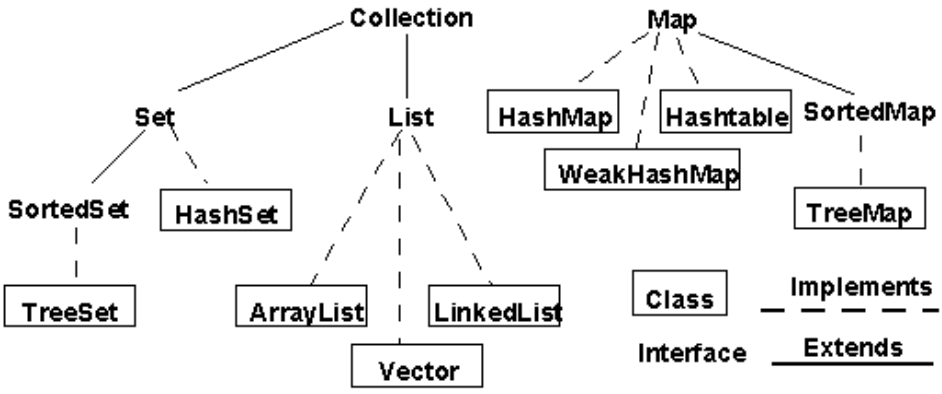
Това са съвкупност от класове написани с цел бързо-действие и универсалност спрямо групи от подобни обекти. Както масивите те се използват за да можете с едно име на променлива да се обръщате към повече от един обект, но важно е да се запомни че те не са масиви и дори най-близко приближаващият се до масиви клас ArrayList се различава по това, че може да съдържа неопределен брой обекти(и не само).

В основата на Java колекциите седят няколко основни интерфейса, които обявяват главните методи ,чрез които работим с различните имплементации на тези интерфейси: Collection, Map. А директно под Collection седят Set и List.

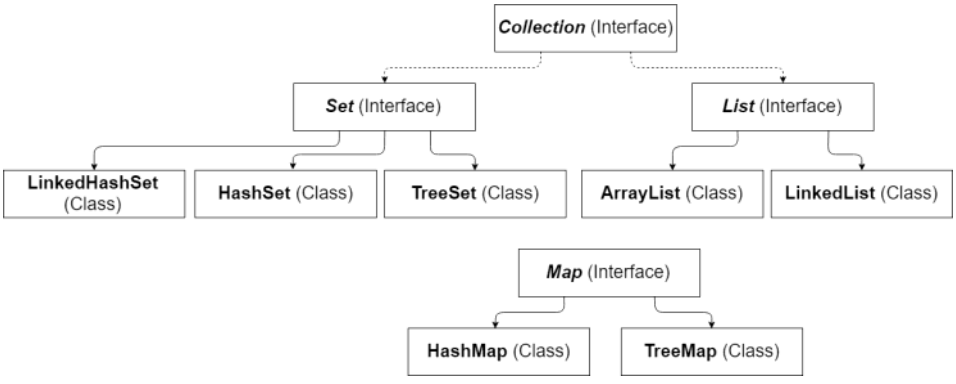
Това са и главните структури, с които се работи:

- Map - група от двойки (ключ, стойност)
- List - група от стойности индексирани чрез целочислен тип(int като масивите 0,1,2,3,4)
- Set - група от **уникални** стойности

Структурата на наследяване на тази област от класове и интерфейси изглежда така:



С цел улеснение обаче можем да я намалим до главните използвани класове:



Всеки от тези 7 класа се използва с различни цели. Следва кратко обяснение на всеки един от класовете.

- **HashSet** - Използва се за съхраняване на **уникални стойности** без да се запазва реда на въвеждане.
- **LinkedHashSet** - Използва се за съхраняване на уникални стойности като **се запазва реда на въвеждане**.
- **TreeSet** - Използва се за съхраняване на уникални стойности като **автоматично се нареждат по възходящ или низходящ ред**.
- **ArrayList** - използва се за съхранение на стойности (може да има дублиране) това е най-близката структура до масив, която още се нарича динамичен масив или вектор(В Java имаше клас Vector, но вече не се използва) т.е. това е масив който може да не обявяваме колко елемента ще съдържа и той **винаги ще има място за добавяне**.
- **LinkedList** - като идея е същото като ArrayList, но се използва ако ще се **добавят или премахват елементи от края или началото** на списъка често.
- **HashMap** - използва се за съхранение на двойки от типа **ключ -> стойност**, като пример в реалния свят е един реален речник където ключа се явява думата, а стойността е значението на думата (В други езици структурата се нарича Dictionary или HashTable(в Java има такъв клас но не се използва)).
- **TreeMap** - като идея е същото като HashMap, обаче автоматично нарежда *ключовете* по възходящ или низходящ ред (не стойностите).

Главните методи в Collection имплементиращите класовете (т.е. без Map):
"E" типът представлява произволен тип(обяснението е малко по-надолу)

```
boolean add(E e)
boolean contains(Object o)
boolean remove(Object o)
boolean isEmpty()
int size()
```

Главните методи в Map имплементиращите класове:
"V" типът е произволен тип, а "K" типът е друг произволен тип

```
V put(K key, V value)
Boolean containsKey(Object key)
Boolean containsValue(Object value)
V remove(Object key)
Boolean isEmpty()
Int size()
V get(Object key)
Collection<V> values()
Set<K> keySet()
```

Вижда се, че тези методи работят или с произволни типове, за които не сме обяснили още или с тип Object, което просто означава, че всичко което наследява типът Object може да бъде поставено на негово място. В Java всеки клас без значение дали е означено наследява от Object по директен или индиректен начин. Друго, което е важно да се спомене ,е че Map интерфейсът описва метод get(Object key), а Collection няма такъв, това е защото Set няма такъв метод (защото Set се използва с друга цел), а List има.

Използване на ArrayList:

```
//Файл:ArrayListTest.java
import java.util.ArrayList;

class ArrayListTest {

    public static void main(String [] args) {

        ArrayList list = new ArrayList(); // създаване на обект и променлива която сочи
        към него
        list.add(10); // добавяне на елемент
        list.add(5);
        list.add(15);
        int sum = 0;
        for(int i = 0; i < list.size(); i++) {
            sum += (Integer)list.get(i); // кастване на обекта който получаваме към
            Integer
        }
        System.out.println(sum);
    }

}
```

Интересният ред в тази програма е този:

```
sum += (Integer)list.get(i);
```

Методът който използваме за взимане на обект от списъка, винаги връща обект от тип Object, което означава, че ние трябва да знаем какво се намира в списъка и ръчно да го преобразуваме(кастнем(cast)) в правилният тип(в случая Integer).

Това може да доведе, до някой доста сериозни проблеми, като например:

- можем да добавим нещо друго освен Integer в списъка - ще ни бъде разрешено, но това разваля правилото, че в колекция се поставят подобни един на друг обекти;
- когато използваме get метода може да объркаме какъв обект има в списъка или изобщо да не знаем .

За това доста отдавна не се ползва този начин за създаване на колекции, а се използва нещо наречено Generic типове. Това са тези произволни типове, които гледахме по рано. Това означава, че можем изрично да кажем какъв тип ще съдържа списъка и да не се притесняваме, че можем да сложим грешен тип или, че ще извадим грешен тип защото Java просто няма да ни разреши да го направи.(ще извади грешки)

```
//Файл:ArrayListGenericTest.java
import java.util.ArrayList;

class ArrayListGenericTest {

    public static void main(String [] args) {

        ArrayList<Integer> list = new ArrayList<>(); // изрично оказваме какъв тип
        обекти ще има в списъка
        list.add(10);
        list.add(5);
        list.add(15);
        int sum = 0;
        for(int i = 0; i < list.size(); i++) {
            sum += list.get(i); // няма нужда да кастваме понеже излиза обекта който
            сме обявили
        }
        System.out.println(sum);
    }

}
```

Чрез този ред изрично казваме на Java , че в този списък ще се съдържат само обекти от тип Integer и подобни на него.

`ArrayList<Integer> list = new ArrayList<>();`

Забележка: Когато използваме Generic сме длъжни винаги да използваме обекти, в случай, че искаме примитивен тип използваме неговия wrapper клас.

Използване на HashSet:

```
//Файл:HashSetTest.java
import java.util.HashSet;

class HashSetTest {

    public static void main(String [] args) {
        HashSet<String> names = new HashSet<>();
        names.add("Dimitar");
        names.add("Ivan");
        names.add("Stefan");
        names.add("Dimitar");//няма да се добави втори път понеже вече се съдържа в
        списъка

        for(String name:names) {// използване на foreach loop
            System.out.println(name);
        }

    }

}
```

Понеже в Set интерфейса елементите трябва да са уникални, не можем да добавяме дублиращи се елементи и така след изпълнението на програмата на екрана ще се изведе:

Dimitar
Ivan
Stefan
Но не можем да разчитаме на реда, в който са наредени, ако искахме задължително да се запазва реда щяхме да използваме LinkedHashSet.
Използваме foreach стил цикъл, защото Set не разполага с индекси, през които да минаваме с обикновен for цикъл.

Използване на HashMap:

```
//Файл:HashMapTest.java
import java.util.HashMap;
import java.util.Set;

class HashMapTest {

    public static void main(String [] args) {
        HashMap<String,Boolean> attendance = new HashMap<>(); // създаваме списък на
        присъстващите
        attendance.put("Dimitar Tomov",true); // добавяме като слагаме двойки
        (ключ,стойност)
        attendance.put("Stefan Stefanov",false);
        attendance.put("Ivan Ivanov",true);

        Set<String> names = attendance.keySet(); // взимаме сет от ключовете на картата

        for(String name:names) { // използваме foreach loop защото преминаваме през Set
            System.out.print(name + " ");
            // ако на ключа отговаря true изкарваме на екрана "присъства", ако е false
            "не присъства"
            if(attendance.get(name)) {
                System.out.println("prisustva");
            } else {
                System.out.println("ne prisustva");
            }

        }

    }

}
```

За да обиколим цялата карта трябва да използваме нейния Set от ключове:
`Set<String> names = attendance.keySet();`

Тази програма може да се опрости с тернареният оператор(Ternary operator):

//Файл:HashMapTestRefactored.java

```
import java.util.HashMap;
import java.util.Set;

class HashMapTestRefactored {

    public static void main(String [] args) {
        HashMap<String,Boolean> attendance = new HashMap<>();
        attendance.put("Dimitar Tomov",true);
        attendance.put("Stefan Stefanov",false);
        attendance.put("Ivan Ivanov",true);

        Set<String> names = attendance.keySet();

        for(String name:names) {
            String attStr = attendance.get(name)?"prisustva":"ne prisustva";
            System.out.println(name + " " + attStr);
        }

    }
}
```

Тернарният оператор се използва за съкратено присвояване на стойност спрямо някакво условие и има следният вид:

```
<boolean> ? <if boolean true> : <if boolean false>
```

И обикновено резултата от него или се връща от метод или се присвоява на променлива.

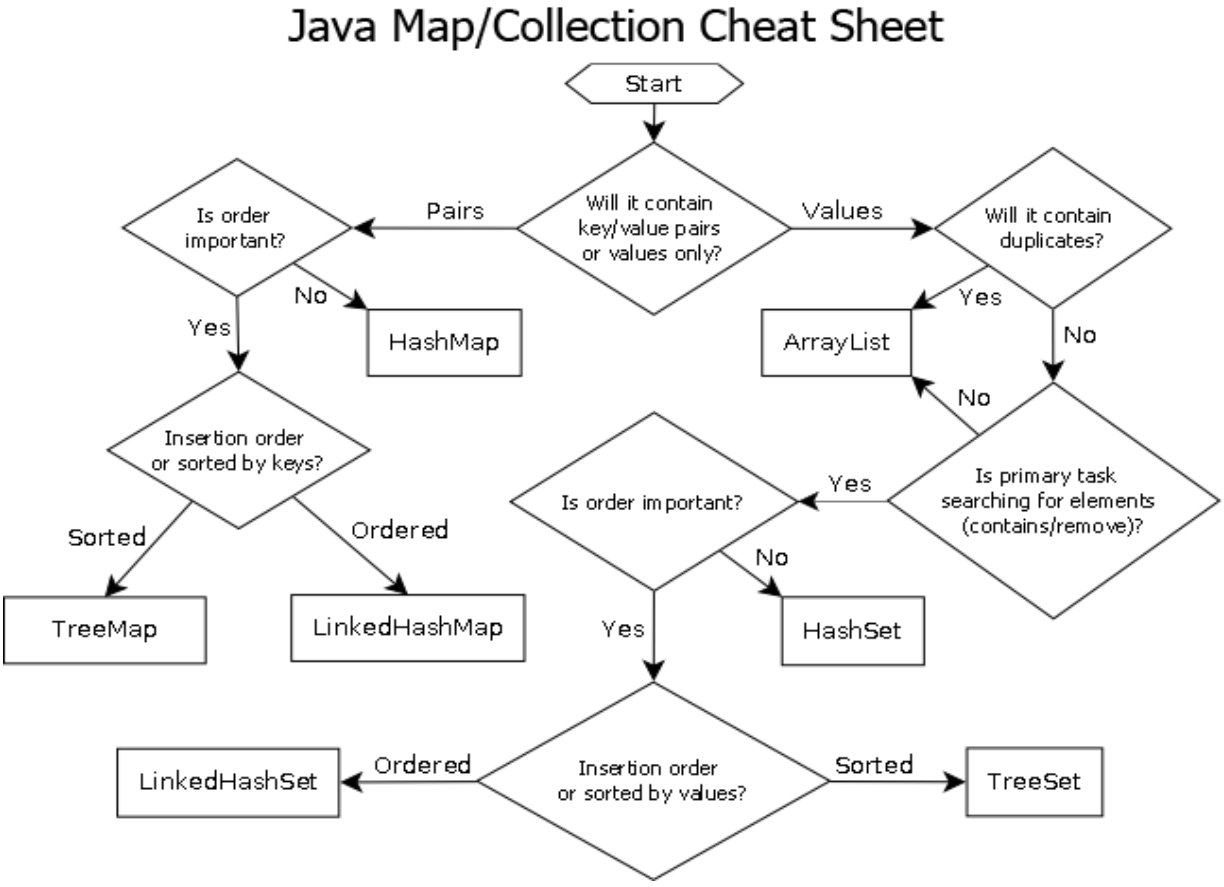
```
String str = 1>0?"edno e po golqmo ot nula":"edno ne e po golqmo ot nula";
```

Колекциите могат да се използват в комбинации една с друга, но трябва да се внимава. Примерно:

//Файл:CombineCollections.java

```
import java.util.*;// добавяне на всички колекции

class CombineCollections {

    public static void main(String [] args) {

        ArrayList<HashSet<String>> classRoomList = new ArrayList<>();

        HashSet<String> classRoomA = new HashSet<>();
        classRoomA.add("Dimitar Tomov");
        classRoomA.add("Ivan Popov");
        classRoomA.add("Stefan Stefanov");
        HashSet<String> classRoomB = new HashSet<>();
        classRoomB.add("Dimitarinka Tomova");
        classRoomB.add("Ivanka Popova");
        classRoomB.add("Stefani Stefanova");

        classRoomList.add(classRoomA);
        classRoomList.add(classRoomB);

        for(int i = 0; i < classRoomList.size(); i++) {
            System.out.println("Classroom " + (i+1));
            printSet(classRoomList.get(i));
        }

    }

    private static void printSet(HashSet<String> set){
        for(String str:set) {
            System.out.println(str);
        }
    }
}
```

Диаграма как да изберете правилната колекция:



Задача:

//Файл:SynonymDictionary.java, Файл:SynonymDictionaryTest.java

Използвайте HashMap и ArrayList за да създадете синонимен речник, на който като му се подаде дума, да извежда всички нейни синоними.

Примерни думи:

програма -> разписание, план, система, платформа, представление, спектакъл
задача -> проблем, загадка, командировка, проект, интерес
колекция -> сбирка, подбор, галерия, музей
колега -> другар, съученик, събрат, съучастник
училище -> школа, учебно заведение, колеж, техникум

Как ще добавите синоним на дума, която вече я има в вашият речник?

Допълнителни линкове към темата:
https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html
https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html
http://how2examples.com/java/collections

Автор: Димитър Томов - xdtomov@gmail.com