Структури от Данни и Обектно-Ориентирано Програмиране
спец. Компютърни Науки
Упражнение №6
31.03.2009г.

ТЕМА: Вложени класове и интерфейси.
Параметризирани класове и интерфейси

Деклариране на вложен клас:

```
class EnclosingClass {
   ...
   class NestedClass {
      ...
   }
}
```

Статичен вложен клас – има статут на член на класа, който се отразява на правото му на достъп до членове на класа и обекта. Вложените интерфейси са по премълчаване статични.

```
class EnclosingClass {
   ...
   static class StaticNestedClass {
      ...
   }
}
```

За да се създаде и да съществува обект от статичния вложен клас не е необходимо да съществува обект от външния клас. Аналогично, създаването на обект от външния клас не води автоматично до създаване на обект от вложения клас.
Методите на вложения клас, независимо дали са статични или не, имат достъп ***само до статичните членове*** на външния клас.

```
public class R {
  int y=1;
  static int z=2;

   public static class S {
      int x;
```

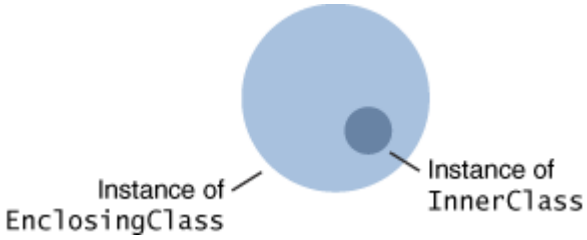
```
        void f () {
            System.out.println("f(): x=" + x);
            //System.out.println("f(): y=" + y); //Static reference to a non-static field
            System.out.println("f(): z=" + z);
        }

        static void g() {
            //System.out.println("g(): x=" + x);  //Static reference to a
                                                  // non-static  field
            // System.out.println("g(): y=" + y); //Static reference to a
                                                  // non-static  field
            System.out.println("g(): z=" + z);
        }
    } // end of class S
}  // end of class R

public class UseR {

  public static void main(String[] args) {
      R.S obj = new R.S();
      obj.f();
      R.S.g();
      obj.g();  // Warning
  }
}
```

Нестатичен вложен клас – вътрешен клас. Не може да декларира статични членове, с изключение на константи. Обект на вътрешния клас съществува само в обект на обхващащия клас и има достъп до неговите членове. Обект от външния клас може да бъде асоцииран с повече от един обект от вътрешния клас.



Тъй като всеки екземпляр на вътрешния клас е вложен в екземпляр на външния, то неговите методи имат достъп до всички членове - полета и методи на външния.

```
public class A {
  int y=1;
  static int z=2;
```

```
  public void useB() {
        B b=new B();
        b.f();
  }

  void h() {
        System.out.println("Outer metod called from inner");
  }


   public class B {  // innner class
        int x;
        public void f () {
              System.out.println("f(): x=" + x);
              System.out.println("f(): y=" + y);
              System.out.println("f(): z=" + z);
              h();
        }
        //static void g() {}  // cannot be declared static
  }
}

public class UseA {
  public static void main(String[] args) {
        //A.B obj = new A.B();  // no enclosing instance of A accessible
        A obj=new A();
        obj.useB();
        obj.new B().f();
  }
}
```

Параметризирани класове и интерфейси

Родов тип – референтен тип, който има параметър-тип. Всеки параметър-тип се заменя с аргумент-тип когато се създава екземпляр на родовия тип.
Типът параметър може да се използва като тип на нестатично поле, тип на параметър или връщана стойност на нестатичен метод, или локална променлива.

```
public class GenericClass<T> {
  T field;

  public T getField () {
    return field;
  }
```

```
  public void setField (T field) {
    this.field = field;
  }

  public void someOtherMethod () {
    T local = field;
    // Use local in some way.
  }
}
```

Параметризиран тип – екземпляр на родов тип, в който типът-параметър е заместен с тип актуален аргумент, например:

```
GenericClass <String> str = new GenericClass <String>();
```

е конкретен параметризиран тип, получен при заместване на типа-параметър ***T*** с типа – актуален аргумент ***String*** и може да се използва подобно на другите референтни типове.

Задаване на граници (долна и горна) на типа-параметър:

```
class CMP<T extends Number & Comparable<T>> {
    CMP (T first, T second) {
        System.out.println (first.compareTo (second));
    }
}
```

Задача 1: Да се създаде итерфейс **Accumulator** и негови класове – наследници за представяне на данна-акумулатор, с основна операция добавяне.

```
public interface Accumulator<T extends Number> {
        public void add(T v);
}

public class NumberAccumulator implements Accumulator <Number>{
        private double total;       // accumulation of numeric values

        // constructor initializes total to 0.0
        public NumberAccumulator ()
        { total = 0.0; }

        // return the total
        public double getTotal()
        { return total; }
```

```
        // update the total by the value of v as a double
        public void add(Number v)
        { total = total + v.doubleValue(); }
}
```

```
public class IntegerAccumulator implements Accumulator <Integer> {

        private int total; // accumulation of numeric values

        public IntegerAccumulator () { total = 0; }

        public int getTotal() { return total; }

        public void add(Integer v) { total = total + v.intValue(); }
}
```

```
public class DoubleAccumulator implements Accumulator <Double> {
        private double total;       // accumulation of numeric values

        public DoubleAccumulator () { total = 0.0; }

        public double getTotal() { return total; }

        public void add(Double v) { total = total + v.doubleValue(); }
}
```

Задача 2: Да се напише програма, която сортира масив от данни по метода на бързото сортиране. Масивът може да съдържа данни от произволен тип, който реализира интерфейса *Comparable*.

```
import java.util.Scanner;

public class QuickSortTest {

        static Integer[] array1;
        static Double[] array2;
        static String[] array3;
        static Scanner sc = new Scanner(System.in);

        public static void main(String[] args) {
                getIntegerArray(array1);
                quickSort(array1);
                printArray(array1);
                 getDoubleArray(array2);
```

```
        quickSort(array2);
        printArray(array2);
        getStringArray(array3);
        quickSort(array3);
        printArray(array3);
}

static void getIntegerArray(Integer[] array) {
        int n = sc.nextInt();
        array1 = new Integer[n];
        for(int i=0; i<n; i++)
                array1[i] = sc.nextInt();
}

static void getDoubleArray(Double[] array) {
        int n = sc.nextInt();
        array2 = new Double[n];
        for(int i=0; i<n; i++)
                array2[i] = sc.nextDouble();
}

static void getStringArray(String[] array) {
        int n = sc.nextInt();
        array3 = new String[n];
        sc.nextLine();
        for(int i=0; i<n; i++)
                array3[i] = sc.nextLine();
}

static <T extends Comparable<T>> void printArray(T [] array)  {
        for (int i=0; i<array.length; i++)
                System.out.println(array[i].toString());
}

static <T extends Comparable<T>> void quickSort(T [] array)  {
        qSort(0, array.length-1, array);
}

static <T extends Comparable<T>> void qSort(int left, int right, T [] array) {
        int pivotIndex=-1;
        if (right>left)  {// more than 1 element
                pivotIndex=partition(left, right, array);
                qSort(left, pivotIndex-1, array);
                qSort(pivotIndex+1, right, array);
   }
}
```

```
    static <T extends Comparable<T>> int partition(int left, int right, T [] array)  {
        int leftPtr=left;
        int rightPtr=right;
        if (array[leftPtr].compareTo(array[rightPtr])>0)
            swap(leftPtr, rightPtr, array);
        T pivot=array[left];
        do {
            do  leftPtr++;
            while (array[leftPtr].compareTo(pivot)<0);
            do  rightPtr--;
            while (array[rightPtr].compareTo(pivot)>0);
            if (leftPtr<rightPtr)
                swap(leftPtr, rightPtr, array);
        } while (leftPtr<rightPtr);
        int pivotIndex=rightPtr;
        swap(left, pivotIndex, array);
        return pivotIndex;
    }

    static <T extends Comparable<T>> void swap(int left, int right, T[] array) {
        T temp=array[left];
        array[left]=array[right];
        array[right]=temp;
    }
}
```