Problem 1

Create class **Square** with one private field of type **double** representing the length of the side. Add the constructor and "getters" returning the side, length of the diagonal, perimeter and surface area of the square represented by an object of the class. Do *not* define "setters" — object of this class should be immutable.

Also, define a static function taking references to two squares and returning the reference to a square, created in the function, with the surface area equal to the sum of the areas of the two squares passed to it.

Test your class in the **main** function of a separate class.

In order to ensure immutability of objects of our class, its sole field should be declared not as a **double**, but rather as a **final double**. Then, after construction, it will be impossible to modify it even by methods of the class.

Problem 2

Temperature is measured mainly in three units: in degrees Celsius, degrees Fahrenheit and in kelvins. It's easy to convert any of them to the two others:

```
Kelvin to Celsius:     C = K - 273.15
Celsius to Kelvin:     K = C + 273.15
Kelvin to Fahrenheit:  F = 9./5*(K - 273.15) + 32
Fahrenheit to Kelvin:  K = 5./9*(F - 32) + 273.15
Celsius to Fahrenheit: F = 9./5*C + 32
Fahrenheit to Celsius: C = 5./9*(F - 32)
```

Write class **Temperature** with one (and only one!) private field of type **double**; objects of the class describe temperature. The class has one constructor and three methods:

- `Temperature(double tm, char unit)` — constructor taking temperature (as a **double**) and symbol of the unit used: 'C' for Celsius, 'F' for Fahrenheit and 'K' for kelvins;
- three methods („getters") returning the temperature represented by an object, but in different units:

```
public double getInC()
public double getInF()
public double getInK()
```

For example, the program

download *Temperatures.java*

```
public class Temperatures {
    public static void main(String[] args) {
        Temperature t1 = new Temperature(25, 'C');
```

```
        System.out.printf("C: %6.2f%n", t1.getInC());
        System.out.printf("F: %6.2f%n", t1.getInF());
        System.out.printf("K: %6.2f%n", t1.getInK());
        Temperature t2 = new Temperature(77, 'F');
        System.out.printf("C: %6.2f%n", t2.getInC());
        System.out.printf("F: %6.2f%n", t2.getInF());
        System.out.printf("K: %6.2f%n", t2.getInK());
        Temperature t3 = new Temperature(298.15, 'K');
        System.out.printf("C: %6.2f%n", t3.getInC());
        System.out.printf("F: %6.2f%n", t3.getInF());
        System.out.printf("K: %6.2f%n", t3.getInK());
    }
}
```

should print

```
C:  25,00
F:  77,00
K: 298,15
C:  25,00
F:  77,00
K: 298,15
C:  25,00
F:  77,00
K: 298,15
```

Problem 3

Create a class **Child** with two non-static public fields:

- name of type **String**,
- numCandies — number of candies — of type **int**; by default 2.

The class defines also one non-static method

```
void givesCandyTo(Child other)
```

— *this* child (on which the method was invoked) gives one candy to the other child, i.e., the number of its candies decreases and the number of the other's candies increases. If *this* child doesn't have any candies left, a message is displayed and the numbers of candies do not change.

The **Child** class defines also two static methods

- `static Child getChildByName(Child[] children, String name)` — takes an array of children and a name (**String**) and returns the child from the array with a given name, or **null**, if there was no child with this name in the array.
- `static void printChildren(Child[] children)` — takes an array of children and prints information on them in one line (as in the example below).

Then, in the **main** function *of a separate class* **Main**, test your class **Child** as desribed below:

The program creates an array of children (for simplicity, with length 3, but it could be any number).

In a loop, the program asks the user to enter names of these children. It creates objects representing the children with given names (and number of candies equal to 2) and stores references to these objects in the array.

In a loop, the program asks the user to enter two names, say name1 and name2; it means that the child with name name1 gives one candy to its friend with name name2.

After each such operation, the program prints the array to demonstrate that it has been completed correctly.

The loop ends when the first name starts with the letter 'q'; then the second name doesn't have to be read.

A run of the program could be, for example, as below:

```
3 children will be created
Name of the child no 0: Bob
Name of the child no 1: Jane
Name of the child no 2: Kate
3 children: [ (Bob, 2 candies) (Jane, 2 candies) (Kate, 2 candies) ]
Entering the name of the 'giver' starting
with 'q' terminates the program

Enter names of 'giver' and 'receiver': Jane Kate
[ (Bob, 2 candies) (Jane, 1 candies) (Kate, 3 candies) ]
Enter names of 'giver' and 'receiver': Jane Cindy
No child with name Cindy
Enter names of 'giver' and 'receiver': Cindy Jane
No child with name Cindy
Enter names of 'giver' and 'receiver': Cindy John
No child with name Cindy
No child with name John
Enter names of 'giver' and 'receiver': Jane Bob
[ (Bob, 3 candies) (Jane, 0 candies) (Kate, 3 candies) ]
Enter names of 'giver' and 'receiver': Jane Kate
No candies to give!
[ (Bob, 3 candies) (Jane, 0 candies) (Kate, 3 candies) ]
Enter names of 'giver' and 'receiver': Kate Jane
[ (Bob, 3 candies) (Jane, 1 candies) (Kate, 2 candies) ]
Enter names of 'giver' and 'receiver': Bob Kate
[ (Bob, 2 candies) (Jane, 1 candies) (Kate, 3 candies) ]
Enter names of 'giver' and 'receiver': q
```