Problem 1

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

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

The class defines also one non-static method

```
public 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

- `public static Child getChildByName(Child[] children, String name)` which 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.
- `public static void printChildren(Child[] children)` which 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
```

Problem 2

Create two classes – **Person** and **Car**. Class **Person** has two fields: name (**String**) and car (**Car**). Not all persons have cars — for those who haven't, the field car will be **null**. The **Car** class has fields make (**String**) and price (**int**). In both classes, all fields are private, so appropriate *getters* will probably be needed. Both classes override method **toString** (or, if you don't know what's that, define a method **info** which just prints information on the object it's been invoked on).

Write three *static* member functions in class **Person**:

- **getCars** which takes an array of references to persons, creates and returns a, perhaps empty, array of (references to) cars owned by these persons. Remember that not all persons own cars. The array returned cannot contain **null**s;
- **expensiveCars** which takes an array of (references to) persons and a minimum price (**int**), creates and returns an array of (references to) cars owned by the persons, but only those cars with prices not less than the given minimum. The function returns an empty array if there is no car which meets the criterion. The array returned cannot contain **null**s.
- **getPrice** which takes an array of (references to) persons and a name (**String**), and returns the price of the car of a person with this name, or -2 if there was no person with this name, or -1 if person with this name doesn't have a car.

In function **main** (in a separate class), create an array of persons and test both

functions.

For example. the following program

download *Objs1.java*

```
public class Objs1 {
    public static void main (String[] args) {
        Car c1 = new Car("Ford", 15000);
        Car c2 = new Car("Dodge", 11100);
        Car c3 = new Car("Buick", 11000);
        Person john = new Person("John", c1);
        Person mary = new Person("Mary", c2);
        Person kate = new Person("Kate");
        Person lucy = new Person("Lucy", c3);
        Person adam = new Person("Adam",
                                 new Car("Opel", 18000));
        Person[] persons = {
            john, mary, kate, lucy, adam, new Person("Jill"),
            new Person("Josh", new Car("Buick",12500))};

        Car[] cars = Person.getCars(persons);
        for (Car c : cars)
            System.out.print(c + " "); // or c.info();
        System.out.println();

        cars = Person.expensiveCars(persons, 12000);
        for (Car c : cars)
            System.out.print(c + " "); // or c.info();
        System.out.println();
        String name = "Adam";
        System.out.println("Price of " + name + "'s car: " +
                           Person.getPrice(persons, name));
    }
}
```

should print something like

```
Ford(15000) Dodge(11100) Buick(11000) Opel(18000) Buick(12500)
Ford(15000) Opel(18000) Buick(12500)
Price of Adam's car: 18000
```

Problem 3

Create two classes, **Square**, with a field corresponding to the length of its side, and **Circle**, with the field describing its radius. Both classes should have one-parameter constructors, methods returning side/radius (**getSide**/**getRadius**), area (**getArea**) and perimeter (**getPerimeter**) and should override method **toString** from class **Object**.

The class **Square** should contain methods **getInscribedCircle** and **getCircumscribedCircle** returning objects of class **Circle**[1] corresponding to circles inscribed in and circumscribed about *this* square. Similarly, the class **Circle** should contain methods **getInscribedSquare** and **getCircumscribedSquare** returning objects of class **Square** corresponding to squares inscribed in and circumscribed about *this* circle.

Add constructors to both classes: in class **Square** a constructor taking an object of class **Circle** and constructing a square with the same area as the given circle, and analogous constructor in class **Circle** taking an object of class **Square**.

Class **Circle** should define *static* function

```
public static Square[] getSquares(Circle[] arr)
```

which takes an array of references to objects of type **Circle** and creates and returns an array of references to objects of type **Square**, which have the same area as the corresponding circles.

Class **Square** should define an analogous function

```
public static Circle[] getCircles(Square[] arr)
```

which takes an array of references to objects of type **Square** and creates and returns an array of references to objects of type **Circle**, which have the same area as the corresponding squares.

In the **main** function *of another class* (e.g., **Main**) create a few objects of both classes and test the constructors and methods. For example, the following program

download *SquareCirc.java*

```
import java.util.Locale;

public class SquareCirc {
    public static void main (String[] args) {
          // we want doubles to be printed with a dot
        Locale.setDefault(Locale.US);

        Square[] sqs = {
                new Square(2), new Square(1), new Square(3)
        };

        System.out.print("Squares: ");
        for (Square s : sqs)
            System.out.print(s + " ");
        System.out.print("\nAreas: ");
        for (Square s : sqs)
            System.out.print(s.getArea() + " ");
        System.out.print("\nPerimeters: ");
        for (Square s : sqs)
            System.out.print(s.getPerimeter() + " ");
        System.out.print("\nInscribed circles: ");
        for (Square s : sqs)
            System.out.print(s.getInscribedCircle() + " ");
```

[1] Strictly speaking — references to objects...

```
            System.out.print("\nCircumscribed circles: ");
            for (Square s : sqs)
                System.out.print(s.getCircumscribedCircle() + " ");

            System.out.println("\n\n...and now circles from squares...");
            Circle[] circles = Square.getCircles(sqs);
            System.out.print("Circles: ");
            for (Circle c : circles)
                System.out.print(c + " ");
            System.out.print("\nAreas: ");
            for (Circle c : circles)
                System.out.print(c.getArea() + " ");
            System.out.print("\nInscribed squares: ");
            for (Circle c : circles)
                System.out.print(c.getInscribedSquare() + " ");
            System.out.print("\nCircumscribed squares: ");
            for (Circle c : circles)
                System.out.print(c.getCircumscribedSquare() + " ");

            System.out.println("\n\n...and back to squares...");
            Square[] squares = Circle.getSquares(circles);
            System.out.print("Squares: ");
            for (Square s : squares)
                System.out.print(s + " ");
            System.out.println();
        }
    }
```

should print (formatting may be different):

```
Squares: Square[2.0] Square[1.0] Square[3.0]
Areas: 4.0 1.0 9.0
Perimeters: 8.0 4.0 12.0
Inscribed circles: Circle[1.00] Circle[0.50] Circle[1.50]
Circumscribed circles: Circle[1.41] Circle[0.71] Circle[2.12]

...and now circles from squares...
Circles: Circle[1.13] Circle[0.56] Circle[1.69]
Areas: 3.9999999999999996 0.9999999999999999 9.0
Inscribed squares: Square[1.6] Square[0.8] Square[2.4]
Circumscribed squares: Square[2.3] Square[1.1] Square[3.4]

...and back to squares...
Squares: Square[2.0] Square[1.0] Square[3.0]
```

NOTE: The value of π is available as Math.PI; square root of x can be calculated as Math.sqrt(x).