Problem 1

Write a class **Square** representing squares and containing one private field side and

- constructor initializing the field side;
- default constructor setting side to 1 (use delegation to the first constructor);
- overridden method `public String toString()`;
- method `public double getSide()` returning the length of the side of the square;
- method `public double getArea()` returning the area of the square;
- method `public double getDiagonal()` returning the length of the diagonal of the square.
- method `public double getPerimeter()` returning the length of the perimeter of the square;
- method `public Square getDoubled()` returning a **Square** with its side two times larger that that of the object the method has been invoked on;
- static method `public static double getTotalArea(Square[] sqs)` taking an array of references to **Square**s and returning the sum of their areas;
- static method `public static Square getMax(Square[] sqs)` taking an array of references to **Square**s and returning the reference to the greatest of them.

Test your class from the **main** function *of another class* **Main**. For example, the following **main**

download *BaseClassSquare.java*

```
public static void main(String[] args) {
    Square s2 = new Square(2);
    Square s1 = new Square();
    System.out.println(s2 + " " + s1);
    System.out.println("Side    " + s2.getSide());
    System.out.println("Area    " + s2.getArea());
    System.out.println("Diag    " + s2.getDiagonal());
    System.out.println("Perim   " + s2.getPerimeter());
    System.out.println("Doubled " + s2.getDoubled());
    Square[] sqs = {s1, s2, new Square(3)};
    System.out.println("Total area: " +
                       Square.getTotalArea(sqs));
    System.out.println("Greatest  : " +
                       Square.getMax(sqs));
}
```

should print

```
Square[2.0] Square[1.0]
```

```
Side    2.0
Area    4.0
Diag    2.8284271247461903
Perim   8.0
Doubled Square[4.0]
Total area: 14.0
Greatest  : Square[3.0]
```

You can comment out the contents of the **main** function and then uncomment line by line while you add methods.

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 two classes:

- **Point** describing points on the Cartesian plane
  - with fields x and y of type **int** (coordinates of the point)
  - and one static method **getPoint** returning an object of the class based on two arguments passed to it (such method plays the rôle of a constructor);
- **Rect** describing rectangles on the Cartesian plane with sides paralel to the axes.
  - It has two fields of type **Point**: ul (upper-left vertex) and br (bottom-right);
  - and a static method **getRect** which returns an object of the class given one point (upper-left vertex) and width and hight of the rectangle;
  - static method **getContainingRect** which accepts an array of points and returns an object of the class corresponding to the smallest rectangle containing all points from the array;
  - and a non-static method **showInfo** printing information on the object.

For example, the following program (classes should be in separate files!)

download *PointsAndRects.java*

```
public class Point {
    int x, y;
    static Point getPoint(int x, int y) {
        // ...
    }
}

public class Rect {
    Point ul;
```

```
    Point br;
    static Rect getRect(Point ull, int w, int h) {
        // ...
    }
    static Rect getContainingRect(Point[] arr) {
        // ...
    }
    void showInfo() {
        // ...
    }
}

public class PointsAndRects {
    public static void main(String[] args) {
        Rect rec = Rect.getRect(Point.getPoint(2, 6), 6, 4);
        rec.showInfo();

        Point[] points = {
            Point.getPoint(3, 4), Point.getPoint(5, 6),
            Point.getPoint(1, 3), Point.getPoint(5, 3),
            Point.getPoint(4, 1), Point.getPoint(3, 7)
        };
        Rect cont = Rect.getContainingRect(points);
        cont.showInfo();
    }
}
```

should print

```
UL=(2, 6) BR=(8, 2)
UL=(1, 7) BR=(5, 1)
```

Problem 4

Create class **Person** with fields `name` of type **String** and `age` of type **int**. Define the constructor, getters and override the **toString** method (or, if you don't know what's that, write a method **info** which just prints information on the object it's been invoked on). Also, define a non-static method **compareTo** which takes the reference to another object of the same class and returns an **int** with value

- negative (otherwise arbitrary), if the object on which the method has been invoked (the *receiver*) corresponds to the younger person;
- zero, if both objects correspond to persons of the same age;
- positive, if the person passed to the function as the argument is younger.

In class **Person** add the static function **sort** which sorts (using, for example, the selection sort algorithm.) an array of references to **Person**s in ascending order according to their age; use the **compareTo** method to compare persons.