[GUI] Ćwiczenia II:

```
interface In1 {
    // publiczne metody abstrakcyjne
    public void metoda1();
    public String method1();

    // publiczna statyczna stała
    public static final int STALA = 1;
}
```

```
interface In2 {
    // (publiczne) metody abstrakcyjne
    void metoda2();
    String method2();

    // (publiczna) metoda domyślna
    default void methDef2() { System.out.println("Default method"); }

    // (publiczna) statyczna metoda
    static void meth2() {}
}
```

```
class K implements In1, In2 {

    @Override
    public void metoda1() { System.out.println("Metoda1");}

    @Override
    public String method1() { return "method1"; }

    @Override
    public void metoda2() { System.out.println("Metoda2");}

    @Override
    public String method2() { return "method2";}

//  @Override
//  public void methDef2() { System.out.println("Default method 2");}

}
```

```
class K2 extends K {
    public void metoda() { System.out.println("Metoda");}
}
```

```
public class gui_note_03_01 {

    public static void main(String[] args) {

        In1 o = new K();
        o.metoda1();            // "Metoda1"
        //o.metoda2();          // Błąd

        In2 ob = new K();
        ob.metoda2();           // "Metoda2"
        ob.methDef2();          // "Default method"

        //K2 obi = new K();     // Błąd
        K obi = new K2();       // OK

        //obi.metoda();         // Błąd
        ((K2)obi).metoda(); // "Metoda

        System.out.println(ob instanceof In2);          // true

        System.out.println(ob instanceof K);            // true
        System.out.println(ob instanceof K2);           // false

        System.out.println(obi instanceof K);           // true
        System.out.println(obi instanceof K2);          // true

        System.out.println(ob.getClass().getName());    // "K"
        System.out.println(obi.getClass().getName());   // "K2"

    }
}
```

```
pulic class FlySpeakTest {

    public static void main(String[] args) {

        Flyable f[] = {new Bird("sparrow"), new Bird("eagle"), null};
        f[2] = Flyable.hybryd(f[0], f[1]);

        Speakable s[] = {
                            new Bird("seagull"),
                            () -> "parrot",
                            new Speakable() {
                                @Override
                                public String speak() {
                                    return "aaaaa";
                                }
                            }, () -> "something else"
        };
        //...
```

```
        System.out.println(Flyable.shortest(f));
        System.out.println(Speakable.loudest(s));
    }
}
```

```
interface Flyable {

    double distance();
    String drive();

    static Flyable hybryd(Flyable f1, Flyable f2) {

        return new Flyable() {

            @Override
            public String drive() {
                    return f1.drive() + f2.drive();
            }

            @Override
            public double distance() {
                return ...;
            }

            @Override
            public  String toString() {
                return ...;
            }
        };
    }

     static Flyable shortest(Flyable[] f) {
         // ...
     }
}
```

```
interface Speakable {

    String speak();

    static Speakable loudest(Speakable[] s) {
        Speakable loudest = s[0];
        for (Speakable sp: s) {
            if (sp.speak().length() > loudest.speak().length())
                loudest = sp;
        }
        return loudest;
```

```
    }
}
```

```
class Bird implements Flyable, Speakable{

    private String name;

    Bird(String name) {
        this.name = name;
    }
    //...

    @Override
    public String speak() {
        return "...";
    }

    @Override
    public double distance() {
        return 500;
    }

    @Override
    public String drive() {
        return "Wings";
    }

    @Override
    public  String toString() {
        return "...";
    }
}
```

```
interface Obliczenie {
    double pole();
    double obwod();
}
```

```
abstract class Figura implements Obliczenie {

    protected int x, y;

    // konstruktor
    public Figura(int x, int y)
    {
        //...
    }

    // metody abstrakcyjne
```

```
    public abstract String fig();
    public abstract void pozycja(int x, int y);

    @Override
    public String toString()
    {
        //...
    }

    //...
}
```

```
class Kolo extends Figura {

    private int promien;

        // konstruktor
    public Kolo(int x, int y, int r)
    {
        //...
    }

    @Override
    public String fig() {
        return "Koło";
    }

    @Override
    public void pozycja(int x, int y)
    {
        //...
    }

    @Override
    public double pole()
    {
        //...
    }

    @Override
    public double obwod()
    {
        //...
    }

    @Override
    public String toString()
    {
        //...
    }
```

```
    //...
}
```

```
class Prostokat extends Figura{

    protected int szer, wys;

        // konstruktor
    public Prostokat(int x, int y, int s, int w)
    {
        //...
    }

    @Override
    public String fig() {
        return "Prostokat";
    }

    @Override
    public void pozycja(int x, int y)
    {
        //...
    }

    @Override
    public double pole()
    {
        //...
    }

    @Override
    public double obwod()
    {
        //...
    }

    @Override
    public String toString()
    {
        return super.toString() + "\nLewy górny - (" + x + ',' + y + ")" + "\nSz
erokość: " + szer + ", " + "Wysokość: " + wys + "\n";
    }
}
```