

* Single Responsibility - принцип единственной ответственности.

Каждый модуль выполняет свою задачу.

Если есть большой класс который выполняет много разных задач - его следует разбить на разные классы, которые можно будет потом переиспользовать. (модульность)

```
class Util {  
    public void PhysicsUtil() {};  
    public void MathUtil() {};  
    public void NetworkUtil() {};  
    //...  
    //...  
    //...  
}
```


```
class Math {  
    public void PhysicsUtil() {};  
}  
  
class Physics {  
    public void MathUtil() {};  
}  
  
class Networking {  
    public void NetworkUtil() {};  
}
```

* Open/Close Principle - принцип открытости/закрытости.

программные сущности должны быть открыты для расширения, но закрыты для модификации.

Новый функционал в классах появляется только за счет наследования, но не за счет изменений в классе.

```
class Display {  
  
}
```



```
class _4kDisplay extends Display {  
  
}  
  
class _144hzDisplay extends Display {  
  
}
```

* Liskov's Substitution Principle - принцип подстановки Лисков.

Наследующий класс должен дополнять, а не заменять поведение.

Если есть класс Б наследующий класс А то в коде они должны быть взаимозаменяемы.

```
class A {
```

```
}
```

```
class B extends A {
```

```
    // должен дополнять, а не заменять
```

```
}
```

```
A a = new A();
```

```
//use(a);  
//process(a);  
//....
```

```
A a = new A();  
B b = new B();
```

```
//use(b);  
//process(b);  
//....
```

* Interface Segregation - принцип разделения интерфейса.

Интерфейсы не должны быть громоздкими.

Для каждой сущности которая пользуется интрфейсом, должны быть только нужные ей методы.

```
interface IVehicle {  
    public void drive();  
    public void fly();  
    // ...  
    // ...  
    // ...  
}
```

```
class Car implements IVehicle {  
    public void drive() {};  
    public void fly() {}; //wtf  
}
```

```
interface ICar {  
    public void drive();  
}
```


```
interface IPlane {  
    public void fly();  
}
```

```
class Car implements ICar {  
    public void drive() {};  
}
```

```
class Plane implements IPlane {  
    public void fly() {};  
}
```

* Dependency Inversion - принцип инверсии зависимостей.
Объекты должны зависеть от абстракций, а не от реализаций.
Взаимодействие классов должно идти через интерфейс.

```
class A {  
    B b;  
}  
  
class B {  
}
```



```
interface Ib {  
    //  
}  
  
class B implements Ib {  
    //  
}  
  
class A {  
    Ib b;  
}
```