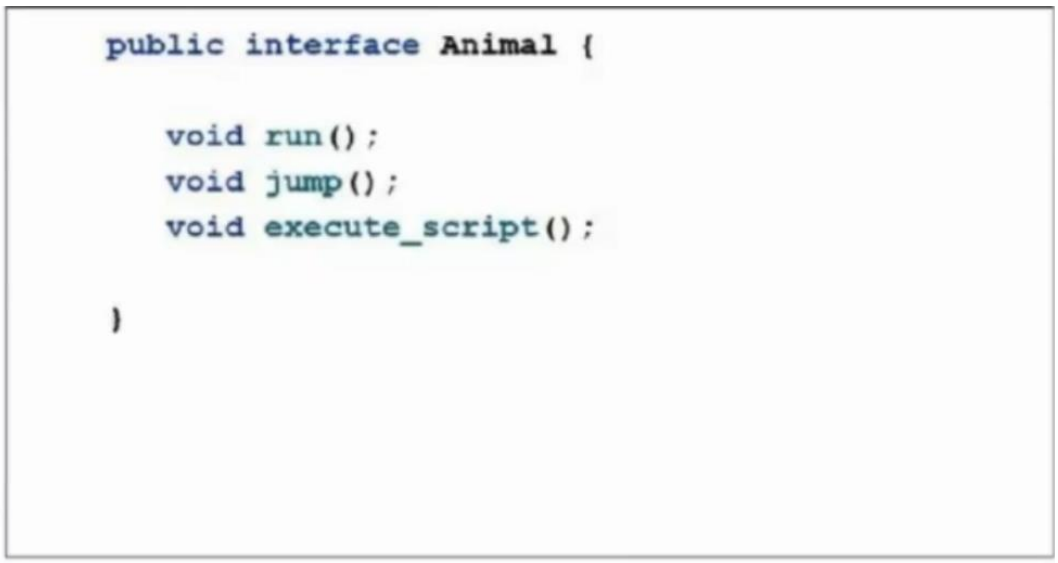


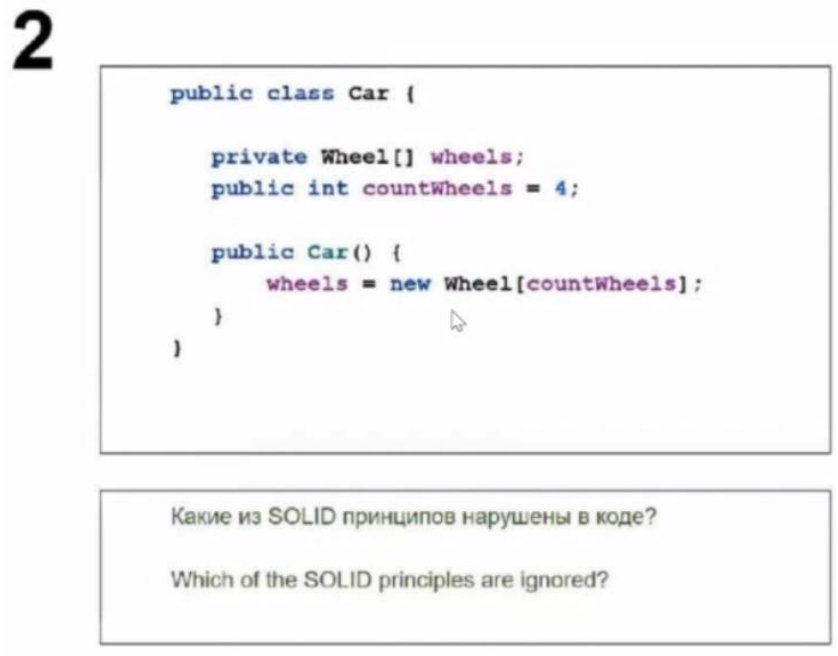
```
public interface Animal {

    void run();
    void jump();
    void execute_script();

}
```

Какие из SOLID принципов нарушены в коде?

Which of the SOLID principles are ignored?

Нарушен 4 принцип SOLID – принцип **Принцип разделения интерфейса** (interface segregation principle / ISP) в формулировке Роберта Мартина: «клиенты не должны зависеть от методов, которые они не используют». Принцип разделения интерфейсов говорит о том, что слишком «толстые» интерфейсы необходимо разделять на более маленькие и специфические, чтобы клиенты маленьких интерфейсов знали только о методах, которые необходимы им в работе. В итоге, при изменении метода интерфейса не должны меняться клиенты, которые этот метод не используют.

2



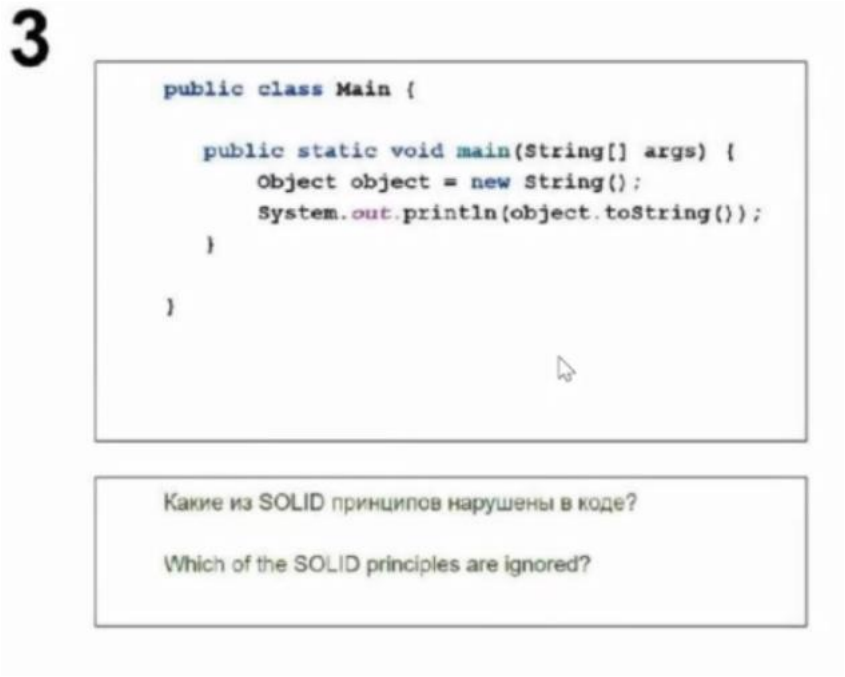
```
public class Car {

    private Wheel[] wheels;
    public int countWheels = 4;

    public Car() {
        wheels = new Wheel[countWheels];
    }
}
```

Какие из SOLID принципов нарушены в коде?

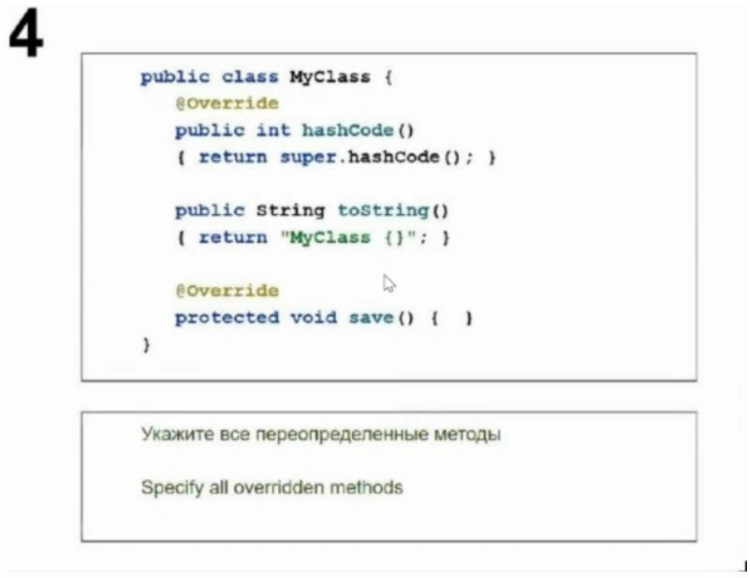
Which of the SOLID principles are ignored?

Здесь нарушено 2 принципа SOLID:

1. Принцип инверсии зависимостей (dependency inversion principle / DIP) — модули верхних уровней не должны зависеть от модулей нижних уровней, а оба типа модулей должны зависеть от абстракций; сами абстракции не должны зависеть от деталей, а вот детали должны зависеть от абстракций.
2. Принцип открытости / закрытости (open-closed principle / OCP) декларирует, что программные сущности (классы, модули, функции и т. п.) должны быть открыты для расширения, но закрыты для изменения. Это означает, что эти сущности могут менять свое поведение без изменения их исходного кода.



Здесь нарушен 3 принцип SOLID - Принцип подстановки Барбары Лисков (Liskov substitution principle / LSP) в формулировке Роберта Мартина: «функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа не зная об этом».



Переопределены hashCode и toString. Потому что MyClass наследуется от Object, а в нем нет метода save. Аннотации в данном случае для программиста.

5

```
public class Animal {

    void waiting() {}

    public class Head {
        public void think() {
            //
            // code here
            //
        }
    }
}
```

Замените комментарий на строку кода, которая вызывает метод *waiting* класса *Animal*.
Replace the comment with the line of code that calls the *waiting* method of the *Animal* class.

Просто пишем:

waiting();

Потому что класс Head имеет доступ ко всем полям класса Animal

6

```
public abstract class Animal {

    abstract void jump();

}
```

Напишите код, создающий экземпляр анонимного класса, унаследованного от *Animal*.
Write code that creates an instance of an anonymous class inherited from *Animal*.

```
public abstract class Animal {
    abstract void jump();
    Animal animal = new Animal() {
        void jump() {
            System.out.println("Jump!");
        }
    };
}
```

7

```
1. public class FailThisTest
2.         extends RuntimeException {}
3.
4. class Lesson {
5.     public void doLesson() throws FailThisTest {
6.         try {
7.             throw new FailThisTest();
8.         } catch (Throwable e) {
9.             // some code
10.        } finally { /* some code */ }
11.    }
12. }
```

Укажите все ошибки в коде

Specify all mistakes in this code

2 ошибки:

1. Исключение и обрабатывается, и пробрасывается
2. В блоке catch Throwable e что неправильно. Правильно было бы сузить ошибку (например написать FailThisTest e)

3

```
class Thrower {

    public static void throwException(X x) {
        throw x;
    }

}
```

Укажите максимально абстрактный допустимый тип данных X

Specify the maximum abstract allowed data type of X

Это тип Throwable

9

Укажите все свойства проверяемых исключений

Specify all properties of checked exceptions

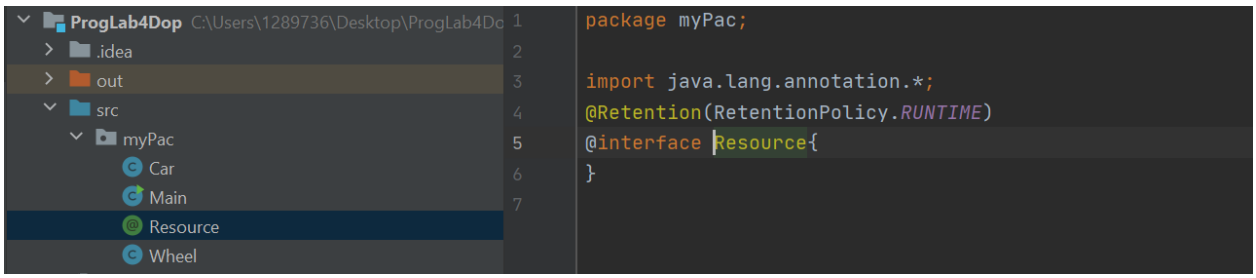
Вроде давали варианты ответов.

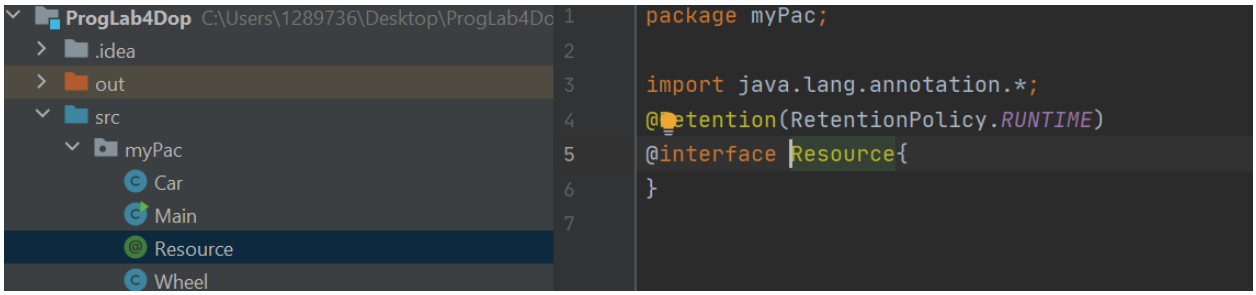
1. И можно кидать
2. Они наследуются от RuntimeException
3. Его нужно либо прокинуть либо обернуть в try-catch-finally

10

```
public class Car {

    @Resource
    private Wheel[] wheels;
    public int countWheels = 4;

    // another fields there
}
// Class.getDeclaredFields()
// Field.getAnnotation(...)
// Field.getName()
```

Напишите код, который выводит имена всех полей класса, которые отмечены аннотацией *@Resource*
Write code that prints the names of all fields in a class that are annotated with *@Resource*





В Main написан ответ. Остальное – для понимания как вообще рефлексия работает.

Из полезного:

Принципы SOLID:

Принцип единственной обязанности / ответственности (single responsibility principle / SRP) обозначает, что каждый объект должен иметь одну обязанность и эта обязанность должна быть полностью инкапсулирована в класс. Все его сервисы должны быть направлены исключительно на обеспечение этой обязанности.

Принцип открытости / закрытости (open-closed principle / OCP) декларирует, что программные сущности (классы, модули, функции и т. п.) должны быть открыты для расширения, но закрыты для изменения. Это означает, что эти сущности могут менять свое поведение без изменения их исходного кода.

Принцип подстановки Барбары Лисков (Liskov substitution principle / LSP) в формулировке Роберта Мартина: «функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа не зная об этом».

Принцип разделения интерфейса (interface segregation principle / ISP) в формулировке Роберта Мартина: «клиенты не должны зависеть от методов, которые они не используют». Принцип разделения интерфейсов говорит о том, что слишком «толстые» интерфейсы необходимо разделять на более маленькие и специфические, чтобы клиенты маленьких интерфейсов знали только о методах, которые необходимы им в работе. В итоге, при изменении метода интерфейса не должны меняться клиенты, которые этот метод не используют.

Принцип инверсии зависимостей (dependency inversion principle / DIP) — модули верхних уровней не должны зависеть от модулей нижних уровней, а оба типа модулей должны зависеть от абстракций; сами абстракции не должны зависеть от деталей, а вот детали должны зависеть от абстракций.

Структура исключений в Java: