Object-oriented programming

Lecture №4

Contract, abstract, interface

PhD, Alexander Vlasov

Question?

- Object
- Class
- Inheritance
- Parameterized Class
- Generics

Contract

- Контракт – совокупность соглашений о функциональности элемента программы (модуля, класса и т.д.).
- Контракт определяет обязательства 2-х сторон: Сервер предоставляет некоторую функциональность, гарантирует выполнение определенных правил, а также требует выполнения определенных правил клиентом.
- Сервер обязуется возвратить корректный результат, если клиент передаст ему корректные параметры.
- Понятие контракта введено в контрактном программировании(Б. Мейер) в середине 80-х. Цель КП – объединить ООП и формальные методы доказательства корректности программ(в первую очередь, на основе логики Хоара)

Contract

В контракт входят:

- набор операций
- параметры операций, область допустимых значений, интерпретация
- допустимые возвращаемые значения, их интерпретация
- сообщения об ошибках (исключения и т.д.)
- предусловия (напр. перед выполнением любой операции сервер должен быть инициализирован)
- постусловия (напр. после выполнения операции init можно осуществлять другие операции);
- инварианты (напр. имя пользователя всегда уникально);
- побочные эффекты (изменение глобального состояния, базы данных и т.д.); дополнительные гарантии (напр. производительность).

interface

```
interface Vehicle {
    // all are the abstract methods.
    void changeGear(int a);
    void speedUp(int a);
    void applyBrakes(int a);
}
```

Implements interface

```
class Bicycle implements Vehicle{
   int speed;
   int gear;
    // to change gear
   @Override
   public void changeGear(int newGear){
      gear = newGear;
   }
   // to increase speed
   @Override
   public void speedUp(int increment){
      speed = speed + increment;
   }
   // to decrease speed
   @Override
   public void applyBrakes(int decrement){
      speed = speed - decrement;
   }
   public void printStates() {
       System.out.println("speed: " + speed
           + " gear: " + gear);
   }
}
```

```
class Bike implements Vehicle {
   int speed;
   int gear;
   // to change gear
   @Override
   public void changeGear(int newGear){
      gear = newGear;
   }
   // to increase speed
   @Override
   public void speedUp(int increment){
      speed = speed + increment;
   }
   // to decrease speed
   @Override
   public void applyBrakes(int decrement){
      speed = speed - decrement;
   }
   public void printStates() {
       System.out.println("speed: " + speed
           + " gear: " + gear);
   }
}
```

Use interface

```
class Main {

    public static void main (String[] args) {
        var vehicles=new Vehicle[2];
        vehicles[0] = new Bicycle();
        vehicles[1] = new Bike();
        for (var v : vehicles ){
                v.changeGear(2);
                v.speedUp(3);
                v.applyBrakes(1);
                v.printStates();
        }
    }
}
```

Key word «default»

```
// An example to show that interfaces can
// have methods from JDK 1.8 onwards
interface in1
{
    final int a = 10;
    default void display()
    {
        System.out.println("hello");
    }
}
// A class that implements interface.
class testClass implements in1
{
    // Driver Code
    public static void main (String[] args)
    {
        testClass t = new testClass();
        t.display();
    }
}
```

Implementation of multiple interfaces

```
interface One { void doIt();}

interface Two { void doIt(); void doAnother();}

class MyClass implements One, Two {
    void doIt() {} //Joint realization for One и Two
    void doAnother{}
}
```

key word «abstract»

```
public abstract class Animal
{
    public void eat(Food food)
    {
      // do something with food....
    }
    public void sleep(int hours)
    {
      try
      {
          // 1000 milliseconds * 60 seconds * 60
          // minutes * hours
          Thread.sleep ( 1000 * 60 * 60 * hours);
       }
       catch (InterruptedException ie) { /* ignore */ }
     }
     public abstract void makeNoise();
}
```

```
public Dog extends Animal
{
    public void makeNoise() {
        System.out.println ("Bark! Bark!");
    }
}
public Cow extends Animal
{
     public void makeNoise() {
         System.out.println ("Moo! Moo!");
     }
}
```

abstract

```
class Main {
    public static void main (String[] args) {
        var animals=new Animal[2];
        animals [0] = new Dog();
        animals [1] = new Cow();
        for (var a : animals ){
                a.sleep(5);
                a.makeNoise();
        }
    }
}
```

Difference between Class and Interface

Class	Interface
In class, you can instantiate variable and create an object.	In an interface, you can't instantiate variable and create an object.
Class can contain concrete(with implementation) methods	The interface cannot contain concrete(with implementation) methods
The access specifiers used with classes are private, protected and public.	In Interface only one specifier is used- Public.

Must know facts about Interface

- A Java class can implement multiple Java Interfaces. It is necessary that the class must implement all the methods declared in the interfaces.
- Class should override all the abstract methods declared in the interface
- The interface allows sending a message to an object without concerning which classes it belongs.
- Class needs to provide functionality for the methods declared in the interface.
- All methods in an interface are implicitly public and abstract
- An interface cannot be instantiated
- An interface reference can point to objects of its implementing classes
- An interface can extend from one or many interfaces. Class can extend only one class but implement any number of interfaces
- An interface cannot implement another Interface. It has to extend another interface if needed.
- An interface which is declared inside another interface is referred as nested interface
- At the time of declaration, interface variable must be initialized. Otherwise, the compiler will throw an error.
- The class cannot implement two interfaces in java that have methods with same name but different return type.

Liskov Substitution Principle, ***LSP***

- *Функции, которые используют ссылки на базовые классы, должны иметь возможность использовать объекты производных классов, не зная об этом.*
- **Derived classes must be substitutable for their base classes**

Original:

> **If for each object o1 of type S there is an object o2 of type T such that for all programs P defined in terms of T, the behavior of P is unchanged when o1 is substituted for o2, then S is a subtype of T**

Нарушение LSP

```
public class DoubleList<T> : IList<T>
{
    private readonly IList<T> innerList = new List<T>();
    public void Add(T item)
    {
        innerList.Add(item);
        innerList.Add(item);
    }
    ...
```

```
public void CheckBehaviourForRegularList()
{
    IList<int> list = new List<int>();

    list.Add(1);

    Assert.Equal(1, list.Count);
}
```

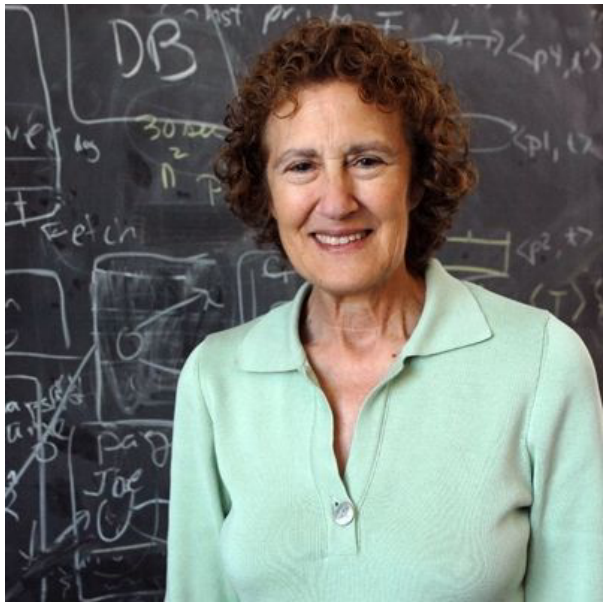
Нарушение LSP = дефект объектной модели
Что делать?

```
public void CheckBehaviourForDoubleList()
{
    IList<int> list = new DoubleList<int>();

    list.Add(1);

    Assert.Equal(1, list.Count); // fail
}
```

Barbara Liskov



Turing Award (2008) - for contributions to practical and theoretical foundations of programming language and system design, especially related to data abstraction, fault tolerance, and distributed computing.

Data Abstraction and Hierarchy, Barbara Liskov - 1988.