Принципы проектирования и дизайна ПО

Лекция №4

Агошков Илья 2016

Сценарии использования (Use Cases).
Идентификация классов/объектов и их обязанностей.
Обзор UML диаграмм.
Основы ООП.
Принципы SOLID. High cohesion, loose coupling.
Dependency Inversion Principle, Inversion of Control,
Dependency Injection
Шаблоны GoF (12-14 шаблонов).
Архитектурные стили:

- Client-server, SOA, Event sourcing, Layered Systems, Ports & Adapters (hexagonal architecture), CQRS

Монолитная архитектура и микросервисы.

В предыдущих сериях...

В предыдущих сериях...

Unit-тесты
Test driven development
Abstract classes/interfaces
Абстракция
Полиморфизм
Наследование

Инкапсуляция

Параллельные массивы

Структуры

Объекты

Инкапсуляция

```
String[] firstname = new String[100];
String[] lastname = new String[100];
String[] paternity = new String[100];
int personsCount = 0;

public static void main(String[] args) {
   addPerson('Иван', 'Иванов', 'Иванович')
}

public void addPerson(String firstName, String lastName, String paternity) {
   firstname[personsCount] = firstName;
   lastname[personsCount] = lastName;
   paternity[personsCount] = paternity;
   personsCount++;
}
```

Инкапсуляция

```
struct Person {
   String firstName;
   String lastName;
   String paternity;
}

Person[] persons = new Person[100];
```

Инкапсуляция

```
public class Person {
  public String firstName;
  public String lastName;
  public String paternity;

  public Person(String firstName, String lastName, String paternity) {
     this.firstName = firstName;
     this.lastName = lastName;
     this.paternity = paternity;
  }
}
```

Инкапсуляция

```
public Person(String firstName, String lastName, String paternity) {
    checkNotNull(firstName);
    checkNotNull(lastName);
    checkNotNull(paternity);
    this.firstName = firstName;
    this.lastName = lastName;
    this.paternity = paternity;
}
```

Инкапсуляция

```
public class Person {
  private String firstName;
  private String lastName;
  private String paternity;

  ...
}
```

Инкапсуляция

```
...
public String getFirstName() {
    return firstName;
}

public void setFirstName(String firstName) {
    this.firstName = firstName;
}
...
```

Инкапсуляция

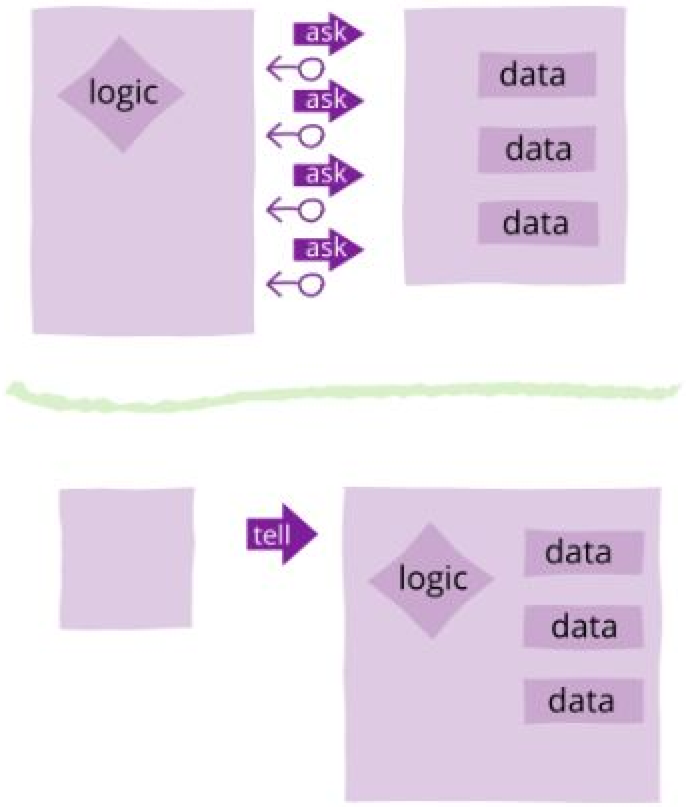
```
...
public String getFirstName() {
    return firstName;
}

public void setFirstName(String firstName) {
    checkNotNull(firstName);
    this.firstName = firstName;
}
...
```

Инкапсуляция

```
public PdfReport buildReport(Collection<Person> employees) {
  PdfReport report = new PdfReport();
  for (Person employee : employees) {
     String shortName = employee.getLastName() + " " +
           employee.getFirstName().charAt(0) + ". " +
           employee.getPaternity().charAt(0) + ". ";
     report.addLine(shortName);
  }
  return report;
}
```

Tell don’t ask http://martinfowler.com/bliki/TellDontAsk.html



Инкапсуляция

```
public PdfReport buildReport(Collection<Person> employees) {
  PdfReport report = new PdfReport();
  for (Person employee : employees) {
     report.addLine(employee.getShortName());
  }
  return report;
}
```

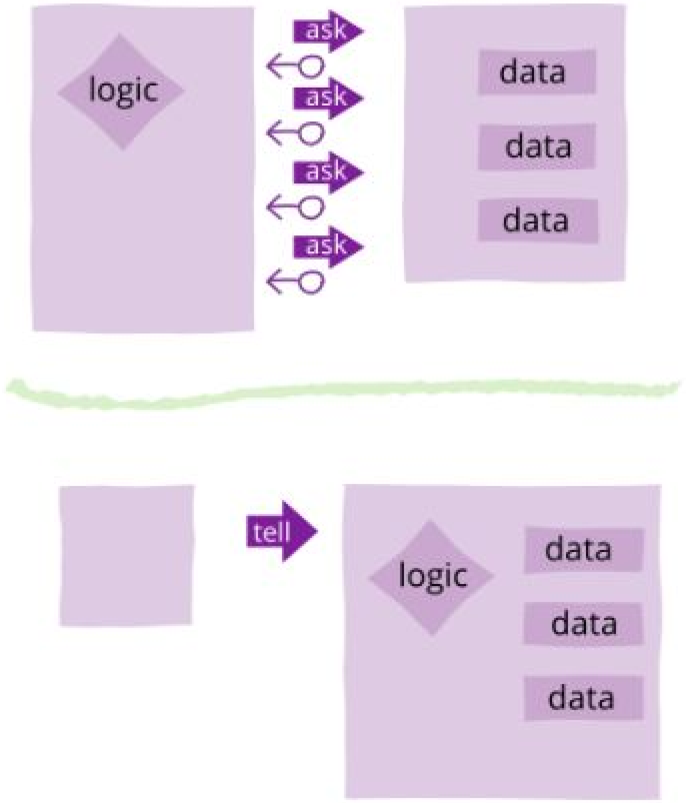
Инкапсуляция

```
public String getShortName() {
  return lastName + " " +
        firstName.charAt(0) + ". " +
        paternity.charAt(0) + ". ";
}
```

Инкапсуляция

```
public String getShortName() {
  return lastName + " " +
          firstName.charAt(0) + ". " +
          (paternity == null ? "" : paternity.charAt(0) + ". ");
}
```

Инкапсуляция



Инкапсуляция

```
public class Robot
{
  private double x = 0;
  private double y = 0;
  private double course = 0; // degrees

  public Robot() {}
  public Robot(double x, double y) {
     this.x = x;
     this.y = y;
  }
  public void forward(int distance) {
     x = x + distance * Math.cos(course / 180 * Math.PI);
     y = y + distance * Math.sin(course / 180 * Math.PI);
  }
```

Инкапсуляция

```
private double maxWalkingDistance = 50;
```

Инкапсуляция

```
private double maxWalkingDistance = 50;

public void forward(int distance) {
   if (distance > maxWalkingDistance)
       throw new IllegalArgumentException("This robot can walk "
               + maxWalkingDistance + " meters maximum at once.");
   x = x + distance * Math.cos(course / 180 * Math.PI);
   y = y + distance * Math.sin(course / 180 * Math.PI);
}
```

Инкапсуляция

```
private double fuel = 1500;
```

Инкапсуляция

```
public void forward(int distance) {
  if (distance > maxWalkingDistance)
     throw new IllegalArgumentException("This robot can walk "
           + maxWalkingDistance + " meters maximum at once.");
  if (fuel < minFuel)
     throw new IllegalArgumentException("This robot has only "
           + fuel + "l of fuel left. But at least"
           + minFuel + "l is required to make a move.");
  x = x + distance * Math.cos(course / 180 * Math.PI);
  y = y + distance * Math.sin(course / 180 * Math.PI);
  fuel -= 100;
}
```

Инкапсуляция

Implementation Hiding

This is what encapsulation is all about: exposing a solution to a problem without requiring the consumer to fully understand the problem domain.

Protection of Invariants

Hiding the internals of the object protects its integrity by preventing users from setting the internal data of the component into an invalid or inconsistent state.

Инкапсуляция

Примеры

HashMap - внутренние оптимизации в новых версиях Java.

ArrayList, HashSet - итерация без необходимости знать внутреннее устройство.

Инкапсуляция

- По-умолчанию, делайте все поля **private** (и желательно **final**).
- Не делайте set-методы, пока они действительно не нужны.
- Не делайте get-методы, пока они действительно не нужны.
- Делайте конструкторы, проверяющие корректность входных параметров