КАЖДЫЙ JAVA-РАЗРАБОТЧИК ДЕЛАЛ ЭТО...

Григорий Кошелев

Контур

SnowOne 2023

О докладчике

с 2011
промышленная разработка на Java

с 2013
Проведение стажировок на Java

с 2014
Контур

с 2017
преподавание курсов по ООП и Java

с 2021
функциональный руководитель Java

О чём доклад

Доклад про

- **практики написания кода**
- **распространённые ошибки начинающих Java-разработчиков**

О чём доклад

Доклад про

- **практики написания кода**
- **распространённые ошибки начинающих Java-разработчиков**

Контекст (перформанс)

- **Java 17, Liberica OpenJDK**
- **x64, Apple M1**

VS

VS

Недостаток знаний

- «Да, в проекте уже так было...»

VS

Недостаток знаний

- «Да, в проекте уже так было...»

Избыток знаний

- «А вот, Шипилёв говорил, что...»

VS

Недостаток знаний

- «Да, в проекте уже так было...»

Избыток знаний

- «А вот, Шипилёв говорил, что...»

- «Нет времени разбираться, надо код писать»

0. == *VS* equals

0. == VS equals

Сколько строк нужно, чтобы определить C#-разработчика?

0. == VS equals

Сколько строк нужно, чтобы определить C#-разработчика?

"Hello, world!" == myString

1. Комментарии VS Самодокументированный код

1. Комментарии VS Самодокументированный код

«Прекратите усердствовать
с комментариями в коде»

Brian Norlander

https://habr.com/ru/company/piter/blog/458990/

1. Комментарии VS Самодокументированный код

«Прекратите усердствовать
с комментариями в коде»

Brian Norlander

«Самодокументируемый код
- это (как правило) чушь»

Christopher Laine

https://habr.com/ru/company/piter/blog/460725/

1. Комментарии VS Самодокументированный код

```
public static double quadraticMean(double... values) {
   int n = values.length;// Value count
   if (n == 0) {// If values is empty, then throw an exception
      throw new IllegalArgumentException("Need at least 1 value to calculate quadratic mean");
   }
   double s = 0;// Init sum with 0
   for (double v : values) {// Iterate over the values
      s += v * v;// Add squared value to the sum
   }
   s = Math.sqrt(s / n);// Divide the sum by count  and calculate the square root
   return s; // Return the average
}
```

1. Комментарии VS Самодокументированный код

```
public static double quadraticMean(double... values) {
  int n = values.length;// Value count
  if (n == 0) {// If values is empty, then throw an exception
    throw new IllegalArgumentException("Need at least 1 value to calculate quadratic mean");
  }
  double s = 0;// Init sum with 0
  for (double v : values) {// Iterate over the values
    s += v * v;// Add squared value to the sum
  }
  s = Math.sqrt(s / n);// Divide the sum by count  and calculate the square root
  return s; // Return the average
}
```

1. Комментарии VS Самодокументированный код

```
public static double quadraticMean(double... values) {
    int n = values.length;// Value count
    if (n == 0) {// If values is empty, then throw an exception
        throw new IllegalArgumentException("Need at least 1 value to calculate quadratic mean");
    }
    double s = 0;// Init sum with 0
    for (double v : values) {// Iterate over the values
        s += v * v;// Add squared value to the sum
    }
    s = Math.sqrt(s / n);// Divide the sum by count  and calculate the square root
    return s; // Return the average
}
```

1. Комментарии VS Самодокументированный код

```
public static double quadraticMean(double... values) {
    int n = values.length;// Value count
    if (n == 0) {// If values is empty, then throw an exception
        throw new IllegalArgumentException("Need at least 1 value to calculate quadratic mean");
    }
    double s = 0;// Init sum with 0
    for (double v : values) {// Iterate over the values
        s += v * v;// Add squared value to the sum
    }
    s = Math.sqrt(s / n);// Divide the sum by count  and calculate the square root
    return s; // Return the average
}
```

1. Комментарии VS Самодокументированный код

```
public static double quadraticMean(double... values) {
    int n = values.length;// Value count
    if (n == 0) {// If values is empty, then throw an exception
        throw new IllegalArgumentException("Need at least 1 value to calculate quadratic mean");
    }
    double s = 0;// Init sum with 0
    for (double v : values) {// Iterate over the values
        s += v * v;// Add squared value to the sum
    }
    s = Math.sqrt(s / n);// Divide the sum by count  and calculate the square root
    return s; // Return the average
}
```

1. Комментарии VS Самодокументированный код

```
public static double rootMeanSquare(double... values) {
    int valueCount = values.length;
    if (valueCount == 0) {
        throw new IllegalArgumentException("Need at least 1 value to calculate quadratic mean");
    }
    double sumOfValueSquares = 0;
    for (double value : values) {
        sumOfValueSquares += value * value;
    }
    double rootMeanSquare = Math.sqrt(sumOfValueSquares / valueCount);
    return rootMeanSquare;
}
```

1. Комментарии VS Самодокументированный код

```
(rootMeanSquare( ...values: 1, 2, 3, 4, 5)); }
```

```
ru.gnkoshelev.snowone2023._1.SelfDocumentedCo
de

public static double rootMeanSquare(
    @NotNull double... values
)
```

snowone2023

1. Комментарии VS Самодокументированный код

```
/**
 * Calculate root mean square (or quadratic mean).
 * <p>
 * Need at least 1 value to calculate quadratic mean.
 *
 * @param values non-empty array of values
 * @return quadratic mean
 * @throws IllegalArgumentException if no values provided
 * @see <a href="https://en.wikipedia.org/wiki/Root_mean_square">root mean square (wiki)</a>
 */
public static double quadraticMean(double... values) {
```

1. Комментарии VS Самодокументированный код

```
/**
 * Calculate root mean square (or quadratic mean).
 * <p>
 * Need at least 1 value to calculate quadratic mean.
 *
 * @param values non-empty array of values
 * @return quadratic mean
 * @throws IllegalArgumentException if no values provided
 * @see <a href="https://en.wikipedia.org/wiki/Root_mean_square">root mean square (wiki)</a>
 */
public static double quadraticMean(double... values) {
```

1. Комментарии VS Самодокументированный код

```
/**
 * Calculate root mean square (or quadratic mean).
 * <p>
 * Need at least 1 value to calculate quadratic mean.
 *
 * @param values non-empty array of values
 * @return quadratic mean
 * @throws IllegalArgumentException if no values provided
 * @see <a href="https://en.wikipedia.org/wiki/Root_mean_square">root mean square (wiki)</a>
 */
public static double quadraticMean(double... values) {
```

1. Комментарии VS Самодокументированный код

```
/**
 * Calculate root mean square (or quadratic mean).
 * <p>
 * Need at least 1 value to calculate quadratic mean.
 *
 * @param values non-empty array of values
 * @return quadratic mean
 * @throws IllegalArgumentException if no values provided
 * @see <a href="https://en.wikipedia.org/wiki/Root_mean_square">root mean square (wiki)</a>
 */
public static double quadraticMean(double... values) {
```

1. Комментарии VS Самодокументированный код

```
/**
 * Calculate root mean square (or quadratic mean).
 * <p>
 * Need at least 1 value to calculate quadratic mean.
 *
 * @param values non-empty array of values
 * @return quadratic mean
 * @throws IllegalArgumentException if no values provided
 * @see <a href="https://en.wikipedia.org/wiki/Root_mean_square">root mean square (wiki)</a>
 */
public static double quadraticMean(double... values) {
```

1. Комментарии VS Самодокументированный код

```
(quadraticMean( ...values: 1, 2, 3, 4, 5)); }
```

```
ru.gnkoshelev.snowone2023._1.WellDocumentedCode

public static double quadraticMean(
    @NotNull double... values
)
```

Calculate root mean square (or quadratic mean).
Need at least 1 value to calculate quadratic mean.

Params: values – non-empty array of values

Returns: quadratic mean

Throws: IllegalArgumentException – if no values provided

See Also: root mean square (wiki)

snowone2023

1. Комментарии VS Самодокументированный код

Google Java Style Guide

https://google.github.io/styleguide/javaguide.html#s7.3-javadoc-where-required

1. Комментарии VS Самодокументированный код

Google Java Style Guide

7.3 Where Javadoc is used

At the *minimum*, Javadoc is present for every `public` class, and every `public` or `protected` member of such a class, with a few exceptions noted below.

Additional Javadoc content may also be present, as explained in Section 7.3.4, Non-required Javadoc.

https://google.github.io/styleguide/javaguide.html#s7.3-javadoc-where-required

1. Комментарии VS Самодокументированный код

CheckStyle - https://checkstyle.org

1. Комментарии VS Самодокументированный код

CheckStyle - https://checkstyle.org

- Десятки «code style» проверок

https://checkstyle.org/checks.html

1. Комментарии VS Самодокументированный код

CheckStyle - https://checkstyle.org

- Десятки «code style» проверок (в том числе проверки для Java Doc)

https://checkstyle.org/config_javadoc.html

1. Комментарии VS Самодокументированный код

CheckStyle - https://checkstyle.org

- Десятки «code style» проверок (в том числе проверки для Java Doc)
- Поддержка Sun Code Conventions

https://checkstyle.org/sun_style.html

1. Комментарии VS Самодокументированный код

CheckStyle - https://checkstyle.org

- Десятки «code style» проверок (в том числе проверки для Java Doc)
- Поддержка Sun Code Conventions
- Поддержка Google Java Style Guide

https://checkstyle.org/google_style.html

1. Комментарии VS Самодокументированный код

CheckStyle - https://checkstyle.org

- Десятки «code style» проверок (в том числе проверки для Java Doc)
- Поддержка Sun Code Conventions
- Поддержка Google Java Style Guide
- Интегрируется в сборку Maven

https://maven.apache.org/plugins/maven-checkstyle-plugin/

1. Комментарии VS Самодокументированный код

CheckStyle - https://checkstyle.org

- Десятки «code style» проверок (в том числе проверки для Java Doc)
- Поддержка Sun Code Conventions
- Поддержка Google Java Style Guide
- Интегрируется в сборку Maven и Gradle

https://docs.gradle.org/current/userguide/checkstyle_plugin.html

1. Комментарии VS Самодокументированный код

CheckStyle - https://checkstyle.org

- Десятки «code style» проверок (в том числе проверки для Java Doc)
- Поддержка Sun Code Conventions
- Поддержка Google Java Style Guide
- Интегрируется в сборку Maven и Gradle
- MUST HAVE в проекте!

2. Связь VS Связь

2. Coupling VS Cohesion

2. Loose Coupling VS High Cohesion

2. Loose Coupling VS High Cohesion

Как разбить код между классами?

2. Loose Coupling VS High Cohesion

Как разбить код между классами?

- **Классы побольше**

2. Loose Coupling VS High Cohesion

Как разбить код между классами?

- Классы побольше
- Классы поменьше

2. Loose Coupling VS High Cohesion

Как разбить код между классами?

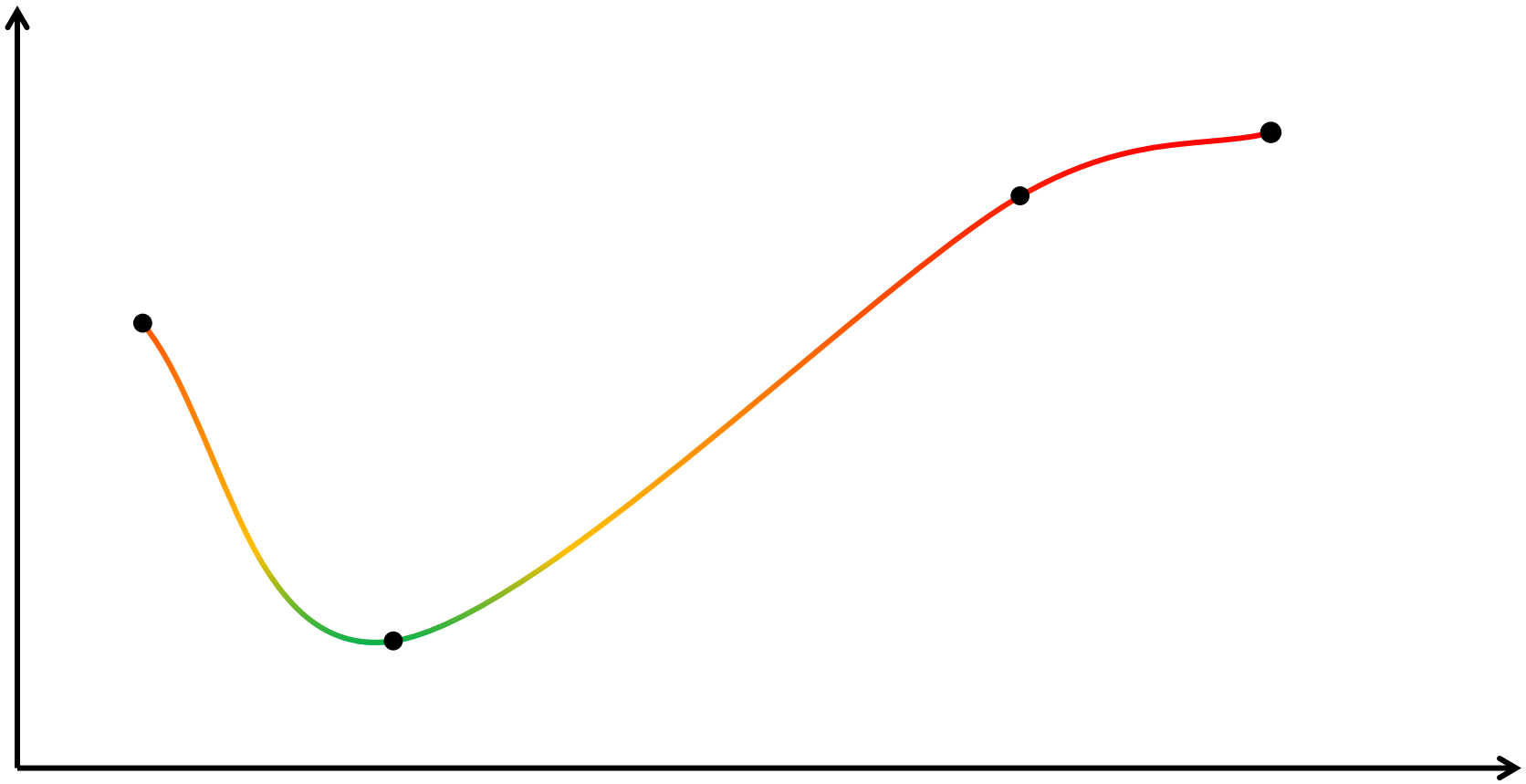
- **Классы побольше (быстро написать)**
- **Классы поменьше**

2. Loose Coupling VS High Cohesion

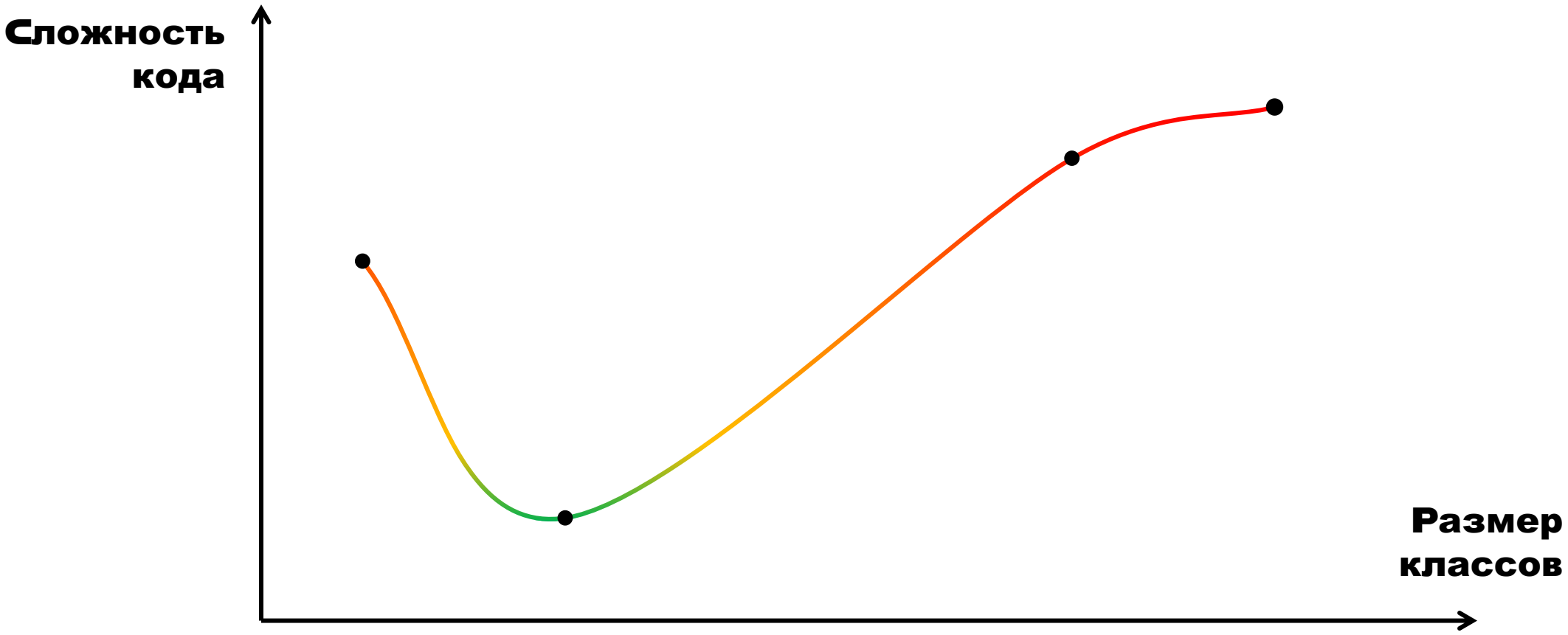
Как разбить код между классами?

- Классы побольше (быстро написать)
- Классы поменьше (SRP!)

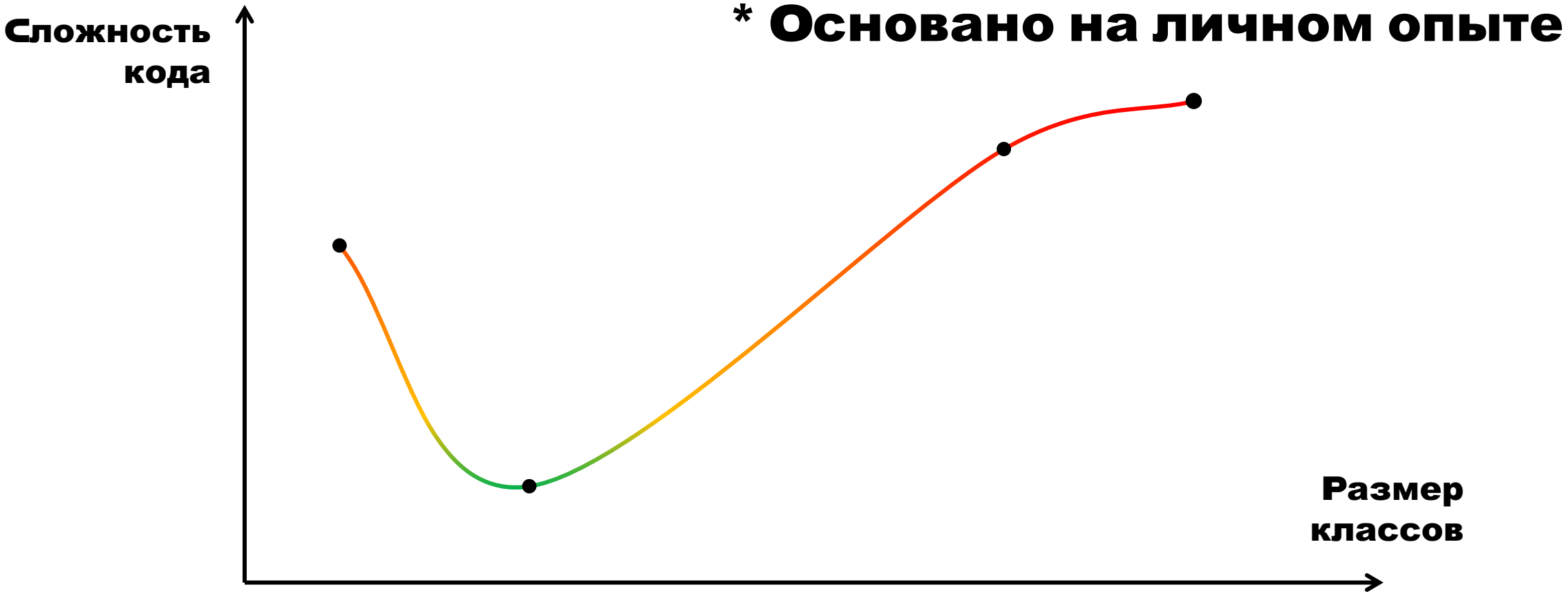
2. Loose Coupling VS High Cohesion



2. Loose Coupling VS High Cohesion



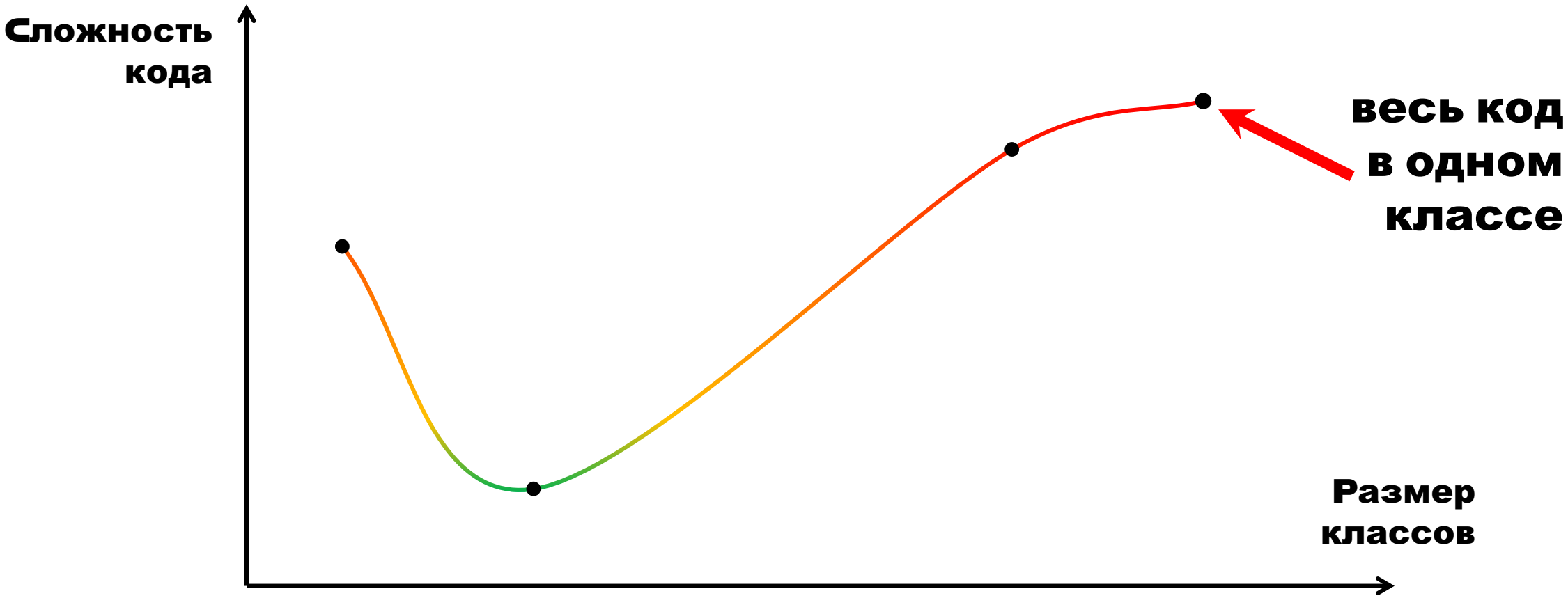
2. Loose Coupling VS High Cohesion



2. Loose Coupling VS High Cohesion

2. Loose Coupling VS High Cohesion

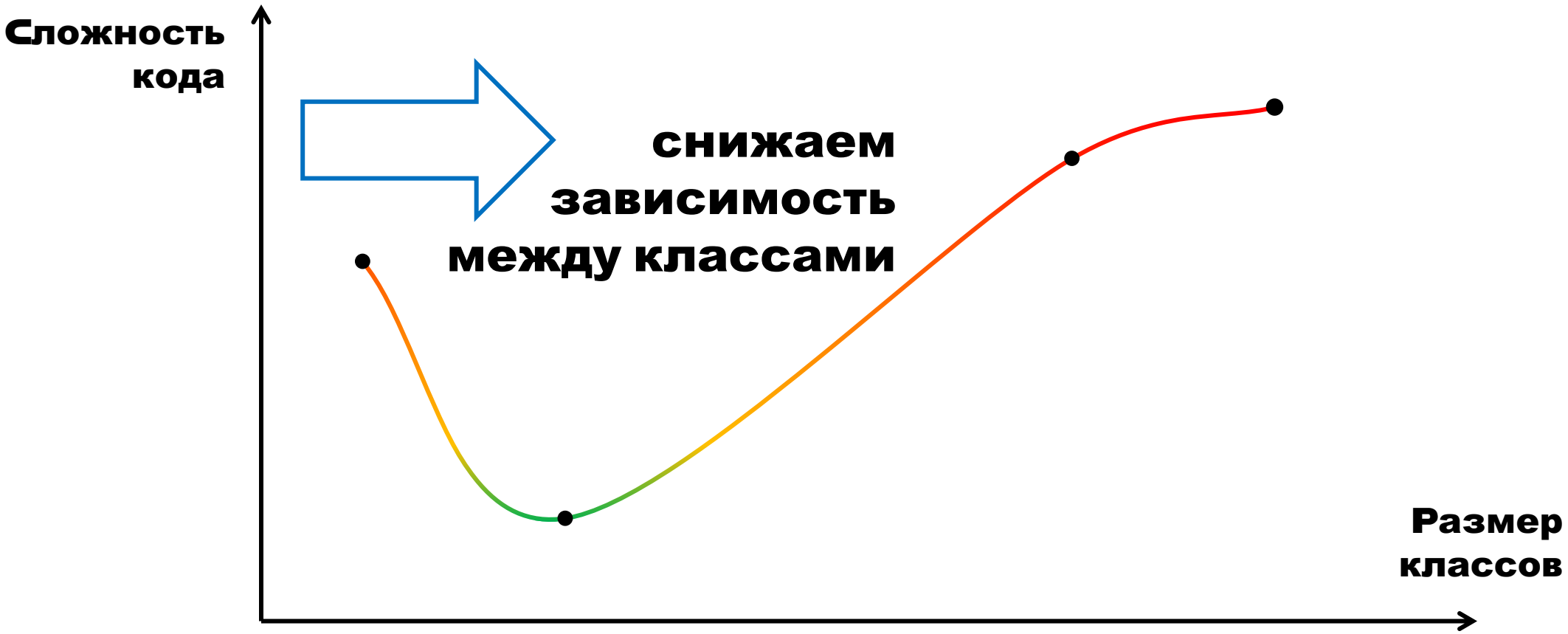
2. Loose Coupling VS High Cohesion



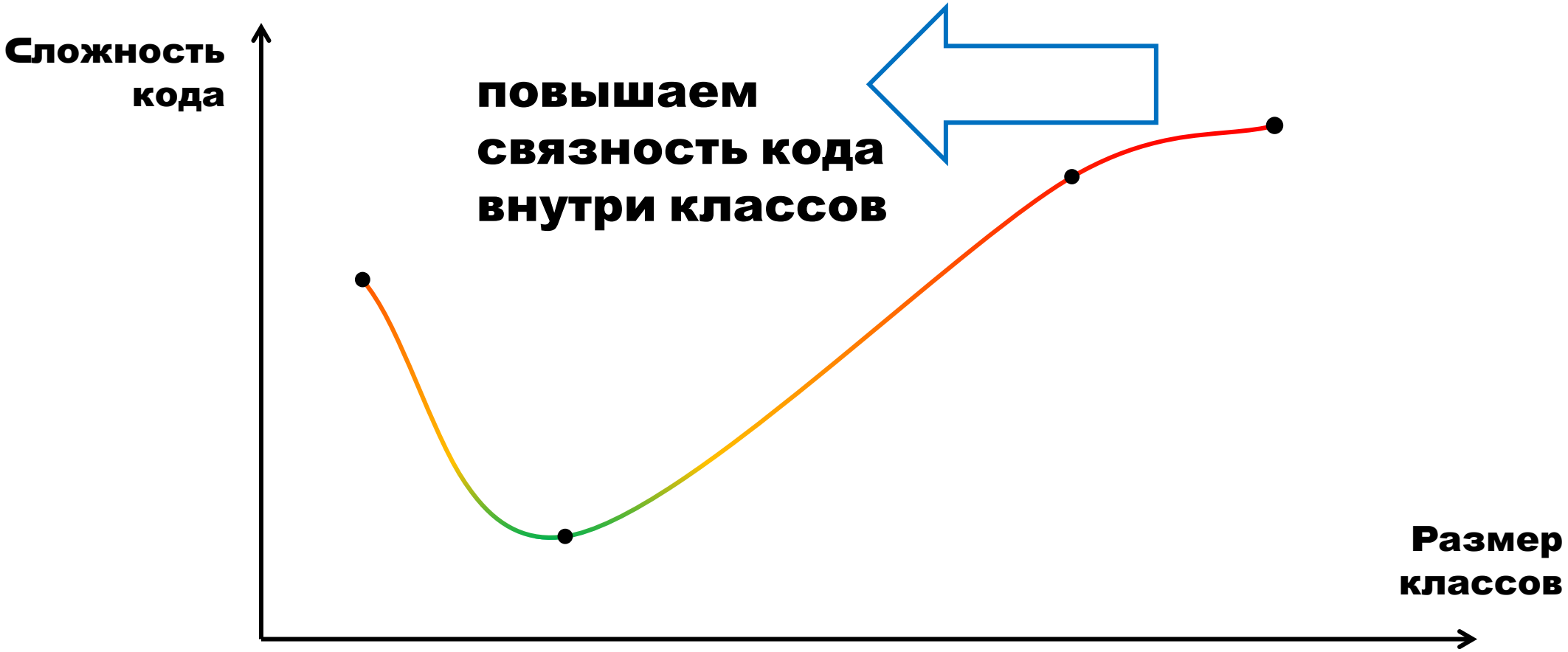
2. Loose Coupling VS High Cohesion

2. Loose Coupling VS High Cohesion

2. Loose Coupling VS High Cohesion



2. Loose Coupling VS High Cohesion



2. Loose Coupling VS High Cohesion

java.lang.System

2. Loose Coupling VS High Cohesion

java.lang.System.arraycopy

2. Loose Coupling VS High Cohesion

java.lang.System.arraycopy

- **20+ использований в java.util.Arrays**

2. Loose Coupling VS High Cohesion

java.lang.System.arraycopy

- 20+ использований в java.util.Arrays
- 0 использований в System

2. Loose Coupling VS High Cohesion

java.lang.System.arraycopy

- 20+ использований в java.util.Arrays
- 0 использований в System

НО

- arraycopy с Java 1.0
- Arrays с 1.2

2. Loose Coupling VS High Cohesion

java.lang.System.arraycopy

- 20+ использований в java.util.Arrays
- 0 использований в System

НО

- arraycopy с Java 1.0
- Arrays с 1.2

2. Loose Coupling VS High Cohesion

java.lang.Boolean

2. Loose Coupling VS High Cohesion

java.lang.Boolean.getBoolean

2. Loose Coupling VS High Cohesion

java.lang.Boolean.getBoolean

```
public static boolean getBoolean(String name) {
    boolean result = false;
    try {
        result = parseBoolean(System.getProperty(name));
    } catch (IllegalArgumentException | NullPointerException e) {
    }
    return result;
}
```

2. Loose Coupling VS High Cohesion

java.lang.Boolean.getBoolean

```
public static boolean getBoolean(String name) {
    boolean result = false;
    try {
        result = parseBoolean(System.getProperty(name));
    } catch (IllegalArgumentException | NullPointerException e) {
    }
    return result;
}
```

2. Loose Coupling VS High Cohesion

java.lang.Boolean.getBoolean

```
public static boolean getBoolean(String name) {
    boolean result = false;
    try {
        result = parseBoolean(System.getProperty(name));
    } catch (IllegalArgumentException | NullPointerException e) {
    }
    return result;
}
```

2. Loose Coupling VS High Cohesion

java.lang.Boolean.getBoolean

```
public static boolean getBoolean(String name) {
    boolean result = false;
    try {
        result = parseBoolean(System.getProperty(name));
    } catch (IllegalArgumentException | NullPointerException e) {
    }
    return result;
}
```

2. Loose Coupling VS High Cohesion

java.lang.Integer.getInteger

```
public static Integer getInteger(String nm) {
    return getInteger(nm, null);
}
```

2. Loose Coupling VS High Cohesion

java.lang.Integer.getInteger

```
public static Integer getInteger(String nm, Integer val) {
    String v = null;
    try {
        v = System.getProperty(nm);
    } catch (IllegalArgumentException | NullPointerException e) {
    }
    if (v != null) {
        try {
            return Integer.decode(v);
        } catch (NumberFormatException e) {
        }
    }
    return val;
}
```

2. Loose Coupling VS High Cohesion

java.lang.Integer.getInteger

```
public static Integer getInteger(String nm, Integer val) {
    String v = null;
    try {
        v = System.getProperty(nm);
    } catch (IllegalArgumentException | NullPointerException e) {
    }
    if (v != null) {
        try {
            return Integer.decode(v);
        } catch (NumberFormatException e) {
        }
    }
    return val;
}
```

2. Loose Coupling VS High Cohesion

java.lang.Integer.getInteger

```
public static Integer getInteger(String nm, Integer val) {
    String v = null;
    try {
        v = System.getProperty(nm);
    } catch (IllegalArgumentException | NullPointerException e) {
    }
    if (v != null) {
        try {
            return Integer.decode(v);
        } catch (NumberFormatException e) {
        }
    }
    return val;
}
```

There isn't the code you're looking for

2. Loose Coupling VS High Cohesion

java.lang.Long.getLong

2. Loose Coupling VS High Cohesion

java.lang.Long.getLong

That's enough!

2. Loose Coupling VS High Cohesion

Выводы

2. Loose Coupling VS High Cohesion

Выводы

- DRY
- KISS

2. Loose Coupling VS High Cohesion

Выводы

- DRY: Don't Repeat Yourself
- KISS: Keep It Simple, Smart

2. Loose Coupling VS High Cohesion

Выводы

- DRY: Don't Repeat Yourself
- KISS: Keep It Simple, Smart
- Чем меньше требуется навигации по коду, тем лучше: между пакетами и классами, переключение экрана с кодом класса

3. Наследование VS Делегирование

3. Наследование VS Делегирование

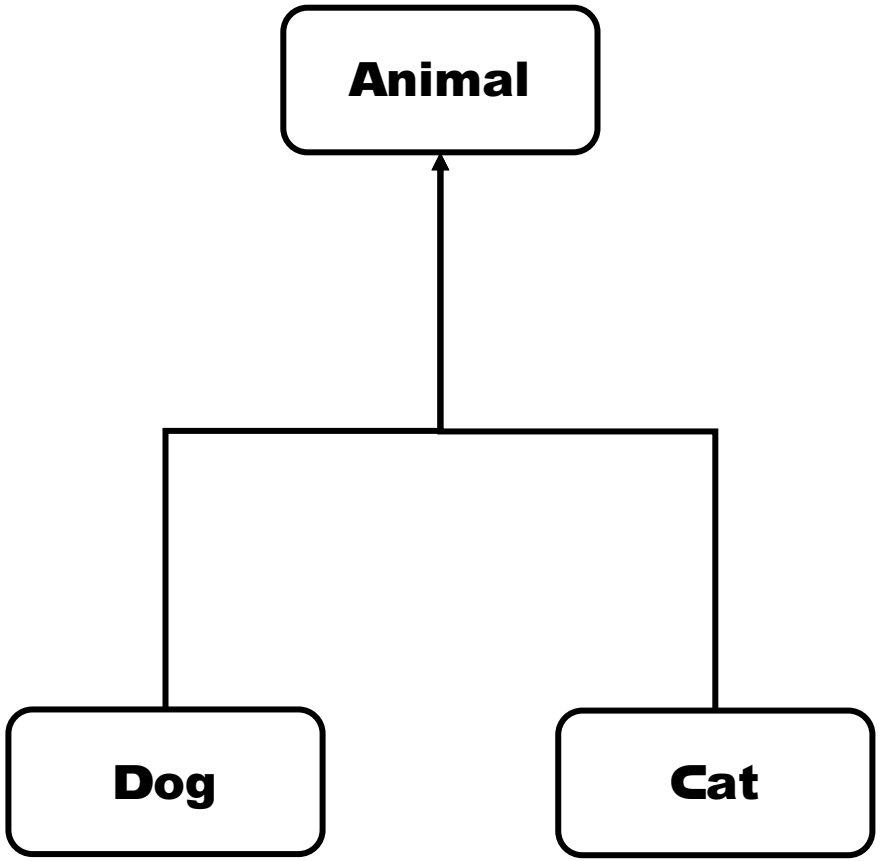
«В любой непонятной ситуации - делегируй»

Александр Кучук

https://www.youtube.com/watch?v=jl_4g2QxqMg

3. Наследование VS Делегирование

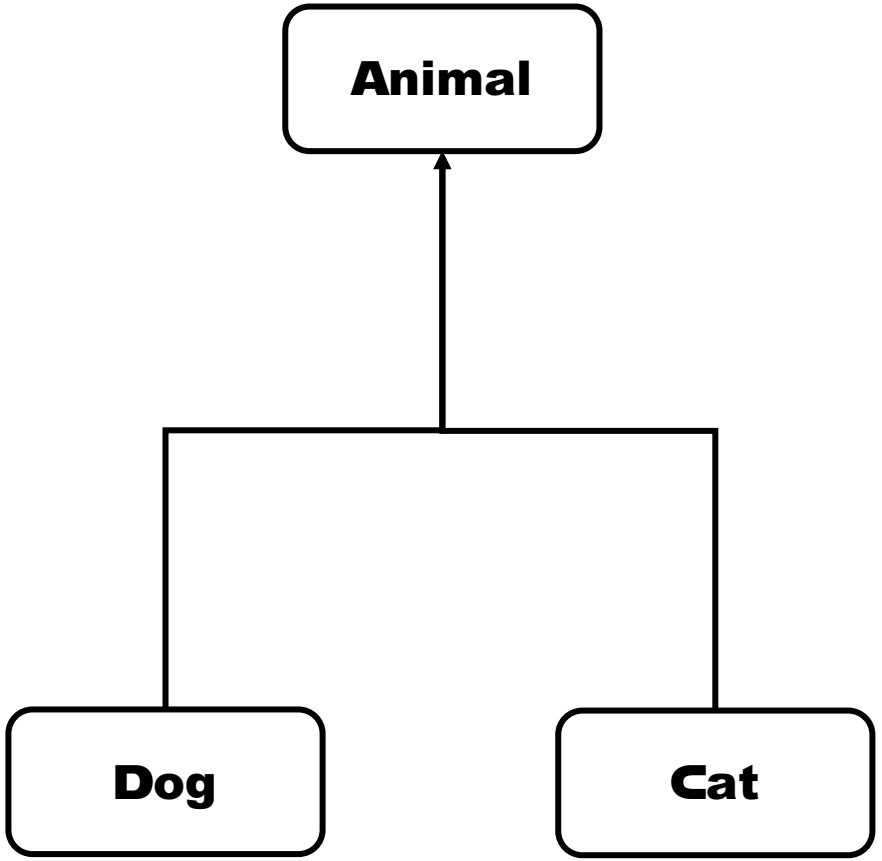
Классический пример



3. Наследование VS Делегирование

Классический пример

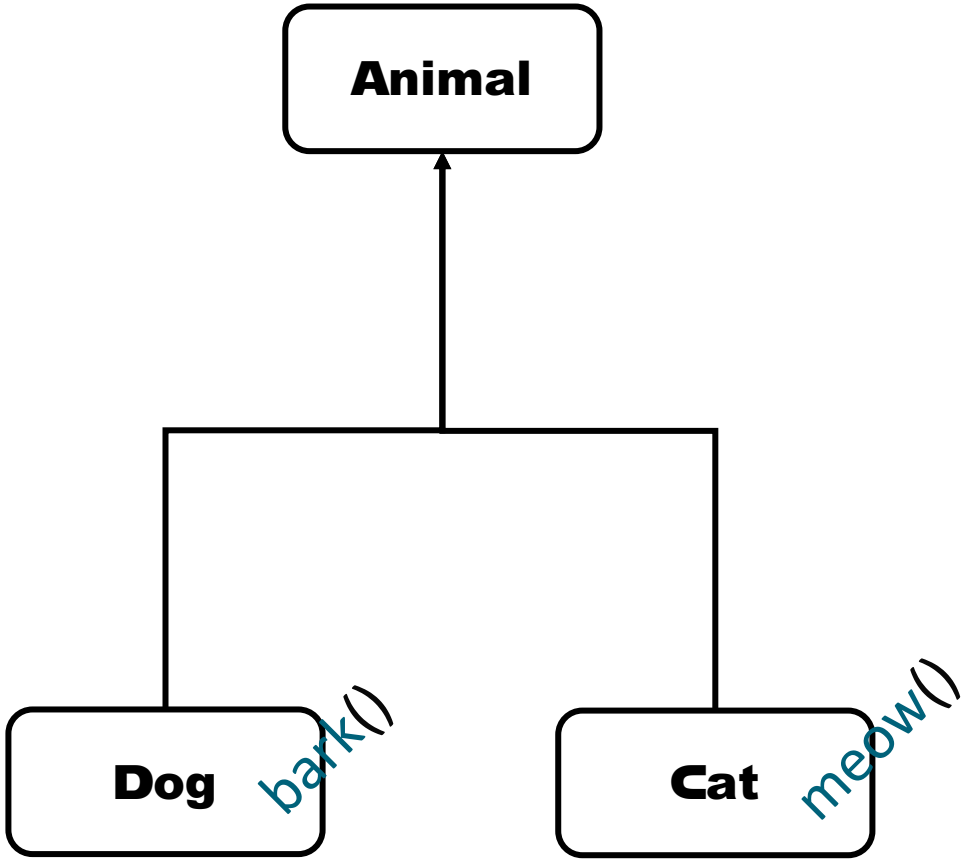
```
public class Dog extends Animal {

    public String bark() {  return "Гав!";  }
}

public class Cat extends Animal {

    public String meow() {  return "Мяу!";  }
}
```



3. Наследование VS Делегирование

Классический пример

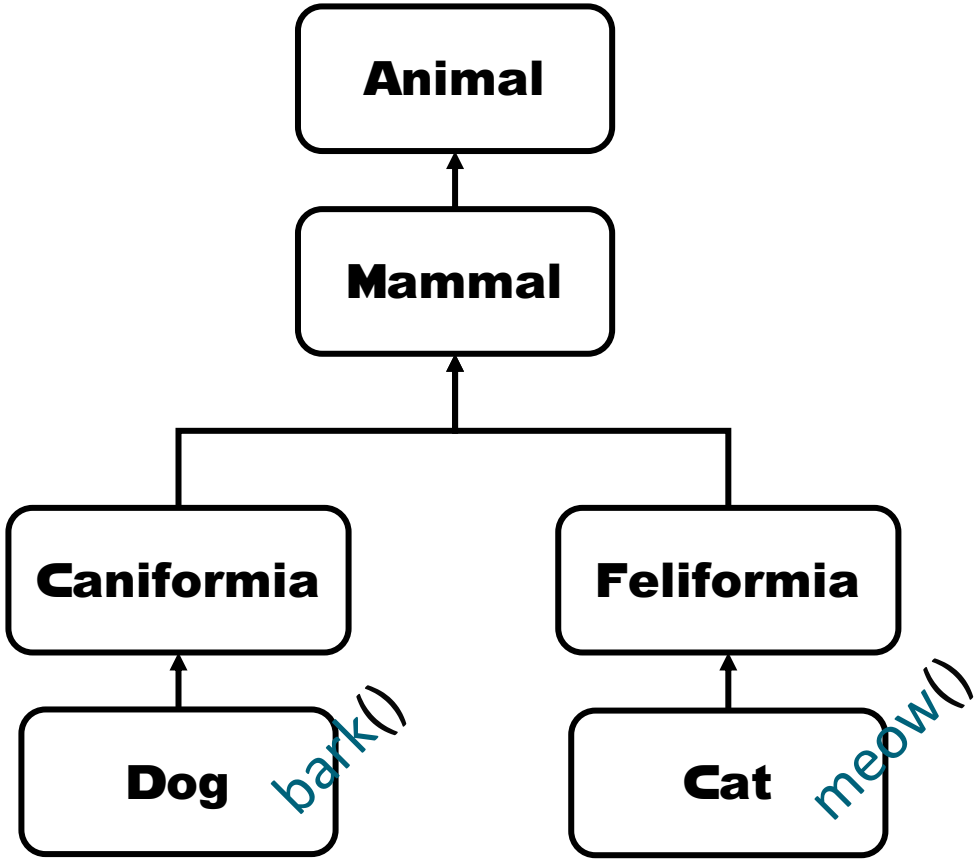
```
public class Dog extends Animal {

    public String bark() {  return "Гав!";  }
}

public class Cat extends Animal {

    public String meow() {  return "Мяу!";  }
}
```



3. Наследование VS Делегирование

Классический пример

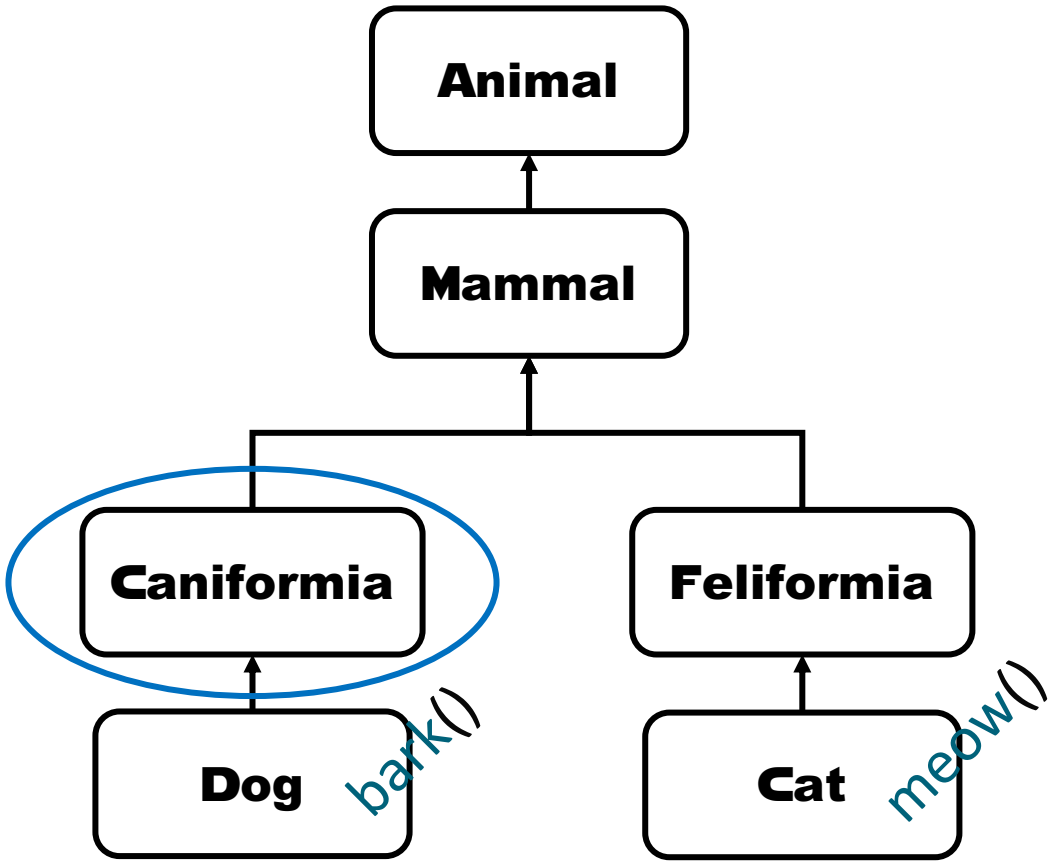
```
public class Dog extends Caniformia {
    @Override
    public String bark() {  return "Гав!";  }
}

public class Cat extends Feliformia {
    @Override
    public String meow() {  return "Мяу!";  }
}
```



3. Наследование VS Делегирование

Классический пример

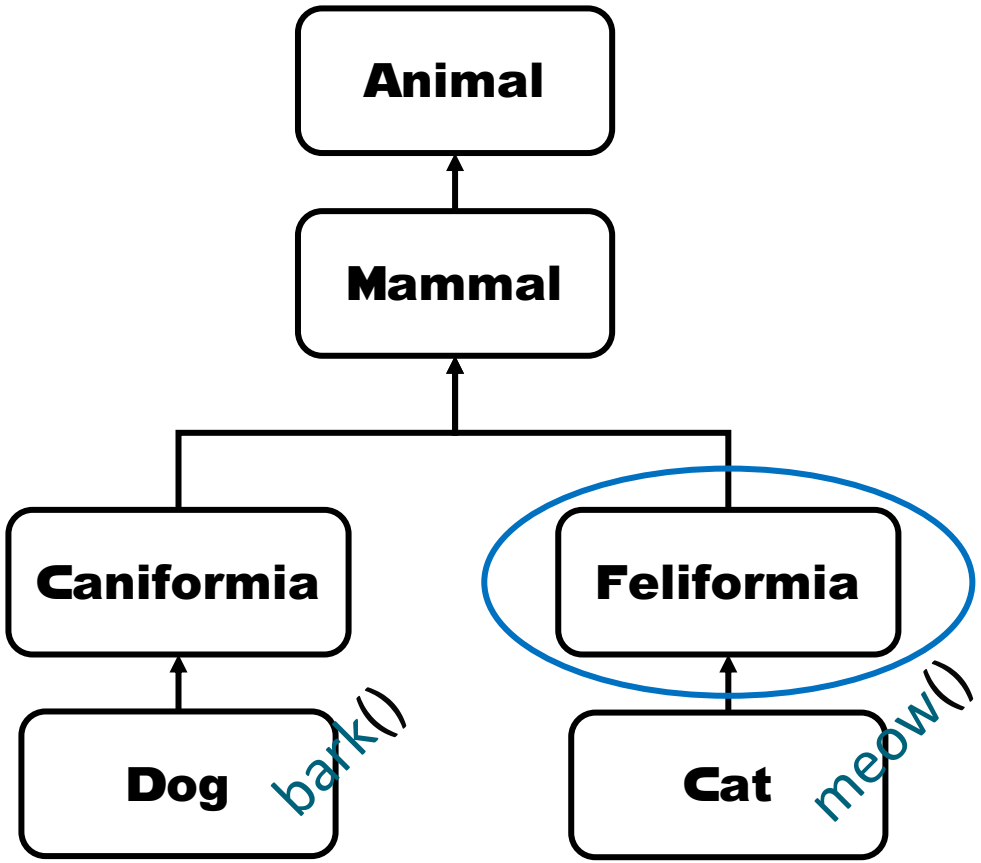
```
public class Dog extends Caniformia {
    @Override
    public String bark() {  return "Гав!";  }
}

public class Cat extends Feliformia {
    @Override
    public String meow() {  return "Мяу!";  }
}
```



3. Наследование VS Делегирование

Классический пример

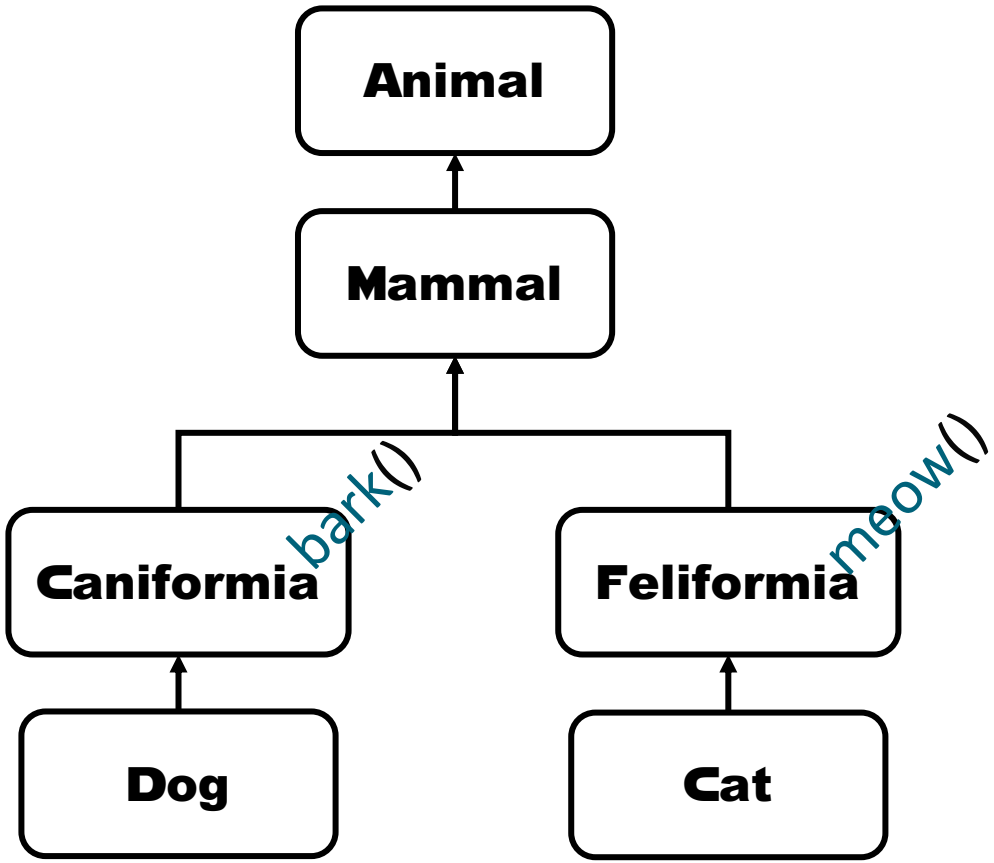
```
public class Dog extends Caniformia {
    @Override
    public String bark() {  return "Гав!";  }
}

public class Cat extends Feliformia {
    @Override
    public String meow() {  return "Мяу!";  }
}
```



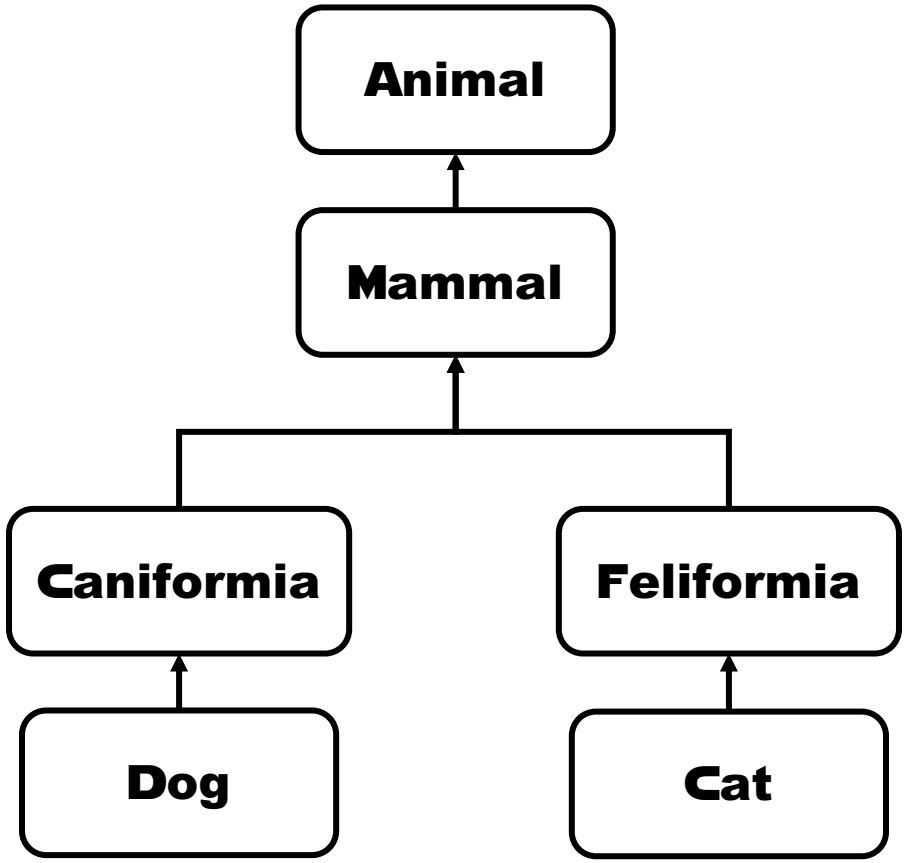
3. Наследование VS Делегирование

Классический пример

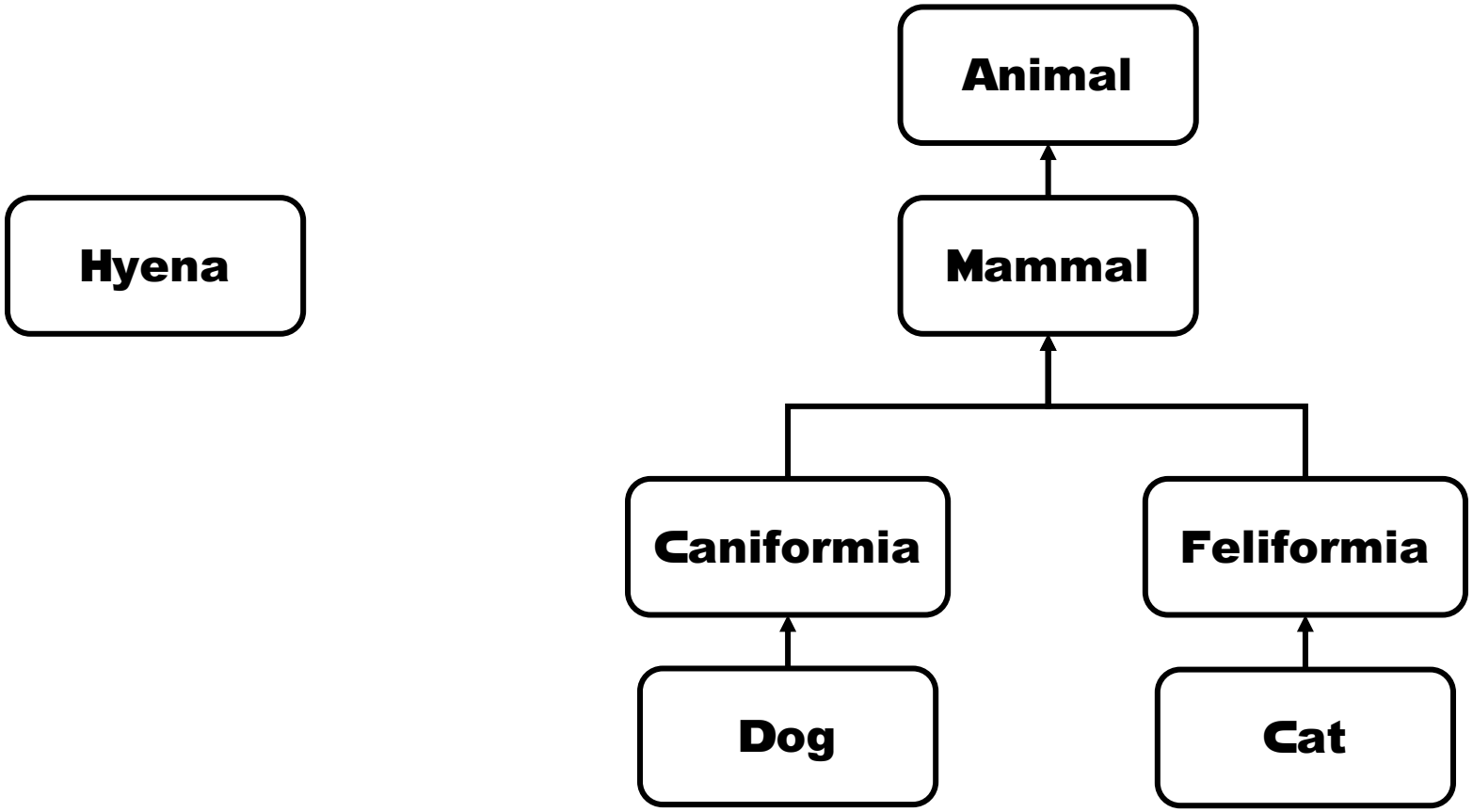
```
public class Dog extends Caniformia {
    @Override
    public String bark() {  return "Гав!";  }
}

public class Cat extends Feliformia {
    @Override
    public String meow() {  return "Мяу!";  }
}
```



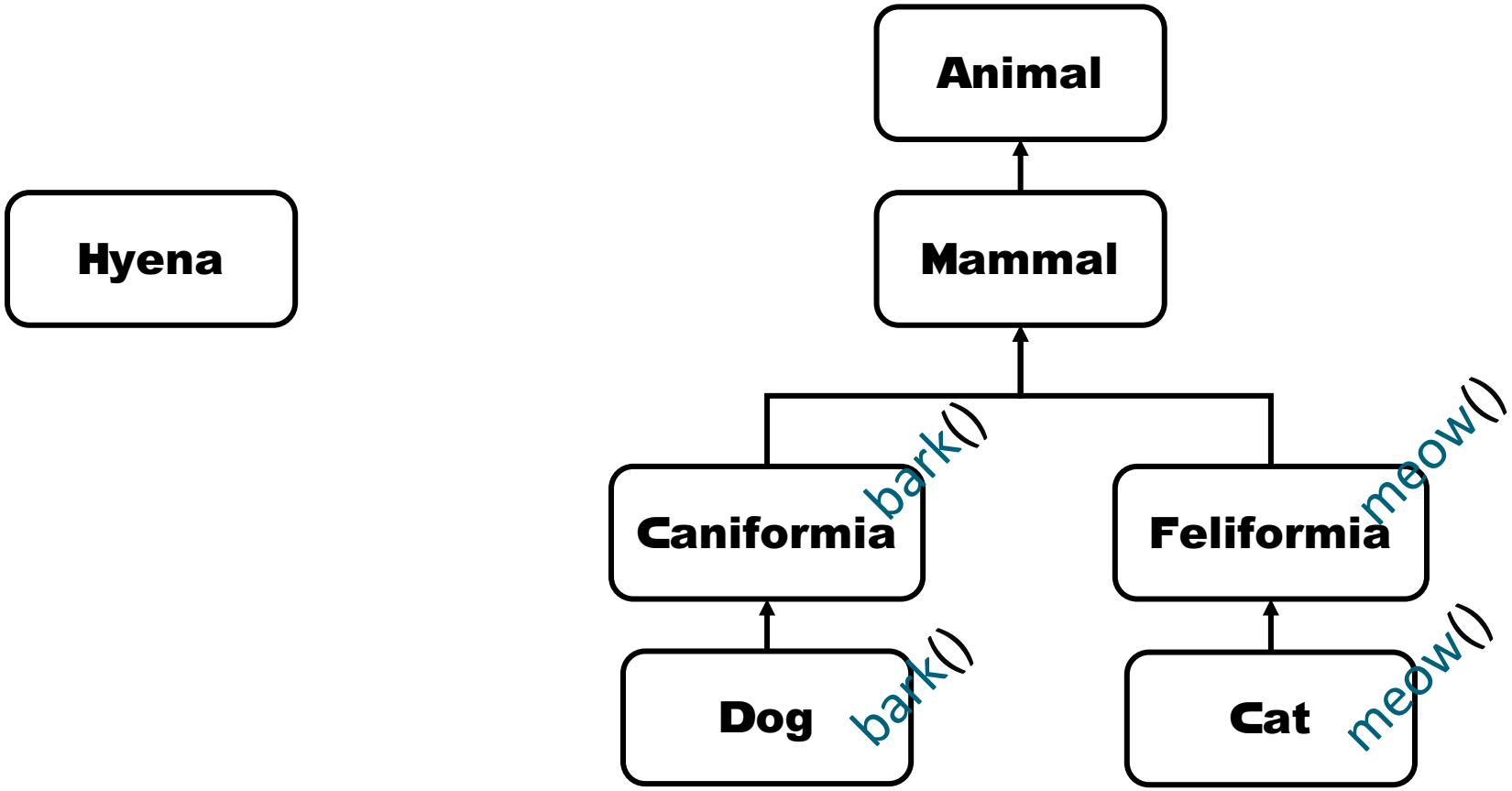
3. Наследование VS Делегирование



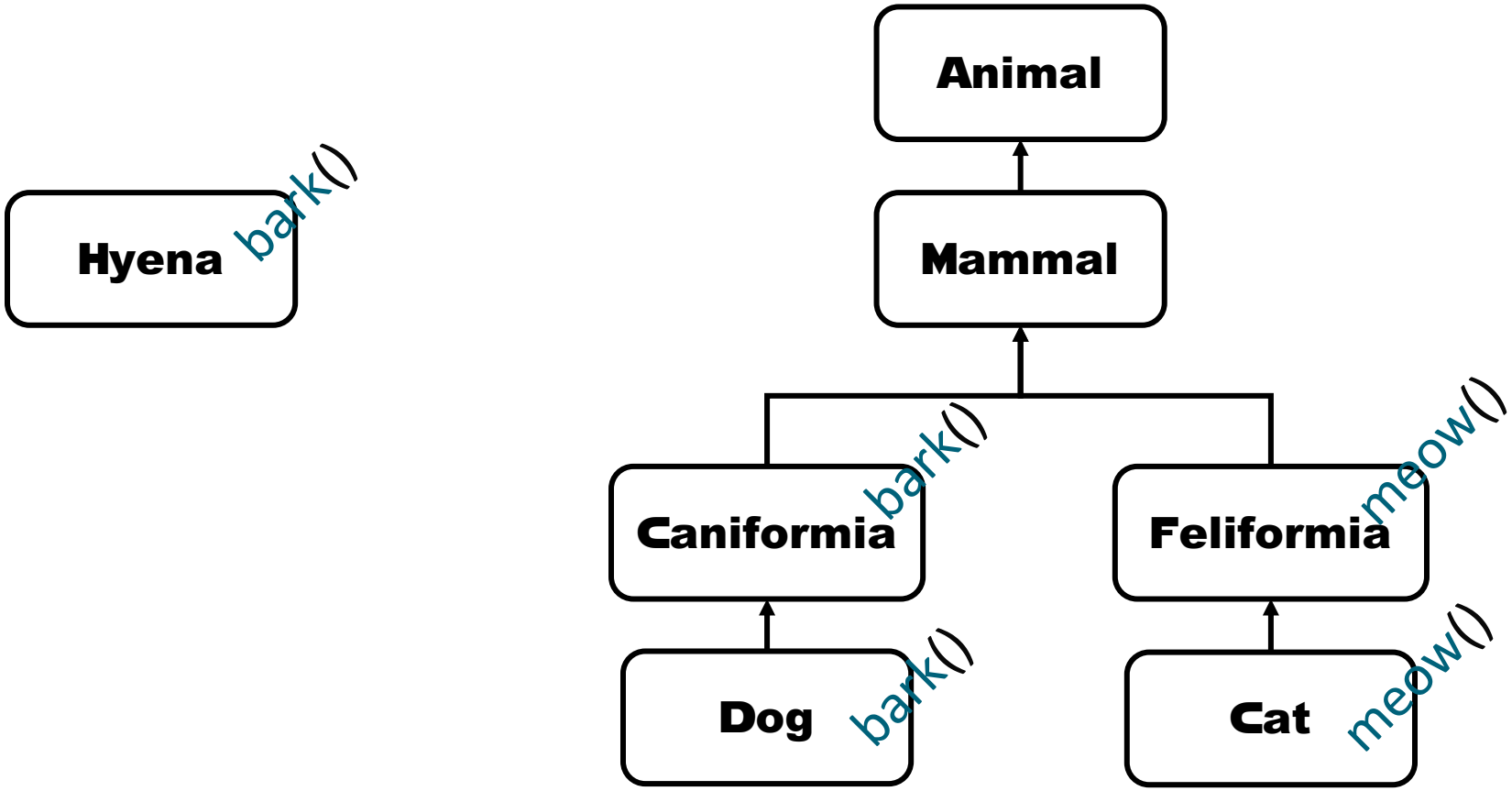
3. Наследование VS Делегирование



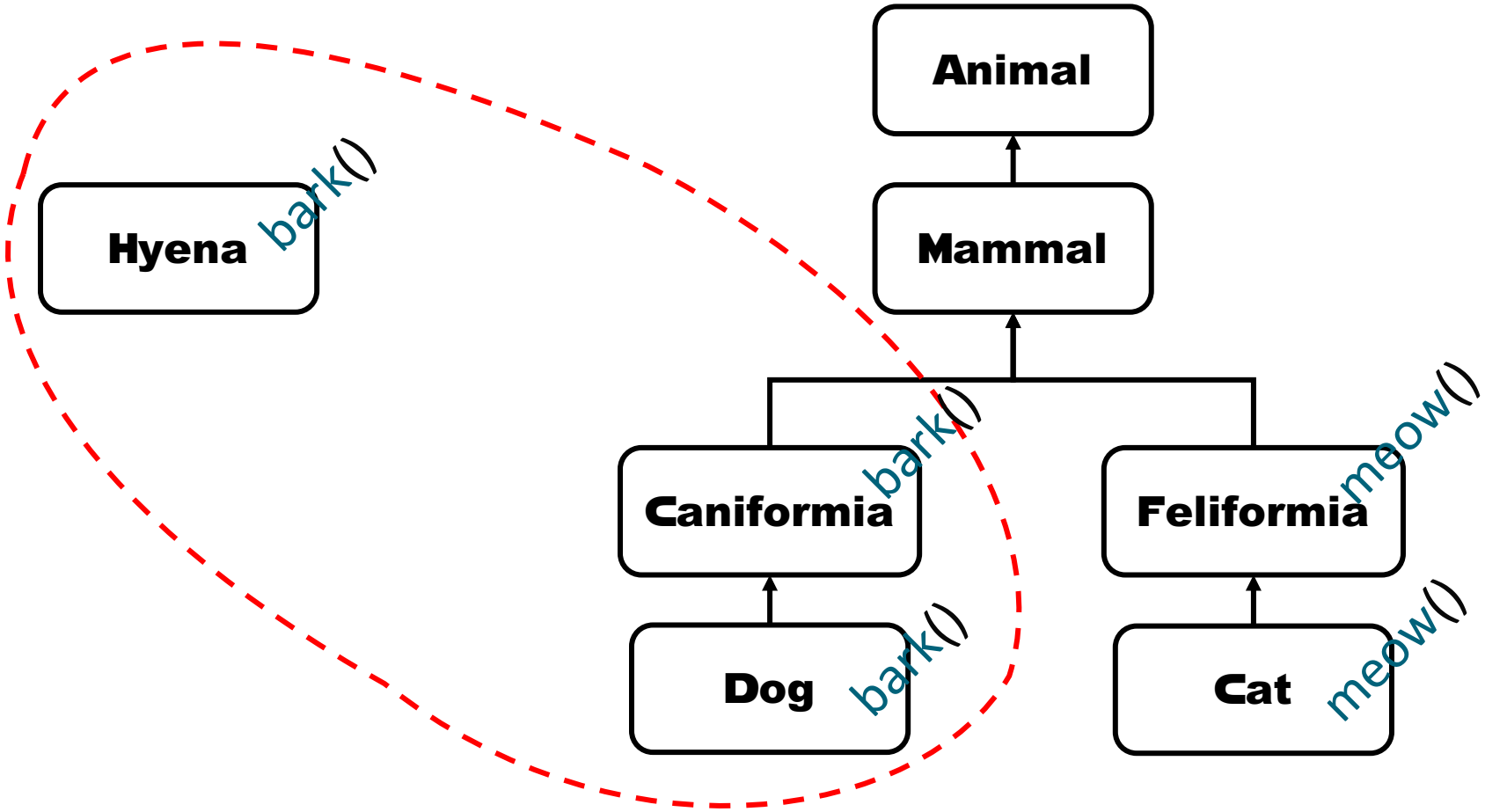
3. Наследование VS Делегирование



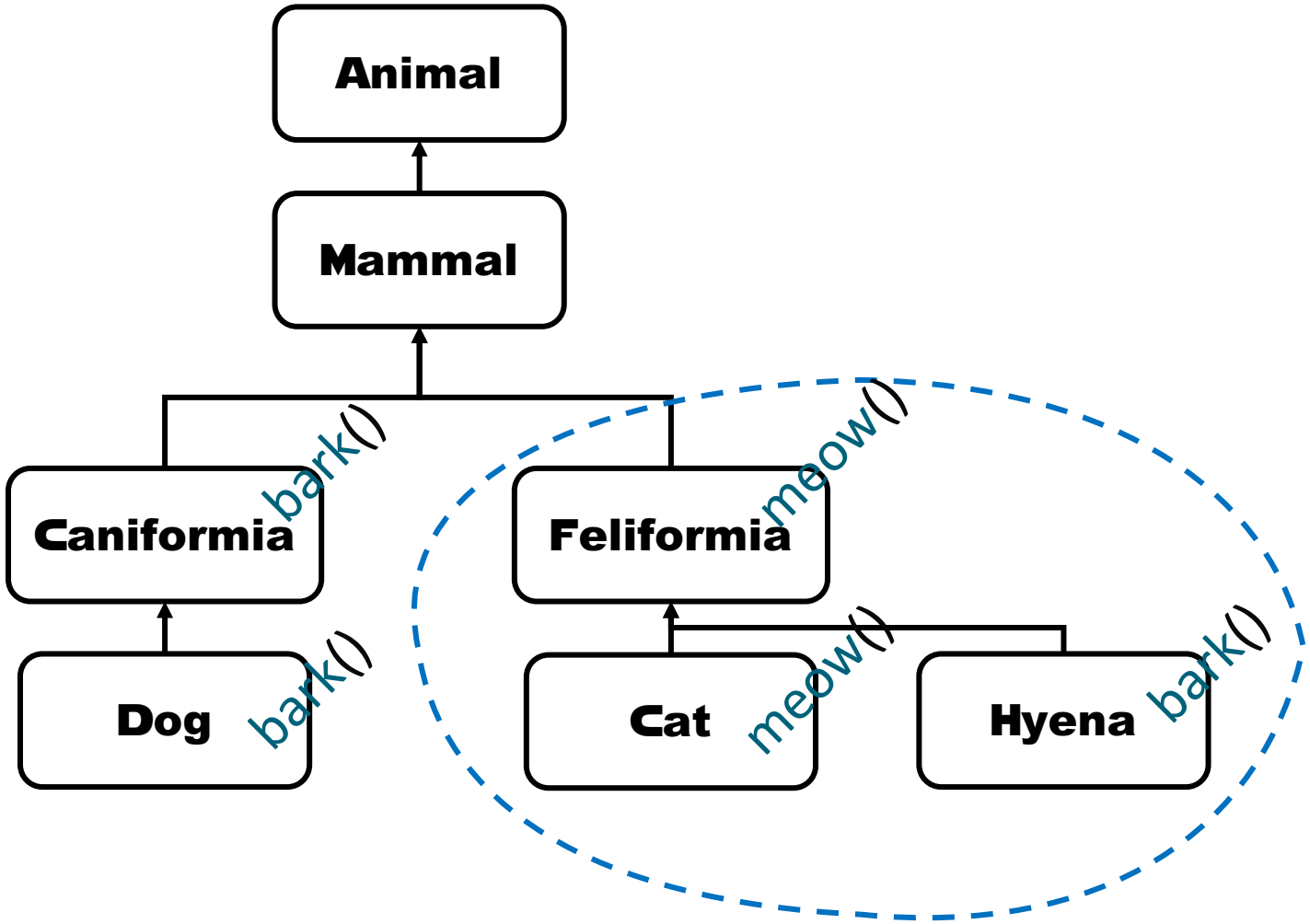
3. Наследование VS Делегирование



3. Наследование VS Делегирование

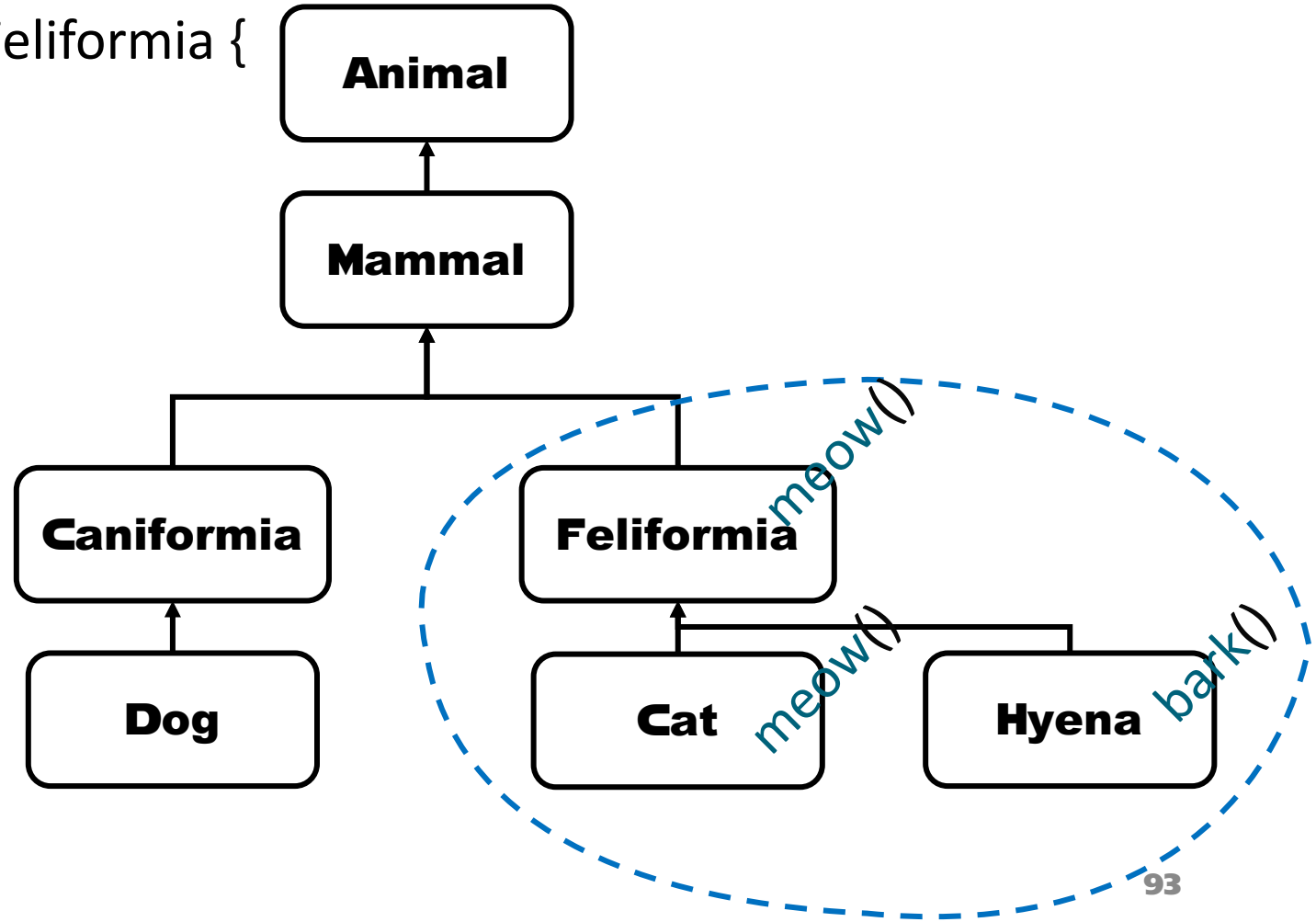


3. Наследование VS Делегирование

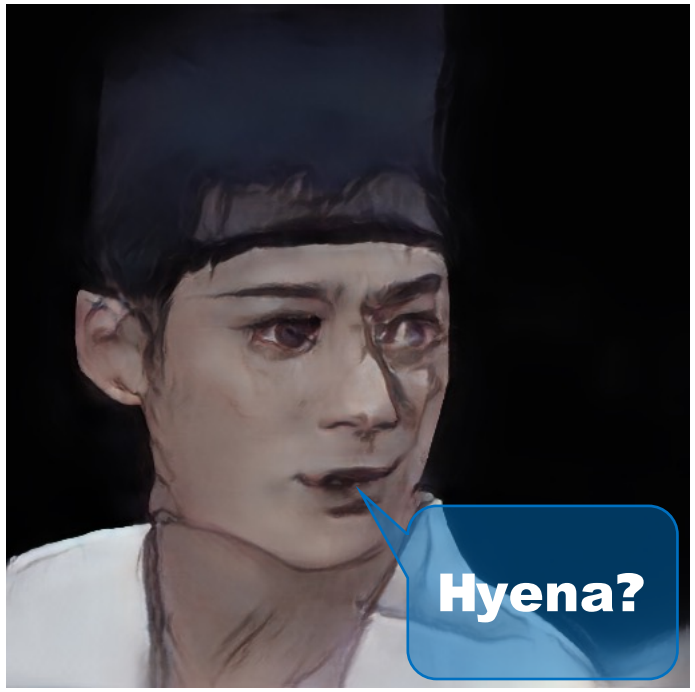


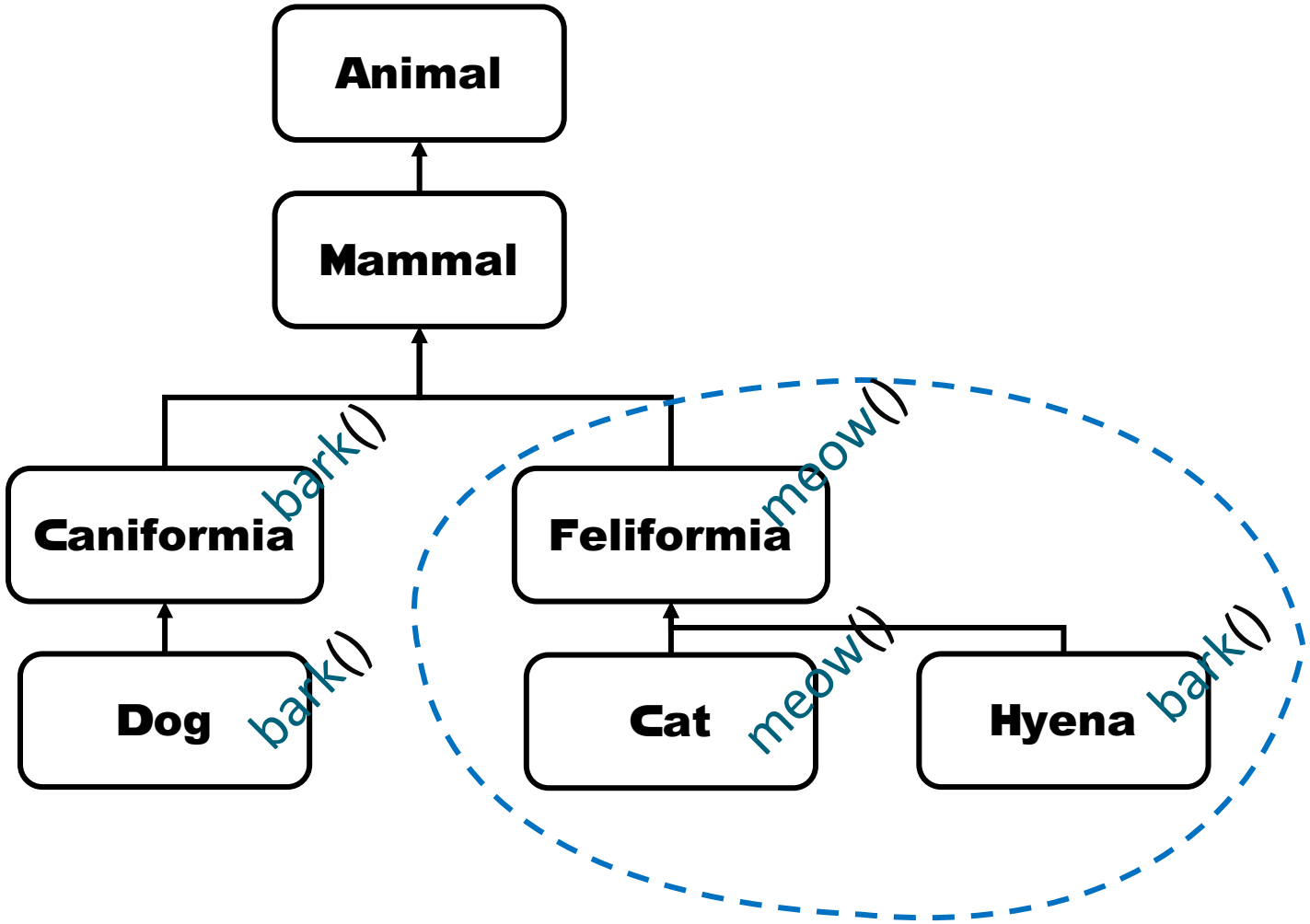
3. Наследование VS Делегирование

```
public static class Hyena extends Feliformia {
    public String bark() {
        return "Ай!";
    }

    //WTF???
    @Override
    public String meow() {
        return bark();
    }
}
```

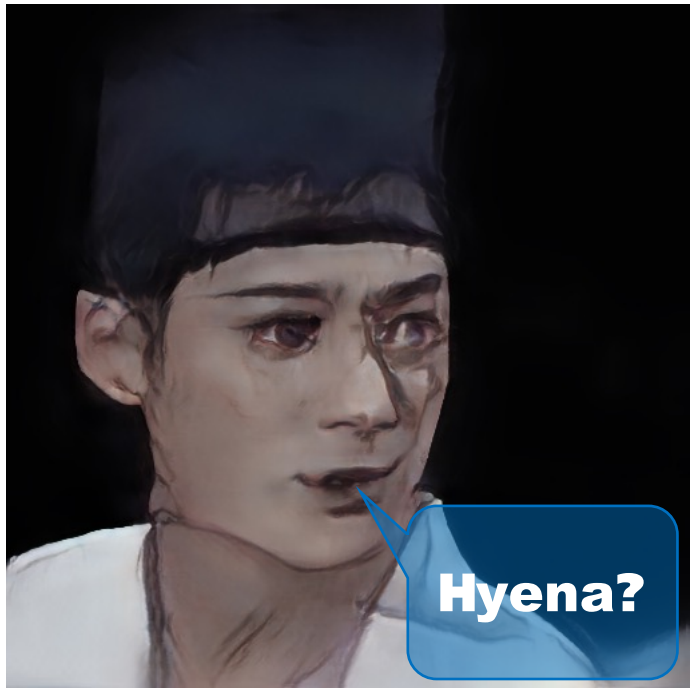


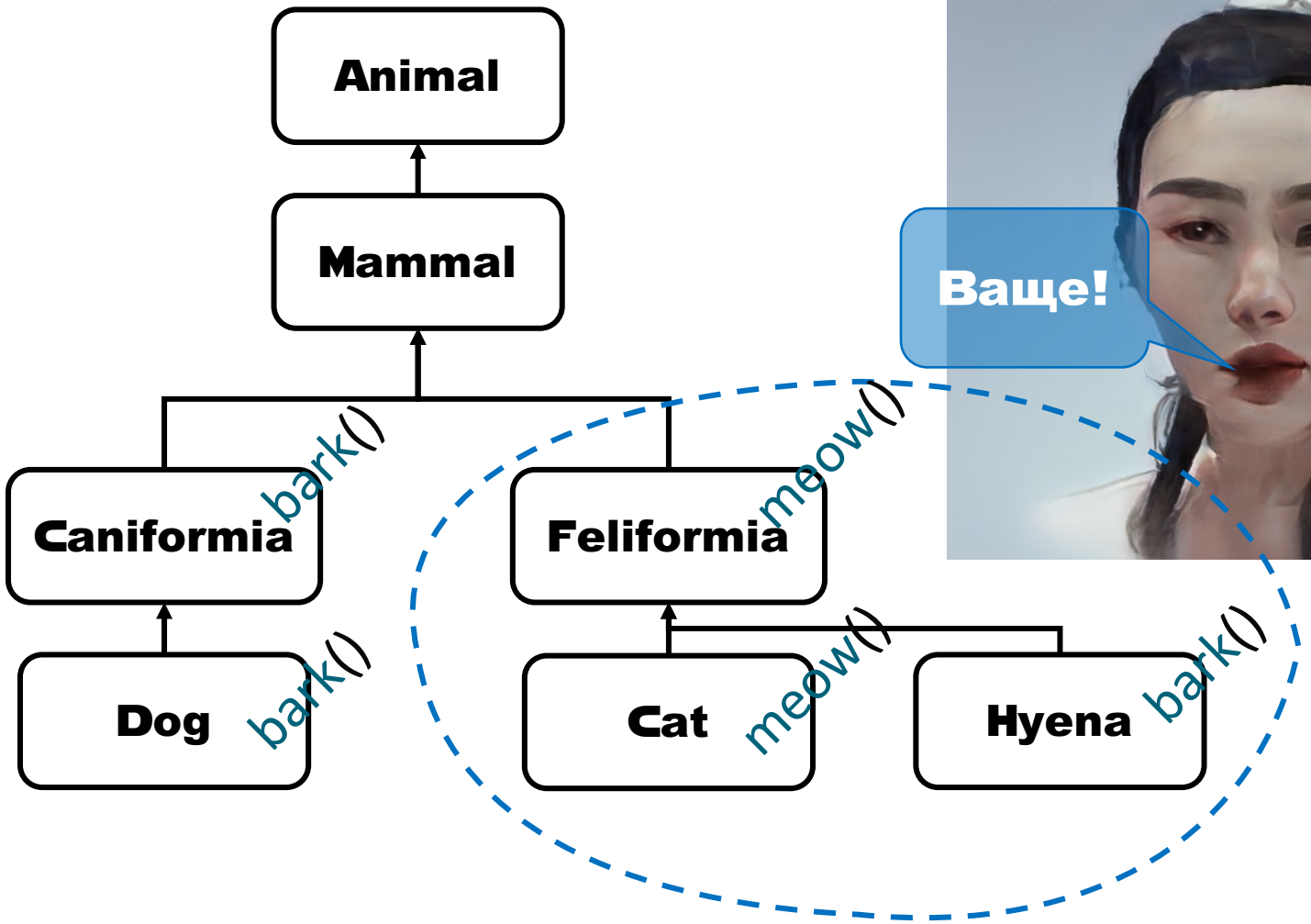
3. Наследование VS Делегирование

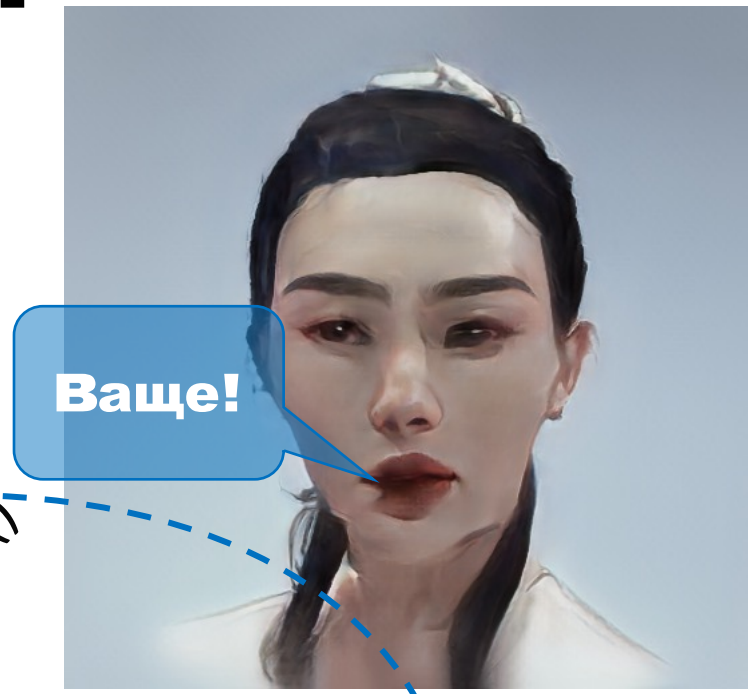




3. Наследование VS Делегирование







3. Наследование VS Делегирование

```
public interface Talkable {
   String talk();
}

public abstract class Animal implements Talkable {
}

interface Barkable extends Talkable {
   @Override
   default String talk() {  return bark();  }
   String bark();
}
```

3. Наследование VS Делегирование

```
public static class Dog extends Caniformia implements Barkable {
   public String bark() {  return "Гав!";  }
}

public static class Cat extends Feliformia implements Meowable {
   public String meow() {  return "Мяу!";  }
}

public static class Hyena extends Feliformia implements Barkable {
   public String bark() {  return "Ай!";  }
}
```

3. Наследование VS Делегирование

```
interface Talkable {
    fun talk(): String
}

abstract class Animal : Talkable

interface Barkable : Talkable {
    override fun talk(): String = bark()
    fun bark(): String
}

class Barking(val s: String) : Barkable {
    override fun bark(): String = s
}
```

https://kotlinlang.org/docs/delegation.html

3. Наследование VS Делегирование

```
interface Talkable {
    fun talk(): String
}

abstract class Animal : Talkable

interface Barkable : Talkable {
    override fun talk(): String = bark()
    fun bark(): String
}

class Barking(val s: String) : Barkable {
    override fun bark(): String = s
}
```

https://kotlinlang.org/docs/delegation.html

3. Наследование VS Делегирование

```
interface Talkable {
    fun talk(): String
}

abstract class Animal : Talkable

interface Barkable : Talkable {
    override fun talk(): String = bark()
    fun bark(): String
}

class Barking(val s: String) : Barkable {
    override fun bark(): String = s
}
```

```
class Dog : Caniformia(), Barkable by Barking("Гав!")

class Cat : Feliformia(), Meowable by Meowing("Мяу!")

class Hyena : Feliformia(), Barkable by Barking("Ай!")
```

https://kotlinlang.org/docs/delegation.html

3. Наследование VS Делегирование

```
interface Talkable {
    fun talk(): String
}

abstract class Animal : Talkable

interface Barkable : Talkable {
    override fun talk(): String = bark()
    fun bark(): String
}

class Barking(val s: String) : Barkable {
    override fun bark(): String = s
}
```

```
class Dog : Caniformia(), Barkable by Barking("Гав!")

class Cat : Feliformia(), Meowable by Meowing("Мяу!")

class Hyena : Feliformia(), Barkable by Barking("Ай!")
```

https://kotlinlang.org/docs/delegation.html

3. Наследование VS Делегирование

«В любой непонятной ситуации - делегируй»

Александр Кучук

https://www.youtube.com/watch?v=jl_4g2QxqMg

3. Наследование VS Делегирование

Выводы

3. Наследование VS Делегирование

Выводы

- Использовать наследование, когда это очевидно

3. Наследование VS Делегирование

Выводы

- Использовать наследование, когда это очевидно
- Всегда помнить, что наследуется поведение

3. Наследование VS Делегирование

Выводы

- Использовать наследование, когда это очевидно
- Всегда помнить, что наследуется поведение
- Если кто-то крякает как утка, ходит как утка и плавает как утка, то это ещё ничего не значит

4. Абстракции:
создавать VS не создавать

4. Абстракции:
создавать VS не создавать

«Не следует множить сущности без необходимости»

«Бритва Оккама», William of Ockham

4. Абстракции:
создавать VS не создавать

«Не следует множить сущности без необходимости»

«Бритва Оккама», William of Ockham

«Loose Coupling & High Cohesion»

Larry Constantine

4. Абстракции: создавать VS не создавать

```
public static double computeWithRawScalars(
    double x1, double y1, double z1,
    double x2, double y2, double z2) {
  double x = y1 * z2 - z1 * y2;
  double y = z1 * x2 - x1 * z2;
  double z = x1 * y2 - y1 * x2;
  return x * x + y * y + z * z;
}
```

```
public static double computeWithVectors(
    double x1, double y1, double z1,
    double x2, double y2, double z2) {
  Vector v1 = new Vector(x1, y1, z1);
  Vector v2 = new Vector(x2, y2, z2);
  return v1.crossProduct(v2).squared();
}
```

4. Абстракции: создавать VS не создавать

```
public static double computeWithRawScalars(
        double x1, double y1, double z1,
        double x2, double y2, double z2) {
    double x = y1 * z2 - z1 * y2;
    double y = z1 * x2 - x1 * z2;
    double z = x1 * y2 - y1 * x2;
    return x * x + y * y + z * z;
}
```

```
public static double computeWithVectors(
        double x1, double y1, double z1,
        double x2, double y2, double z2) {
    Vector v1 = new Vector(x1, y1, z1);
    Vector v2 = new Vector(x2, y2, z2);
    return v1.crossProduct(v2).squared();
}
```

Junior

4. Абстракции: создавать VS не создавать

```
public static double computeWithRawScalars(
        double x1, double y1, double z1,
        double x2, double y2, double z2) {
    double x = y1 * z2 - z1 * y2;
    double y = z1 * x2 - x1 * z2;
    double z = x1 * y2 - y1 * x2;
    return x * x + y * y + z * z;
}
```

Middle

```
public static double computeWithVectors(
        double x1, double y1, double z1,
        double x2, double y2, double z2) {
    Vector v1 = new Vector(x1, y1, z1);
    Vector v2 = new Vector(x2, y2, z2);
    return v1.crossProduct(v2).squared();
}
```

Junior

4. Абстракции: создавать VS не создавать

```
public static double computeWithRawScalars(
        double x1, double y1, double z1,
        double x2, double y2, double z2) {
    double x = y1 * z2 - z1 * y2;
    double y = z1 * x2 - x1 * z2;
    double z = x1 * y2 - y1 * x2;
    return x * x + y * y + z * z;
}
```

Middle

```
public static double computeWithVectors(
        double x1, double y1, double z1,
        double x2, double y2, double z2) {
    Vector v1 = new Vector(x1, y1, z1);
    Vector v2 = new Vector(x2, y2, z2);
    return v1.crossProduct(v2).squared();
}
```

Junior

Senior

4. Абстракции: создавать VS не создавать

```
Benchmark                Mode Cnt   Score   Error Units
computeWithRawScalars avgt  50   4,783 ± 0,031 ns/op
computeWithVectors    avgt  50   4,785 ± 0,040 ns/op
```

4. Абстракции: создавать VS не создавать

```
Benchmark                Mode Cnt   Score    Error Units
computeWithRawScalars avgt  50   4,783 ± 0,031 ns/op
computeWithVectors    avgt  50   4,785 ± 0,040 ns/op
```

«Разбор перформансных задач с JBreak (часть 3)»

JBreak 2018

https://habr.com/ru/company/skbkontur/blog/351540/

4. Абстракции: создавать VS не создавать

```
Benchmark                  Mode Cnt   Score    Error Units
computeWithRawScalars avgt  50   4,783 ± 0,031 ns/op
computeWithVectors    avgt  50   4,785 ± 0,040 ns/op
```

«Разбор перформансных задач с JBreak (часть 3)»

JBreak 2018

https://habr.com/ru/company/skbkontur/blog/351540/

4. Абстракции: создавать VS не создавать

Выводы

4. Абстракции: создавать VS не создавать

Выводы

- При создании новой абстракции не нужно думать о мифической производительности

4. Абстракции: создавать VS не создавать

Выводы

- При создании новой абстракции не нужно думать о мифической производительности
- На первом месте должна быть читаемость кода!

4. Абстракции: создавать VS не создавать

Выводы

- При создании новой абстракции не нужно думать о мифической производительности
- На первом месте должна быть читаемость кода!
- См. Loose Coupling & High Cohesion

5. Integer VS int

5. Integer VS int

java.lang.Integer

```
public final class Integer extends Number
      implements Comparable<Integer>, Constable, ConstantDesc {

   private final int value;

   /* Methods, static fields and static methods */
}
```

5. Integer VS int

Auto-boxing и unboxing

```
Integer integer = 42;// Auto-boxing
int i = integer;// Auto-unboxing
```

5. Integer VS int

Auto-boxing и unboxing

```
Integer integer = 42;// Auto-boxing
int i = integer;// Auto-unboxing
```

А как быть с null?

```
Integer integer = null;
int i = integer;
```

5. Integer VS int

Auto-boxing и unboxing

```
Integer integer = 42;// Auto-boxing
int i = integer;// Auto-unboxing
```

А как быть с null?

```
Integer integer = null;
int i = integer;
```

```
Exception in thread "main" java.lang.NullPointerException:
Cannot invoke "java.lang.Integer.intValue()" because "integer" is null
```

5. Integer VS int

```
private class IntegerContainer {
    private int value;
}


IntegerContainer container = new IntegerContainer();
if (container.value == 0) {
        System.out.println("Value is 0");
}
```

5. Integer VS int

```
private class IntegerContainer {
    private int value;
}


IntegerContainer container = new IntegerContainer();
if (container.value == 0) {
        System.out.println("Value is 0");
}
```

5. Integer VS int

```
private class IntegerContainer {
    private Integer value;
}


IntegerContainer container = new IntegerContainer();
if (container.value == 0) {
        System.out.println("Value is 0");
}
```

5. Integer VS int

```
private class IntegerContainer {
    private Integer value;
}


IntegerContainer container = new IntegerContainer();
if (container.value == 0) {
        System.out.println("Value is 0");
}
```

```
Exception in thread "main”
java.lang.NullPointerException:
Cannot invoke "java.lang.Integer.intValue()"
because "container.value" is null
```

5. Integer VS int

sizeOf

5. Integer VS int

sizeOf

	x32 HotSpot	x64 -XX:+UseCompressedOops	x64 -XX:-UseCompressedOops
int	4 bytes	4 bytes	4 bytes
Object	8 bytes	16 bytes	16 bytes
Integer	16 bytes	16 bytes	24 bytes

5. Integer VS int

sizeOf

	x32 HotSpot	x64 -XX:+UseCompressedOops	x64 -XX:-UseCompressedOops
int	4 bytes	4 bytes	4 bytes
Object	8 bytes	16 bytes	16 bytes
Integer	16 bytes	16 bytes	24 bytes
int[]	4 bytes	4 bytes	4 bytes
Integer[]	20 bytes	20 bytes	32 bytes

https://shipilev.net/jvm/objects-inside-out/

5. Integer VS int

sizeOf

	x32 HotSpot	x64 -XX:+UseCompressedOops	x64 -XX:-UseCompressedOops
int	4 bytes	4 bytes	4 bytes
Object	8 bytes	16 bytes	16 bytes
Integer	16 bytes	16 bytes	24 bytes
int[]	4 bytes	4 bytes	4 bytes
Integer[]	20 bytes	20 bytes	32 bytes

*** В пересчёте на одно значение**
без учёта затрат на сам объект массива

5. Integer VS int

Если всё так плохо, то зачем нужен Integer?

5. Integer VS int

Если всё так плохо, то зачем нужен Integer?

- **Collection<E>: List<E>, Set<E>, Queue<E>, ...**
- **Map<K,V>**

5. Integer VS int

Google Guava

5. Integer VS int

Google Guava
- Ints.asList - IntArrayAsList

```
int[] backingArray = /*  ...  */
List<Integer> guavaArrayList = Ints.asList(backingArray);
```

https://github.com/google/guava/blob/master/guava/src/com/google/common/primitives/Ints.java

5. Integer VS int

Arrays.asList vs Ints.asList

5. Integer VS int

Arrays.asList vs Ints.asList

- **IntArrayAsList занимает меньше памяти в 5 или 8 раз**

5. Integer VS int

Arrays.asList vs Ints.asList

- IntArrayAsList занимает меньше памяти в 5 или 8 раз
- IntArrayAsList медленее на итерирование до 5 раз, если используется boxing
- IntArrayAsList быстрее на итерирование на 25%, если не используется boxing

5. Integer VS int

Будущее - Project Valhalla

https://openjdk.org/projects/valhalla/

5. Integer VS int

Будущее - Project Valhalla

- https://openjdk.org/projects/valhalla/

6. ArrayList *VS* LinkedList

6. ArrayList VS LinkedList

- Кто использует LinkedList?

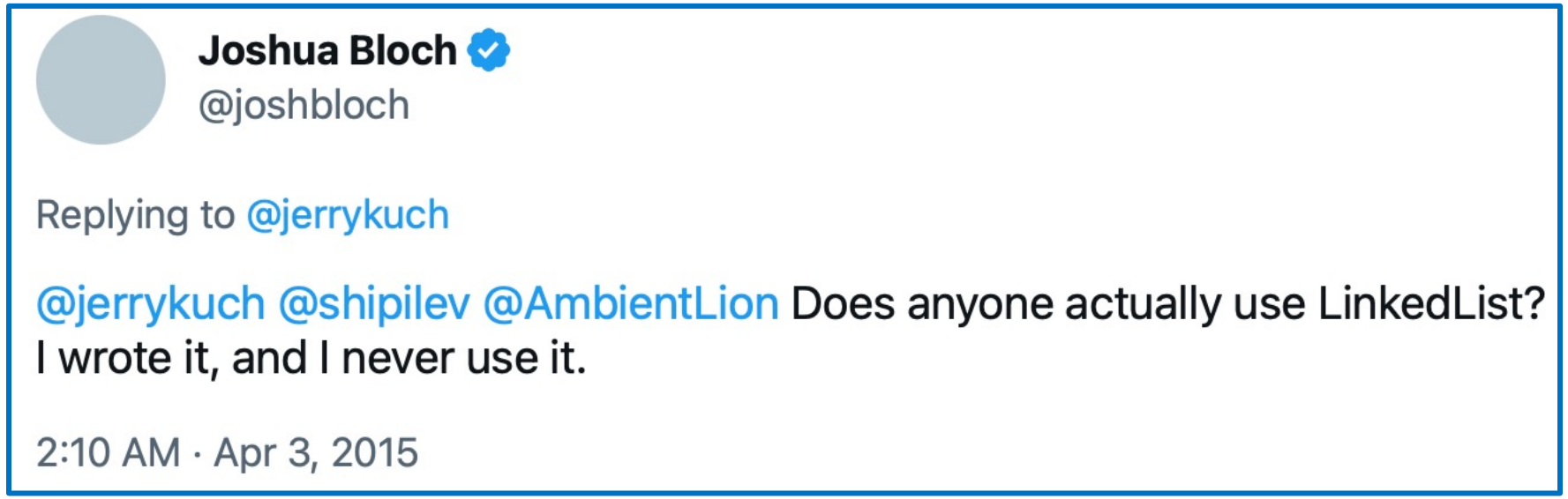
6. ArrayList VS LinkedList

- Кто использует LinkedList?
- А ArrayList?

6. ArrayList VS LinkedList

- Кто использует LinkedList?
- А ArrayList?
- Может быть, что-то другое?

6. ArrayList VS LinkedList



6. ArrayList VS LinkedList

Примеры использования связных списков

6. ArrayList *VS* LinkedList

Примеры использования связных списков

- HashMap.Node (Josh Bloch)

6. ArrayList *VS* LinkedList

Примеры использования связных списков

- **HashMap.Node (Josh Bloch)**
- **LinkedHashMap.Entry (Josh Bloch)**

6. ArrayList VS LinkedList

Примеры использования связных списков

- HashMap.Node (Josh Bloch)
- LinkedHashMap.Entry (Josh Bloch)
- ConcurrentDoublyLinkedList (Doug Lea)

http://www.java2s.com/Code/Java/Collections-Data-Structure/ConcurrentDoublyLinkedList.htm

6. ArrayList VS LinkedList

Плохой сценарий для ArrayList -

6. ArrayList *VS* LinkedList

Плохой сценарий для ArrayList -
добавление и удаление из начала

6. ArrayList *VS* LinkedList

Плохой сценарий для ArrayList -
добавление и удаление из начала

ArrayDeque - ваш выбор!

6. ArrayList VS LinkedList

Выводы

6. ArrayList VS LinkedList

Выводы

- Каждой задаче своя структура данных

6. ArrayList *VS* LinkedList

Выводы

- Каждой задаче своя структура данных
- Вместо java.util.LinkedList
  есть структуры получше

7. Result<D,E> VS Exception

7. Result<D,E> VS Exception

```
public class Result<D, E> {
    private final D data;
    private final E error;

    private Result(D data, E error) {
        this.data = data;
        this.error = error;
    }

    public D data() {
        return data;
    }

    public E error() {
        return error;
    }
```

7. Result<D,E> VS Exception

```
public class Result<D, E> {
    private final D data;
    private final E error;

    private Result(D data, E error) {
        this.data = data;
        this.error = error;
    }

    public D data() {
        return data;
    }

    public E error() {
        return error;
    }

    public static <D, E> Result<D, E> of(@NotNull D data) {
        return new Result<>(data, null);
    }

    public static <D, E> Result<D, E> ofError(@NotNull E error) {
        return new Result<>(null, error);
    }
}
```

7. Result<D,E> VS Exception

```
public record Result<D, E>(D data, E error) {

    public static <D, E> Result<D, E> ofError(@NotNull E error) {
        return new Result<>(null, error);
    }

    public static <D, E> Result<D, E> of(@NotNull D data) {
        return new Result<>(data, null);
    }
}
```

7. Result<D,E> *VS* Exception

Exception-as-a-Result

7. Result<D,E> *VS* Exception

Exception-as-a-Result

PlayFramework?

7. Result<D,E> VS Exception

play.mvc.results

Class Result

```
java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ java.lang.RuntimeException
              └ play.utils.FastRuntimeException
                  └ play.mvc.results.Result
```

All Implemented Interfaces:

java.io.Serializable

Direct Known Subclasses:

BadRequest, Error, Forbidden, NoResult, NotFound, NotModified, Ok, Redirect, RedirectToStatic, RenderBinary, Unauthorized, WebSocketResult

---

```
public abstract class Result
extends FastRuntimeException
```

Result support

7. Result<D,E> VS Exception

play.mvc.results

Class Result

```
java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ java.lang.RuntimeException
              └ play.utils.FastRuntimeException
                  └ play.mvc.results.Result
```

All Implemented Interfaces:

java.io.Serializable

Direct Known Subclasses:

BadRequest, Error, Forbidden, NoResult, NotFound, NotModified, Ok, Redirect, RedirectToStatic, RenderBinary, Unauthorized, WebSocketResult

---

```
public abstract class Result
extends FastRuntimeException
```

Result support

7. Result<D,E> VS Exception

play.mvc.results

Class Result

```
java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ java.lang.RuntimeException
              └ play.utils.FastRuntimeException
                  └ play.mvc.results.Result
```

All Implemented Interfaces:

java.io.Serializable

Direct Known Subclasses:

BadRequest, Error, Forbidden, NoResult, NotFound, NotModified, Ok, Redirect, RedirectToStatic, RenderBinary, Unauthorized, WebSocketResult

---

```
public abstract class Result
extends FastRuntimeException
```

Result support

https://www.playframework.com/documentation/1.2.1/api/play/mvc/results/Result.html

7. Result<D,E> VS Exception

nile black Dec 11, 2009, 10:06:57 AM

to play-fr...@googlegroups.com

Hi,Everyone!
I felt curious "Why use throw Exception" why not a return ?

public abstract class Result extends RuntimeException

all the return are RuntimeExecption.
Error,Forbidden,NotFound,NotModified,OK,Redirect,RedirectToStatic,RenderBinary,
RenderStatic,RenderTemplate,RenderText,RenderXml,Unauthorized

Nile Black

7. Result<D,E> VS Exception

Guillaume Bort Dec 11, 2009, 3:30:24 PM

to play-fr...@googlegroups.com

At first, it was driven by some experiments, like to be able to auto-redirect when calling an action method, etc ... Now I'm pretty happy with that. But I agree that we could do almost the same with plain return constructs. It would be a little more verbose in some cases however ...

2009/12/11 nile black <nile....@gmail.com>:

•••

> --
>

7. Result<D,E> VS Exception

Guillaume Bort Dec 11, 2009, 3:30:24 PM

to play-fr...@googlegroups.com

At first, it was driven by some experiments, like to be able to auto-redirect when calling an action method, etc ... Now I'm pretty happy with that. But I agree that we could do almost the same with plain return constructs. It would be a little more verbose in some cases however ...

2009/12/11 nile black <nile....@gmail.com>:

•••

> --
>

7. Result<D,E> VS Exception

Guillaume Bort Dec 11, 2009, 3:30:24 PM

to play-fr...@googlegroups.com

At first, it was driven by some experiments, like to be able to auto-redirect when calling an action method, etc ... Now I'm pretty happy with that. But I agree that we could do almost the same with plain return constructs. It would be a little more verbose in some cases however ...

2009/12/11 nile black <nile....@gmail.com>:

•••

> --
>

7. Result<D,E> *VS* Exception

7. Result<D,E> *VS* Exception

7. Result<D,E> VS Exception

Benchmark	(failRatio)	Score		Error	Units
baseline		*2.725*	±	*0.051*	*ns/op*
result		*2.799*	±	*0.018*	*ns/op*
exceptionDriven	*1/1*	*645.310*	±	*2.572*	*ns/op*
exceptionDriven	*1/2*	*647.682*	±	*4.328*	*ns/op*
exceptionDriven	*1/5*	*644.445*	±	*4.775*	*ns/op*
exceptionDriven	*1/10*	*645.337*	±	*3.772*	*ns/op*
exceptionDriven	*1/100*	*641.115*	±	*6.571*	*ns/op*
exceptionDriven	*1/1000*	*642.891*	±	*4.471*	*ns/op*
exceptional	*1/1*	*642.602*	±	*5.368*	*ns/op*
exceptional	*1/2*	*323.427*	±	*2.723*	*ns/op*
exceptional	*1/5*	*130.551*	±	*0.856*	*ns/op*
exceptional	*1/10*	*66.263*	±	*0.629*	*ns/op*
exceptional	*1/100*	*9.216*	±	*0.234*	*ns/op*
exceptional	*1/1000*	*3.512*	±	*0.494*	*ns/op*

7. Result<D,E> VS Exception

Benchmark	(failRatio)	Score		Error	Units
baseline		*2.725*	±	*0.051*	*ns/op*
result		*2.799*	±	*0.018*	*ns/op*
exceptionDriven	*1/1*	*645.310*	±	*2.572*	*ns/op*
exceptionDriven	*1/2*	*647.682*	±	*4.328*	*ns/op*
exceptionDriven	*1/5*	*644.445*	±	*4.775*	*ns/op*
exceptionDriven	*1/10*	*645.337*	±	*3.772*	*ns/op*
exceptionDriven	*1/100*	*641.115*	±	*6.571*	*ns/op*
exceptionDriven	*1/1000*	*642.891*	±	*4.471*	*ns/op*
exceptional	*1/1*	*642.602*	±	*5.368*	*ns/op*
exceptional	*1/2*	*323.427*	±	*2.723*	*ns/op*
exceptional	*1/5*	*130.551*	±	*0.856*	*ns/op*
exceptional	*1/10*	*66.263*	±	*0.629*	*ns/op*
exceptional	*1/100*	*9.216*	±	*0.234*	*ns/op*
exceptional	*1/1000*	*3.512*	±	*0.494*	*ns/op*

7. Result<D,E> VS Exception

Benchmark	(failRatio)	Score		Error	Units
baseline		*2.725*	*±*	*0.051*	*ns/op*
result		*2.799*	*±*	*0.018*	*ns/op*
exceptionDriven	*1/1*	*645.310*	*±*	*2.572*	*ns/op*
exceptionDriven	*1/2*	*647.682*	*±*	*4.328*	*ns/op*
exceptionDriven	*1/5*	*644.445*	*±*	*4.775*	*ns/op*
exceptionDriven	*1/10*	*645.337*	*±*	*3.772*	*ns/op*
exceptionDriven	*1/100*	*641.115*	*±*	*6.571*	*ns/op*
exceptionDriven	*1/1000*	*642.891*	*±*	*4.471*	*ns/op*
exceptional	*1/1*	*642.602*	*±*	*5.368*	*ns/op*
exceptional	*1/2*	*323.427*	*±*	*2.723*	*ns/op*
exceptional	*1/5*	*130.551*	*±*	*0.856*	*ns/op*
exceptional	*1/10*	*66.263*	*±*	*0.629*	*ns/op*
exceptional	*1/100*	*9.216*	*±*	*0.234*	*ns/op*
exceptional	*1/1000*	*3.512*	*±*	*0.494*	*ns/op*

7. Result<D,E> VS Exception

Benchmark	(failRatio)	Score		Error	Units
baseline		2.725	±	0.051	ns/op
result		2.799	±	0.018	ns/op
exceptionDriven	1/1	645.310	±	2.572	ns/op
exceptionDriven	1/2	647.682	±	4.328	ns/op
exceptionDriven	1/5	644.445	±	4.775	ns/op
exceptionDriven	1/10	645.337	±	3.772	ns/op
exceptionDriven	1/100	641.115	±	6.571	ns/op
exceptionDriven	1/1000	642.891	±	4.471	ns/op
exceptional	1/1	642.602	±	5.368	ns/op
exceptional	1/2	323.427	±	2.723	ns/op
exceptional	1/5	130.551	±	0.856	ns/op
exceptional	1/10	66.263	±	0.629	ns/op
exceptional	1/100	9.216	±	0.234	ns/op
exceptional	1/1000	3.512	±	0.494	ns/op

7. Result<D,E> VS Exception

Benchmark	(failRatio)	Score		Error	Units
baseline		*2.725*	±	*0.051*	*ns/op*
result		*2.799*	±	*0.018*	*ns/op*
exceptionDriven	*1/1*	*645.310*	±	*2.572*	*ns/op*
exceptionDriven	*1/2*	*647.682*	±	*4.328*	*ns/op*
exceptionDriven	*1/5*	*644.445*	±	*4.775*	*ns/op*
exceptionDriven	*1/10*	*645.337*	±	*3.772*	*ns/op*
exceptionDriven	*1/100*	*641.115*	±	*6.571*	*ns/op*
exceptionDriven	*1/1000*	*642.891*	±	*4.471*	*ns/op*
exceptional	*1/1*	*642.602*	±	*5.368*	*ns/op*
exceptional	*1/2*	*323.427*	±	*2.723*	*ns/op*
exceptional	*1/5*	*130.551*	±	*0.856*	*ns/op*
exceptional	*1/10*	*66.263*	±	*0.629*	*ns/op*
exceptional	*1/100*	*9.216*	±	*0.234*	*ns/op*
exceptional	*1/1000*	*3.512*	±	*0.494*	*ns/op*

7. Result<D,E> VS Exception

Benchmark	(failRatio)	Score		Error	Units
baseline		*2.725*	±	*0.051*	*ns/op*
result		*2.799*	±	*0.018*	*ns/op*
exceptionDriven	*1/1*	*645.310*	±	*2.572*	*ns/op*
exceptionDriven	*1/2*	*647.682*	±	*4.328*	*ns/op*
exceptionDriven	*1/5*	*644.445*	±	*4.775*	*ns/op*
exceptionDriven	*1/10*	*645.337*	±	*3.772*	*ns/op*
exceptionDriven	*1/100*	*641.115*	±	*6.571*	*ns/op*
exceptionDriven	*1/1000*	*642.891*	±	*4.471*	*ns/op*
exceptional	*1/1*	*642.602*	±	*5.368*	*ns/op*
exceptional	*1/2*	*323.427*	±	*2.723*	*ns/op*
exceptional	*1/5*	*130.551*	±	*0.856*	*ns/op*
exceptional	*1/10*	*66.263*	±	*0.629*	*ns/op*
exceptional	*1/100*	*9.216*	±	*0.234*	*ns/op*
exceptional	*1/1000*	*3.512*	±	*0.494*	*ns/op*

7. Result<D,E> VS Exception

Benchmark	(failRatio)	Score		Error	Units
baseline		*2.725*	±	*0.051*	*ns/op*
result		*2.799*	±	*0.018*	*ns/op*
exceptionDriven	*1/1*	*645.310*	±	*2.572*	*ns/op*
exceptionDriven	*1/2*	*647.682*	±	*4.328*	*ns/op*
exceptionDriven	*1/5*	*644.445*	±	*4.775*	*ns/op*
exceptionDriven	*1/10*	*645.337*	±	*3.772*	*ns/op*
exceptionDriven	*1/100*	*641.115*	±	*6.571*	*ns/op*
exceptionDriven	*1/1000*	*642.891*	±	*4.471*	*ns/op*
exceptional	*1/1*	*642.602*	±	*5.368*	*ns/op*
exceptional	*1/2*	*323.427*	±	*2.723*	*ns/op*
exceptional	*1/5*	*130.551*	±	*0.856*	*ns/op*
exceptional	*1/10*	*66.263*	±	*0.629*	*ns/op*
exceptional	*1/100*	*9.216*	±	*0.234*	*ns/op*
exceptional	*1/1000*	*3.512*	±	*0.494*	*ns/op*

7. Result<D,E> VS Exception

Benchmark	(failRatio)	Score		Error	Units
baseline		*2.725*	±	*0.051*	*ns/op*
result		*2.799*	±	*0.018*	*ns/op*
exceptionDriven	*1/1*	*645.310*	±	*2.572*	*ns/op*
exceptionDriven	*1/2*	*647.682*	±	*4.328*	*ns/op*
exceptionDriven	*1/5*	*644.445*	±	*4.775*	*ns/op*
exceptionDriven	*1/10*	*645.337*	±	*3.772*	*ns/op*
exceptionDriven	*1/100*	*641.115*	±	*6.571*	*ns/op*
exceptionDriven	*1/1000*	*642.891*	±	*4.471*	*ns/op*
exceptional	*1/1*	*642.602*	±	*5.368*	*ns/op*
exceptional	*1/2*	*323.427*	±	*2.723*	*ns/op*
exceptional	*1/5*	*130.551*	±	*0.856*	*ns/op*
exceptional	*1/10*	*66.263*	±	*0.629*	*ns/op*
exceptional	*1/100*	*9.216*	±	*0.234*	*ns/op*
exceptional	*1/1000*	*3.512*	±	*0.494*	*ns/op*

7. Result<D,E> VS Exception

Benchmark	(failRatio)	Score		Error	Units
baseline		*2.725*	±	*0.051*	*ns/op*
result		*2.799*	±	*0.018*	*ns/op*
exceptionDriven	*1/1*	*645.310*	±	*2.572*	*ns/op*
exceptionDriven	*1/2*	*647.682*	±	*4.328*	*ns/op*
exceptionDriven	*1/5*	*644.445*	±	*4.775*	*ns/op*
exceptionDriven	*1/10*	*645.337*	±	*3.772*	*ns/op*
exceptionDriven	*1/100*	*641.115*	±	*6.571*	*ns/op*
exceptionDriven	*1/1000*	*642.891*	±	*4.471*	*ns/op*
exceptional	*1/1*	*642.602*	±	*5.368*	*ns/op*
exceptional	*1/2*	*323.427*	±	*2.723*	*ns/op*
exceptional	*1/5*	*130.551*	±	*0.856*	*ns/op*
exceptional	*1/10*	*66.263*	±	*0.629*	*ns/op*
exceptional	*1/100*	*9.216*	±	*0.234*	*ns/op*
exceptional	*1/1000*	*3.512*	±	*0.494*	*ns/op*

7. Result<D,E> VS Exception

Benchmark	(failRatio)	Score		Error	Units
baseline		*2.725*	±	*0.051*	*ns/op*
result		*2.799*	±	*0.018*	*ns/op*
exceptionDriven	*1/1*	*645.310*	±	*2.572*	*ns/op*
exceptionDriven	*1/2*	*647.682*	±	*4.328*	*ns/op*
exceptionDriven	*1/5*	*644.445*	±	*4.775*	*ns/op*
exceptionDriven	*1/10*	*645.337*	±	*3.772*	*ns/op*
exceptionDriven	*1/100*	*641.115*	±	*6.571*	*ns/op*
exceptionDriven	*1/1000*	*642.891*	±	*4.471*	*ns/op*
exceptional	*1/1*	*642.602*	±	*5.368*	*ns/op*
exceptional	*1/2*	*323.427*	±	*2.723*	*ns/op*
exceptional	*1/5*	*130.551*	±	*0.856*	*ns/op*
exceptional	*1/10*	*66.263*	±	*0.629*	*ns/op*
exceptional	*1/100*	*9.216*	±	*0.234*	*ns/op*
exceptional	*1/1000*	*3.512*	±	*0.494*	*ns/op*

7. Result<D,E> VS Exception

Benchmark	(failRatio)	Score		Error	Units
baseline		*2.725*	±	*0.051*	*ns/op*
result		*2.799*	±	*0.018*	*ns/op*
exceptionDriven	*1/1*	*645.310*	±	*2.572*	*ns/op*
exceptionDriven	*1/2*	*647.682*	±	*4.328*	*ns/op*
exceptionDriven	*1/5*	*644.445*	±	*4.775*	*ns/op*
exceptionDriven	*1/10*	*645.337*	±	*3.772*	*ns/op*
exceptionDriven	*1/100*	*641.115*	±	*6.571*	*ns/op*
exceptionDriven	*1/1000*	*642.891*	±	*4.471*	*ns/op*
exceptional	*1/1*	*642.602*	±	*5.368*	*ns/op*
exceptional	*1/2*	*323.427*	±	*2.723*	*ns/op*
exceptional	*1/5*	*130.551*	±	*0.856*	*ns/op*
exceptional	*1/10*	*66.263*	±	*0.629*	*ns/op*
exceptional	*1/100*	*9.216*	±	*0.234*	*ns/op*
exceptional	*1/1000*	*3.512*	±	*0.494*	*ns/op*

7. Result<D,E> VS Exception

Benchmark	(failRatio)	Score		Error	Units
baseline		*2.725*	±	*0.051*	*ns/op*
result		*2.799*	±	*0.018*	*ns/op*
exceptionDriven	*1/1*	*645.310*	±	*2.572*	*ns/op*
exceptionDriven	*1/2*	*647.682*	±	*4.328*	*ns/op*
exceptionDriven	*1/5*	*644.445*	±	*4.775*	*ns/op*
exceptionDriven	*1/10*	*645.337*	±	*3.772*	*ns/op*
exceptionDriven	*1/100*	*641.115*	±	*6.571*	*ns/op*
exceptionDriven	*1/1000*	*642.891*	±	*4.471*	*ns/op*
exceptional	*1/1*	*642.602*	±	*5.368*	*ns/op*
exceptional	*1/2*	*323.427*	±	*2.723*	*ns/op*
exceptional	*1/5*	*130.551*	±	*0.856*	*ns/op*
exceptional	*1/10*	*66.263*	±	*0.629*	*ns/op*
exceptional	*1/100*	*9.216*	±	*0.234*	*ns/op*
exceptional	*1/1000*	*3.512*	±	*0.494*	*ns/op*

7. Result<D,E> VS Exception

Выводы

7. Result<D,E> *VS* Exception

Выводы

- При обработке исключительных (редких) ситуаций разница между Result и Exception не существенная

7. Result<D,E> VS Exception

Выводы

- При обработке исключительных (редких) ситуаций разница между Result и Exception не существенная
- Result в большинстве случаев оптимизируется JIT-компилятором

7. Result<D,E> VS Exception

Выводы

- При обработке исключительных (редких) ситуаций разница между Result и Exception не существенная
- Result в большинстве случаев оптимизируется JIT-компилятором
- Exception медленные,
  а Exception для happy-path - дурной тон

7.1. NullPointerException

7.1. NullPointerException

java.lang.Boolean.getBoolean

```
public static boolean getBoolean(String name) {
    boolean result = false;
    try {
        result = parseBoolean(System.getProperty(name));
    } catch (IllegalArgumentException | NullPointerException e) {
    }
    return result;
}
```

7.1. NullPointerException

java.lang.Boolean.getBoolean

```
public static boolean getBoolean(String name) {
    boolean result = false;
    try {
        result = parseBoolean(System.getProperty(name));
    } catch (IllegalArgumentException | NullPointerException e) {
    }
    return result;
}
```

7.1. NullPointerException

java.lang.System.getProperty

```
public static String getProperty(String key) {
  checkKey(key);
  @SuppressWarnings("removal")
  SecurityManager sm = getSecurityManager();
  if (sm != null) {
    sm.checkPropertyAccess(key);
  }

  return props.getProperty(key);
}
```

7.1. NullPointerException

java.lang.System.checkKey

```
private static void checkKey(String key) {
    if (key == null) {
        throw new NullPointerException("key can't be null");
    }
    if (key.isEmpty()) {
        throw new IllegalArgumentException("key can't be empty");
    }
}
```

7.1. NullPointerException

Выводы

7.1. NullPointerException

Выводы

- Fail-fast

7.1. NullPointerException

Выводы

- Fail-fast
- NPE - это ошибка в коде,
  который кинул Exception

7.1. NullPointerException

Выводы

- Fail-fast
- NPE - это ошибка в коде,
  который кинул Exception
- IAE - это ошибка в коде,
  который вызвал код,
  который кинул Exception

7.2. Exception VS RuntimeException

7.2. Exception VS RuntimeException

«Какой вы веры:
checked или unchecked?»

Александр Кучук

https://www.youtube.com/watch?v=jl_4g2QxqMg

7.2. Exception VS RuntimeException

«Какой вы веры:
checked или unchecked?»

Александр Кучук

«Я делаю исключения от RuntimeException
и молюсь, чтобы их словил кто-нибудь»

Александр Кучук

https://www.youtube.com/watch?v=jl_4g2QxqMg

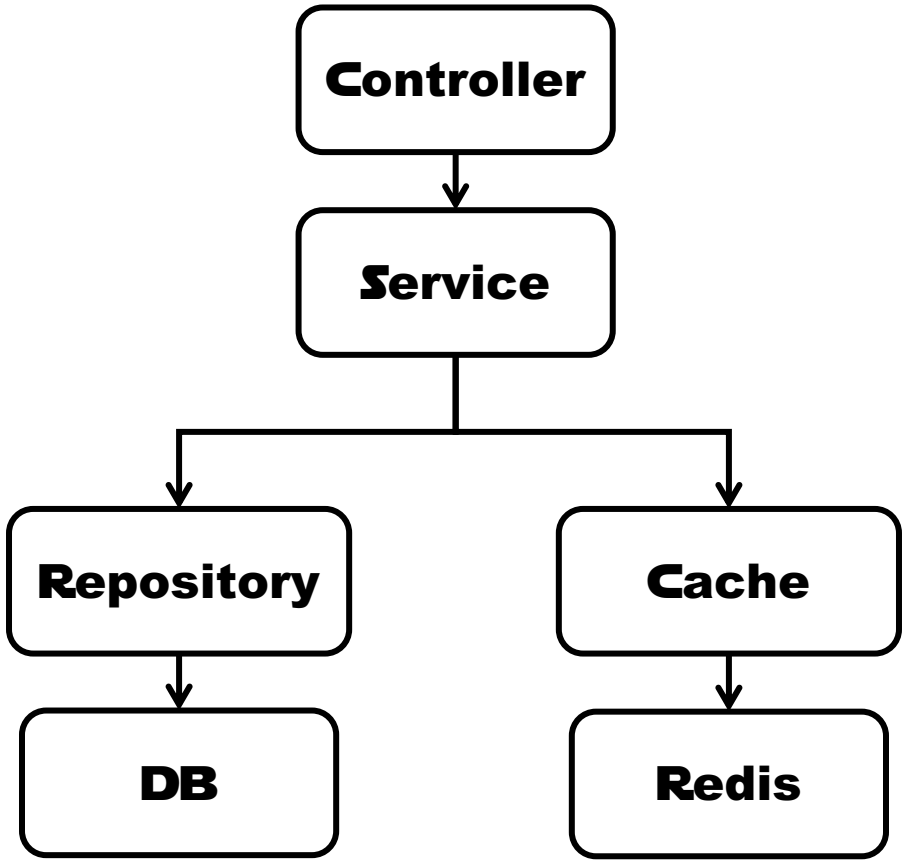
7.2. Exception VS RuntimeException

7.2. Exception VS RuntimeException

Бизнес-исключения наследуются от Exception

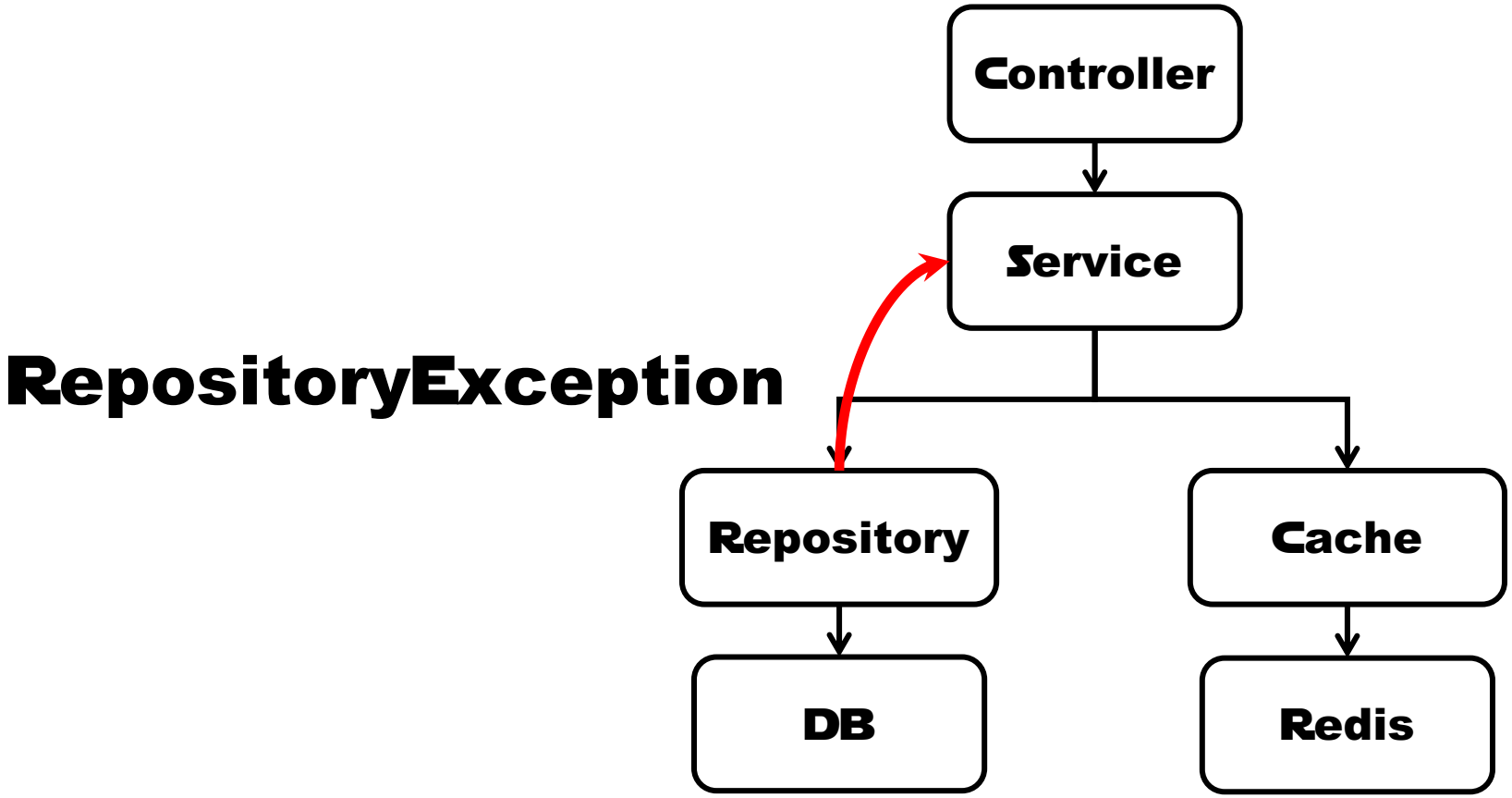
7.2. Exception VS RuntimeException

Бизнес-исключения наследуются от Exception



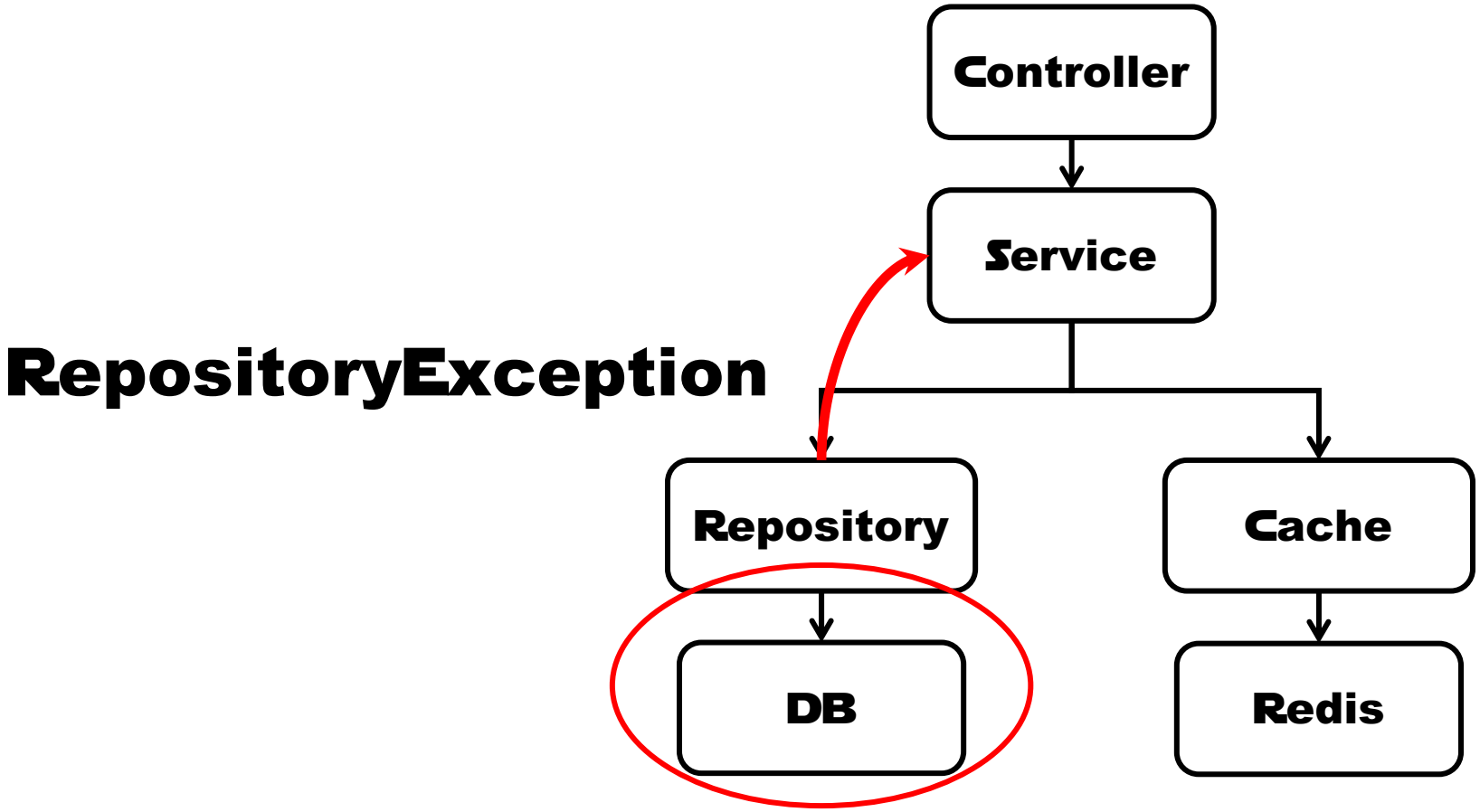
7.2. Exception VS RuntimeException

Бизнес-исключения наследуются от Exception



7.2. Exception VS RuntimeException

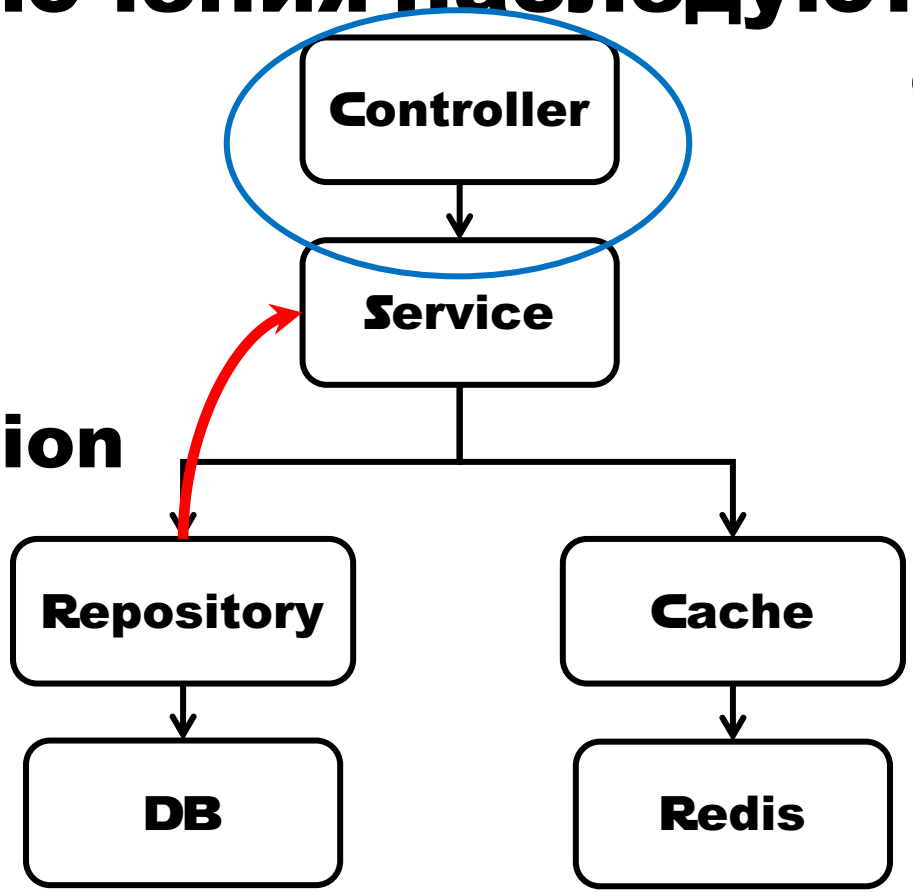
Бизнес-исключения наследуются от Exception



7.2. Exception VS RuntimeException

Бизнес-исключения наследуются от Exception

RepositoryException



7.2. Exception VS RuntimeException

java.io.UnsupportedEncodingException extends IOException

7.2. Exception VS RuntimeException

java.io.UnsupportedEncodingException extends IOException

java.io.UncheckedIOException extends RuntimeException

7.2. Exception VS RuntimeException

java.util.stream.Stream

7.2. Exception VS RuntimeException

java.util.stream.Stream

- filter(Predicate)
- map(Function)
- ...

7.2. Exception VS RuntimeException

java.util.stream.Stream

- filter(Predicate)
- map(Function)
- ...

Optional, Result
и другие монады

8. Null VS Optional

8. Null VS Optional

«Null - это очень быстро»

Неизвестный автор

8. Null VS Optional

«Null - это очень быстро»

Неизвестный автор

«Null Reference -
это моя ошибка на миллиард долларов»

Tony Hoare,
автор ALGOL
и Null Reference

https://en.wikipedia.org/wiki/Tony_Hoare#Research_and_career

8. Null VS Optional

```
private Object findNullable() {
    Object obj = /* ... */
    return obj;
}

Object result = findNull();
if (result != null) {
    /* ... */
} else {
    /* ... */
}
```

8. Null VS Optional

```
private Object findNullable() {
    Object obj = /* ... */
    return obj;
}

Object result = findNull();
if (result != null) {
    /* ... */
} else {
    /* ... */
}
```

```
private Optional<Object> findOptional() {
    Object obj = /* ... */
    return Optional.ofNullable(obj);
}

Optional<Object> result = findOptional();
if (result.isPresent()) {
    /* ... */
} else {
    /* ... */
}
```

8. Null VS Optional

Вариант с Optional

8. Null VS Optional

Вариант с Optional медленнее

8. Null VS Optional

Вариант с Optional медленнее на

8. Null VS Optional

Вариант с Optional медленнее на 1-1.5

8. Null VS Optional

Вариант с Optional медленнее
на 1-1.5 наносекунды

8. Null VS Optional

Вариант с Optional медленнее
на 1-1.5 наносекунды

Вывод: практической выгоды
от использования Null нет!

8.1. ??? *VS* Empty

8.1. ??? VS Empty

Collections

- emptyList
- emptySet
- emptyMap

8.1. ??? VS Empty

```
private List<Object> findAllOrEmpty() {
    if (count == 0) {
        return Collections.emptyList();
    }
    /*  make result list */
    return list;
}

List<Object> objects = findAllOrEmpty();
for (var object : objects) {
        /*  consume objects */
}
```

Collections

- **emptyList**
- **emptySet**
- **emptyMap**

8.2. @NotNull VS @Nullable

8.2. @NotNull VS @Nullable

JetBrains Annotations

8.2. @NotNull VS @Nullable

JetBrains Annotations

```
public static class Container<T> {
    private final T value;

    public Container(@NotNull T value) {
        this.value = value;
    }

    @NotNull
    public T value() {  return value;  }
}
```

https://github.com/JetBrains/java-annotations

8.2. @NotNull VS @Nullable

JetBrains Annotations

```
Container<Object> container = new Container<>( value: null);
```

Passing 'null' argument to parameter annotated as @NotNull

```
ru.gnkoshelev.snowone2023._08._02.NullabilityAnalysis.Container<T>

@Contract(pure = true)
public Container(
    @NotNull T value
)
```

snowone2023

https://github.com/JetBrains/java-annotations

8.2. @NotNull VS @Nullable

JetBrains Annotations

```
Container<Object> container = new Container<>( value: null);
```

Passing 'null' argument to parameter annotated as @NotNull

```
ru.gnkoshelev.snowone2023._08._02.NullabilityAnalysis.Container<T>

@Contract(pure = true)
public Container(
    @NotNull T value
)
```

snowone2023

https://github.com/JetBrains/java-annotations

8.2. @NotNull VS @Nullable

JetBrains Annotations

```
Container<Object> container = new Container<>( value: null);
```

Passing 'null' argument to parameter annotated as @NotNull

```
ru.gnkoshelev.snowone2023._08._02.NullabilityAnalysis.Container<T>

@Contract(pure = true)
public Container(
    @NotNull T value
)
```

snowone2023

https://github.com/JetBrains/java-annotations

8.2. @NotNull VS @Nullable

SpotBugs

8.2. @NotNull VS @Nullable

SpotBugs

- Maven
- Gradle

8.2. @NotNull VS @Nullable

SpotBugs

- Maven
- Gradle
- Intellij IDEA
- Eclipse

8.2. @NotNull VS @Nullable

SpotBugs
- Maven
- Gradle
- Intellij IDEA
- Eclipse

SpotBugs Annotations

https://github.com/spotbugs/spotbugs/tree/master/spotbugs-annotations

8.3. Nullability check

8.3. Nullability check

java.util.Objects

- **requireNonNull**
- **nonNull**
- **...**

8.3. Nullability check

```
/**
 * Returns {@code true} if the provided reference is non-{@code null} otherwise returns {@code false}.
 *
 * @apiNote This method exists to be used as a
 * {@link java.util.function.Predicate}, {@code filter(Objects::nonNull)}
 *
 * @param obj a reference to be checked against {@code null}
 * @return {@code true} if the provided reference is non-{@code null} otherwise {@code false}
 *
 * @see java.util.function.Predicate
 * @since 1.8
 */
public static boolean nonNull(Object obj) {
   return obj != null;
}
```

8.3. Nullability check

```
if (object != null) {
    /* process */
}
```

```
if (Objects.nonNull(object)) {
    /* process */
}
```

8.3. Nullability check

```
if (object != null) {
    /* process */
}
```

```
if (Objects.nonNull(object)) {
    /* process */
}
```

8.3. Nullability check

```
if (object != null) {
    /* process */
}
```

```
if (Objects.nonNull(object)) {
    /* process */
}
```

8.3. Nullability check

```
/**
 * Returns {@code true} if the provided reference is non-{@code null} otherwise returns {@code false}.
 *
 * @apiNote This method exists to be used as a
 * {@link java.util.function.Predicate}, {@code filter(Objects::nonNull)}
 *
 * @param obj a reference to be checked against {@code null}
 * @return {@code true} if the provided reference is non-{@code null} otherwise {@code false}
 *
 * @see java.util.function.Predicate
 * @since 1.8
 */
public static boolean nonNull(Object obj) {
   return obj != null;
}
```

8.3. Nullability check

```
/**
 * Returns {@code true} if the provided reference is non-{@code null} otherwise returns {@code false}.
 *
 * @apiNote This method exists to be used as a
 * {@link java.util.function.Predicate}, {@code filter(Objects::nonNull)}
 *
 * @param obj a reference to be checked against {@code null}
 * @return {@code true} if the provided reference is non-{@code null} otherwise {@code false}
 *
 * @see java.util.function.Predicate
 * @since 1.8
 */
public static boolean nonNull(Object obj) {
    return obj != null;
}
```

8.3. Nullability check

Returns `true` if the provided reference is non-`null` otherwise returns `false`.

Params: `obj` – a reference to be checked against `null`

Returns: `true` if the provided reference is non-`null` otherwise `false`

API Note: This method exists to be used as a `java.util.function.Predicate`, `filter(Objects::nonNull)`

Since: 1.8

See Also: `java.util.function.Predicate`

8.3. Nullability check

Kotlin:

```
!Holders.config?.ibank?.payment?.budget?.deny?.payee?.kpp?.zeroes
```

8.3. Nullability check

Kotlin:

```
!Holders.config?.ibank?.payment?.budget?.deny?.payee?.kpp?.zeroes
```

Снимок экрана 2022-09-27 в 16.56.05

9. Mutable VS Immutable

9. Mutable VS Immutable

«Использование Immutable-структур
требует много памяти и много CPU»

Неизвестный автор

9. Mutable VS Immutable

«Использование Immutable-структур
требует много памяти и много CPU»

Неизвестный автор

«Mutable-структуры -
корень всех Concurrent-ошибок»

Неизвестный автор

9. Mutable *VS* Immutable

java.util.HashMap<K,V>

java.util.concurrent.ConcurrentHashMap<K,V>

9. Mutable VS Immutable

java.util.HashMap<K,V>
java.util.concurrent.ConcurrentHashMap<K,V>

K
- hashCode
- equals

9. Mutable VS Immutable

java.util.Collections

9. Mutable VS Immutable

java.util.Collections

- **unmodifiableCollections**
- **unmodifiableList**
- **unmodifiableSet**
- **unmodifiableMap**
- **...**

9. Mutable VS Immutable

Record

9. Mutable VS Immutable

Record

```
public record ImmutableDataClass(String foo, String bar) {
}

ImmutableDataClass obj = new ImmutableDataClass("foo", "bar");
System.out.println(obj.foo() + " and " + obj.bar());
```

9. Mutable VS Immutable

Immutable

9. Mutable VS Immutable

Immutable

- Read-only структура
- Долго живёт, но редко меняется
- DTO (Data Transfer Object)
- Передаётся между слоями приложения в качестве результата
- Требуется потокобезопасность, а расходы на создание незначительны

10. AutoCloseable vs finalize()

10. AutoCloseable vs finalize()

```
public class CloseableResource implements AutoCloseable {
    public void use() {  /* use resource */  }

    @Override
    public void close() {  /* close resource */  }
}
```

10. AutoCloseable vs finalize()

```
public class CloseableResource implements AutoCloseable {
    public void use() {  /* use resource */  }

    @Override
    public void close() {  /* close resource */  }
}
```

```
try (var resource = new CloseableResource()) {
    resource.use();
}
```

10. AutoCloseable vs finalize()

```
public class FinalizableResource {
    public void use() {  /* use resource */  }

    @Override
    protected void finalize() {  /* close resource */  }
}
```

10. AutoCloseable vs finalize()

```
public class FinalizableResource {
    public void use() { /* use resource */ }

    @Override
    protected void finalize() { /* close resource */ }
}
```

```
var resource = new FinalizableResource();
resource.use();
```

10. AutoCloseable vs finalize()

```
public class FinalizableResource implements Closeable {
    public void use() {  /* use resource */  }

    @Override
    protected void finalize() {  close();  }

    @Override
    public void close() {  /* close resource */  }
}
```

10. AutoCloseable vs finalize()

```
public class FinalizableResource implements Closeable {
    public void use() {  /* use resource */  }

    @Override
    protected void finalize() {  close();  }

    @Override
    public void close() {  /* close resource */  }
}
```

Deprecated c Java 9

10. AutoCloseable vs finalize()

- Не гарантируется момент выполнения finalize(), реализация JVM-специфична
- Дорогое создание и удаление объектов с переопределённым finalize()
- Финализация выполняется в одном выделенном потоке (HotSpot)
- Невозможно перехватить исключение, брошенное из finalize

https://www.infoq.com/articles/Finalize-Exiting-Java/

10. AutoCloseable vs finalize()

```
public class CleanableResource implements AutoCloseable {
   private static final Cleaner cleaner = Cleaner.create();

   private final Cleaner.Cleanable cleanable;

   public CleanableResource() {
      Runnable clean = () -> { /* Cleaning operation without referencing to 'this'. */ };
      cleanable = cleaner.register(this, clean);
   }

   public void use() { /* use resource */ }

   @Override
   public void close() { cleanable.clean(); }
}
```

10. AutoCloseable vs finalize()

```
public class CleanableResource implements AutoCloseable {
   private static final Cleaner cleaner = Cleaner.create();

   private final Cleaner.Cleanable cleanable;

   public CleanableResource() {
      Runnable clean = () -> { /* Cleaning operation without referencing to 'this'. */ };
      cleanable = cleaner.register(this, clean);
   }

   public void use() { /* use resource */ }

   @Override
   public void close() { cleanable.clean(); }
}
```

Лямбду можно,
но случайно можно сослаться на this,
и clean автоматически не отработает

10. AutoCloseable vs finalize()

```
public static void main(String[] args) throws InterruptedException {
    CleanableResource resource = new CleanableResource();
    resource.use();
    resource = null;// Resource is phantom reachable from now
    System.gc();// There is no guarantee but normally GC should start
    Thread.sleep(5_000);
    // Resource cleaned successfully if GC succeeded before time out
}
```

10. AutoCloseable vs finalize()

- Не гарантируется вызов clean до выхода приложения, реализация зависит от JVM
- У каждого Cleaner свой поток очистки
- Игнорируются исключения, брошенные при очистке

https://www.infoq.com/articles/Finalize-Exiting-Java/

Выводы

Выводы

«Только Ситхи всё возводят в абсолют»

Оби-Ван Кеноби

Выводы

DRY - Don't Repeat Yourself

KISS - Keep It Simple, Smart

- https://t.me/chnl_GregoryKoshelev
- https://t.me/chat_GregoryKoshelev

Код

- https://github.com/gnkoshelev/snowone2023