On Variability Debt in Object-Oriented Implementations

Johann Mortara – Philippe Collet – Anne-Marie Pinna-Dery

Université Côte d'Azur, CNRS, I3S, France

TD4ViS @ SPLC '22 – Graz, Austria

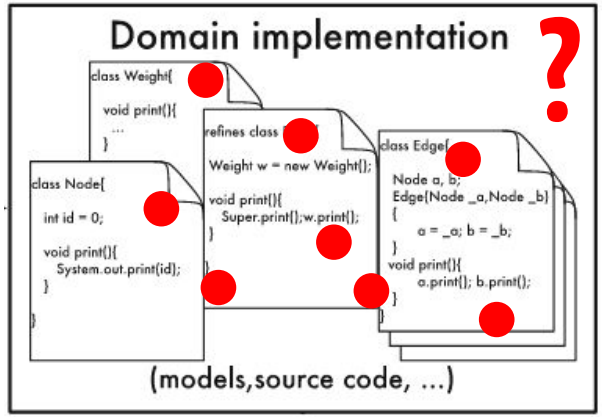
September 13, 2022

OO variability implementations

Undocumented OO variability implementations

OO codebases use OO mechanisms to implement variability in a single codebase

- inheritance
- overloading of methods and constructors
- design patterns



Creation of **complex zones** in the system

⇒ **understanding them is crucial** to comprehend the codebase variability

Variation points and variants

```java
public abstract class Shape {
  public abstract double area();
  public abstract double perimeter(); /*...*/
}
```

```java
public class Circle extends Shape {
  private final double radius;
  // Constructor omitted
  public double area() {
    return Math.PI * Math.pow(radius, 2);
  }
  public double perimeter() {
    return 2 * Math.PI * radius;
  }
}
```

```java
public class Rectangle extends Shape {
  private final double width, length;
  // Constructor omitted
  public double area() {
    return width * length;
  }
  public double perimeter() {
    return 2 * (width + length);
  }
  public void draw(int x, int y) {
  // rectangle at (x, y, width, length)
  }
  public void draw(Point p) {
  // rectangle at (p.x, p.y, width, length)
  }
}
```

Variation points and variants

vp_Shape

```java
public abstract class Shape {
  public abstract double area();
  public abstract double perimeter(); /*...*/
}
```

v_Circle

```java
public class Circle extends Shape {
  private final double radius;
  // Constructor omitted
  public double area() {
    return Math.PI * Math.pow(radius, 2);
  }
  public double perimeter() {
    return 2 * Math.PI * radius;
  }
}
```

v_Rectangle

```java
public class Rectangle extends Shape {
  private final double width, length;
  // Constructor omitted
  public double area() {
    return width * length;
  }
  public double perimeter() {
    return 2 * (width + length);
  }
  public void draw(int x, int y) {
  // rectangle at (x, y, width, length)
  }
  public void draw(Point p) {
  // rectangle at (p.x, p.y, width, length)
  }
}
```

Variation points and variants

vp_Shape

```
1  public abstract class Shape {
2    public abstract double area();
3    public abstract double perimeter(); /*...*/
4  }
```

v_Circle

```
5  public class Circle extends Shape {
6    private final double radius;
7    // Constructor omitted
8    public double area() {
9      return Math.PI * Math.pow(radius, 2);
10   }
11   public double perimeter() {
12     return 2 * Math.PI * radius;
13   }
14 }
```

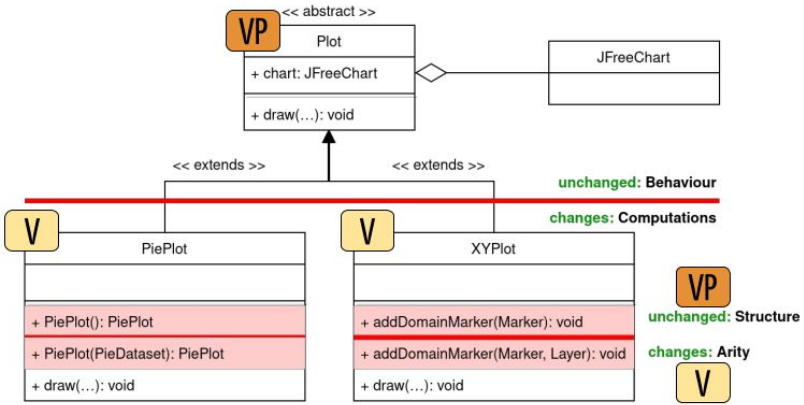
v_Rectangle

```
15 public class Rectangle extends Shape {
16   private final double width, length;
17   // Constructor omitted
18   public double area() {
19     return width * length;
20   }
21   public double perimeter() {
22     return 2 * (width + length);
23   }
24   public void draw(int x, int y) {
25   // rectangle at (x, y, width, length)
26   }
27   public void draw(Point p) {
28   // rectangle at (p.x, p.y, width, length)
29   }
30 }
```

vp_draw

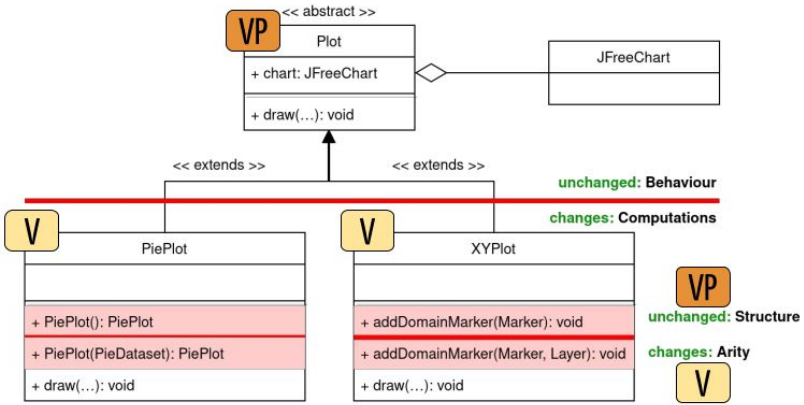
The reality of OO variability implementation

Variability implemented using mechanisms



The reality of OO variability implementation

Variability implemented using mechanisms

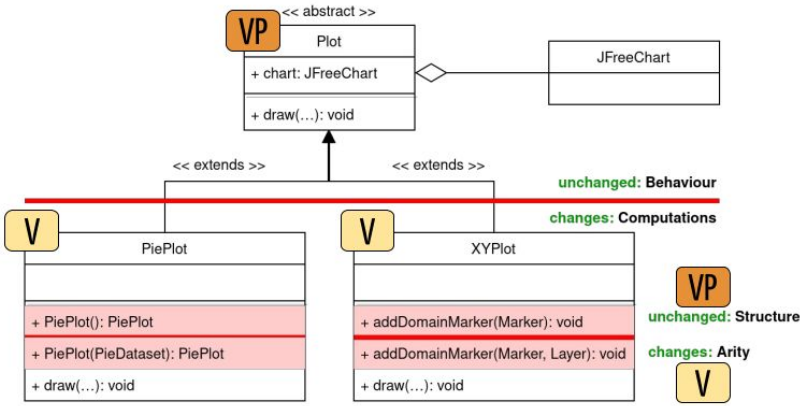


Variability implemented without using mechanisms

Plot
drawPiePlot(): void
drawXYPlot(): void

The reality of OO variability implementation

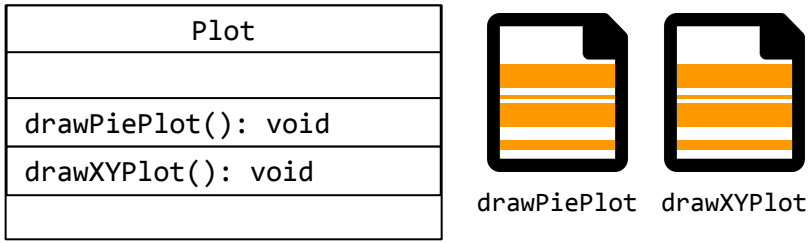
Variability implemented using mechanisms



Duplicated blocks: 3

Duplicated blocks: 2

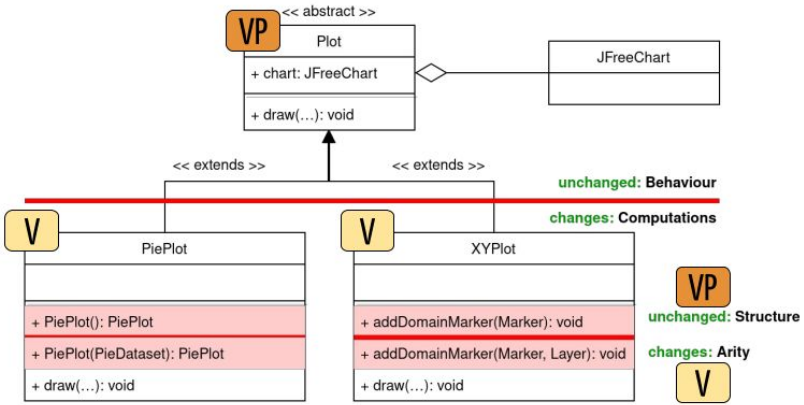
Variability implemented without using mechanisms



Duplicated blocks: 25

The reality of OO variability implementation

Variability implemented using mechanisms



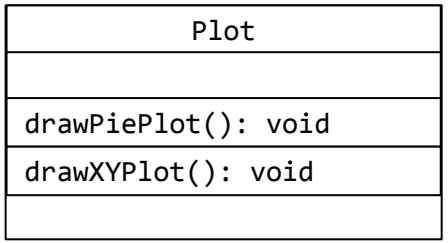
Duplicated blocks: 3

Code coverage: 80%

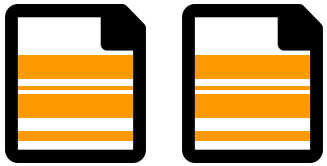
Duplicated blocks: 2

Code coverage: 75%

Variability implemented without using mechanisms



Duplications

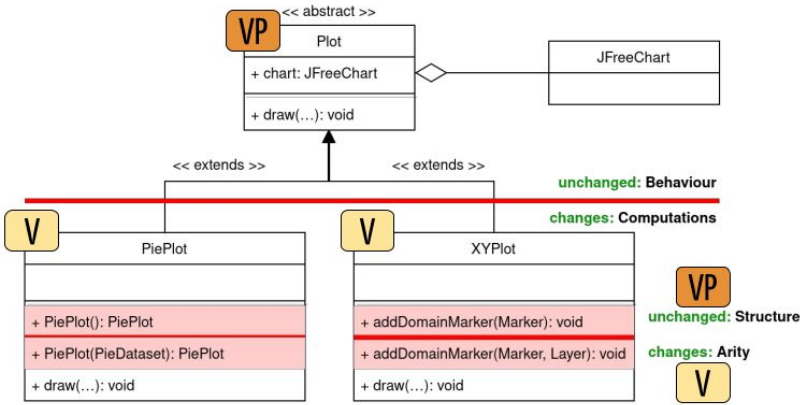


Duplicated blocks: 25

Code coverage: 55%

The reality of OO variability implementation

Variability implemented using mechanisms



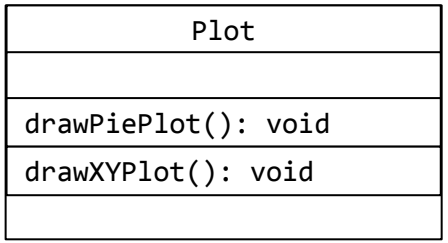
Duplicated blocks: 3

Code coverage: 80%

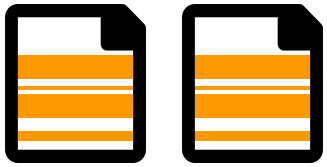
Duplicated blocks: 2

Code coverage: 75%

Variability implemented without using mechanisms



Duplications



drawPiePlot drawXYPlot

Duplicated blocks: 25

Code coverage: 55%

⇒ technical debt

Variability debt

"Technical debt **caused by defects and sub-optimal solutions in the implementation of variability** management in software systems. [...] Variability debt **leads to maintenance and evolution difficulties** to manage families of systems or highly configurable systems."

Lack of knowledge of the implemented variability
+ absence of traceability

Variability model is rarely available
Documentations are rarely up to date and exhaustive
No mapping in the implementations

no known variability
implementation mechanisms
⇒ artifact duplication + ↑ code complexity

Identical implementation mechanisms
for variability and business logic

OO variability implementations are prone to variability debt

Daniele Wolfart, Wesley Klewerton Guez Assunção, and Jabier Martinez. 2021. Variability Debt: Characterization, Causes and Consequences. In XX Brazilian Symposium on Software Quality. 1–10.

How to measure OO variability debt?

Different types of variability debt

- System-level structure quality issues
- Code Duplication
- Lack of tests
- Out-of-date or incomplete documentation
- Architectural antipatterns
- Expensive tests
- Multi-version support
- Old technology in use
- Duplicate documentation
- Poor test of feature interactions

Daniele Wolfart, Wesley Klewerton Guez Assunção, and Jabier Martinez. 2021. Variability Debt: Characterization, Causes and Consequences. In XX Brazilian Symposium on Software Quality. 1–10.

How to measure OO variability debt?

Different types of variability debt applicable to our experimented OO codebases

- System-level structure quality issues
  in the implementation
- Code Duplication
- Lack of tests
- ~~Out-of-date or incomplete documentation~~
  no documentation
- ~~Architectural antipatterns~~
  no information on architecture
- ~~Expensive tests~~
  no information on architecture
- ~~Multi-version support~~
  no information on versions
- ~~Old technology in use~~
  no information on versions
- ~~Duplicate documentation~~
  no documentation
- ~~Poor test of feature interactions~~
  no list of features with mapping

Daniele Wolfart, Wesley Klewerton Guez Assunção, and Jabier Martinez. 2021. Variability Debt: Characterization, Causes and Consequences. In XX Brazilian Symposium on Software Quality. 1–10.

How to measure OO variability debt?

Different types of variability debt

Chosen OO metrics

- System-level structure quality issues → Cognitive complexity

in the implementation

- Code Duplication → Duplicated code blocks
- Lack of tests → Unit tests coverage

Daniele Wolfart, Wesley Klewerton Guez Assunção, and Jabier Martinez. 2021. Variability Debt: Characterization, Causes and Consequences. In XX Brazilian Symposium on Software Quality. 1–10.

Example of code duplication

Duplications can be pure technical debt in classes concentrating variability implementations, but can also be **improperly managed variability implementations**

```
if (isVerticalTickLabels()) {
    anchor = TextAnchor.CENTER_RIGHT;
    rotationAnchor = TextAnchor.CENTER_RIGHT;
    if (edge == RectangleEdge.TOP) {
        angle = Math.PI / 2.0;
    }
    else {
        angle = -Math.PI / 2.0;
    }
}
else {
    if (edge == RectangleEdge.TOP) {
        anchor = TextAnchor.BOTTOM_CENTER;
        rotationAnchor = TextAnchor.BOTTOM_CENTER;
    }
    else {
        anchor = TextAnchor.TOP_CENTER;
        rotationAnchor = TextAnchor.TOP_CENTER;
    }
}
```

refreshTicksHorizontal

```
if (isVerticalTickLabels()) {
    anchor = TextAnchor.BOTTOM_CENTER;
    rotationAnchor = TextAnchor.BOTTOM_CENTER;
    if (edge == RectangleEdge.LEFT) {
        angle = -Math.PI / 2.0;
    }
    else {
        angle = Math.PI / 2.0;
    }
}
else {
    if (edge == RectangleEdge.LEFT) {
        anchor = TextAnchor.CENTER_RIGHT;
        rotationAnchor = TextAnchor.CENTER_RIGHT;
    }
    else {
        anchor = TextAnchor.CENTER_LEFT;
        rotationAnchor = TextAnchor.CENTER_LEFT;
    }
}
```

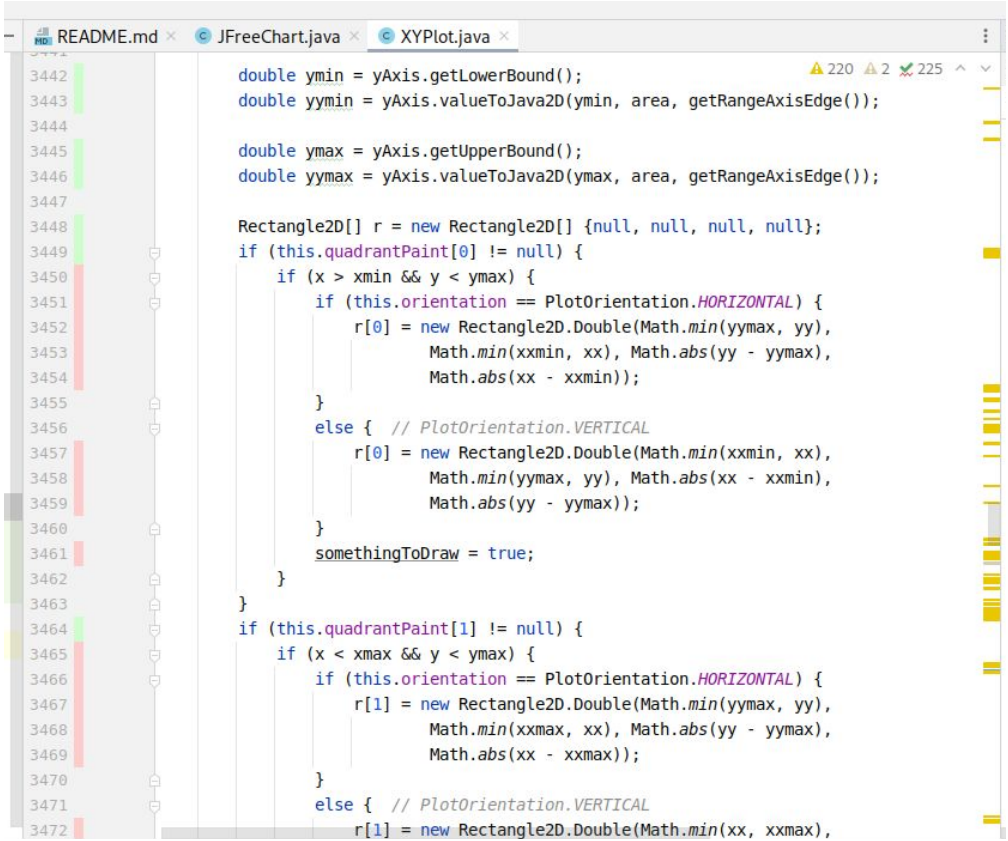
refreshTicksVertical

Common part

Variable part

Common part

(Obvious) Example of low coverage



org.jfree.chart.plot.XYPlot

```
protected Map drawAxes(Graphics2D g2, Rectangle2D plotArea,
        Rectangle2D dataArea, PlotRenderingInfo plotState) {

    AxisCollection axisCollection = new AxisCollection();

    // add domain axes to lists...
    for (CategoryAxis xAxis : this.domainAxes.values()) {
        if (xAxis != null) {
            int index = getDomainAxisIndex(xAxis);
            axisCollection.add(xAxis, getDomainAxisEdge(index));
        }
    }

    // add range axes to lists...
    for (ValueAxis yAxis : this.rangeAxes.values()) {
        if (yAxis != null) {
            int index = findRangeAxisIndex(yAxis);
            axisCollection.add(yAxis, getRangeAxisEdge(index));
        }
    }

    Map axisStateMap = new HashMap();

    // draw the top axes
    double cursor = dataArea.getMinY() - this.axisOffset.calculateTopOutset(
            dataArea.getHeight());
    Iterator iterator = axisCollection.getAxesAtTop().iterator();
    while (iterator.hasNext()) {
        Axis axis = (Axis) iterator.next();
        if (axis != null) {
            AxisState axisState = axis.draw(g2, cursor, plotArea, dataArea,
                    RectangleEdge.TOP, plotState);
            cursor = axisState.getCursor();
            axisStateMap.put(axis, axisState);
        }
    }

    // draw the bottom axes
    cursor = dataArea.getMaxY()
             + this.axisOffset.calculateBottomOutset(dataArea.getHeight());
    iterator = axisCollection.getAxesAtBottom().iterator();
    while (iterator.hasNext()) {
        Axis axis = (Axis) iterator.next();
        if (axis != null) {
            AxisState axisState = axis.draw(g2, cursor, plotArea, dataArea,
                    RectangleEdge.BOTTOM, plotState);
            cursor = axisState.getCursor();
            axisStateMap.put(axis, axisState);
        }
    }

    // draw the left axes
    cursor = dataArea.getMinX()
             - this.axisOffset.calculateLeftOutset(dataArea.getWidth());
    iterator = axisCollection.getAxesAtLeft().iterator();
    while (iterator.hasNext()) {
        Axis axis = (Axis) iterator.next();
        if (axis != null) {
            AxisState axisState = axis.draw(g2, cursor, plotArea, dataArea,
                    RectangleEdge.LEFT, plotState);
            cursor = axisState.getCursor();
            axisStateMap.put(axis, axisState);
        }
    }

    // draw the right axes
    cursor = dataArea.getMaxX()
             + this.axisOffset.calculateRightOutset(dataArea.getWidth());
    iterator = axisCollection.getAxesAtRight().iterator();
    while (iterator.hasNext()) {
        Axis axis = (Axis) iterator.next();
        if (axis != null) {
            AxisState axisState = axis.draw(g2, cursor, plotArea, dataArea,
                    RectangleEdge.RIGHT, plotState);
            cursor = axisState.getCursor();
            axisStateMap.put(axis, axisState);
        }
    }

    return axisStateMap;

}
```

refactor

pure technical debt

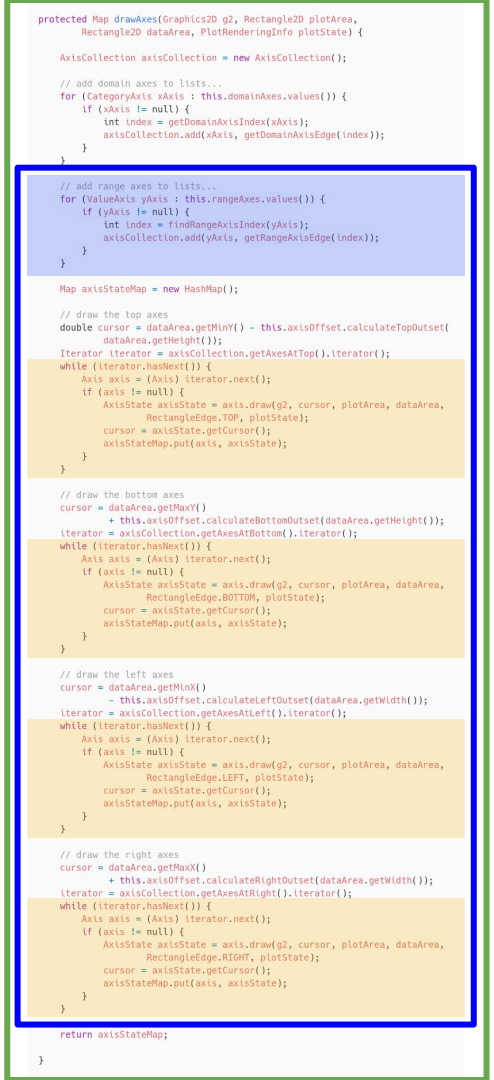
```
protected void addRangesToList(AxisCollection axisCollection) {
    for (ValueAxis axis : this.rangeAxes.values()) {
        if (axis != null) {
            int index = findRangeAxisIndex(axis);
            axisCollection.add(axis, getRangeAxisEdge(index));
        }
    }
}
```

refactor

variability debt

```
private void drawTheAxes(Iterator iterator, double cursor, Graphics2D g2, Rectangle2D plotArea, Rectangle2D
dataArea, PlotRenderingInfo plotState, Map<Axis, AxisState> axisStateMap, RectangleEdge rectangleEdge) {
    while (iterator.hasNext()) {
        Axis axis = (Axis) iterator.next();
        if (axis != null) {
            AxisState axisState = axis.draw(g2, cursor, plotArea, dataArea,
                    rectangleEdge, plotState);
            cursor = axisState.getCursor();
            axisStateMap.put(axis, axisState);
        }
    }
}
```

Example of high cognitive complexity



refactor

core part

```
protected Map<Axis, AxisState> drawAxes(Graphics2D g2, Rectangle2D plotArea,
        Rectangle2D dataArea, PlotRenderingInfo plotState) {

    AxisCollection axisCollection = new AxisCollection();

    // add domain axes to lists...
    for (ValueAxis axis : this.domainAxes.values()) {
        if (axis != null) {
            int axisIndex = findDomainAxisIndex(axis);
            axisCollection.add(axis, getDomainAxisEdge(axisIndex));
        }
    }

    return extractedDraw(g2, plotArea, dataArea, plotState, axisCollection);

}
```

call

variability resolution part

call

variable part

Limitations

- Variable parts in the code are visible but not the variability resolution
- Pure technical and variability debt have the same shape

Thanks for your attention!

Time for questions!