Projektowanie Obiektowe 2

dr inż. Paweł Trajdos

Politechnika Wrocławska, Katedra Systemów i Sieci Komputerowych
Wyb. Wyspianskiego 27, 50-370 Wroclaw

5 lutego 2023

Politechnika Wrocławska

Spis treści

Politechnika Wrocławska

Section 1

Powtórzenie wiadomości

Politechnika Wrocławska

Powtórzenie wiadomości

Czy pamiętasz co znaczą następujące pojęcia?

- Analiza czasownikowo-rzeczownikowa
- Diagram przypadków użycia
- Karta CRC
- Zasada DRY
- Zasada SRP

Politechnika Wrocławska

Section 2

Liczby zespolone – kontynuacja

Zbieranie wymagań

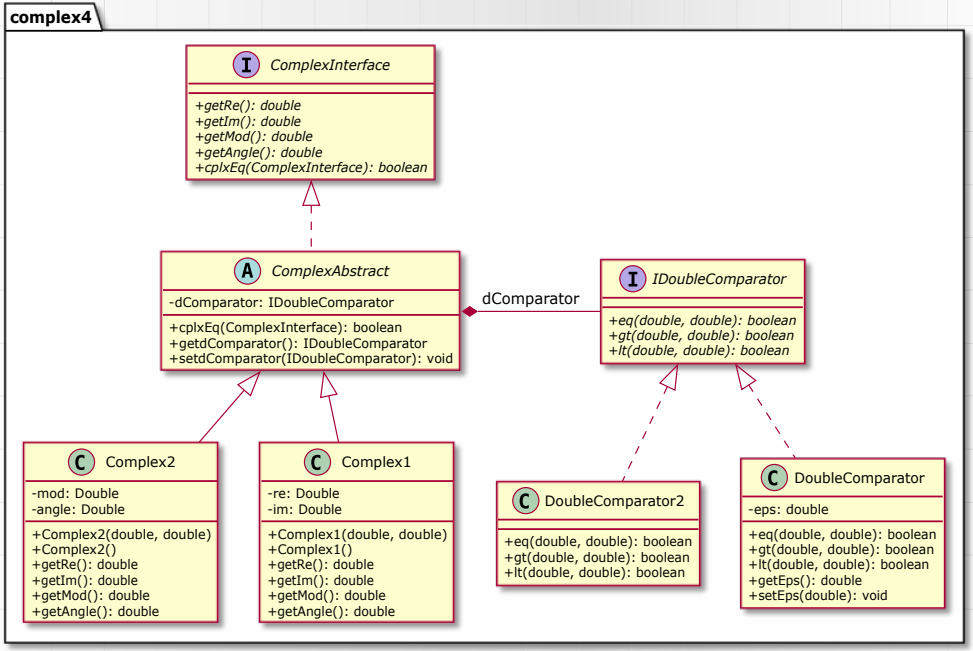
Analiza czasownikowo-rzeczownikowa

Potrzebny jest obiekt umożliwiający wykonywanie operacji na liczbach zespolonych. Chcemy, aby umożliwiał on wyłuskanie części rzeczywistej, części urojonej, kąta oraz modułu liczby zespolonej. Ponadto potrzebujemy możliwości sprawdzenia czy dwie liczby zespolone są sobie równe.
Będzie nam też potrzebna możliwość wprowadzenia różnych porządków częściowych dla liczb zespolonych.

Zbieranie wymagań

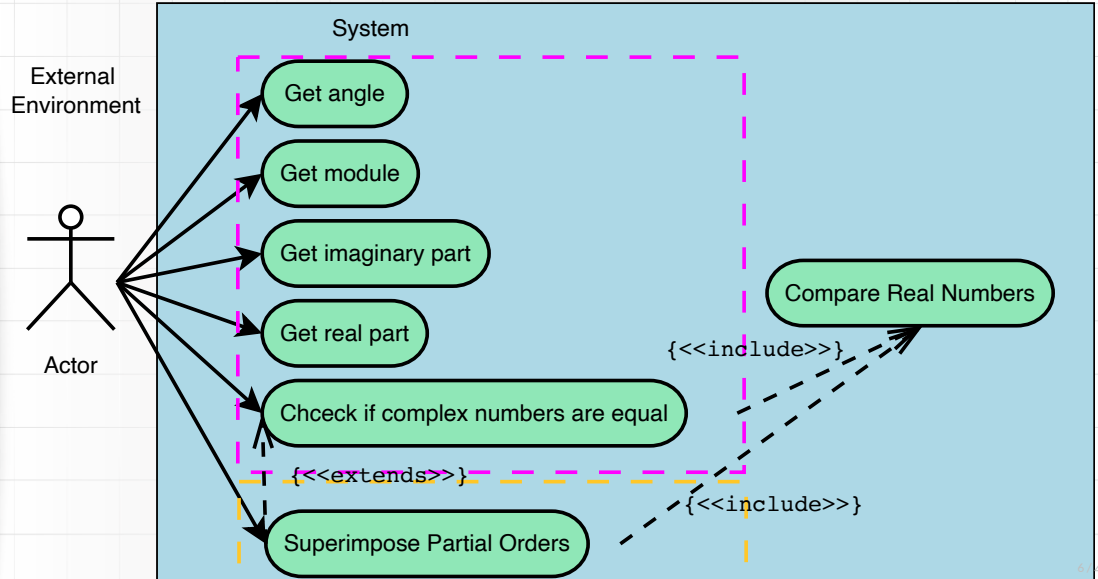
Analiza czasownikowo-rzeczownikowa

Potrzebny jest obiekt umożliwiający wykonywanie operacji na liczbach zespolonych. Chcemy, aby umożliwiał on wyłuskanie części rzeczywistej, części urojonej, kąta oraz modułu liczby zespolonej. Ponadto potrzebujemy możliwości sprawdzenia czy dwie liczby zespolone są sobie równe. Będzie nam też potrzebna możliwość wprowadzenia różnych porządków częściowych dla liczb zespolonych.


complex4
ComplexInterface
+getRe(): double
+getIm(): double
+getMod(): double
+getAngle(): double
+cplxEq(ComplexInterface): boolean
ComplexAbstract
-dComparator: IDoubleComparator
+cplxEq(ComplexInterface): boolean
+getdComparator(): IDoubleComparator
+setdComparator(IDoubleComparator): void
dComparator
IDoubleComparator
+eq(double, double): boolean
+gt(double, double): boolean
+lt(double, double): boolean
Complex2
-mod: Double
-angle: Double
+Complex2(double, double)
+Complex2()
+getRe(): double
+getIm(): double
+getMod(): double
+getAngle(): double
Complex1
-re: Double
-im: Double
+Complex1(double, double)
+Complex1()
+getRe(): double
+getIm(): double
+getMod(): double
+getAngle(): double
DoubleComparator2
+eq(double, double): boolean
+gt(double, double): boolean
+lt(double, double): boolean
DoubleComparator
-eps: double
+eq(double, double): boolean
+gt(double, double): boolean
+lt(double, double): boolean
+getEps(): double
+setEps(double): void

Complex Number

Nowy diagram przypadków użycia



Politechnika Wrocławska

Nowe karty CRC

Classname: ComplexComparator

Superclass: java.util.Comparator
Subclass(es): ComplexReComparator,...

Responisibilities:

- Compare complex numbers
- Superimpose partial orders

Collaboration:
Complex

Politechnika Wrocławska

Nowe karty CRC

Classname: Complex

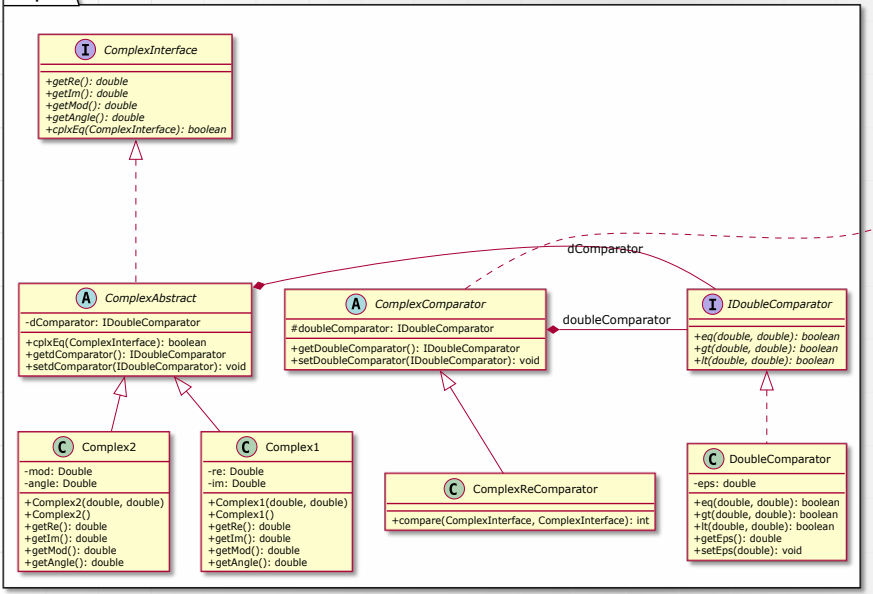
Superclass: none
Subclass(es): none

Responisibilities:

- get real part
- get imaginary part
- get module
- get angle
- compare coplex numbers

Collaboration:

- IDoubleComparator
- ComplexComparator


complex5
ComplexInterface
+getRe(): double
+getIm(): double
+getMod(): double
+getAngle(): double
+cplxEq(ComplexInterface): boolean
ComplexAbstract
-dComparator: IDoubleComparator
+cplxEq(ComplexInterface): boolean
+getdComparator(): IDoubleComparator
+setdComparator(IDoubleComparator): void
ComplexComparator
#doubleComparator: IDoubleComparator
+getDoubleComparator(): IDoubleComparator
+setDoubleComparator(IDoubleComparator): void
IDoubleComparator
+eq(double, double): boolean
+gt(double, double): boolean
+lt(double, double): boolean
dComparator
doubleComparator
Complex2
-mod: Double
-angle: Double
+Complex2(double, double)
+Complex2()
+getRe(): double
+getIm(): double
+getMod(): double
+getAngle(): double
Complex1
-re: Double
-im: Double
+Complex1(double, double)
+Complex1()
+getRe(): double
+getIm(): double
+getMod(): double
+getAngle(): double
ComplexReComparator
+compare(ComplexInterface, ComplexInterface): int
DoubleComparator
-eps: double
+eq(double, double): boolean
+gt(double, double): boolean
+lt(double, double): boolean
+getEps(): double
+setEps(double): void
java.util
+compare(T, T): int
+reversed(): Comparator<T>
+thenComparing(Comparator<? super
+thenComparing(Function<? super T, ?
+thenComparing(Function<? super T, ?
+thenComparingInt(ToIntFunction<? su
+thenComparingLong(ToLongFunction<
+thenComparingDouble(ToDoubleFunct
+reverseOrder(): Comparator<T extend
+naturalOrder(): Comparator<T extend
+nullsFirst(Comparator<? super T>): C
+nullsLast(Comparator<? super T>): C
+comparing(Function<? super T, ? exte
+comparing(Function<? super T, ? exte
+comparingInt(ToIntFunction<? super
+comparingLong(ToLongFunction<? su
+comparingDouble(ToDoubleFunction<

Implementacja

Listing: ComplexComparator.java

```
1  package complex5;
2
3  import java.util.Comparator;
4
5  public abstract class ComplexComparator implements Comparator<ComplexInterface> {
6
7    protected IDoubleComparator doubleComparator;
8
9    public ComplexComparator() {doubleComparator = new DoubleComparator();}
10
11   public IDoubleComparator getDoubleComparator() {return doubleComparator;}
12
13   public void setDoubleComparator(IDoubleComparator doubleComparator) {this.doubleComparator =
       doubleComparator;  }
14 }
```

Implementacja

Listing: ComplexReComparator.java

```
1  package complex5;
2
3  public class ComplexReComparator extends ComplexComparator {
4    @Override
5    public int compare(ComplexInterface o1, ComplexInterface o2) {
6
7      if(doubleComparator.gt(o1.getRe(), o2.getRe())) return 1;
8
9      if(doubleComparator.lt(o1.getRe(), o2.getRe())) return -1;
10
11     return 0;
12   }
13 }
```

Testy

Listing: ComplexReComparatorTest.java

```
12  @Test
13  public void test() {
14    ComplexInterface c1 = new Complex1(3,5);
15    ComplexInterface c2 = new Complex1(1,2);
16    ComplexInterface c3 = new Complex1(4,-1);
17
18    List<ComplexInterface> origList = new ArrayList<>(3);
19    origList.add(c1);origList.add(c2);origList.add(c3);
20
21    ComplexComparator cmp = new ComplexReComparator();
22    Collections.sort(origList, cmp);
23
24    List<ComplexInterface> desiredList = new ArrayList<>(3);
25    desiredList.add(c2);desiredList.add(c1);desiredList.add(c3);
26
27    for(int i=0;i<desiredList.size();i++)
28      if(!desiredList.get(i).cplxEq(origList.get(i)))
29        fail("Complex list sorting failed");
30  }
```

Politechnika Wrocławska

Section 3

Projekt – geometria 2D

Analiza czasownikowo - rzeczownikowa

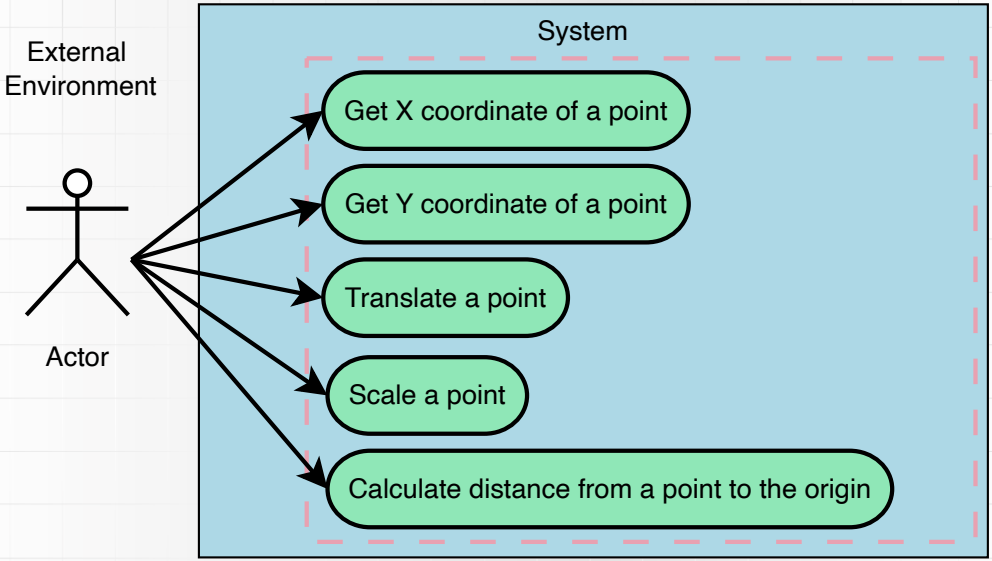
Potrzebny jest obiekt umożliwiający wykonywanie operacji na punktach w przestrzeni dwuwymiarowej. Dla każdego z punktu potrzebna jest operacja pobrania jego współrzędnych, translacja punktu o zadany wektor, skalowanie współrzędnych oraz obliczenie odległości punktu od środka układu współrzędnych.

Politechnika Wrocławska

Analiza czasownikowo - rzeczownikowa

Potrzebny jest obiekt umożliwiający wykonywanie operacji na punktach w przestrzeni dwuwymiarowej. Dla każdego z punktu potrzebna jest operacja pobrania jego współrzędnych, translacja punktu o zadany wektor, skalowanie współrzędnych oraz obliczenie odległości punktu od środka układu współrzędnych.

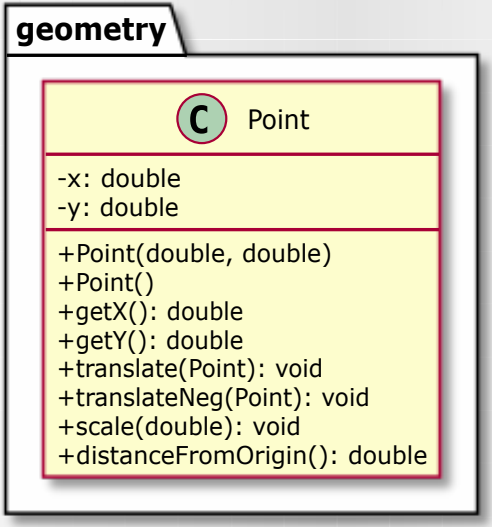
Diagram Przypadków użycia



Karty CRC

Classname: Point	
Superclass: none **Subclass(es):** none	
Responisibilities: Handle Point specific operations. - translate - scale - distance from origin	**Collaboration:** none

Diagram klas



Implementacja

Listing: Point.java

```
1  package geometry;
2
3  public class Point {
4    private double x;
5    private double y;
6
7    public Point(double x, double y) {this.x=x; this.y=y;}
8    public Point() {this(0,0);}
9    public double getX() {return x;}
10   public double getY() {return y;}
11
12   public void translate(Point a) {x+=a.getX(); y+=a.getY(); }
13   public void translateNeg(Point a) {x-=a.getX(); y-=a.getY();}
14   public void scale(double sfactor) {x*=sfactor;y*=sfactor;}
15
16   public double distanceFromOrigin() {
17     return Math.sqrt(x*x + y*y);
18   }
19 }
```

Testy

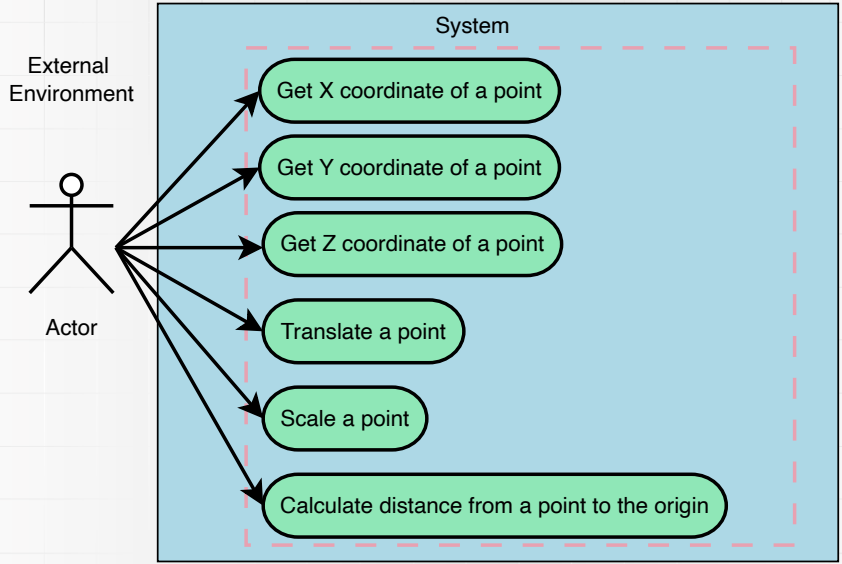
Listing: PointTest.java

```
12  @Test
13  public void test() {
14    double x=1;
15    double y=1;
16
17    Point p1 = new Point();
18    IDoubleComparator dc = new DoubleComparator();
19
20    assertTrue("Default Constructor", dc.eq(p1.getX(), 0) && dc.eq(p1.getY(), 0));
21    assertTrue("Distance1: ", dc.eq(p1.distanceFromOrigin(), 0));
22
23    Point p2 = new Point(x,y);
24    assertTrue("Constructor", dc.eq(p2.getX(), x) && dc.eq(p2.getY(), y));
25    assertTrue("Distance2: ", dc.eq(p2.distanceFromOrigin(), Math.sqrt(2.0)));
26  }
```

Subsection 1

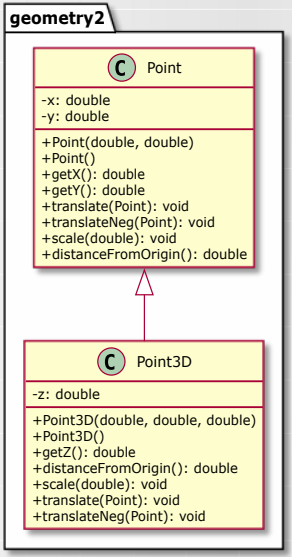
Dodajemy trzeci wymiar

Diagram Przypadków użycia



Politechnika Wrocławska

Diagram klas



Karty CRC

Classname: Point3D	
Superclass: Point **Subclass(es):** none	
Responisibilities: Handle Point specific operations for a 3D point. - translate - scale - distance from origin	**Collaboration:** none

Implementacja

Listing: Point3D.java

```
1  package geometry2;
2
3  public class Point3D extends Point {
4
5    private double z;
6
7    public Point3D(double x, double y, double z) {super(x,y);this.z=z;}
8    public Point3D() {this(0,0,0);}
9
10   public double getZ() {return z; }
11
12   @Override
13   public double distanceFromOrigin() {
14     double s =super.distanceFromOrigin();
15     return Math.sqrt(s*s + z*z);
16   }
17
18   @Override
19   public void scale(double sfactor) {
20     super.scale(sfactor);
21     z*=sfactor;
22   }
```

Implementacja

Listing: Point3D.java

```
24    @Override
25    public void translate(Point a) {
26      super.translate(a);
27      if(a instanceof Point3D)z+= ((Point3D) a).getZ();//!
28    }
29
30    @Override
31    public void translateNeg(Point a) {
32      super.translateNeg(a);
33      if(a instanceof Point3D)z-= ((Point3D) a).getZ();//!
34    }
35  }
```

Testy

Listing: Point3DTest.java

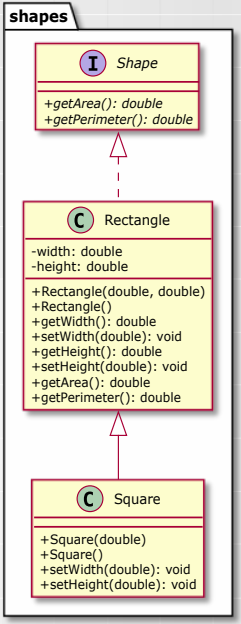
```java
package geometry2;

import static org.junit.Assert.assertTrue;

import org.junit.Test;

import complex4.DoubleComparator;
import complex4.IDoubleComparator;

public class Point3DTest {

  @Test
  public void test() {
    double x=0;
    double y=0;
    double z=1;

    Point p1 = new Point3D(x, y, z);
    IDoubleComparator dc = new DoubleComparator();

    assertTrue("Distance", dc.eq(p1.distanceFromOrigin(), 1));//Incompatible with Point
  }
```

Section 4

Liskov Substitution Principle

Liskov Substitution Principle

Funkcje, które używają wskaźników lub referencji do klas bazowych, muszą być w stanie używać również obiektów klas dziedziczących po klasach bazowych, bez dokładnej znajomości tychobiektów


shapes
Shape
+getArea(): double
+getPerimeter(): double
Rectangle
-width: double
-height: double
+Rectangle(double, double)
+Rectangle()
+getWidth(): double
+setWidth(double): void
+getHeight(): double
+setHeight(double): void
+getArea(): double
+getPerimeter(): double
Square
+Square(double)
+Square()
+setWidth(double): void
+setHeight(double): void

Przykład

Kod

Listing: Shape.java

```
1 package shapes;
2
3 public interface Shape {
4
5   public double getArea();
6   public double getPerimeter();
7
8 }
```

Przykład

Kod

Listing: Rectangle.java

```java
package shapes;

public class Rectangle implements Shape {
  private double width;
  private double height;

  public Rectangle(double w, double h) {width=w; height=h;}
  public Rectangle() {this(1,1);  }

  public double getWidth() {return width; }
  public void setWidth(double width) {this.width = width; }

  public double getHeight() { return height;  }
  public void setHeight(double height) {this.height = height; }

  @Override
  public double getArea() {return width*height;}

  @Override
  public double getPerimeter() {return 2*width + 2*height;}
}
```

Przykład

Kod

Listing: Square.java

```
1  package shapes;
2
3  public class Square extends Rectangle {
4
5    public Square(double width) {super(width,width);}
6    public Square() {this(1);}
7
8    @Override
9    public void setWidth(double width) {
10     super.setWidth(width);
11     super.setHeight(width);
12   }
13
14   @Override
15   public void setHeight(double height) {
16     super.setHeight(height);
17     super.setWidth(height);
18   }
19 }
```

Przykład

Testy

Listing: RectangleTest.java

```
12  @Test
13  public void test() {
14    double w=2;
15    double h=3;
16    Rectangle rec = new Rectangle();
17    rec.setWidth(w);
18    rec.setHeight(h);
19    IDoubleComparator dc = new DoubleComparator();
20
21    assertTrue("Area", dc.eq(rec.getArea(), 2*3));
22    assertTrue("Perimeter", dc.eq(rec.getPerimeter(), 2*2 + 2*3));
23  }
```

Przykład

Testy

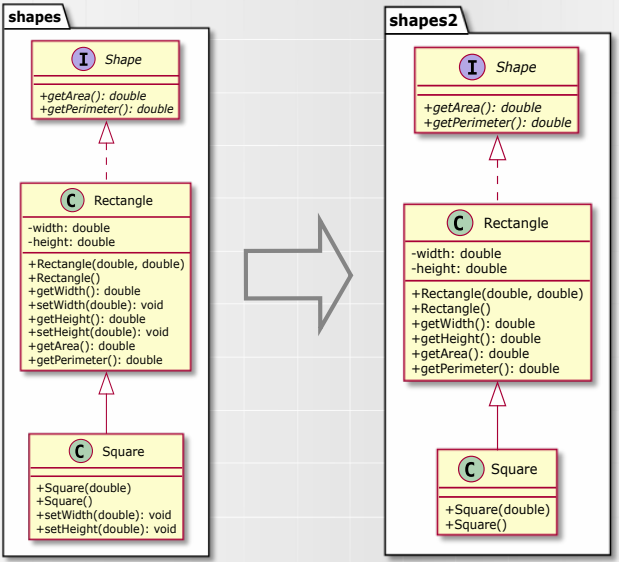
Listing: SquareTest.java

```
10  @Test
11  public void test() {
12    double w=2;
13    double h=3;
14    Rectangle rec = new Rectangle();
15    rec.setWidth(w);
16    rec.setHeight(h);
17    IDoubleComparator dc = new DoubleComparator();
18
19    Rectangle sq = new Square();
20    sq.setWidth(w);
21    sq.setHeight(h);
22
23    //assertTrue("Area", dc.eq(rec.getArea(), sq.getArea()));//Fail!
24    //assertTrue("Perimeter", dc.eq(rec.getPerimeter(), sq.getPerimeter()));//Fail!
25  }
```

Czy da się to naprawić?

Naprawiamy

Sposób 1



Naprawiamy

Sposób 2

Listing: Rectangle.java

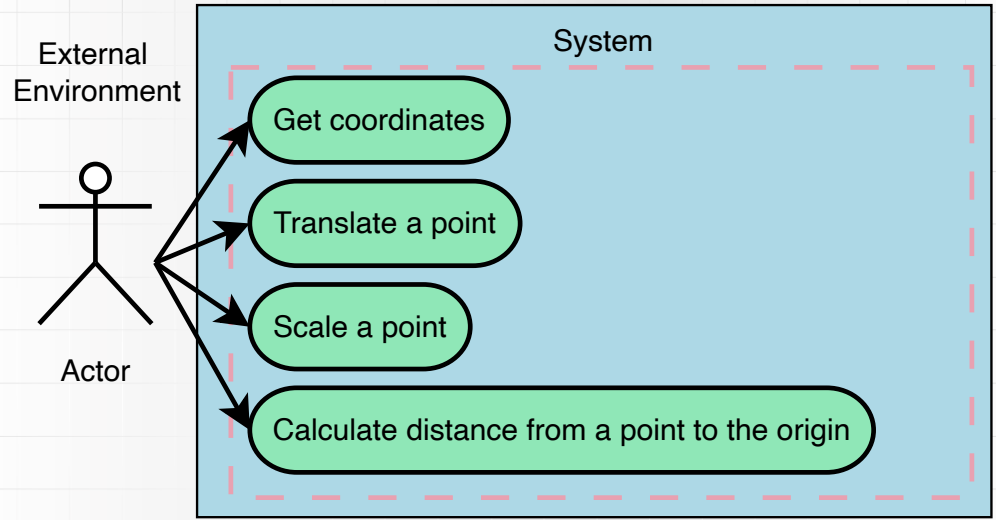
```java
package shapes3;

public final class Rectangle implements Shape {
  private double width;
  private double height;

   public Rectangle(double w, double h) {width=w; height=h;}
   public Rectangle(double w) {this(w,w); }
   public Rectangle() {this(1,1); }

  public double getWidth() {return width; }
  public void setWidth(double width) {this.width = width; }
  public double getHeight() { return height;  }
  public void setHeight(double height) {this.height = height; }
  @Override
  public double getArea() {return width*height;}
  @Override
  public double getPerimeter() {return 2*width + 2*height;}
}
```

Politechnika Wrocławska

Section 5

Geometria 2D – Naprawiamy

Diagram Przypadków użycia



Karty CRC

Classname: IPoint

Superclass: none
Subclass(es): Point

Responisibilities:
Handle Point specific operations.

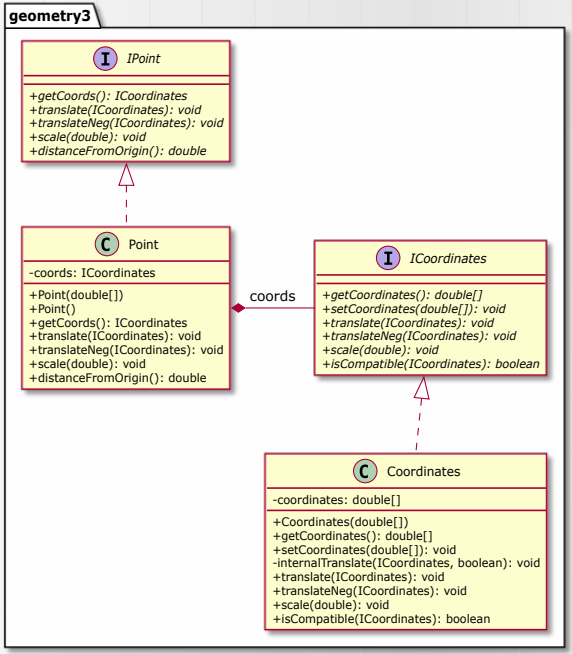
- translate
- scale
- distance from origin

Collaboration:
ICoordinates

Politechnika Wrocławska

Karty CRC

Classname: ICoordinates	
Superclass: none **Subclass(es):** Coordinates	
Responisibilities: Handle Coordinate specific operations. - translate - scale	**Collaboration:** none


geometry3
IPoint
+getCoords(): ICoordinates
+translate(ICoordinates): void
+translateNeg(ICoordinates): void
+scale(double): void
+distanceFromOrigin(): double
Point
-coords: ICoordinates
+Point(double[])
+Point()
+getCoords(): ICoordinates
+translate(ICoordinates): void
+translateNeg(ICoordinates): void
+scale(double): void
+distanceFromOrigin(): double
coords
ICoordinates
+getCoordinates(): double[]
+setCoordinates(double[]): void
+translate(ICoordinates): void
+translateNeg(ICoordinates): void
+scale(double): void
+isCompatible(ICoordinates): boolean
Coordinates
-coordinates: double[]
+Coordinates(double[])
+getCoordinates(): double[]
+setCoordinates(double[]): void
-internalTranslate(ICoordinates, boolean): void
+translate(ICoordinates): void
+translateNeg(ICoordinates): void
+scale(double): void
+isCompatible(ICoordinates): boolean

Politechnika Wrocławska

Implementacja

Listing: IPoint.java

```
1  package geometry3;
2
3  public interface IPoint {
4
5    public ICoordinates getCoords();
6    public void translate(ICoordinates coords)throws Exception;
7    public void translateNeg(ICoordinates coords)throws Exception;
8    public void scale(double sfactor);
9    public double distanceFromOrigin();
10
11 }
```

Implementacja

Listing: ICoordinates.java

```
1  package geometry3;
2
3  public interface ICoordinates {
4    public double[] getCoordinates();
5    public void setCoordinates(double[] coords);
6    public void translate(ICoordinates coords)throws Exception;
7    public void translateNeg(ICoordinates coords)throws Exception;
8    public void scale(double scaleFactor);
9    public boolean isCompatible(ICoordinates coords);
10 }
```

Implementacja

Listing: Coordinates.java

```
1  package geometry3;
2  import java.util.Arrays;
3  public class Coordinates implements ICoordinates {
4
5    private double[] coordinates;
6
7    public Coordinates(double[] coordinates) {this.setCoordinates(coordinates);}
8
9    @Override
10   public double[] getCoordinates() {return Arrays.copyOf(coordinates, coordinates.length);  }
11
12   @Override
13   public void setCoordinates(double[] coords) {coordinates = Arrays.copyOf(coords,
       coords.length);}
14
15   private void internalTranslate(ICoordinates coords, boolean positive)throws Exception{
16     if(!this.isCompatible(coords))throw new Exception("Incompatible Coordinates");
17     double[] transCoords = coords.getCoordinates();
18     for(int i=0;i<coordinates.length;i++)
19       coordinates[i]+= positive? transCoords[i]: -transCoords[i];
20   }
```

Implementacja

Listing: Coordinates.java

```
22  @Override
23  public void translate(ICoordinates coords) throws Exception {internalTranslate(coords, true);}
24
25  @Override
26  public void translateNeg(ICoordinates coords) throws Exception {internalTranslate(coords,
        false);}
27
28  @Override
29  public void scale(double scaleFactor) {
30    for(int i=0;i<coordinates.length;i++)
31      coordinates[i]*=scaleFactor;
32  }
33
34  @Override
35  public boolean isCompatible(ICoordinates coords) {
36    if(this.coordinates.length != coords.getCoordinates().length)
37      return false;
38    return true;
39  }
```

Implementacja

Listing: Coordinates.java

```
41   @Override
42   public boolean equals(Object obj) {
43     if( super.equals(obj))return true;
44
45     if(!( obj instanceof ICoordinates))
46       return false;
47
48     Coordinates tmpCoords = (Coordinates) obj;
49
50     for(int i=0;i<coordinates.length;i++)
51       if(! new Double(coordinates[i]).equals(new Double(tmpCoords.getCoordinates()[i])))
52         return false;
53
54     return true;
55   }
56 }
```

Implementacja

Listing: Point.java

```java
package geometry3;

public class Point implements IPoint {

  private ICoordinates coords;

  public Point(double[] coords) {
    this.coords = new Coordinates(coords);
  }
  public Point(){this(new double[] {0,0});}

  public ICoordinates getCoords() {
    return  new Coordinates(coords.getCoordinates());
  }

  public void translate(ICoordinates coords)throws Exception {
    this.coords.translate(coords);
  }
  public void translateNeg(ICoordinates coords)throws Exception {
    this.coords.translateNeg(coords);
  }
```

Implementacja

Listing: Point.java

```
23    @Override
24    public void scale(double sfactor) {coords.scale(sfactor);}
25
26    public double distanceFromOrigin() {
27      double sum=0;
28      double[] cords = coords.getCoordinates();
29
30      for(int i=0;i<cords.length;i++)
31        sum += cords[i]*cords[i];
32
33      return Math.sqrt(sum);
34    }
35  }
```

Testowanie

Listing: CoordinatesTest.java

```
9   @Test
10  public void test() {
11
12    double[] cordsArray = new double[] {1,3,5,7,8};
13    double[] orgCordsArray = Arrays.copyOf(cordsArray, cordsArray.length);
14    ICoordinates cords = new Coordinates(cordsArray);
15    ICoordinates testCords = new Coordinates(orgCordsArray);
16    double[] retCoords = cords.getCoordinates();
17
18    for(int i=0;i<retCoords.length;i++)
19      if(orgCordsArray[i] != retCoords[i])
20        fail("Returned Coords failure");
```

Testowanie

Listing: CoordinatesTest.java

```
22      try {
23        cords.translate(cords);
24        retCoords = cords.getCoordinates();
25        for(int i=0;i<retCoords.length;i++)
26          if(2*orgCordsArray[i] != retCoords[i])
27            fail("Returned Coords failure");
28
29        cords.translateNeg(testCords);
30        assertTrue("Negative Translation", cords.equals(testCords));
31
32        cords.scale(2.0);
33        retCoords = cords.getCoordinates();
34        for(int i=0;i<retCoords.length;i++)
35          if(2*orgCordsArray[i] != retCoords[i])
36            fail("Returned Coords failure");
37
38      } catch (Exception e) {fail("Coord translation: Exception");}
```

Testowanie

Listing: CoordinatesTest.java

```
40      ICoordinates tmpCords = new Coordinates(new double[] {3,4});
41      try {
42        cords.translate(tmpCords);
43        fail("No Exception");
44      } catch (Exception e) {assertTrue(true);}
45    }
46  }
```

Testowanie

Listing: PointTest.java

```
9   @Test
10  public void test() {
11    double x=1, y=1;
12    IPoint p1 = new Point(new double[] {x,y});
13    double[] coords = p1.getCoords().getCoordinates();
14    DoubleComparator dc = new DoubleComparator();
15
16    assertTrue("Coordinates: ", dc.eq(coords[0], x) && dc.eq(coords[1], y));
17
18    double trueDist = Math.sqrt(x*x+ y*y);
19    assertTrue("Distance: ", dc.eq(p1.distanceFromOrigin(), trueDist));
```

Testowanie

Listing: PointTest.java

```
21      try {
22        p1.translate(p1.getCoords());
23        coords = p1.getCoords().getCoordinates();
24        assertTrue("Translation: ", dc.eq(coords[0], 2*x) && dc.eq(coords[1], 2*y));
25
26        p1.scale(2.0);
27        coords = p1.getCoords().getCoordinates();
28        assertTrue("Scale: ", dc.eq(coords[0], 4*x) && dc.eq(coords[1], 4*y));
29
30        p1.translateNeg(p1.getCoords());
31        coords = p1.getCoords().getCoordinates();
32        assertTrue("Translate Neg: ", dc.eq(coords[0], 0) && dc.eq(coords[1], 0));
33      } catch (Exception e) {
34        fail("Exception Caught");
35      }
36    }
37  }
```

Politechnika Wrocławska

Section 6

Law of Demeter

Low Coupling Principle

Deleguj odpowiedzialności tak, aby zachować jak najmniejsza liczbę powiązań pomiędzy klasami.
Klasy A i B są ze sobą powiązane, gdy:

- obiekt typu A ma atrybuty typu B lub typu C związanego z B;
- obiekt typu A wywołuje metody obiektu typu B;
- obiekt typu A ma metodę związaną z typem B;
- obiekt typu A dziedziczy po typie B.

Klasa ze zbyt wieloma powiązaniami prawdopodobnie jest przeciążona.

Law of Demeter

Zasada minimalnej wiedzy, zasada ograniczania interakcji

Jedna z odmian zasady luźnych powiązań.

Obiekty nie powinny wiedzieć zbyt wiele o obiektach z którymi nie współpracują.

Rozmawiaj tylko z przyjaciółmi, nie rozmawiaj z obcymi.

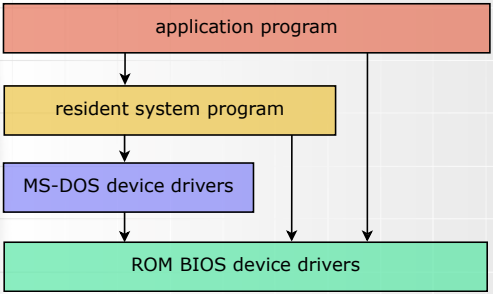
Praktyczne znaczenie

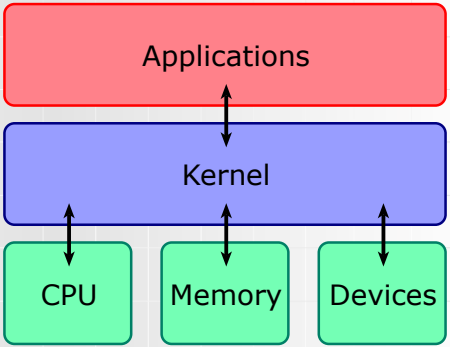
Możesz wywoływać metody z obiektów:

- przekazanych jako argument,
- lokalnie stworzonych,
- będących podobiektami obiektu,
- globalnych,
- statycznych.

Law of Demeter

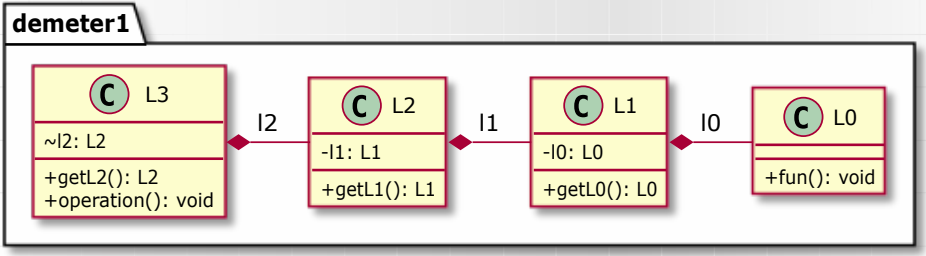
Przykład





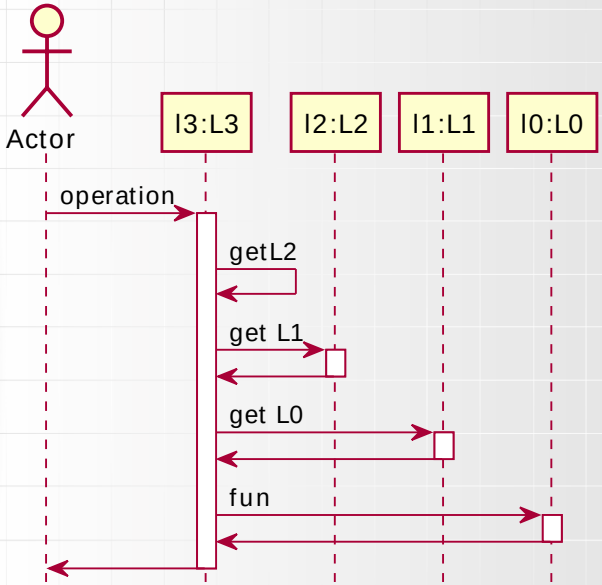
Law of Demeter

Naruszenie



Law of Demeter

Naruszenie



Law of Demeter

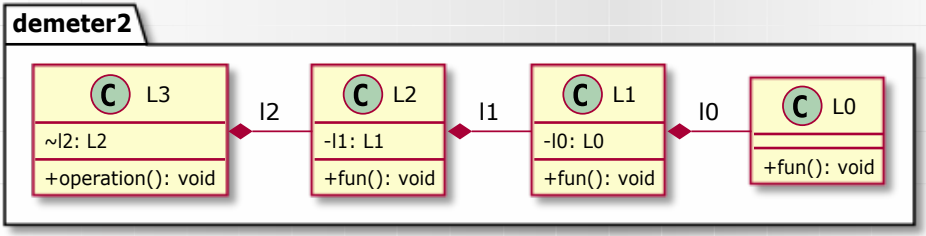
Naruszenie

Listing: L3.java

```
1  package demeter1;
2
3  public class L3 {
4    L2 l2=new L2();
5
6    public L2 getL2() { return l2; }
7
8    public void operation() { getL2().getL1().getL0().fun(); }
9
10 }
```

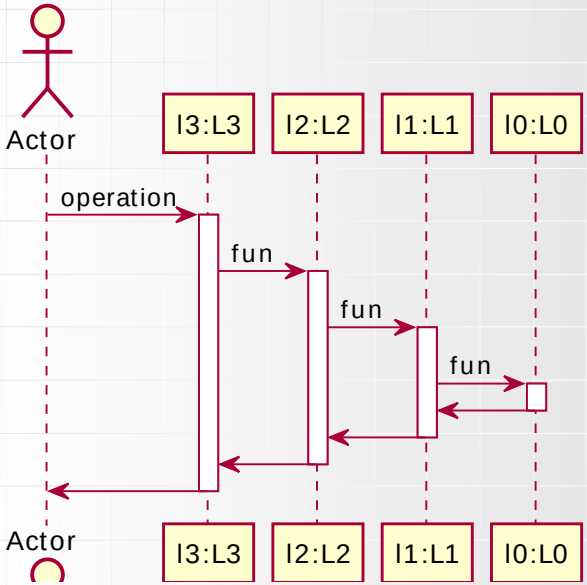
Law of Demeter

Poprawnie



Law of Demeter

Poprawnie



Law of Demeter

Poprawnie

Listing: L0.java

```
1 package demeter2;
2
3 public class L0 {
4   public void fun() {}
5 }
```

Listing: L1.java

```
3 public class L1 {
4   private L0 l0 = new L0();
5   public void fun() { l0.fun();}
6 }
```

Listing: L2.java

```
3 public class L2 {
4   private L1 l1 =  new L1();;
5   public void fun() { l1.fun(); }
6 }
```

Law of Demeter

Poprawnie

Listing: L3.java

```
3 public class L3 {
4   L2 l2=new L2();
5   public void operation() { l2.fun(); }
6 }
```

Politechnika Wrocławska

Section 7

Małe podsumowanie

Politechnika Wrocławska

Czego już się nauczliśmy

- SOLID:
  - S Single Responsibility Principle
  - O Open–close principle
  - L Liskov substitution principle
  - I Interface segregation principle
  - D Dependency inversion principle
- Keep it simple, stupid. (KISS)
- Don’t Repeat Yourself (DRY)

Projektowanie Obiektowe 2

dr inż. Paweł Trajdos

Politechnika Wrocławska, Katedra Systemów i Sieci Komputerowych
Wyb. Wyspianskiego 27, 50-370 Wroclaw

5 lutego 2023