Politechnika Wrocławska

Programowanie Obiektowe – Projektowanie Obiektowe

dr inż. Paweł Trajdos

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

5 lutego 2023

Politechnika Wrocławska

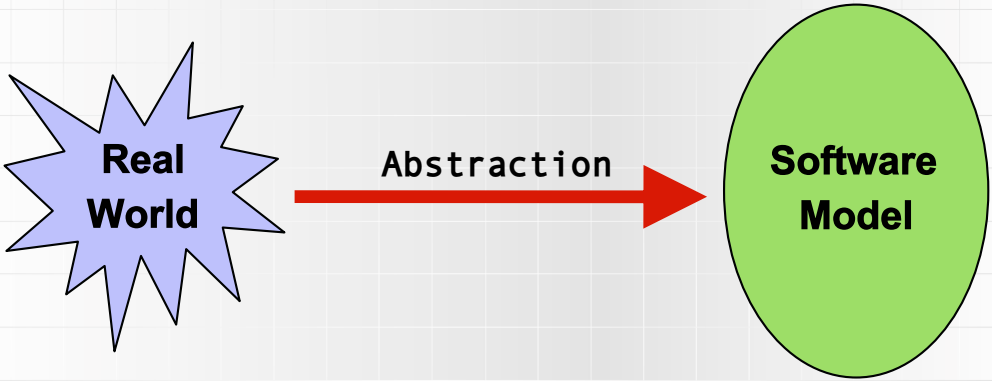
Spis treści

Politechnika Wrocławska

Section 1

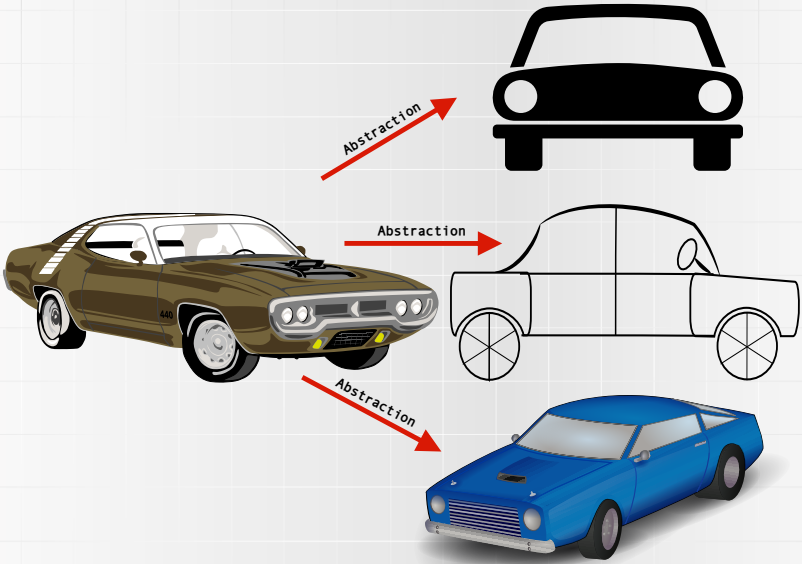
Proces wytwórczy oprogramowania

Od rzeczywistego systemu do abstrakcyjnego modelu



Abstrakcja

Przykład – samochody

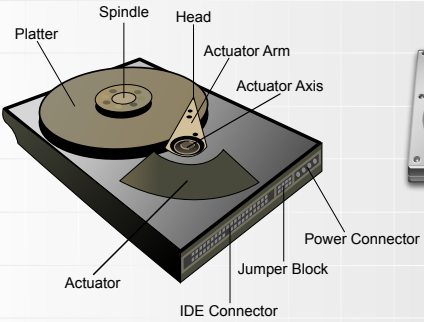


Abstrakcja

Przykład – dysk twardy

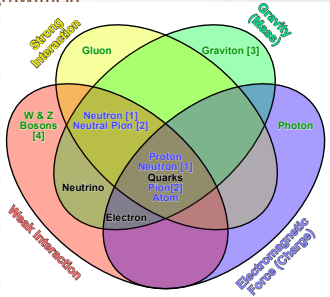
Hardware

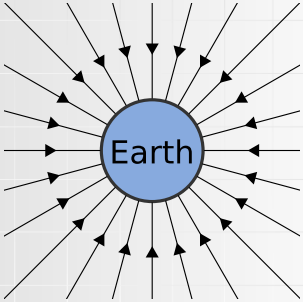
Abstract Filesystem

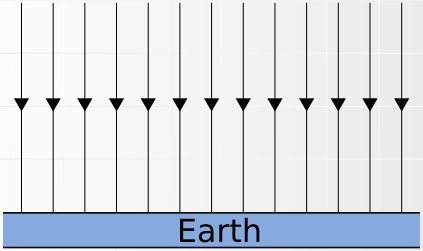


Abstrakcja

Przykład – grawitacja







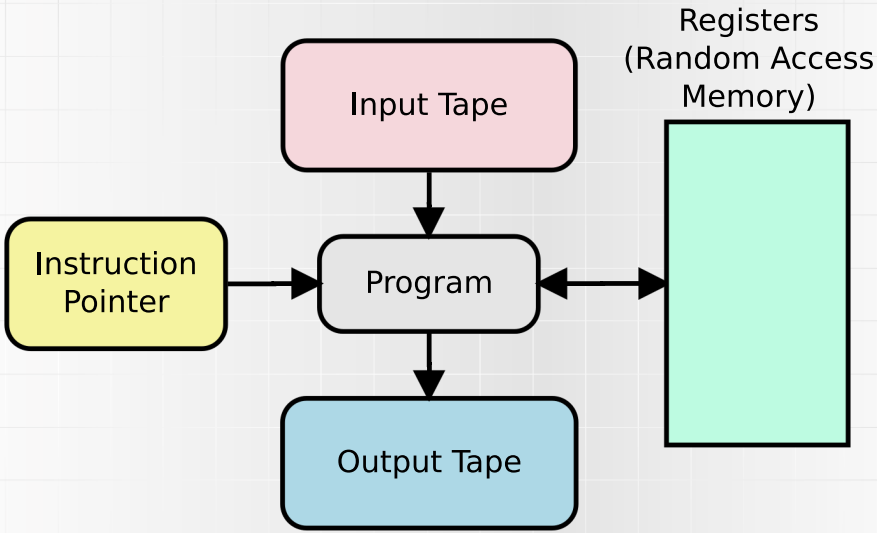
Politechnika Wrocławska

Section 2

Paradygmaty programowania imperatywnego

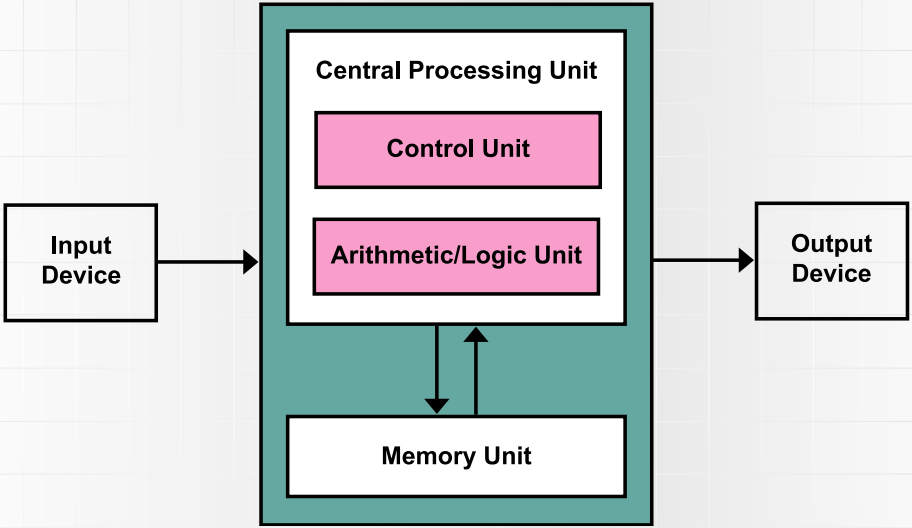
Maszyna RAM

Model Teoretyczny

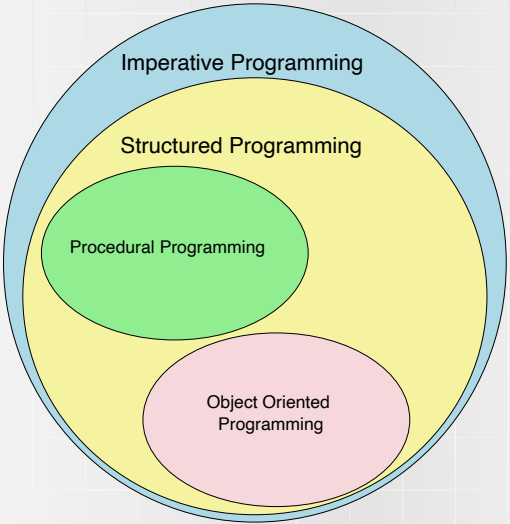


Architektura von Neumanna

Model Praktyczny



Paradygmaty programowania imperatywnego



Programowanie strukturalne i niestrukturalne

Listing: structural.c

```
1 void a(){
2  int a=2;
3  for(int i=0;i<10;i++)a++;
4 }
```

Programowanie strukturalne i niestrukturalne

Listing: nonStructural.as

```
16 a():
17     pushq %rbp
18     movq  %rsp, %rbp
19     movl  $1, -4(%rbp)
20     movl  $0, -8(%rbp)
21 .L3:
22     cmpl $9, -8(%rbp)
23     jg .L4
24     addl $1, -4(%rbp)
25     addl $1, -8(%rbp)
26     jmp .L3
27 .L4:
28     nop
29     popq  %rbp
30     ret
```

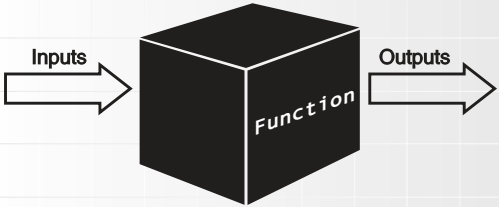
Politechnika Wrocławska

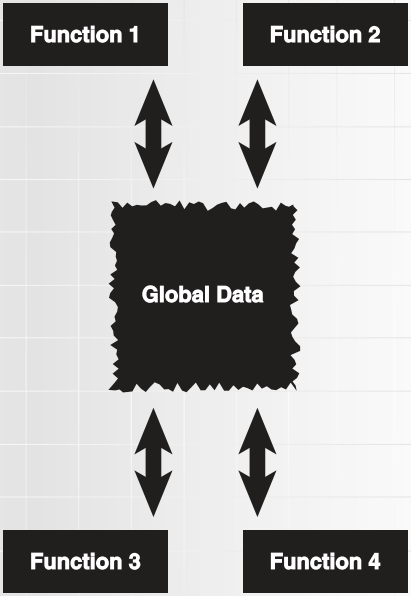
Section 3

Programowanie Proceduralne

Programowanie Proceduralne

Schemat





Programowanie Proceduralne

Przykład – Liczba zespolona

Listing: Complex.h

```
#ifndef COMPLEX_H_
#define COMPLEX_H_

struct Complex{
 double re;
 double im;
};
double complex_abs(const Complex* complex);
double complex_angle(const Complex* complex);
void complex_add(Complex* result, const Complex * a, const Complex *b);
void complex_fancy(Complex* a);
#endif
```

Programowanie Proceduralne

Przykład – Liczba zespolona

Listing: Complex.cpp

```cpp
#include "Complex.h"
#include<math.h>

double complex_abs(const Complex* complex){
  return sqrt((complex->re)*(complex->re) + (complex->im)*(complex->im));
}
double complex_angle(const Complex* complex){
 return atan2(complex->im,complex->re);
}
void complex_add(Complex* result, const Complex * a, const Complex *b){
  result->re = a->re + b->re;
  result->im = a->im + b->im;
}
void complex_fancy(Complex* a){
  a->im=a->re;
}
```

Programowanie Proceduralne

Przykład – Liczba zespolona

Listing: main.cpp

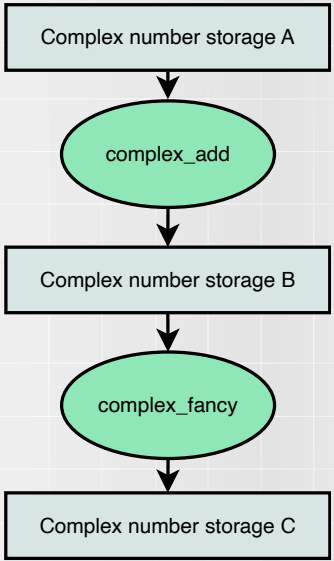
```
#include<iostream>
#include "Complex.h"
using namespace std;
int main(){
  Complex c,d,r;
  c.im = d.re =1;
  complex_add(&r,&c,&d);
  cout<<"Module: " << complex_abs(&r)
   <<"\nAngle: " << complex_angle(&r)<<endl;
}
```

Listing: Kompilacja i uruchomienie

```
g++ ./main.cpp ./Complex.cpp
./a.out
Module: 1.41421
Angle: 0.785398
```

Programowanie Proceduralne

Przepływ danych



Politechnika Wrocławska

Section 4

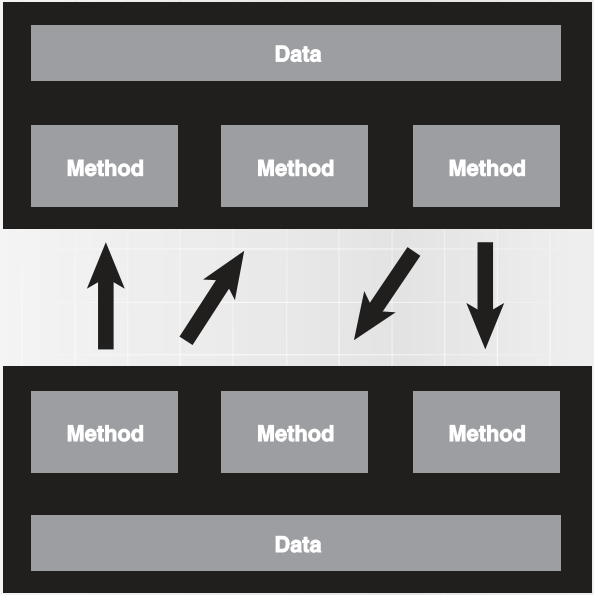
Programowanie Obiektowe

Subsection 1

Podejście Obiektowe

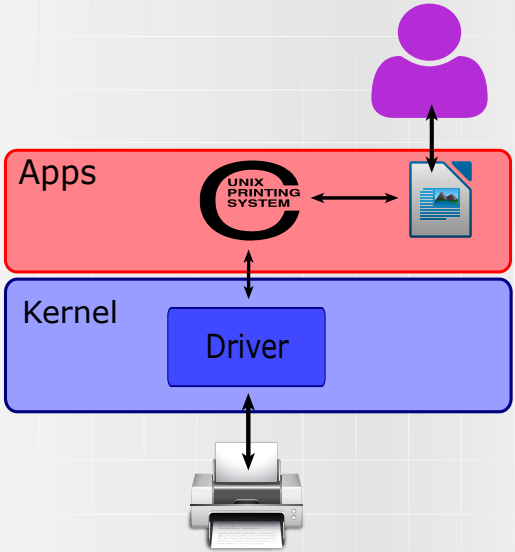
Politechnika Wrocławska

Podejście Obiektowe

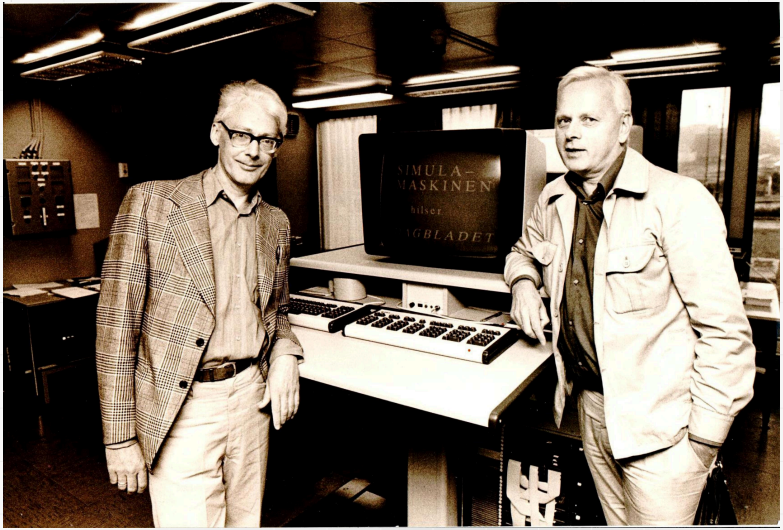


Podejście Obiektowe

Współpraca obiektów – przykład

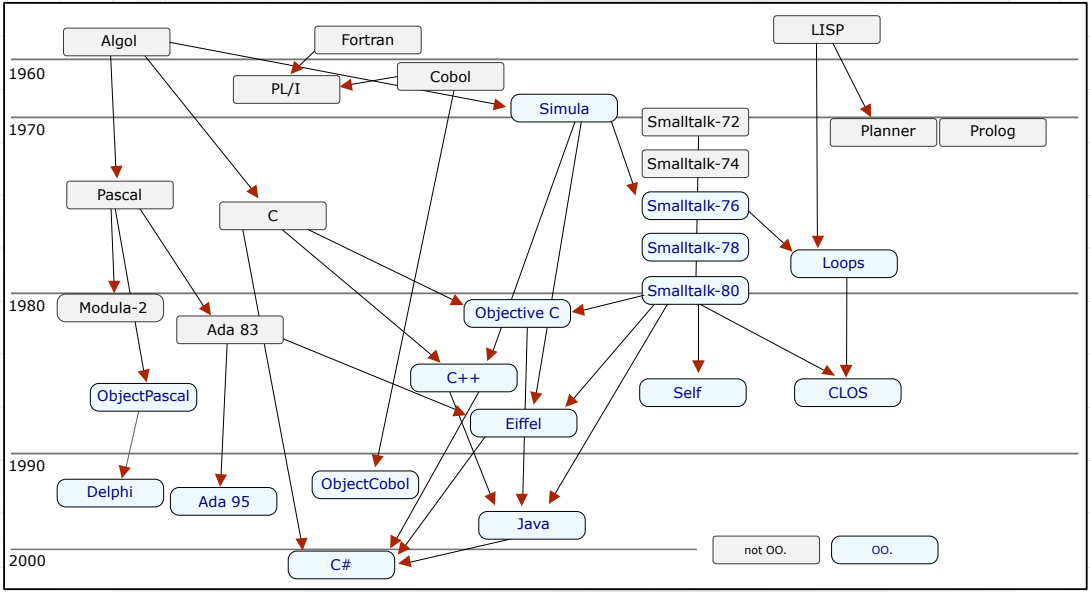


Simula



Rysunek: Ole-Johan Dah i Kristen Nygaard

Obiektowo zorientowane języki programowania

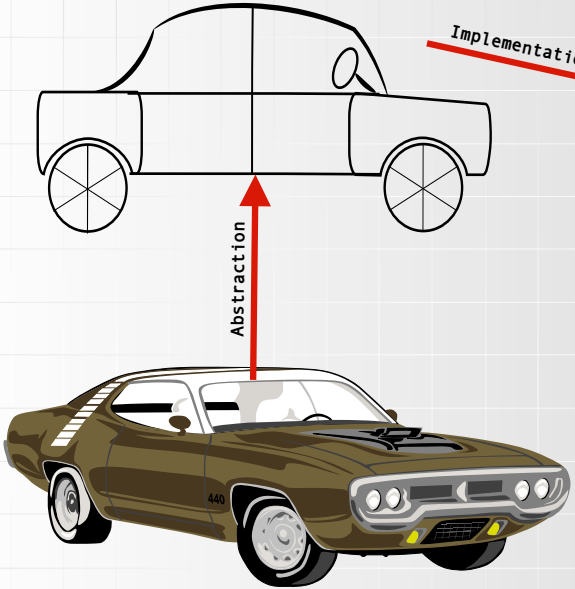


Politechnika Wrocławska

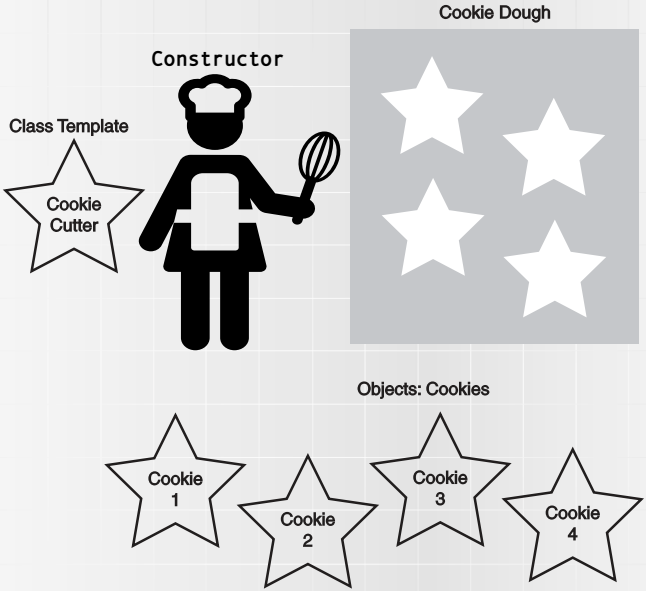
Subsection 2

Klasa a obiekt

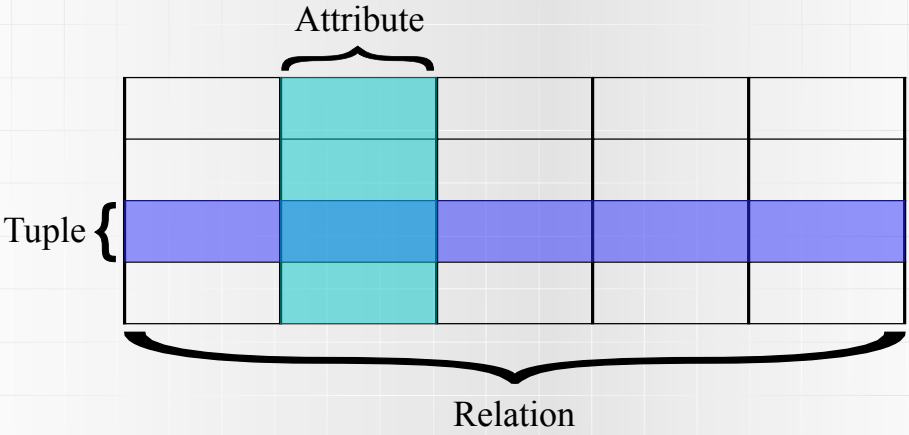
Klasa Obiektów



Klasa a obiekt(y)



Klasa a obiekt(y)



Tworzymy pierwszą klasę

Listing: CarSimple.java

```java
package carSimple;
public class CarSimple{
 private double maxSpeed;
 /**
  Constructor
  @param maxSpeed -- max speed km/h
 */
 public CarSimple(double maxSpeed){this.maxSpeed=maxSpeed;}
 /**
  Converts speed in mph into kmh
  @param speed -- speed in mph
 */
 public static double mph2kmh(double speed){return 1.609*speed;}

 @Override
public String toString() {
  return "Car with max speed: " + this.maxSpeed;
}
```

Tworzymy obiekty pierwszej klasy

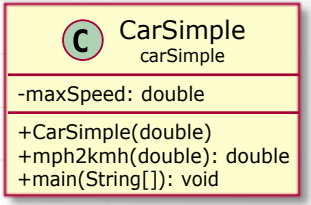
Listing: CarSimple.java

```
24 /**
25   Static method -- main
26   @param args[] -- runtime arguments for the program
27  */
28  public static void main(String args[]){
29   CarSimple car = new CarSimple(100);
30   System.out.println("Car: " + car.toString());
31   CarSimple car2 = new CarSimple(CarSimple.mph2kmh(100));
32   System.out.println("Car2: " + car2);
33  }
34 }
```

```
1 Car: Car with max speed: 100.0
2 Car2: Car with max speed: 160.9
```

Diagram Klas

Java



Listing: CarSimple.java

```java
package carSimple;
public class CarSimple{
 private double maxSpeed;
 /**
  Constructor
  @param maxSpeed -- max speed km/h
 */
 public CarSimple(double
     maxSpeed){this.maxSpeed=maxSpeed;}
 /**
  Converts speed in mph into kmh
  @param speed -- speed in mph
 */
 public static double mph2kmh(double speed){return
     1.609*speed;}

 @Override
public String toString() {
  return "Car with max speed: " + this.maxSpeed;
}
/**
```

Konstruktor bezargumentowy

Listing: CarSimple.java

```
1  package carSimple2;
2  public class CarSimple{
3   private double maxSpeed;
4   /**
5    Constructor
6    @param maxSpeed -- max speed km/h
7   */
8   public CarSimple(double maxSpeed){this.maxSpeed=maxSpeed;}
9
10  public CarSimple() {
11    this(66.6);//DRY
12  }
```

Konstruktor bezargumentowy

Listing: CarSimple.java

```
32  public static void main(String args[]){
33   CarSimple car = new CarSimple();
34   System.out.println("Car: " + car.toString());
35  }
36 }
```

```
2 Car: Car with max speed: 66.6
```

Single Responsibility Principle/ High Cohesion principle

Zasada pojedynczej odpowiedzialności/ Zasada spójności

Jakie dane/operacje umieścić w klasie?

Single Responsibility Principle/ High Cohesion principle

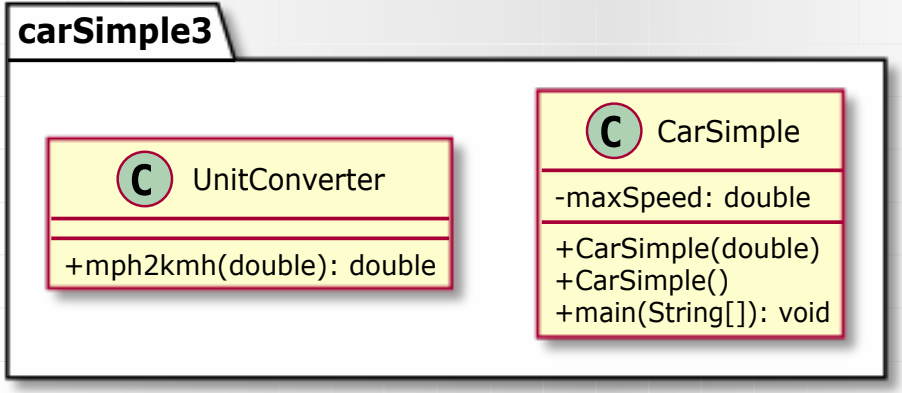
Zasada pojedynczej odpowiedzialności/ Zasada spójności

Nie powinno być więcej niż jednego powodu do modyfikacji klasy

Przypisuj odpowiedzialności do obiektu tak, aby spójność była jak największa.

Obiekt powinien mieć tylko jedną odpowiedzialność. Nie należy tworzyć obiektów odpowiedzialnych za zbyt wiele zadań. Klasy ze zbyt wieloma odpowiedzialnościami lepiej podzielić na kilka mniejszych klas.

Rozdzielenie odpowiedzialności



Zmienne statyczne i inicjatory

Listing: StaticV1.java

```
1  package staticVars;
2  public class StaticV1 {
3
4    public static String staticVar = "Static variable";//static initialization
5    public String objectVar;
6    private Double val = new Double(10);//Instance initializer
7    public StaticV1() {
8      this.objectVar = "object-specific variable";
9    }
10   public void increase() { this.val++;}
11   @Override
12   public String toString() {
13     return this.getClass().getCanonicalName() + " static var: " + staticVar + ", normal var:" +
       this.objectVar+"value: "+this.val +"\n";
14   }
```

Zmienne statyczne i inicjatory

Listing: StaticV1.java

```
16   public static void main(String[] args) {
17     System.out.println(StaticV1.staticVar);//No object is created so far
18     StaticV1 obj1 = new StaticV1();
19     StaticV1 obj2 = new StaticV1();
20     System.out.println(obj1.toString() + obj2 );
21     StaticV1.staticVar="Varr";
22     obj2.increase();
23     System.out.println(obj1.toString() + obj2 );
24   }
25
26 }
```

```
1 Static variable
2 staticVars.StaticV1 static var: Static variable, normal var:object-specific variablevalue: 10.0
3 staticVars.StaticV1 static var: Static variable, normal var:object-specific variablevalue: 10.0
4
5 staticVars.StaticV1 static var: Varr, normal var:object-specific variablevalue: 10.0
6 staticVars.StaticV1 static var: Varr, normal var:object-specific variablevalue: 11.0
```

Statyczna inicjalizacja

Listing: StaticV2.java

```
1  package staticVars;
2  public class StaticV2 {
3    public static int a;
4    public static int b;
5    private int x=44;
6    static {
7      a=3;
8      b = (int) Math.pow(a, 2.0);
9      //x=3;//Compilation error: x is not static
10   }
11
12   @Override
13   public String toString() {
14     return "a= " + a + "\tb= " + b + "\tx : " +x;
15   }
```

Statyczna inicjalizacja

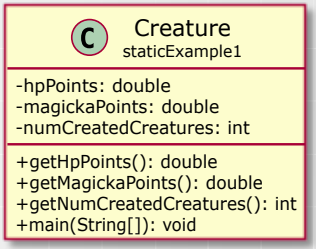
Listing: StaticV2.java

```
17   public static void main(String[] args) {
18     StaticV2 obj = new StaticV2();
19     System.out.println(obj);
20   }
21 }
```

```
1 a= 3^^Ib= 9^^Ix : 44
```

Zmienne statyczne – Przykład zastosowania

Zliczanie obiektów *



Zmienne statyczne – Przykład zastosowania

Zliczanie obiektów *

Listing: Creature.java

```
1  package staticExample1;
2
3  public class Creature {
4
5    private double hpPoints=100;
6    private double magickaPoints=50;
7
8    private static int numCreatedCreatures=0;
9
10   public double getHpPoints() { return hpPoints; }
11
12   public double getMagickaPoints() {  return magickaPoints;}
13
14   public static int getNumCreatedCreatures() {return numCreatedCreatures;}
15
16   public Creature() {
17     numCreatedCreatures++;
18   }
```

Zmienne statyczne – Przykład zastosowania

Zliczanie obiektów *

Listing: Creature.java

```
20   public static void main(String[] args) {
21     System.out.println("Number of creatures created so far: " +
        Creature.getNumCreatedCreatures());
22     Creature creature1 = new Creature();
23     System.out.println("Number of creatures created so far: " +
        Creature.getNumCreatedCreatures());
24   }
25
26 }
```

```
1 Number of creatures created so far: 0
2 Number of creatures created so far: 1
```

Zmienne finalne

Listing: FinalsV1.java

```
1  package finalVars;
2
3  public class FinalsV1 {
4
5    public static final String constVal="CVal"; //have to be initialised here!
6    private final Double value = new Double(33); //have to be initialised here or in constructor!
7    private final Double value2;
8
9    public FinalsV1() {
10     this.value2 = new Double(22);
11     //value = new Double(55);//Compilation error
12     //value++;//Compilation error 'value cannot be assigned'
13   }
14
15   public static void main(String[] args) {
16     //FinalsV1.constVal = "XVal";//Compilation error
17   }
18 }
```

Zmienne finalne

Listing: FinalsV2.java

```
1  package finalVars;
2  import java.util.ArrayList;
3  import java.util.List;
4
5  public class FinalsV2 {
6
7    private final List<Double> list;
8
9    public FinalsV2() {
10     list = new ArrayList<>();
11     list.add(5.5);
12   }
13
14   @Override
15   public String toString() {
16     return list.toString();
17   }
```

Zmienne finalne

Listing: FinalsV2.java

```
19   public static void main(String[] args) {
20     FinalsV2 obj = new FinalsV2();
21     System.out.println(obj.toString());
22
23   }
24 }
```

```
1 [5.5]
```

Finalne argumenty metod

Listing: FinalsV3.java

```java
package finalVars;
import java.util.ArrayList;
import java.util.List;

public class FinalsV3 {

  public static void addElement(List<Double> list) {
    list = new ArrayList<>();
    list.add(1.1);//add to a new list assigned to copied reference
  }

  public static void addElement2(final List<Double> list) {
    //list = new ArrayList<>();//Compilation error
    list.add(2.2);
  }

```

Finalne argumenty metod

Listing: FinalsV3.java

```
17   public static void main(String[] args) {
18     List l1 = new ArrayList<>();
19     l1.add(3.3); l1.add(4.4);
20     List l2 = new ArrayList<>();
21     l2.addAll(l1);
22
23     addElement(l1);
24     addElement2(l2);
25     System.out.println("L1: " + l1.toString() + "\nL2: " + l2 );
26   }
27
28
29 }
```

```
1 L1: [3.3, 4.4]
2 L2: [3.3, 4.4, 2.2]
```

Niszczenie obiektów

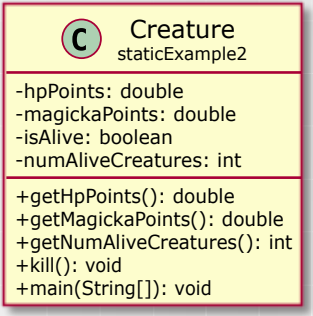
Listing: Cookie.java

```java
package objects;
public class Cookie {
  private String name;
  public Cookie(String name) {this.name = name;}
  @Override protected void finalize() throws Throwable {
    super.finalize();
    System.out.println(this.name+  " has been eaten by the cookie monster!");
  }
  public static void main(String[] args) {
    {
      Cookie cookie = new Cookie("cookie1");
      cookie = null;
    }
      System.gc();

      System.out.println("END");
  }
}
```

```
END
cookie1 has been eaten by the cookie monster!
```

Zliczanie obiektów – Przykład

* *



Zliczanie obiektów – Przykład

**

Listing: Creature.java

```java
package staticExample2;
public class Creature {
  private double hpPoints=100;
  private double magickaPoints=50;
  private boolean isAlive=true;
  private static int numAliveCreatures=0;

  public double getHpPoints() { return isAlive? hpPoints:0; }
  public double getMagickaPoints() {  return isAlive? magickaPoints:0;}
  public static int getNumAliveCreatures() {return numAliveCreatures;}

  public Creature() {
    numAliveCreatures++;
  }
  public void kill() {
    this.isAlive=false;
    numAliveCreatures--;
  }
```

Zliczanie obiektów – Przykład

**

Listing: Creature.java

```
20    public static void main(String[] args) {
21      System.out.println("Number of alive creatures: " + Creature.getNumAliveCreatures());
22      Creature creature1 = new Creature();
23      System.out.println("Number of alive creatures: " + Creature.getNumAliveCreatures());
24      creature1.kill();
25      System.out.println("Number of alive creatures: " + Creature.getNumAliveCreatures());
26    }
27
28  }
```

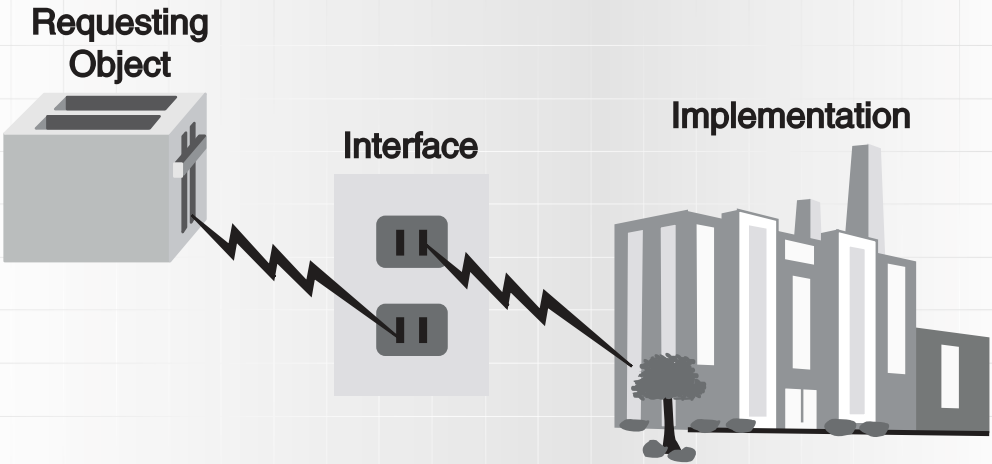
```
1  Number of alive creatures: 0
2  Number of alive creatures: 1
3  Number of alive creatures: 0
```

Politechnika Wrocławska

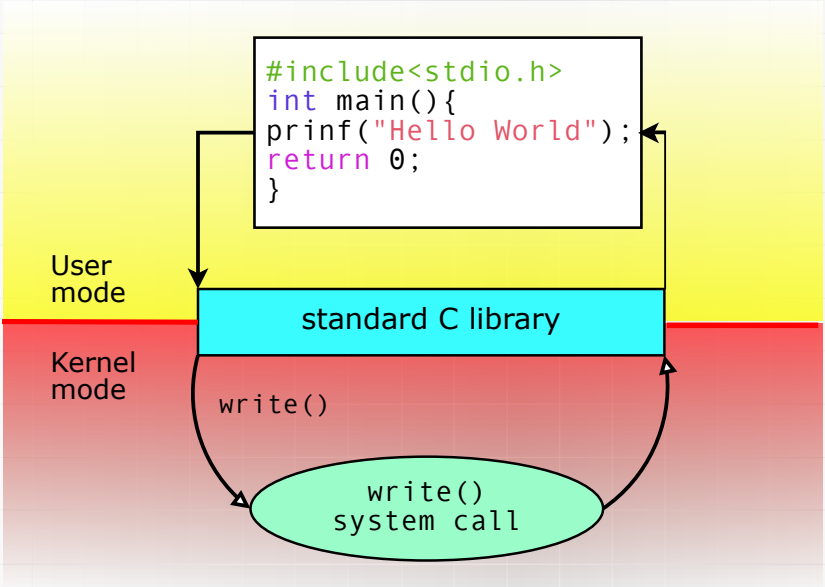
Subsection 3

Interfejs i hermetyzacja

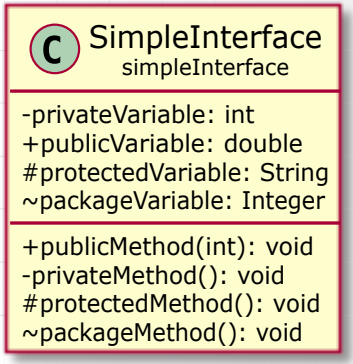
Interfejs i hermetyzacja



Interfejs i hermetyzacja



Interfejs jako kontrakt



Listing: SimpleInterface.java

```java
package simpleInterface;
public  class SimpleInterface{
  private int privateVariable;
  public double publicVariable;
  protected String protectedVariable;
  Integer packageVariable;
  /**
    Public method description
    @version 0.0.1
    @param x -- input parameter
   */
  public void publicMethod(int x){}
  private void privateMethod(){}
  protected void protectedMethod(){}
  void packageMethod(){}
  @Override
  public String toString() {
    return "Private Variable: " + this.privateVariable
     +"\n"
      + "Protected Variable: " + this.protectedVariable +
     "\n";
  }
```

Interfejs jako kontrakt

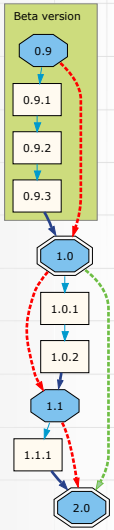
Modyfikator	Uml	Class	Package	Subclass	World
public	+	Y	Y	Y	Y
protected	#	Y	Y	Y	N
	~	Y	Y	N	N
private	-	Y	N	N	N

Interfejs jako kontrakt

Semantic Versioning

MajorVersion.MinorVersion.Patch

- MajorVersion – Dodanie nowej funkcjonalności, API traci wsteczną kompatybilność.
- MinorVersion – Dodanie nowej funkcjonalności, API jest wstecznie kompatybilne.
- Patch – Drobna poprawa błędów, API się nie zmienia.



Testy jednostkowe a pakiety

Listing: Example2.java

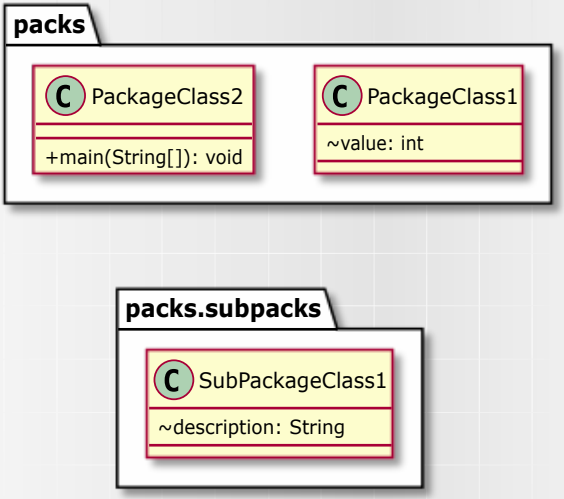
```java
package examples;

public class Example2 {

  public static int factorial1(int n) {
    if(n<=0) return 1;
    int factorial=1;
    for(int i=1;i<=n;i++)
      factorial*=i;
    return factorial;
  }

  public static int factorial2(int n) {
    if(n<=1)return 1;
    return n*Example2.factorial2(n-1);
  }
```

Testy jednostkowe a pakiety

Listing: Example2Test.java

```
1  package examples;
2  import static org.junit.Assert.*;
3  import org.junit.Test;
4  import examples.Example2;
5
6  public class Example2Test {
7    @Test
8    public void testFactorial1() {
9      assertTrue("Factorial 1 test", Example2.factorial1(0) == 1);
10     assertTrue("Factorial 1 test", Example2.factorial1(4) == 24);
11     assertTrue("Factorial 1 test", Example2.factorial1(3) == 6);
12     assertTrue("Factorial 1 test", Example2.factorial1(1) == 1);
13   }
14   @Test
15   public void testFactorial2() {
16     assertTrue("Factorial 2 test", Example2.factorial2(0) == 1);
17     assertTrue("Factorial 2 test", Example2.factorial2(4) == 24);
18     assertTrue("Factorial 2 test", Example2.factorial2(3) == 6);
19     assertTrue("Factorial 2 test", Example2.factorial2(1) == 1);
20   }
```

Pakiety – widoczność zmiennych



Pakiety – widoczność zmiennych

Listing: PackageClass1.java

```java
package packs;

public class PackageClass1 {

  int value=10;//package visibility

  @Override
  public String toString() {
    return this.getClass().getCanonicalName() + " value: " + this.value;
  }

}
```

Pakiety – widoczność zmiennych

Listing: SubPackageClass1.java

```java
package packs.subpacks;

public class SubPackageClass1 {

  String description = "A description";

  @Override
  public String toString() {
    return this.getClass().getCanonicalName() + " description: " + this.description;
  }



}
```

Pakiety – widoczność zmiennych

Listing: PackageClass2.java

```java
package packs;

import packs.subpacks.SubPackageClass1;

public class PackageClass2 {

  public static void main(String[] args) {
    PackageClass1 obj = new PackageClass1();
    obj.value =55;
    System.out.println(obj);

    SubPackageClass1 obj2 = new SubPackageClass1();
    //obj2.description = "ABC";//Compilation error -- not visible
  }
}
```

```
packs.PackageClass1 value: 55
```

Czy prywatne jest zawsze prywatne?

Listing: SimpleInterface.java

```
6  package simpleInterface;
7  public  class SimpleInterface{
8    private int privateVariable;
9    public double publicVariable;
10   protected String protectedVariable;
11   Integer packageVariable;
12   /**
13     Public method description
14     @version 0.0.1
15     @param x -- input parameter
16    */
17   public void publicMethod(int x){}
18   private void privateMethod(){}
19   protected void protectedMethod(){}
20   void packageMethod(){}
21   @Override
22   public String toString() {
23     return "Private Variable: " + this.privateVariable +"\n"
24       + "Protected Variable: " + this.protectedVariable + "\n";
25   }
26 }
```

Czy prywatne jest zawsze prywatne?

Listing: SimpleInterfaceReflection.java

```java
package simpleInterface;

import java.lang.reflect.Field;

public class SimpleInterfaceReflection {
  public static void main( String[] args) {
    SimpleInterface obj = new SimpleInterface();
    try {
      Field f =  obj.getClass().getDeclaredField("privateVariable");
      f.setAccessible(true);
      f.setInt(obj, 10);
      System.out.println("Value: " + obj);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

```
Value: Private Variable: 10
Protected Variable: null
```

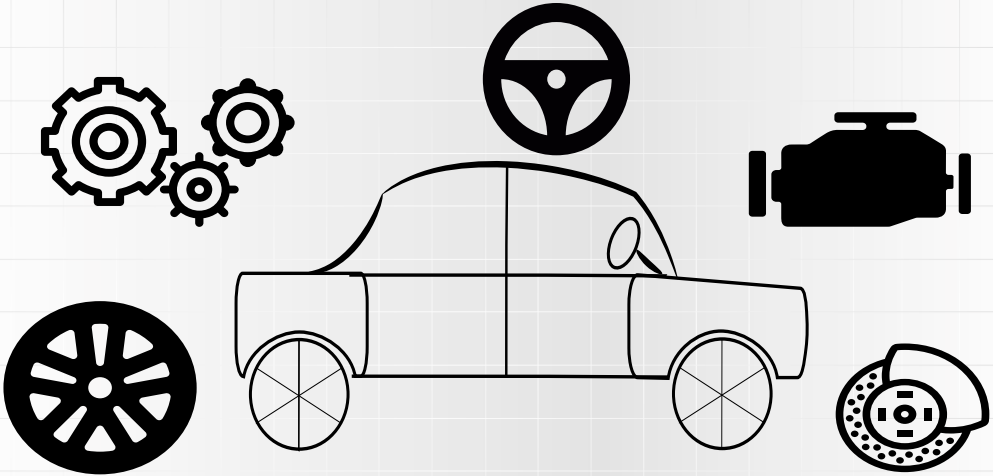
Politechnika Wrocławska

Subsection 4

Kompozycja

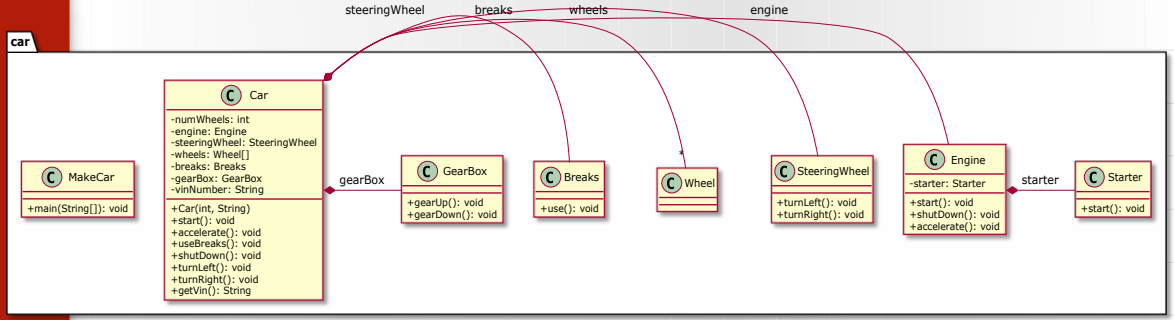
Politechnika Wrocławska

Kompozycja



Kompozycja

Diagram klas



Kompozycja

Listing: Car.java

```
1  package car;
2
3  public class Car{
4    private int numWheels;
5    private Engine engine = null;
6    private SteeringWheel steeringWheel;
7    private Wheel[] wheels;
8    private Breaks breaks;
9    private GearBox gearBox;
10   private String vinNumber;
11
12   public Car(int numWheels,String vinNumber){
13     this.numWheels = numWheels;
14     engine = new Engine();
15     steeringWheel = new SteeringWheel();
16     wheels = new Wheel[this.numWheels];
17     for(int i=0;i<wheels.length;i++)
18       wheels[i]=new Wheel();
19     breaks = new Breaks();
20     gearBox = new GearBox();
21     this.vinNumber = vinNumber;
22   }
```

Kompozycja

Listing: Car.java

```
24    public void start(){ engine.start();}
25    public void accelerate(){engine.accelerate();}
26    public void useBreaks(){breaks.use();}
27    public void shutDown(){
28      this.useBreaks();
29      engine.shutDown();
30    }
31    public void turnLeft(){steeringWheel.turnLeft();}
32    public void turnRight(){steeringWheel.turnRight();}
33    public String getVin() {return this.vinNumber;}
34
35  }
```

Kompozycja

Listing: Engine.java

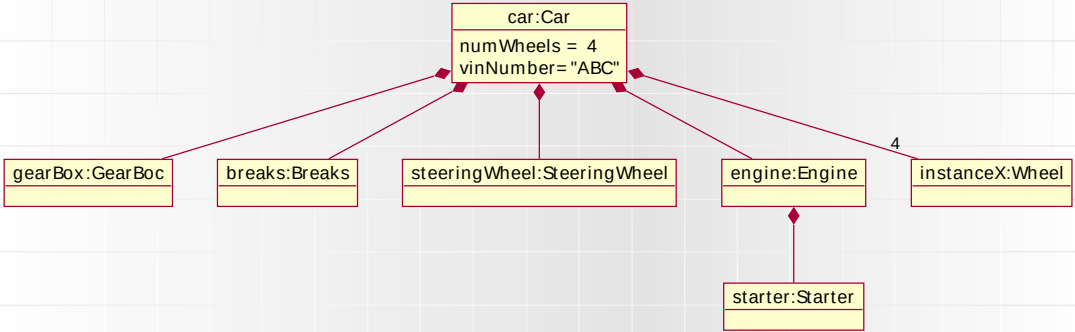
```java
/**
 @author pawel trajdos
 @version 0.0.1
 @date 04.03.2019
**/
package car;
public class Engine{

  private Starter starter;

  public Engine() {
    this.starter = new Starter();
  }

  public void start(){
    this.starter.start();
  }
  public void shutDown(){}
  public void accelerate(){}
}
```

Diagram obiektów

Listing: Car.java

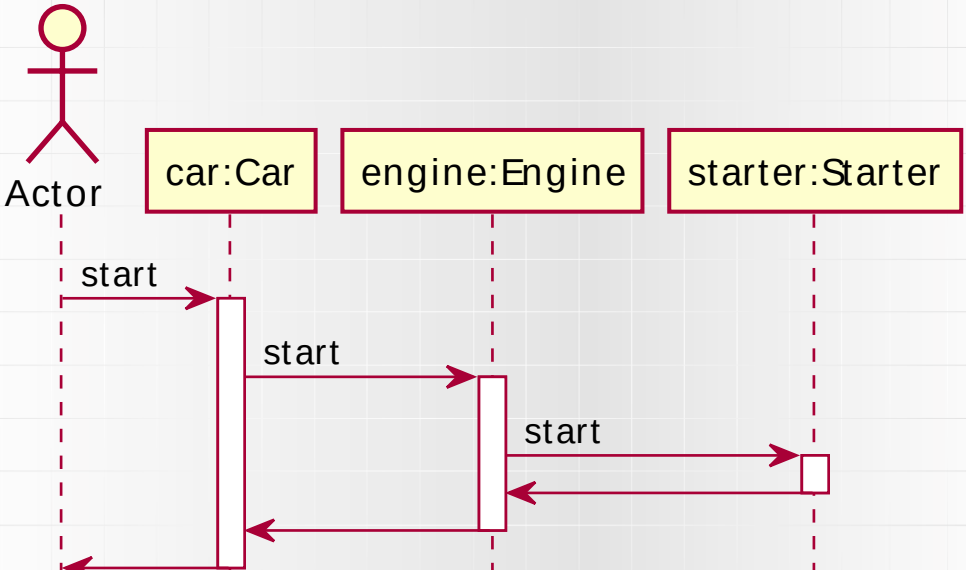
```
1 package car;
2
3 public class MakeCar {
4   public static void main(String[] args) {
5     Car c1 = new Car(4, "ABC");
6   }
7 }
```

Diagram obiektów



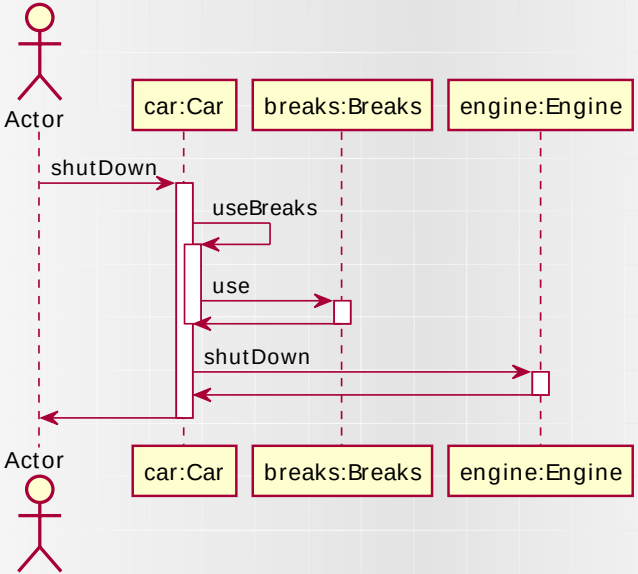
Komunikacja pomiędzy obiektami

Diagram sekwencji



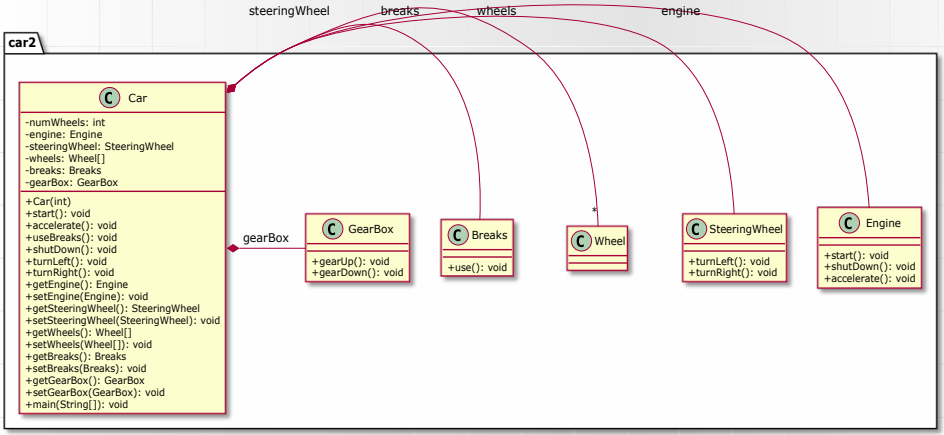
Komunikacja pomiędzy obiektami

Diagram sekwencji



Kompozycja a agregacja

Diagram klas



Agregacja

Listing: Car.java

```
31
32
33  public Engine getEngine() { return engine; }
34
35  public void setEngine(Engine engine) { this.engine = engine; }
36
37  public SteeringWheel getSteeringWheel() { return steeringWheel; }
38
39  public void setSteeringWheel(SteeringWheel steeringWheel) { this.steeringWheel =
       steeringWheel; }
40
41  public Wheel[] getWheels() { return wheels; }
42
43  public void setWheels(Wheel[] wheels) { this.wheels = wheels; }
44
45  public Breaks getBreaks() { return breaks;  }
46
47  public void setBreaks(Breaks breaks) { this.breaks = breaks; }
48
49  public GearBox getGearBox() { return gearBox; }
50
51  public void setGearBox(GearBox gearBox) { this.gearBox = gearBox;}
```

Agregacja

Konsekwencje

Listing: Car.java

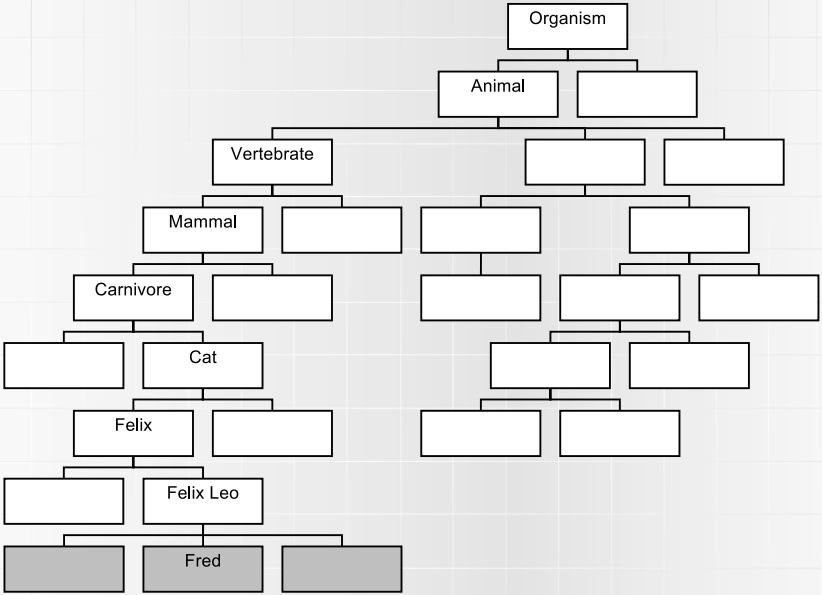
```
53    public static void main(String[] args) {
54      Engine engine;
55      {
56        Car car = new Car(4);
57        engine = car.getEngine();
58        System.out.println("Engine: " + engine);
59      }
60      //car variable is invisible here
61      System.out.println("Engine: " + engine);
62      {
63        Car car = new Car(6);
64        car.setEngine(engine);
65        System.out.println("Engine: " + car.getEngine());
66      }
67    }
68  }
```

```
1  Engine: car2.Engine@15db9742
2  Engine: car2.Engine@15db9742
3  Engine: car2.Engine@15db9742
```

Subsection 5

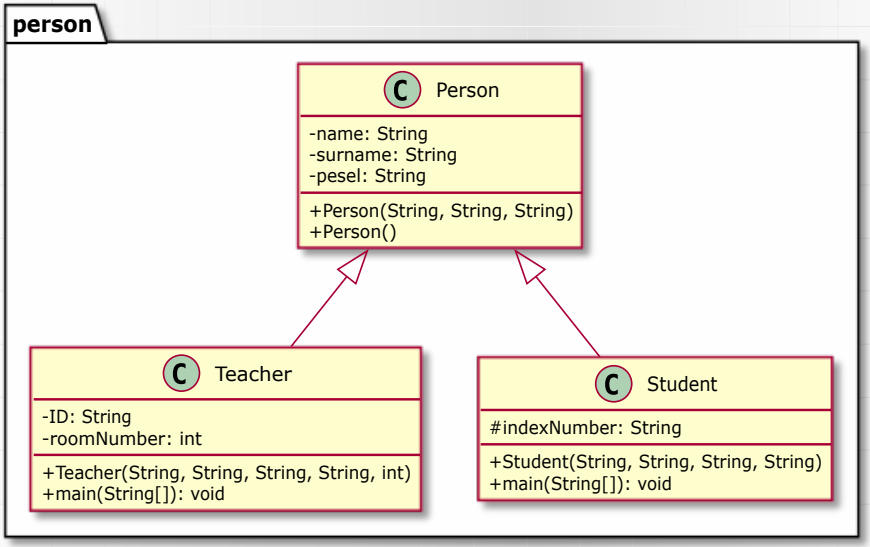
Dziedziczenie

Dziedziczenie



Dziedziczenie

Diagram UML



Dziedziczenie

Listing: Person.java

```
1  package person;
2  public class Person {
3
4    private String name;
5    private String surname;
6    private String pesel;
7
8    public Person(String name, String surname,String pesel) {
9      this.name = name;
10     this.surname = surname;
11     this.pesel = pesel;
12   }
13
14   public Person() {
15     this("John","Doe", null);
16   }
```

Dziedziczenie

Listing: Person.java

```
18    @Override
19    public String toString() {
20
21      return "Name: " + this.name + "\n"
22          + "Surname: " + this.surname + "\n"
23          + "PESEL: " + (this.pesel == null? "Unknown":this.pesel);
24    }
25  }
```

Dziedziczenie

Listing: Student.java

```java
package person;
public class Student extends Person {

  protected String indexNumber;

  public Student(String name, String surname, String pesel,String indexNumber) {
    super(name,surname,pesel);
    this.indexNumber = indexNumber;
  }

  @Override
  public String toString() {
    return "Student: \n" + super.toString() + "\n"
        + "Index number: " + this.indexNumber;
  }

  public static void main(String[] args) {
    Student stud = new Student("Jan","Kowalski","1234","5678");
    System.out.println(stud);
  }
}
```

Dziedziczenie

```
Student:
Name: Jan
Surname: Kowalski
PESEL: 1234
Index number: 5678

```

Dziedziczenie

Listing: Teacher.java

```java
package person;

public class Teacher extends Person {

  private String ID;
  private int roomNumber;

  public Teacher(String name, String surname, String pesel, String ID, int roomNumber) {
    super(name, surname, pesel);
    this.ID = ID;
    this.roomNumber = roomNumber;
  }

  @Override
  public String toString() {
    return "Teacher:\n"
        + "ID: " + ID
        + "\nroomNumber:" + roomNumber
        + "\n" + super.toString();
  }
```

Dziedziczenie

Listing: Teacher.java

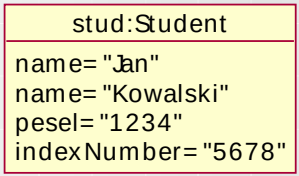
```
22    public static void main(String[] args) {
23      Teacher t = new Teacher("Zenon", "Nowak", "8900", "1000", 404);
24      System.out.println(t);
25    }
26  }
```

Dziedziczenie

```
1 Teacher:
2 ID: 1000
3 roomNumber:404
4 Name: Zenon
5 Surname: Nowak
6 PESEL: 8900
7
```

Dziedziczenie

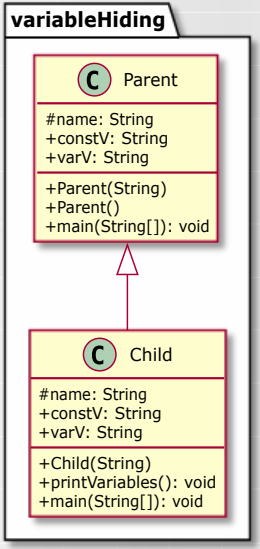
Diagram obiektów



Dziedziczenie a Hermetyzacja

Modyfikator	Uml	Class	Package	Subclass	World
public	+	Y	Y	Y	Y
protected	#	Y	Y	Y	N
	∼	Y	Y	N	N
private	-	Y	N	N	N

Zmienne statyczne i finalne



Zmienne statyczne i finalne

Listing: Parent.java

```
1  package variableHiding;
2  public class Parent {
3
4    protected String name;
5    public static final String constV = "Parent const variable";
6    public static String varV="Parent static variable";
7
8    public Parent(String name) {
9      //Variable shadowing
10     this.name = name;
11   }
12
13   public Parent() {
14     this("Parent variable");
15   }
16
```

Zmienne statyczne i finalne

Listing: Parent.java

```
17    public static void main(String[] args) {
18      //Parent.constV="x";// Compilation error
19      System.out.println("Const: " + Parent.constV);
20      Parent.varV="new value";
21      System.out.println("Static var: " + Parent.varV);
22    }
23
24  }
```

```
1  Const: Parent const variable
2  Static var: new value
3
```

Przesłanianie zmiennych

Listing: Child.java

```
1  package variableHiding;
2
3  public class Child extends Parent {
4    //Variable hiding
5    protected String name;
6    public static final String constV = "Child const variable";
7    public static String varV="Child static variable";
8
9    public Child(String name) {
10     super();
11     this.name = name;
12   }
13
14   public void printVariables() {
15     System.out.println("Child variable: " + name + "\n Parent variable: " + super.name);
16     System.out.println("Child static var: " + varV + "\nParent static var: " + Parent.varV );
17     System.out.println("Child static const: " + constV + "\nParent static var: " + Parent.constV
       );
18   }
```

Przesłanianie zmiennych

Listing: Child.java

```
1
2   public static void main(String[] args) {
3     Child child = new Child("A child");
4     Child.varV="XXX";
5     child.printVariables();
6   }
7 }
```

```
1 Child variable: A child
2  Parent variable: Parent variable
3 Child static var: XXX
4 Parent static var: Parent static variable
5 Child static const: Child const variable
6 Parent static var: Parent const variable
7
```

Klasy finalne

Listing: Parent.java

```
1 package finalClasses;
2
3 public final class Parent {
4   protected int var;
5 }
```

Listing: Child.java

```
1 package finalClasses;
2
3 public class Child  /*extends Parent -- compilation error*/{
4
5 }
```

Metody finalne

Listing: Parent.java

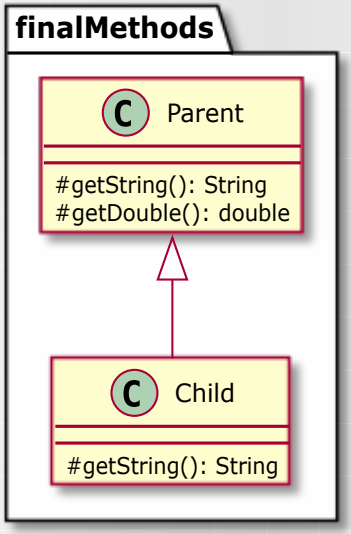
```java
package finalMethods;

public class Parent {

  protected String getString() {
    return "Parent String";
  }

  protected final double getDouble() {
    return 1.0;
  }

}
```

Metody finalne

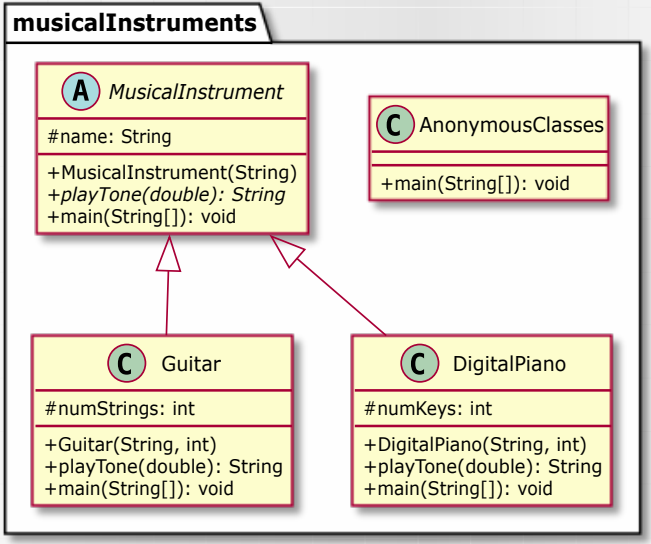
Listing: Child.java

```
1  package finalMethods;
2
3  public class Child extends Parent {
4
5    @Override
6    protected String getString() {
7      return super.getString() + " Child string";
8    }
9
10   /*
11   @Override
12   protected double getDouble() {
13     return 2.0;
14   }
15   Compilation error!
16   */
17
18
19 }
```

Metody finalne



Klasy abstrakcyjne



Klasy abstrakcyjne

Listing: MusicalInstrument.java

```java
package musicalInstruments;

public abstract class MusicalInstrument {

  protected String name;

  public MusicalInstrument(String name) {
    this.name = name;
  }

  public abstract String playTone(double frequency);

  public static void main(String[] args) {
    /*
     * Compilation error
    MusicalInstrument instrument = new MusicalInstrument("An Instrument");
    */

  }

}
```

Klasy abstrakcyjne

Listing: Guitar.java

```java
package musicalInstruments;

public class Guitar extends MusicalInstrument {

  protected int numStrings;

  public Guitar(String name,int numStrings) {
    super(name);
    this.numStrings = numStrings;
  }

  @Override
  public String playTone(double frequency) {
    return "The Guitar plays: " + frequency;
  }

  public static void main(String[] args) {
    Guitar guitar  = new Guitar("Fender Stratocaster",6);
    System.out.println(guitar.playTone(100));
  }

}
```

Klasy abstrakcyjne

Listing: DigitalPiano.java

```
1  package musicalInstruments;
2
3  public class DigitalPiano extends MusicalInstrument {
4
5    protected int numKeys;
6
7    public DigitalPiano(String name, int numKeys) {
8      super(name);
9      this.numKeys = numKeys;
10   }
11
12   @Override
13   public String playTone(double frequency) {
14     return "The digital piano plays: " + frequency;
15   }
16
17   public static void main(String[] args) {
18     DigitalPiano digPiano = new DigitalPiano("Korgg i3", 61);
19     System.out.println(digPiano.playTone(600));
20   }
21
22 }
```

Klasy anonimowe

Listing: AnonymousClasses.java

```java
package musicalInstruments;
public class AnonymousClasses {
  public static void main(String[] args) {
    MusicalInstrument instrument  = new MusicalInstrument("Anonymous") {
      @Override
      public String playTone(double frequency) {
        return this.name + " instrument plays: " + frequency;
      }
    };
    System.out.println(instrument.playTone(440));
  }

}
```

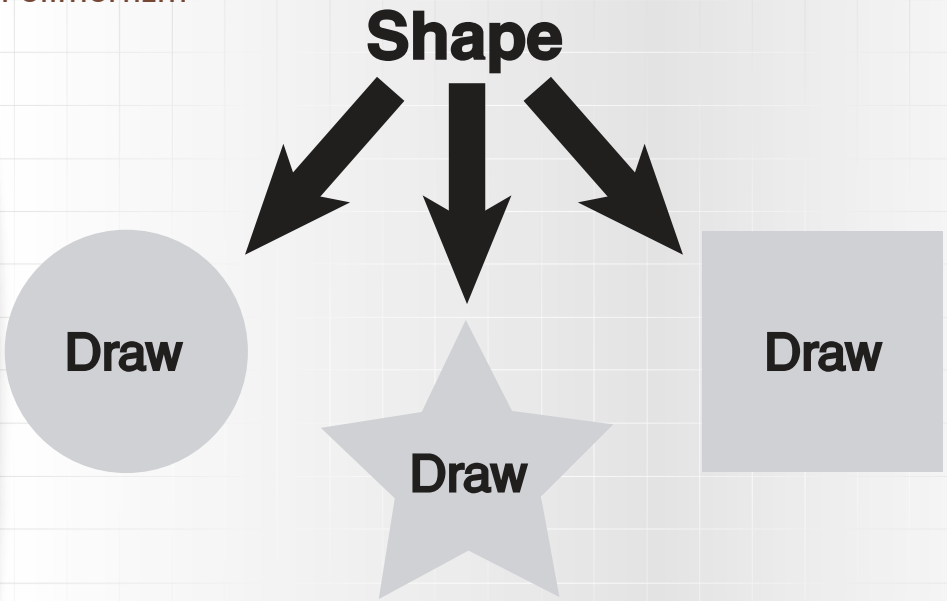
```
Anonymous instrument plays: 440.0

```

Politechnika Wrocławska

Subsection 6

Polimorfizm

Polimorfizm

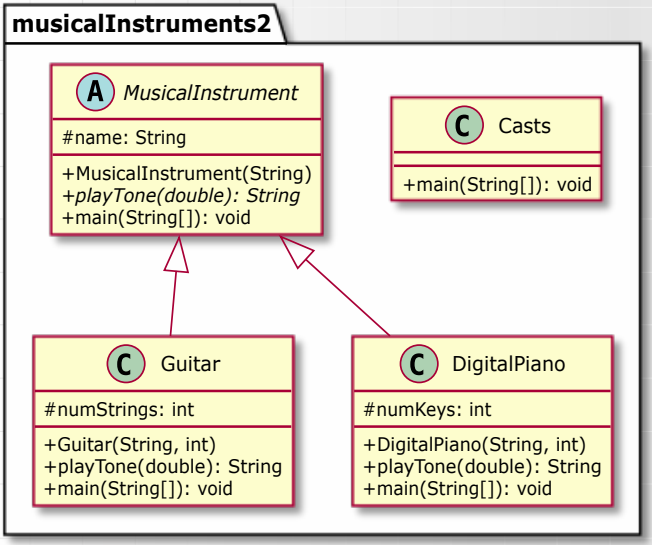


Polimorfizm

Co zrobić, gdy odpowiedzialność różni się w zależności od typu?

Przydziel zobowiązania, przy użyciu polimorfizmu, typom dla których to zachowanie jest różne.

Rzutowanie w górę



Rzutowanie w górę

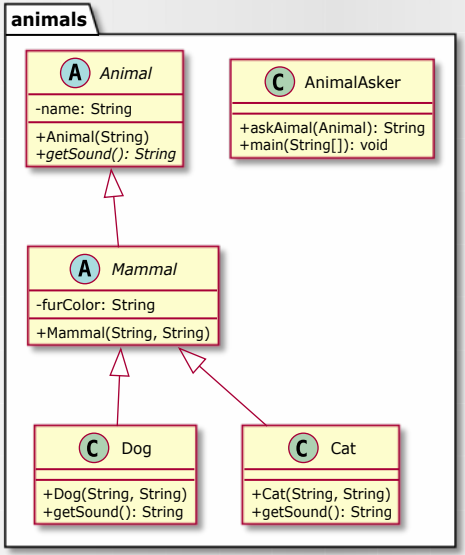
Listing: Casts.java

```java
package musicalInstruments2;

public class Casts {

  public static void main(String[] args) {
    Guitar guitar = new Guitar("Name", 6);
    MusicalInstrument instrument = guitar; //upcasting
    System.out.println(instrument.playTone(1000));
    MusicalInstrument instrument2 = new DigitalPiano("DigPiano", 61);
    //Guitar guitar2 = (Guitar) instrument2;//ClassCastException, downcastings
  }

}
```

```
The Guitar plays: 1000.0

```

Polimorfizm

Hierarchia klas



Polimorfizm

```java
/**
 @author pawel trajdos
 @version 0.0.1
**/
package animals;
public abstract class Animal{
  private String name;
  public abstract String getSound();
  public Animal(String name){this.name=name;}
  @Override
  public String toString() {
    return "An animal with name: " + this.name;
  }

}
```

Polimorfizm

```java
/**
 @author pawel trajdos
 @version 0.0.1
 @date 04.03.2019
**/
package animals;

public abstract class Mammal extends Animal{
  private String furColor;
  public Mammal(String name,String furColor){
    super(name);
    this.furColor=furColor;
  }
  @Override
  public String toString() {
    String parentDesc = super.toString();
    String description = parentDesc + "\t"+
        "Mammal with fur in colour: " + this.furColor;
    return description;
  }

}
```

Polimorfizm

```java
/**
 @author pawel trajdos
 @version 0.0.1
 @date 04.03.2019
**/
package animals;

public class Cat extends Mammal{
  @Override
  public String getSound(){
    return "Meow!";
  }
  public Cat(String name, String furColor){
    super(name,furColor);
  }
  @Override
  public String toString() {
    return super.toString() + "\tCat";
  }

}
```

Polimorfizm

```java
/**
 @author pawel trajdos
 @version 0.0.1
 @date 04.03.2019
**/
package animals;

public class Dog extends Mammal{
  @Override
  public String getSound(){
    return "Bark!";
  }
  public Dog(String name, String furColor){
    super(name,furColor);
  }
  @Override
  public String toString() {
    return super.toString() + "\tDog";
  }

}
```

Wywołania polimorficzne

```java
package animals;
import java.util.LinkedList;
import java.util.List;
public class AnimalAsker{

  public static String askAimal(Animal animal) {
    return animal.toString() + " says: " + animal.getSound();
  }

  public static void main(String[] args){
    List<Animal> animals = new LinkedList<Animal>();
    Cat cat1 = new Cat("Bonifacy","Black");
    Cat cat2 = new Cat("Filemon","White, Ginger red");
    Dog dog1 = new Dog("Reksio", "White, Ginger red");
    animals.add(cat1);
    animals.add(cat2);
    animals.add(dog1);
    for(Animal animal: animals){
      System.out.println(AnimalAsker.askAimal(animal));
    }

  }
}
```

Wywołania polimorficzne

```
1 An animal with name: Bonifacy^^IMammal with fur in colour: Black^^ICat says: Meow!
2 An animal with name: Filemon^^IMammal with fur in colour: White, Ginger red^^ICat says: Meow!
3 An animal with name: Reksio^^IMammal with fur in colour: White, Ginger red^^IDog says: Bark!
4
```

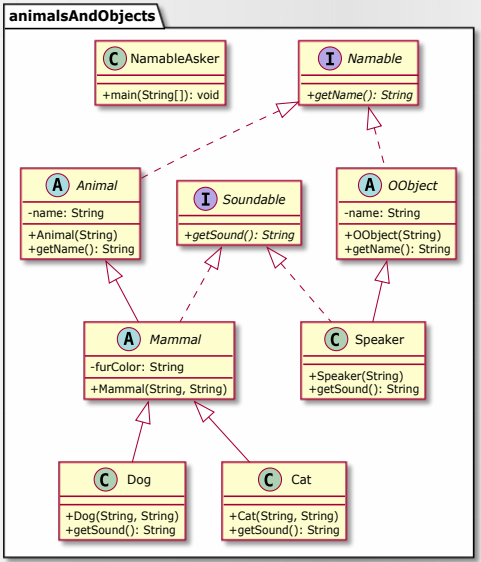
Politechnika Wrocławska

Subsection 7

Interfejsy

Animals

Dodanie interfejsów



Interface Segregation Principle

Zasada segregacji interfejsów

Klasa udostępnia tylko te interfejsy, które są niezbędne do zrealizowania konkretnej operacji.

Klasy nie powinny być zmuszane do zależności od metod, których nie używają.

Klasa powinna udostępniać drobnoziarniste interfejsy dostosowane do potrzeb jej klienta. Czyli, że klienci nie powinni mieć dostępu do metod których nie używają.

Dependency Inversion Principle

Zasada odwrócenia zależności

Wysokopoziomowe moduły nie powinny zależeć od modułów niskopoziomowych – zależności między nimi powinny wynikać z abstrakcji.

animalsAndObjects

Interfejsy

Listing: Namable.java

```
1 package animalsAndObjects;
2 public interface Namable {
3   public String getName();
4 }
```

Listing: Soundable.java

```
1 package animalsAndObjects;
2 public interface Soundable {
3   public String getSound();
4 }
```

animalsAndObjects

Klasy abstrakcyjne

Listing: Animal.java

```java
package animalsAndObjects;
public abstract class Animal implements  Namable{
  private String name;
  public Animal(String name){this.name=name;}
  @Override
  public String getName() {return name;}
}
```

Listing: Mammal.java

```java
package animalsAndObjects;

public abstract class Mammal extends Animal implements Soundable{
  private String furColor;
  public Mammal(String name,String furColor){
    super(name);
    this.furColor=furColor;
  }
}
```

animalsAndObjects

Klasy

Listing: Cat.java

```java
package animalsAndObjects;

public class Cat extends Mammal{
  @Override
  public String getSound(){
    return "Meow!";
  }
  public Cat(String name, String furColor){
    super(name,furColor);
  }
}
```

animalsAndObjects

Klasy

Listing: Dog.java

```java
package animalsAndObjects;

public class Dog extends Mammal{
  @Override
  public String getSound(){
     return "Bark!";
  }
  public Dog(String name, String furColor){
    super(name,furColor);
  }
}
```

animalsAndObjects

Klasy

Listing: OObject.java

```java
package animalsAndObjects;

public abstract class OObject implements Namable {

  private String name;
  public OObject(String name) {
    this.name = name;
  }
  @Override
  public String getName() {return name;}

}
```

animalsAndObjects

Klasy

Listing: Speaker.java

```java
package animalsAndObjects;

public class Speaker extends OObject implements Soundable {

  public Speaker(String name) {
    super(name);
  }

  @Override
  public String getSound() {
    return "Beep!";
  }
}
```

animalsAndObjects

Klasy

Listing: NamableAsker.java

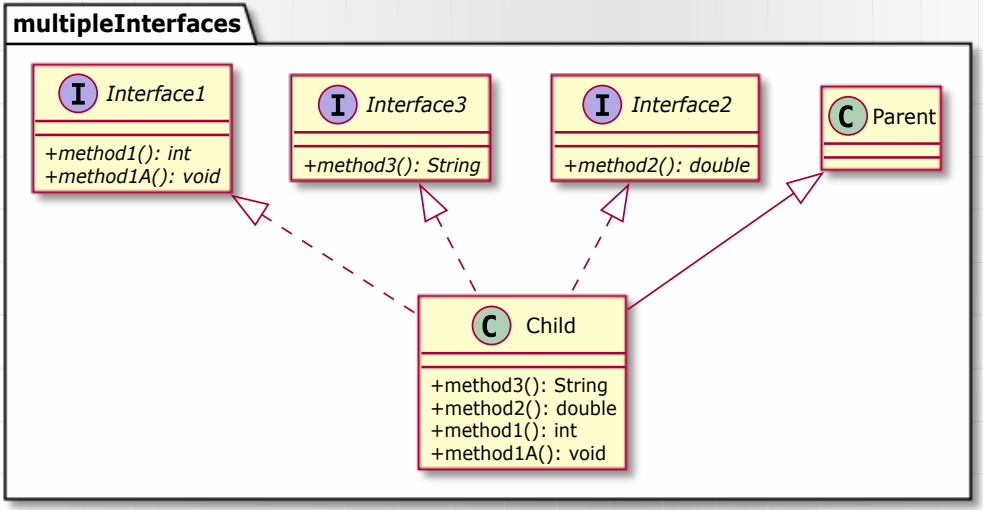
```java
package animalsAndObjects;
import java.util.LinkedList;
import java.util.List;
public class NamableAsker{
  public static void main(String[] args){
    List<Namable> namables = new LinkedList<>();
    Cat cat1 = new Cat("Bonifacy","Black");
    Cat cat2 = new Cat("Filemon","White, Ginger red");
    Dog dog1 = new Dog("Reksio", "White, Ginger red");
    Speaker speaker = new Speaker("Speaker");
    namables.add(cat1);
    namables.add(cat2);
    namables.add(dog1);
    namables.add(speaker);

    for(Namable animal: namables){
      System.out.print(animal.getName() + ": ");
      if(animal instanceof Soundable)
        System.out.print(((Soundable) animal).getSound());
      System.out.println("");
    }
```

animalsAndObjects

Klasy

```
1 Bonifacy: Meow!
2 Filemon: Meow!
3 Reksio: Bark!
4 Speaker: Beep!
5
```

Implementacja kilku interfejsów



Implementacja kilku interfejsów

Listing: Interface1.java

```java
package multipleInterfaces;

public interface Interface1 {

  public int method1();
  void method1A(); //package
  //protected void method1C();// Compilation error -- no protected modifier allowed
  //private void method1D();// Compilation error -- no protected modifier allowed

}
```

Listing: Interface2.java

```java
package multipleInterfaces;

public interface Interface2 {

  public double method2();
}
```

Implementacja kilku interfejsów

Listing: Interface3.java

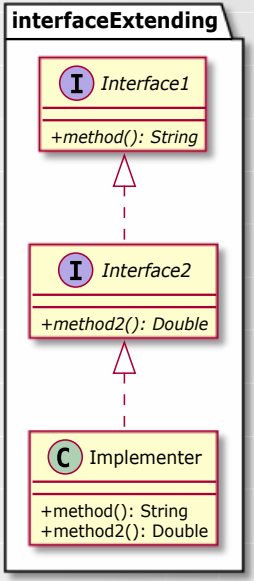
```
1 package multipleInterfaces;
2
3 public interface Interface3 {
4
5   public String method3();
6
7 }
```

Implementacja kilku interfejsów

Listing: Child.java

```java
package multipleInterfaces;

public class Child extends Parent implements Interface1, Interface2,Interface3 {

  @Override
  public String method3() {
    return null;
  }

  @Override
  public double method2() {
    return 0;
  }

  @Override
  public int method1() {
    return 0;
  }

  @Override
  public void method1A() {}
}
```

Rozszerzanie interfejsów



Rozszerzanie interfejsów

Listing: Interface1.java

```
1 package interfaceExtending;
2
3 public interface Interface1 {
4
5   public String method();
6
7 }
```

Listing: Interface2.java

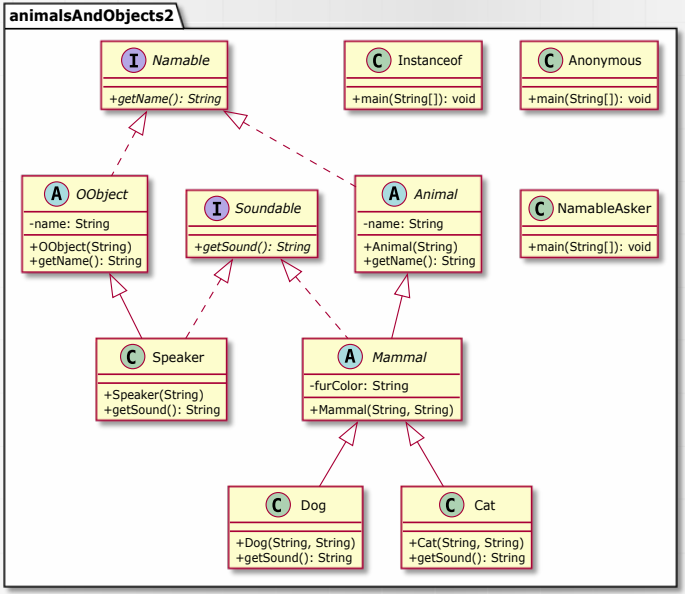
```
1 package interfaceExtending;
2
3 public interface Interface2 extends Interface1 {
4
5   public Double method2();
6
7 }
```

Rozszerzanie interfejsów

Listing: Implementer.java

```java
package interfaceExtending;

public class Implementer implements Interface2 {

  @Override
  public String method() {
    return "Method implementation";
  }

  @Override
  public Double method2() {
    return new Double(5);
  }
}
```

Interfejsy a klasy anonimowe



Interfejsy a klasy anonimowe

Listing: Anonymous.java

```
1  package animalsAndObjects2;
2
3  public class Anonymous {
4    public static void main(String[] args) {
5
6      Soundable snd = new Soundable() {
7        @Override
8        public String getSound() {
9          return "Some sound";
10       }
11     };
12     System.out.println(snd.getSound());
13   }
14
15 }
```

```
1 Some sound
2
```

Operator 'instanceof' a klasy

Listing: Instanceof.java

```java
package animalsAndObjects2;

public class Instanceof {

  public static void main(String[] args) {
    Animal anim = new Cat("Bonifacy", "Black");
    if(anim instanceof Cat)
      System.out.println(((Cat) anim).getSound());
    //VS
    if(anim instanceof Soundable)
      System.out.println(((Soundable) anim).getSound());
  }
}
```

```
Meow!
Meow!

```

Politechnika Wrocławska

Subsection 8

Projektowanie współpracy opartej na interfejsach

Low Coupling Principle

Deleguj odpowiedzialności tak, aby zachować jak najmniejszą 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.

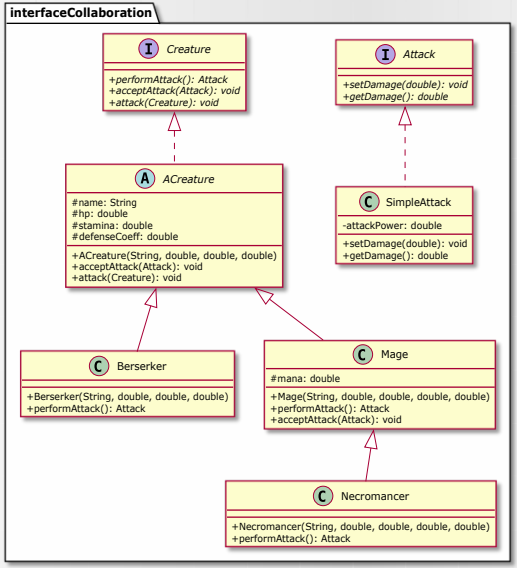
Indirection

Komu przydzielić zobowiązanie, jeśli zależy nam na uniknięciu bezpośredniego powiązania między obiektami?

Przypisz te odpowiedzialności do nowego pośredniego obiektu. Obiekt ten będzie służył do komunikacji innych klas/komponentów/usług/pakietów tak, że nie będą one zależne bezpośrednio od siebie.

Współpraca oparta o interfejsy

Przykład



Współpraca oparta o interfejsy

Przykład

Listing: Attack.java

```
1  package interfaceCollaboration;
2
3  public interface Attack {
4
5    public void setDamage(double damage);
6
7    public double getDamage();
8
9  }
```

Współpraca oparta o interfejsy

Przykład

Listing: SimpleAttack.java

```
1  package interfaceCollaboration;
2
3  public class SimpleAttack implements Attack {
4
5    private double attackPower=0;
6
7    @Override
8    public void setDamage(double damage) {this.attackPower = damage;}
9
10   @Override
11   public double getDamage() {return attackPower;}
12
13 }
```

Współpraca oparta o interfejsy

Przykład

Listing: Creature.java

```
1  package interfaceCollaboration;
2
3  public interface Creature {
4
5    public Attack performAttack();
6
7    public void acceptAttack(Attack attack);
8
9    public void attack(Creature creature);
10
11 }
```

Współpraca oparta o interfejsy

Przykład

Listing: ACreature.java

```java
package interfaceCollaboration;

public abstract class ACreature implements Creature {

  protected String name;
  protected double hp;
  protected double stamina;
  protected double defenseCoeff;

  public ACreature(String name, double hp, double stamina, double defenseCoeff) {
    this.name = name;
    this.hp = hp;
    this.stamina = stamina;
    this.defenseCoeff = defenseCoeff;
  }

  @Override
  public void acceptAttack(Attack attack) {
    hp-= attack.getDamage()*defenseCoeff;
  }
```

Współpraca oparta o interfejsy

Przykład

Listing: ACreature.java

```
22    @Override
23    public void attack(Creature creature) {
24      creature.acceptAttack(this.performAttack());
25      this.acceptAttack(creature.performAttack());
26    }
27
28  }
```

Współpraca oparta o interfejsy

Przykład

Listing: Berserker.java

```java
package interfaceCollaboration;

public class Berserker extends ACreature {

  public Berserker(String name, double hp, double stamina, double defenseCoeff) {
    super(name, hp, stamina, defenseCoeff);
  }

  @Override
  public Attack performAttack() {
    Attack attack = new SimpleAttack();
    attack.setDamage(0.3*this.stamina);
    return attack;
  }

}
```

Współpraca oparta o interfejsy

Przykład

Listing: Mage.java

```java
package interfaceCollaboration;

public class Mage extends ACreature {

  protected double mana;

  public Mage(String name, double hp, double stamina, double defenseCoeff, double mana) {
    super(name, hp, stamina, defenseCoeff);
    this.mana = mana;
  }

  @Override
  public Attack performAttack() {
    Attack attack = new SimpleAttack();
    attack.setDamage(0.05*this.stamina + 0.6*mana);
    return attack;
  }
```

Współpraca oparta o interfejsy

Przykład

Listing: Mage.java

```
19    @Override
20    public void acceptAttack(Attack attack) {
21      super.acceptAttack(attack);
22      hp+= 0.1*mana;
23    }
24  }
```

Współpraca oparta o interfejsy

Przykład

Listing: Necromancer.java

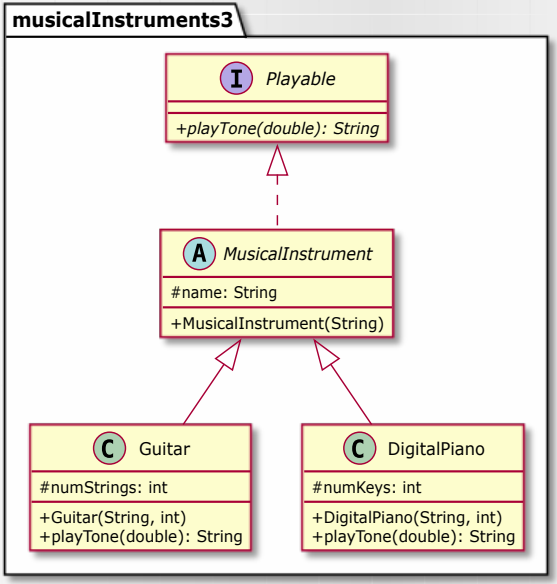
```java
package interfaceCollaboration;

public class Necromancer extends Mage {

  public Necromancer(String name, double hp, double stamina, double defenseCoeff, double mana) {
    super(name, hp, stamina, defenseCoeff, mana);
  }

  @Override
  public Attack performAttack() {
    Attack attack = super.performAttack();
    double damage = attack.getDamage();
    damage+= 0.1*hp;
    hp=.9*hp;
    attack.setDamage(damage);
    return attack;
  }
}
```

Politechnika Wrocławska

Subsection 9

Interfejsy w Java 8

Instrumenty muzyczne z interfejsem



Instrumenty muzyczne z interfejsem

Listing: Playable.java

```
1 package musicalInstruments3;
2
3 public interface Playable {
4
5   public String playTone(double frequency);
6
7 }
```

Instrumenty muzyczne z interfejsem

Listing: MusicalInstrument.java

```java
package musicalInstruments3;

public abstract class MusicalInstrument implements Playable{

  protected String name;

  public MusicalInstrument(String name) {
    this.name = name;
  }

}
```

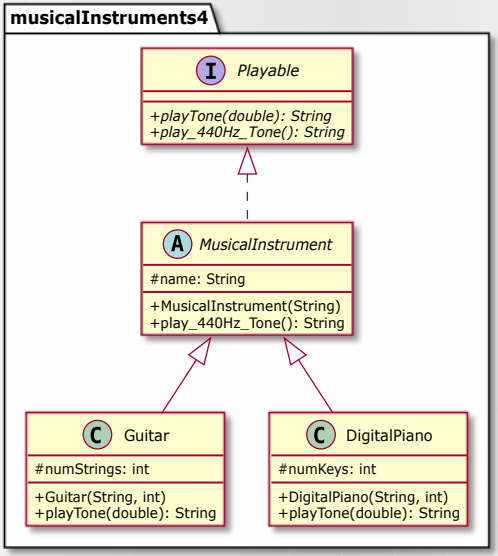
Instrumenty muzyczne z interfejsem

Listing: Guitar.java

```java
package musicalInstruments3;

public class Guitar extends MusicalInstrument {

  protected int numStrings;

  public Guitar(String name,int numStrings) {
    super(name);
    this.numStrings = numStrings;
  }

  @Override
  public String playTone(double frequency) {
    return "The Guitar plays: " + frequency;
  }
}
```

Dodanie metody do interfejsu

Przed Java 8



Dodanie metody do interfejsu

Przed Java 8

Listing: Playable.java

```
1 package musicalInstruments4;
2
3 public interface Playable {
4
5   public String playTone(double frequency);
6   public String play_440Hz_Tone();
7
8 }
```

Dodanie metody do interfejsu

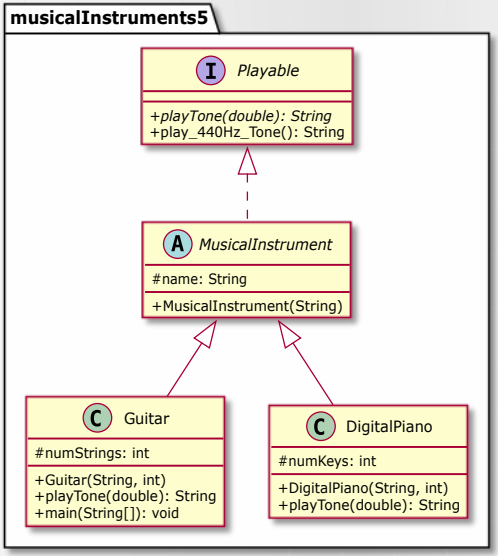
Przed Java 8

Listing: MusicalInstrument.java

```
1  package musicalInstruments4;
2
3  public abstract class MusicalInstrument implements Playable{
4
5    protected String name;
6
7    public MusicalInstrument(String name) {
8      this.name = name;
9    }
10
11   @Override
12   public String play_440Hz_Tone() {
13     return this.playTone(440);
14   }
15
16
17
18 }
```

Dodanie metody do interfejsu

Java 8+



Dodanie metody do interfejsu

Java 8+

Listing: Playable.java

```
1  package musicalInstruments5;
2
3  public interface Playable {
4
5    public String playTone(double frequency);
6    default public String play_440Hz_Tone() {
7      return playTone(440);
8    }
9  }
```

Dodanie metody do interfejsu

Java 8+

Listing: Guitar.java

```java
package musicalInstruments5;

public class Guitar extends MusicalInstrument {

  protected int numStrings;

  public Guitar(String name,int numStrings) {
    super(name);
    this.numStrings = numStrings;
  }

  @Override
  public String playTone(double frequency) {
    return "The Guitar plays: " + frequency;
  }
```

Dodanie metody do interfejsu

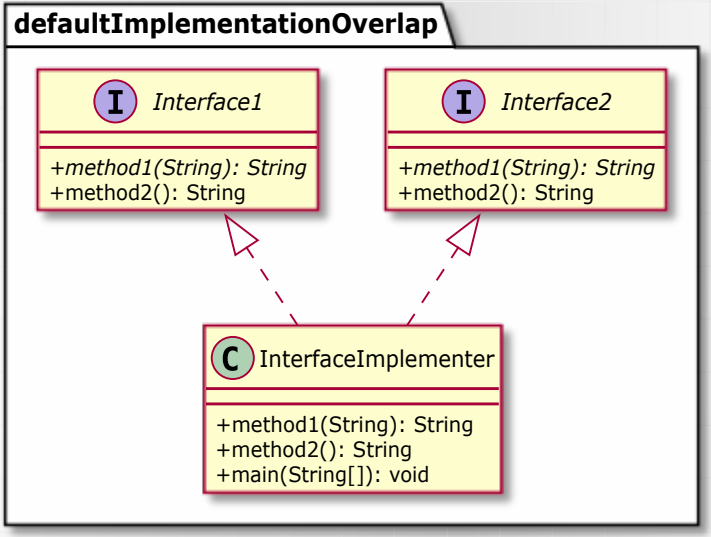
Java 8+

Listing: Guitar.java

```
17    public static void main(String[] args) {
18      Guitar guitar = new Guitar("Fender", 6);
19      System.out.println(guitar.play_440Hz_Tone());
20    }
21  }
```

```
1  The Guitar plays: 440.0
2
```

Problemy


defaultImplementationOverlap
I
Interface1
+method1(String): String
+method2(): String
I
Interface2
+method1(String): String
+method2(): String
C
InterfaceImplementer
+method1(String): String
+method2(): String
+main(String[]): void

Problemy

Listing: Interface1.java

```
1  package defaultImplementationOverlap;
2
3  public interface Interface1 {
4
5    public String method1(String arg);
6    default public String method2() {
7      return "Interface1, method2";
8    }
9
10 }
```

Listing: Interface2.java

```
1  package defaultImplementationOverlap;
2
3  public interface Interface2 {
4
5    public String method1(String arg);
6    default public String method2() {
7      return "Interface2, method2";
8    }
9
```

Problemy

Listing: InterfaceImplementer.java

```java
package defaultImplementationOverlap;

public class InterfaceImplementer implements Interface1, Interface2 {

  @Override
  public String method1(String arg) {
    return "method1: " + arg;
  }
  //Default method MUST be implemented
  @Override
  public String method2() {
    return "Method2";
  }
```

Problemy

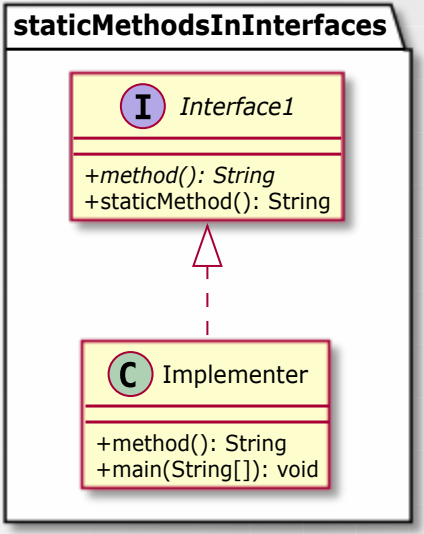
Listing: InterfaceImplementer.java

```
15   public static void main(String[] args) {
16     InterfaceImplementer implementer = new InterfaceImplementer();
17     System.out.println(implementer.method1("ARG") +"\n"+ implementer.method2() );
18   }
19 }
```

```
1 method1: ARG
2 Method2
3
```

Metody statyczne w interfejsach

Java 8+



Metody statyczne w interfejsach

Java 8+

Listing: Interface1.java

```java
package staticMethodsInInterfaces;

public interface Interface1 {

  public String method();
  public static String staticMethod() {
    return "Interface 1, Static method";
  }

}
```

Metody statyczne w interfejsach

Java 8+

Listing: Implementer.java

```java
package staticMethodsInInterfaces;

public class Implementer implements Interface1 {

  @Override
  public String method() {
    return "Implementer, method";
  }

  public static void main(String[] args) {
    Implementer imp = new Implementer();
    System.out.println(imp.method());
    //System.out.println(Implementer.staticMethod());//Compilation error -- undefined method
    System.out.println(Interface1.staticMethod());
  }
}
```

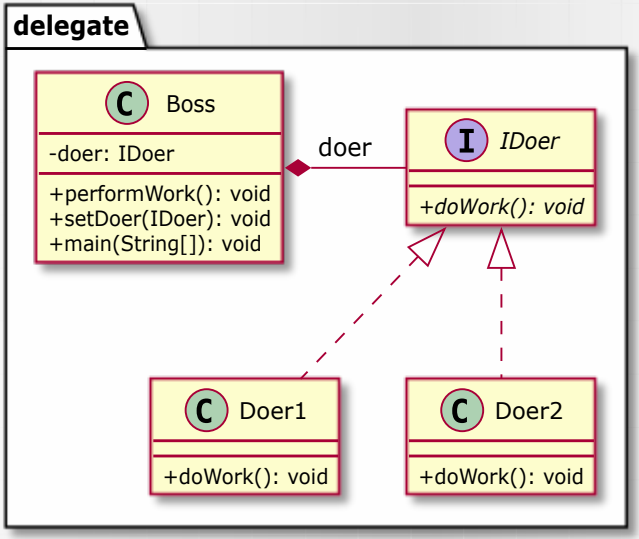
```
Implementer, method
Interface 1, Static method

```

Politechnika Wrocławska

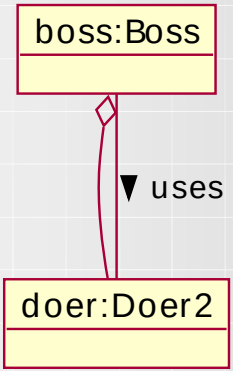
Subsection 10

Delegacja

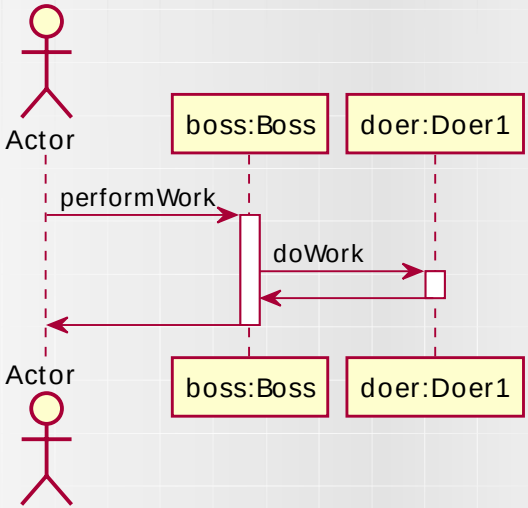
Delegacja


delegate
Boss
-doer: IDoer
+performWork(): void
+setDoer(IDoer): void
+main(String[]): void
doer
IDoer
+doWork(): void
Doer1
+doWork(): void
Doer2
+doWork(): void

Delegacja



Delegacja


Actor
boss:Boss
doer:Doer1
performWork
doWork
Actor
boss:Boss
doer:Doer1

Delegacja

Listing: IDoer.java

```
1 package delegate;
2
3 public interface IDoer {
4
5   public void doWork();
6
7 }
```

Delegacja

Listing: Doer1.java

```java
package delegate;

public class Doer1 implements IDoer {

  @Override
  public void doWork() {
    System.out.println("Doer1 does some work");
  }

}
```

Listing: Doer2.java

```java
package delegate;

public class Doer2 implements IDoer {

  @Override
  public void doWork() {
    System.out.println("Doer2 does some work");
  }

```

Delegacja

Listing: Boss.java

```
1  package delegate;
2
3  public class Boss {
4
5    private IDoer doer = new Doer1();
6
7    public void performWork() {
8      doer.doWork();//Delegate
9    }
10
11   public void setDoer(IDoer doer) { this.doer = doer; }
```

Delegacja

Listing: Boss.java

```
13    public static void main(String[] args) {
14      Boss boss = new Boss();
15      boss.performWork();
16      boss.setDoer(new Doer2());
17      boss.performWork();
18    }
19
20  }
```

```
1  Doer1 does some work
2  Doer2 does some work
3
```

Open-Close Principle

Zasada Otwarte - zamknięte

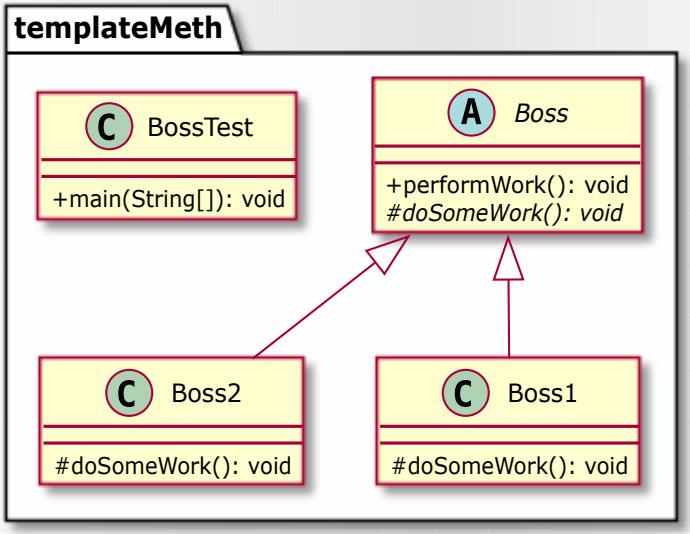
Elementy systemu takie, jak klasy, moduły, funkcje itd. powinny być otwarte na rozszerzenie, ale zamknięte na modyfikacje. Oznacza to, iż można zmienić zachowanie takiego elementu bez zmiany jego kodu.

Politechnika Wrocławska

Subsection 11

Metoda szablonowa

Metoda Szablonowa



Metoda Szablonowa

Listing: Boss.java

```
1  package templateMeth;
2
3  public abstract class Boss {
4
5    public void performWork() {
6      doSomeWork();
7    }
8
9    protected abstract void doSomeWork();
10
11 }
```

Metoda Szablonowa

Listing: Boss1.java

```java
package templateMeth;

public class Boss1 extends Boss {

  @Override
  protected void doSomeWork() {
    System.out.println("Boss 1 is doing some work!");
  }
}
```

Metoda Szablonowa

Listing: Boss2.java

```java
package templateMeth;

public class Boss2 extends Boss {

  @Override
  protected void doSomeWork() {
    System.out.println("Boss 2 is doing some work!");
  }
}
```

Metoda Szablonowa

Listing: BossTest.java

```
1  package templateMeth;
2
3  public class BossTest {
4
5
6    public static void main(String[] args) {
7      Boss b1 = new Boss1();
8      Boss b2 = new Boss2();
9
10     b1.performWork();
11     b2.performWork();
12   }
13 }
```

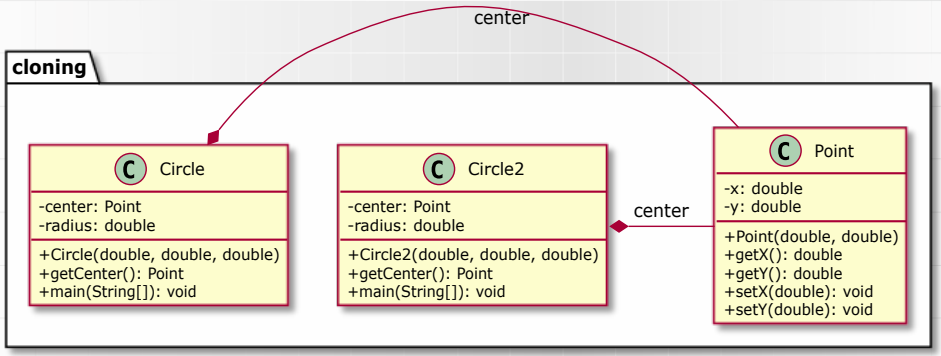
```
1 Boss 1 is doing some work!
2 Boss 2 is doing some work!
3
```

Section 5

Kopiowanie obiektów

Interfejs ’Cloneable’

Płytka kopia



Interfejs 'Cloneable'

Płytka kopia

Listing: Point.java

```
1  package cloning;
2  public class Point implements Cloneable {
3    private double x;
4    private double y;
5
6    public Point(double x, double y) { this.x=x; this.y=y;  }
7
8    public double getX() { return x; }
9
10   public double getY() { return y; }
11
12   public void setX(double x) { this.x = x; }
13
14   public void setY(double y) { this.y = y; }
15
16   @Override
17   public String toString() {
18     return "X: " + x + "\tY: " + y;
19   }
```

Interfejs 'Cloneable'

Płytka kopia

Listing: Point.java

```
20   @Override
21   protected Object clone() throws CloneNotSupportedException {
22     return super.clone();
23   }
24 }
```

Interfejs 'Cloneable'

Płytka kopia

Listing: Circle.java

```java
package cloning;

public class Circle implements Cloneable {

  private Point center;
  private double radius;

  public Circle(double x, double y, double radius) {
    this.center = new Point(x,y);
    this.radius = radius;
  }

  public Point getCenter() { return center;   }

  @Override
  protected Object clone() throws CloneNotSupportedException { return super.clone();  }

  @Override
  public String toString() {
    return "Circle with center: " + center + " and radius: " + radius;
  }
```

Interfejs 'Cloneable'

Płytka kopia

Listing: Circle.java

```
23   public static void main(String[] args) {
24     Circle circle = new Circle(1, 3, 5);
25     try {
26       Circle circle2 = (Circle) circle.clone(); // Shallow copy
27       System.out.println(circle2.toString());
28       circle.getCenter().setX(55);
29       System.out.println(circle2.toString());
30     } catch (CloneNotSupportedException e) {
31       e.printStackTrace();
32     }
33   }
34
35 }
```

```
1 Circle with center: X: 1.0^^IY: 3.0 and radius: 5.0
2 Circle with center: X: 55.0^^IY: 3.0 and radius: 5.0
3
```

Interfejs 'Cloneable'

Głęboka kopia

Listing: Circle2.java

```java
package cloning;

public class Circle2 implements Cloneable {

  private Point center;
  private double radius;

  public Circle2(double x, double y, double radius) {
    this.center = new Point(x,y);
    this.radius = radius;
  }

  public Point getCenter() { return center;   }

  @Override
  public String toString() {
    return "Circle with center: " + center + " and radius: " + radius;
  }
```

Interfejs 'Cloneable'

Głęboka kopia

Listing: Circle2.java

```
20  @Override
21  protected Object clone() throws CloneNotSupportedException {
22    Circle2 circle =(Circle2) super.clone();
23    circle.center = (Point) this.center.clone();
24    return circle;
25  }
```

Interfejs 'Cloneable'

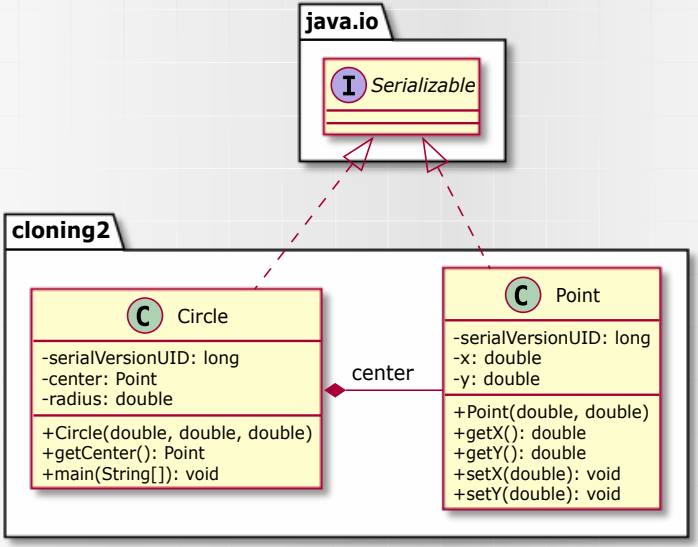
Głęboka kopia

Listing: Circle2.java

```
27   public static void main(String[] args) {
28     Circle2 circle = new Circle2(1, 3, 5);
29     try {
30       Circle2 circle2 = (Circle2) circle.clone(); // Deep copy
31       System.out.println(circle2.toString());
32       circle.getCenter().setX(55);
33       System.out.println(circle2.toString());
34     } catch (CloneNotSupportedException e) {
35       e.printStackTrace();
36     }
37   }
38
39 }
```

```
1 Circle with center: X: 1.0^^IY: 3.0 and radius: 5.0
2 Circle with center: X: 1.0^^IY: 3.0 and radius: 5.0
3
```

Interfejs ’Serializable’



Interfejs 'Serializable'

Listing: Point.java

```
1  package cloning2;
2  import java.io.Serializable;
3  public class Point implements Serializable {
4
5    private static final long serialVersionUID = -4177483652942167078L;
6    private double x;
7    private double y;
8
9    public Point(double x, double y) { this.x=x; this.y=y;  }
10
11   public double getX() { return x; }
12
13   public double getY() { return y; }
14
15   public void setX(double x) { this.x = x; }
16
17   public void setY(double y) { this.y = y; }
18
19   @Override
20   public String toString() {
21     return "X: " + x + "\tY: " + y;
22   }
```

Interfejs 'Serializable'

Listing: Circle.java

```java
package cloning2;
import java.io.Serializable;
import org.apache.commons.lang3.SerializationUtils;

public class Circle implements Serializable {

  private static final long serialVersionUID = 2106764739852167127L;
  private Point center;
  private double radius;

  public Circle(double x, double y, double radius) {
    this.center = new Point(x,y);
    this.radius = radius;
  }

  public Point getCenter() { return center;   }

  @Override
  public String toString() {
    return "Circle with center: " + center + " and radius: " + radius;
  }
```

Interfejs 'Serializable'

Listing: Circle.java

```
23    public static void main(String[] args) {
24      Circle circle = new Circle(1, 3, 5);
25      Circle circle2 = SerializationUtils.clone(circle);
26      System.out.println(circle2.toString());
27      circle.getCenter().setX(55);
28      System.out.println(circle2.toString());
29    }
30  }
```

```
1  Circle with center: X: 1.0^^IY: 3.0 and radius: 5.0
2  Circle with center: X: 1.0^^IY: 3.0 and radius: 5.0
3
```

Politechnika Wrocławska

Programowanie Obiektowe – Projektowanie Obiektowe

dr inż. Paweł Trajdos

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

5 lutego 2023