Politechnika Wrocławska

Wzorce Projektowe

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

Rodzaje wzorców

- Kreacyjne:
  - Singleton
  - Builder
  - Abstract Factory
  - Prototype
  - Factory Method

Politechnika Wrocławska

Rodzaje wzorców

- Strukturalne:
  - Wraper
  - Decorator
  - Facade
  - Composite
  - Bridge
  - Flyweight
  - Proxy

Rodzaje wzorców

- Behawioralne:
  - Command
  - Interpreter
  - Iterator
  - Mediator
  - Memento
  - Observer
  - Visitor
  - State
  - Strategy
  - Chain of responsibility

Politechnika Wrocławska

Rodzaje wzorców

- Pozostałe:
  - Dependency Injection
  - Lazy initialization
  - ....

Politechnika Wrocławska

Section 1

Wzorce kreacyjne

Subsection 1

Singleton

Singleton

Tylko jedna instancja klasy!

Politechnika Wrocławska

Implementacja

Listing: SingletonEager.java

```java
package singleton;

public class SingletonEager {
  private static SingletonEager object = new SingletonEager();

  private SingletonEager() {}

  public static SingletonEager getInstance() {
    return object;
  }


}
```

Test

Listing: SingletonEagerTest.java

```
1  package singleton;
2
3  import static org.junit.Assert.*;
4
5  import org.junit.Test;
6
7  public class SingletonEagerTest {
8
9    @Test
10   public void test() {
11     SingletonEager sing = SingletonEager.getInstance();
12     SingletonEager sing2 = SingletonEager.getInstance();
13     assertTrue("The same instance", sing == sing2);
14   }
15
16 }
```

Alternatywne implementacje

Singleton

Lazy initialization

Listing: SingletonLazy.java

```java
package singleton;

public class SingletonLazy {

  private static SingletonLazy obj;

  private SingletonLazy() {
    if(obj != null) {
      throw new IllegalStateException("Creating another instance is forbidden");
    }
  }

  public static SingletonLazy getInstance() {
    if(obj == null) {
      obj = new SingletonLazy();
    }
    return obj;
  }

}
```

Singleton

Lazy initialization

Listing: SingletonLazyTest.java

```
1  package singleton;
2
3  import static org.junit.Assert.*;
4
5  import org.junit.Test;
6
7  public class SingletonLazyTest {
8
9    @Test
10   public void test() {
11     SingletonLazy sing = SingletonLazy.getInstance();
12     SingletonLazy sing2 = SingletonLazy.getInstance();
13     assertTrue("The same instance", sing == sing2);
14   }
15
16 }
```

Singleton

Lazy initialization, Serialization

Listing: SingletonLazySerializableA.java

```
1  package singleton;
2  import java.io.Serializable;
3
4  public class SingletonLazySerializableA implements Serializable {
5
6    private static final long serialVersionUID = 537639362882624447 1L;
7    private static SingletonLazySerializableA obj;
8
9    private SingletonLazySerializableA() {
10     if(obj != null) {
11       throw new IllegalStateException("Creating another instance is forbidden");
12     }
13   }
14
15   public static SingletonLazySerializableA getInstance() {
16     if(obj == null) {
17       obj = new SingletonLazySerializableA();
18     }
19     return obj;
20   }
21 }
```

Singleton

Lazy initialization, Serialization

Listing: SingletonLazySerializableATest.java

```java
package singleton;

import static org.junit.Assert.*;

import org.apache.commons.lang3.SerializationUtils;
import org.junit.Test;

public class SingletonLazySerializableATest {

  @Test
  public void test() {
    SingletonLazySerializableA sing = SingletonLazySerializableA.getInstance();
    SingletonLazySerializableA sing2 = SingletonLazySerializableA.getInstance();
    assertTrue("The same instance", sing == sing2);
    //Let's do some serialization
    SingletonLazySerializableA sing3 = SerializationUtils.clone(sing);
    assertFalse("Equal?", sing == sing3);
  }

}
```

Singleton

Lazy initialization, Serialization

Listing: SingletonLazySerializableB.java

```java
package singleton;

import java.io.ObjectStreamException;
import java.io.Serializable;

public class SingletonLazySerializableB implements Serializable {

  private static final long serialVersionUID = 5376393628826244471L;
  private static SingletonLazySerializableB obj;

  private SingletonLazySerializableB() {
    if(obj != null) {
      throw new IllegalStateException("Creating another instance is forbidden");
    }
  }
  public static SingletonLazySerializableB getInstance() {
    if(obj == null) {
      obj = new SingletonLazySerializableB();
    }
    return obj;
  }
```

Singleton

Lazy initialization, Serialization

Listing: SingletonLazySerializableB.java

```
23          private Object readResolve() throws ObjectStreamException {
24            return getInstance();
25          }
26  }
```

Singleton

Lazy initialization, Serialization

Listing: SingletonLazySerializableBTest.java

```java
package singleton;

import static org.junit.Assert.*;

import org.apache.commons.lang3.SerializationUtils;
import org.junit.Test;

public class SingletonLazySerializableBTest {

  @Test
  public void test() {
    SingletonLazySerializableB sing = SingletonLazySerializableB.getInstance();
    SingletonLazySerializableB sing2 = SingletonLazySerializableB.getInstance();
    assertTrue("The same instance", sing == sing2);
    //Let's do some serialization
    SingletonLazySerializableB sing3 = SerializationUtils.clone(sing);
    assertTrue("Equal?", sing == sing3);
  }

}
```

Singleton

Lazy initialization, Cloneable interface

Listing: SingletonLazyCloneable.java

```
1  package singleton;
2
3  public class SingletonLazyCloneable implements Cloneable {
4
5    private static SingletonLazyCloneable obj;
6
7    private SingletonLazyCloneable() {
8      if(obj != null) {
9        throw new IllegalStateException("Creating another instance is forbidden");
10     }
11   }
12
13   public static SingletonLazyCloneable getInstance() {
14     if(obj == null) {
15       obj = new SingletonLazyCloneable();
16     }
17     return obj;
18   }
```

Singleton

Lazy initialization, Cloneable interface

Listing: SingletonLazyCloneable.java

```
20    @Override
21    public Object clone() throws CloneNotSupportedException {
22      return getInstance();
23    }
24  }
```

Singleton

Lazy initialization, Cloneable interface

Listing: SingletonLazyCloneableTest.java

```
1  package singleton;
2
3  import static org.junit.Assert.*;
4  import org.junit.Test;
5
6  public class SingletonLazyCloneableTest {
7
8    @Test
9    public void test() {
10     SingletonLazyCloneable sing = SingletonLazyCloneable.getInstance();
11     SingletonLazyCloneable sing2 = SingletonLazyCloneable.getInstance();
12     assertEquals("Equality", sing, sing2);
13
14     try {
15       SingletonLazyCloneable sing3 = (SingletonLazyCloneable) sing.clone();
16       assertEquals("Equality", sing, sing3);
17     } catch (CloneNotSupportedException e) {
18       fail();
19     }
20   }
21 }
```

Singleton jako prywatna statyczna klasa

Listing: SingletonStatic.java

```java
package singleton;

public class SingletonStatic {
  private SingletonStatic() {
  }

  private static class Holder{
    private static final SingletonStatic inst = new SingletonStatic();
  }

  public static SingletonStatic getInstance() {
    return Holder.inst;
  }
}
```

Singleton jako prywatna statyczna klasa

Listing: SingletonStaticTest.java

```java
package singleton;

import static org.junit.Assert.*;

import org.junit.Test;

public class SingletonStaticTest {

  @Test
  public void test() {
    SingletonStatic sing = SingletonStatic.getInstance();
    SingletonStatic sing2 = SingletonStatic.getInstance();
    assertEquals("Equal instances ", sing, sing2);
  }

}
```

Politechnika Wrocławska

Singleton jako Enum

Listing: SingletonEnum.java

```java
package singleton;

public enum SingletonEnum {
  INSTANCE;

  public void doSth() {
    System.out.println("STH");
  }
}
```

Singleton jako Enum

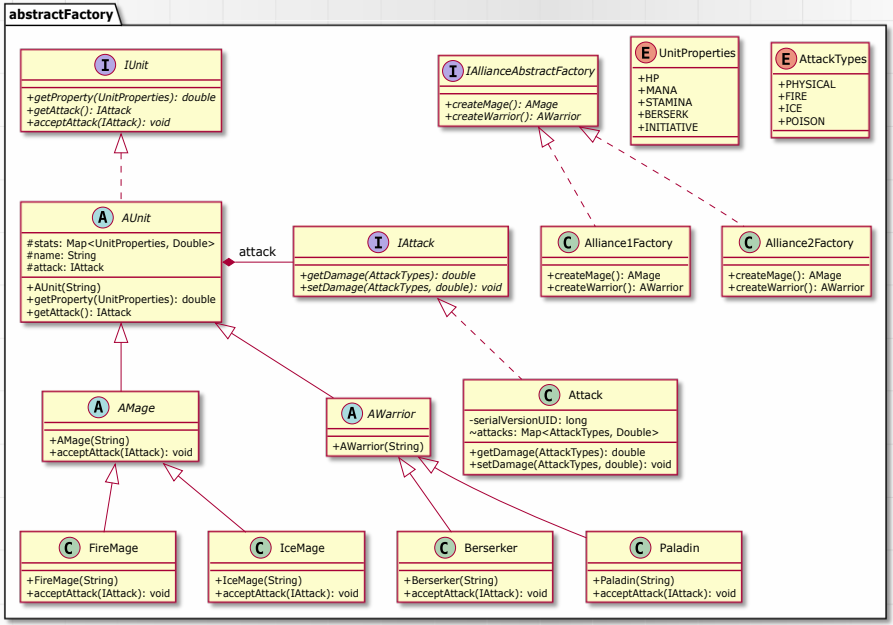
Listing: SingletonEnumTest.java

```
1  package singleton;
2
3  import static org.junit.Assert.*;
4
5  import org.junit.Test;
6
7  public class SingletonEnumTest {
8
9    @Test
10   public void test() {
11     SingletonEnum e1 = SingletonEnum.INSTANCE;
12     SingletonEnum e2 = SingletonEnum.INSTANCE;
13     assertEquals("Equality", e1, e2);
14
15     SingletonEnum.INSTANCE.doSth();
16   }
17
18 }
```

Politechnika Wrocławska

Subsection 2

Fabryka Abstrakcyjna


abstractFactory
IUnit
+getProperty(UnitProperties): double
+getAttack(): IAttack
+acceptAttack(IAttack): void
IAllianceAbstractFactory
+createMage(): AMage
+createWarrior(): AWarrior
UnitProperties
+HP
+MANA
+STAMINA
+BERSERK
+INITIATIVE
AttackTypes
+PHYSICAL
+FIRE
+ICE
+POISON
AUnit
#stats: Map<UnitProperties, Double>
#name: String
#attack: IAttack
+AUnit(String)
+getProperty(UnitProperties): double
+getAttack(): IAttack
attack
IAttack
+getDamage(AttackTypes): double
+setDamage(AttackTypes, double): void
Alliance1Factory
+createMage(): AMage
+createWarrior(): AWarrior
Alliance2Factory
+createMage(): AMage
+createWarrior(): AWarrior
AMage
+AMage(String)
+acceptAttack(IAttack): void
AWarrior
+AWarrior(String)
Attack
-serialVersionUID: long
~attacks: Map<AttackTypes, Double>
+getDamage(AttackTypes): double
+setDamage(AttackTypes, double): void
FireMage
+FireMage(String)
+acceptAttack(IAttack): void
IceMage
+IceMage(String)
+acceptAttack(IAttack): void
Berserker
+Berserker(String)
+acceptAttack(IAttack): void
Paladin
+Paladin(String)
+acceptAttack(IAttack): void

Politechnika Wrocławska

Listing: AttackTypes.java

```
1 package abstractFactory;
2
3 public enum AttackTypes {
4  PHYSICAL, FIRE, ICE, POISON
5 }
```

Politechnika Wrocławska

Listing: UnitProperties.java

```java
package abstractFactory;

public enum UnitProperties {
  HP,MANA,STAMINA,BERSERK,INITIATIVE

}
```

Politechnika Wrocławska

Listing: IUnit.java

```java
package abstractFactory;

public interface IUnit {

  public double getProperty(UnitProperties prop);
  public IAttack getAttack();
  public void acceptAttack(IAttack attack);

}
```

Politechnika Wrocławska

Listing: AUnit.java

```java
package abstractFactory;
import java.util.HashMap;
import java.util.Map;

public abstract class AUnit implements IUnit {
  protected Map<UnitProperties,Double> stats;
  protected String name;
  protected IAttack attack;

  public AUnit(String name) {
    this.name=name;
    stats = new HashMap<>();
    attack = new Attack();
  }
  @Override
  public double getProperty(UnitProperties prop) {
    Double propVal = stats.get(prop);
    if(propVal != null)
      return propVal.doubleValue();
    return 0;
  }
  @Override
  public IAttack getAttack() {return attack;}
}
```

Politechnika Wrocławska

Listing: AMAge.java

```java
package abstractFactory;

public abstract class AMage extends AUnit {

  public AMage(String name) {
    super(name);
    this.stats.put(UnitProperties.HP, 50.0);
    this.stats.put(UnitProperties.INITIATIVE, 30.0);
  }

  @Override
  public void acceptAttack(IAttack attack) {
    Double pDmg = attack.getDamage(AttackTypes.PHYSICAL);
    Double currHP = this.stats.get(UnitProperties.HP);
    if(pDmg !=null) {
      currHP-= 0.9*pDmg;
    }
    this.stats.put(UnitProperties.HP, currHP<0? 0:currHP );
  }

}
```

Politechnika Wrocławska

Listing: IceMAge.java

```java
package abstractFactory;
public class IceMage extends AMage {
  public IceMage(String name) {
    super(name);
    this.attack.setDamage(AttackTypes.ICE, 30);
  }
  @Override
  public void acceptAttack(IAttack attack) {
    super.acceptAttack(attack);
    Double currHP = this.stats.get(UnitProperties.HP);
    Double fDmg = attack.getDamage(AttackTypes.FIRE);
    if(fDmg !=null) {
      currHP-= 0.1*fDmg;
    }
    this.stats.put(UnitProperties.HP, currHP<0? 0:currHP );
  }
}
```

Politechnika Wrocławska

Listing: FireMAge.java

```java
package abstractFactory;
public class FireMage extends AMage {
  public FireMage(String name) {
    super(name);
    this.attack.setDamage(AttackTypes.FIRE, 30);
  }
  @Override
  public void acceptAttack(IAttack attack) {
    super.acceptAttack(attack);
    Double currHP = this.stats.get(UnitProperties.HP);
    Double fDmg = attack.getDamage(AttackTypes.ICE);
    if(fDmg !=null) {
      currHP-= 0.1*fDmg;
    }
    this.stats.put(UnitProperties.HP, currHP<0? 0:currHP );
  }
}
```

Politechnika Wrocławska

Listing: AWarrior.java

```java
package abstractFactory;

public abstract class AWarrior extends AUnit {

  public AWarrior(String name) {
    super(name);
    this.stats.put(UnitProperties.HP, 100.0);
    this.attack.setDamage(AttackTypes.PHYSICAL, 10);
  }

}
```

Politechnika Wrocławska

Listing: Berserker.java

```java
package abstractFactory;

public class Berserker extends AWarrior {
  public Berserker(String name) {
    super(name);
    this.stats.put(UnitProperties.BERSERK, 100.0);
  }
  @Override
  public void acceptAttack(IAttack attack) {
    Double pDmg = attack.getDamage(AttackTypes.PHYSICAL);
    Double currHP = this.stats.get(UnitProperties.HP);
    if(pDmg !=null) {
      currHP-= pDmg;
    }
    Double fDmg = attack.getDamage(AttackTypes.FIRE);
    if(fDmg !=null) {
      currHP-= 0.5*fDmg;
    }
    this.stats.put(UnitProperties.HP, currHP<0? 0:currHP );
  }
}
```

Politechnika Wrocławska

Listing: Paladin.java

```java
package abstractFactory;

public class Paladin extends AWarrior {
  public Paladin(String name) {
    super(name);
    this.stats.put(UnitProperties.INITIATIVE, 100.0);
  }
  @Override
  public void acceptAttack(IAttack attack) {
    Double pDmg = attack.getDamage(AttackTypes.PHYSICAL);
    Double currHP = this.stats.get(UnitProperties.HP);
    if(pDmg !=null) {
      currHP-= 0.1*pDmg;
    }
    this.stats.put(UnitProperties.HP, currHP<0? 0:currHP );
  }
}
```

Politechnika Wrocławska

Listing: IAttack.java

```java
package abstractFactory;

public interface IAttack {

  public double getDamage(AttackTypes type);
  public void setDamage(AttackTypes type, double value);

}
```

Politechnika Wrocławska

Listing: Attack.java

```java
package abstractFactory;

import java.util.HashMap;
import java.util.Map;

public class Attack implements IAttack {

  private static final long serialVersionUID = -1692503483936934114L;
  Map<AttackTypes, Double> attacks;

  public Attack() {
    attacks = new HashMap<>();
  }
  @Override
  public double getDamage(AttackTypes type) {
    Double damage = attacks.get(type);
    return (damage != null)? damage.doubleValue():0;
  }
  @Override
  public void setDamage(AttackTypes type, double value) {
    attacks.put(type, value);
  }
}
```

Politechnika Wrocławska

Listing: IAllianceAbstractFactory.java

```java
package abstractFactory;

public interface IAllianceAbstractFactory {

  public AMage createMage();

  public AWarrior createWarrior();

}
```

Politechnika Wrocławska

Listing: Alliance1Factory.java

```java
package abstractFactory;

public class Alliance1Factory implements IAllianceAbstractFactory {

  @Override
  public AMage createMage() {
    return new FireMage("Merlin");
  }

  @Override
  public AWarrior createWarrior() {
    return new Berserker("Bjorn");
  }

}
```

Politechnika Wrocławska

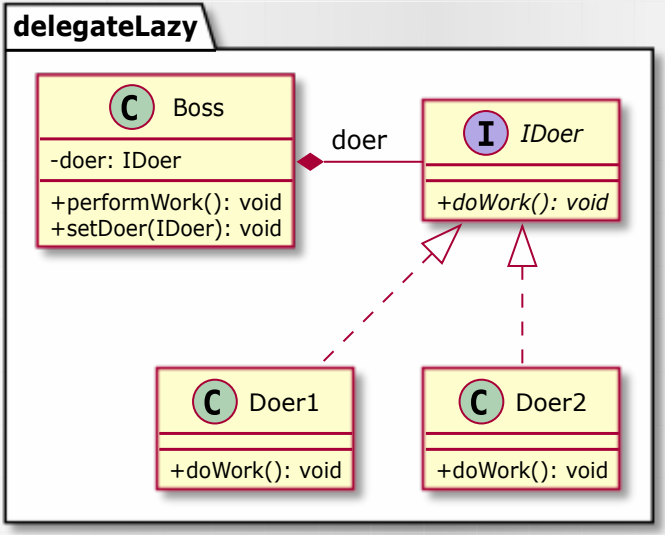
Listing: Alliance2Factory.java

```java
package abstractFactory;

public class Alliance2Factory implements IAllianceAbstractFactory {

  @Override
  public AMage createMage() {
    return new IceMage("Morrigan");
  }

  @Override
  public AWarrior createWarrior() {
    return new Paladin("Lancelot");
  }

}
```

Section 2

Lazy Initialization

Lazy Initialization



Lazy Initialization

Listing: Boss.java

```
1  package delegateLazy;
2
3  public class Boss {
4
5    private IDoer doer;
6
7    public void performWork() {
8      if(doer == null)
9        doer = new Doer1(); //Lazy initialization
10     doer.doWork();
11   }
12
13   public void setDoer(IDoer doer) { this.doer = doer; }
14
15 }
```

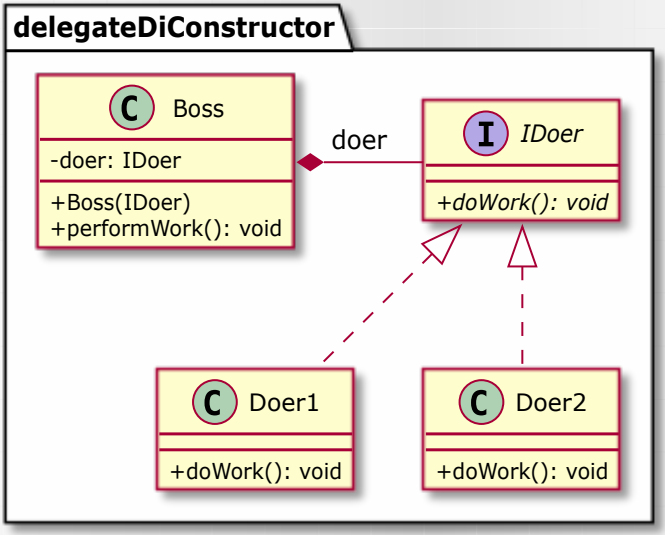
Politechnika Wrocławska

Section 3

Dependency Injection

Dependency Injection

via Constructor



Dependency Injection

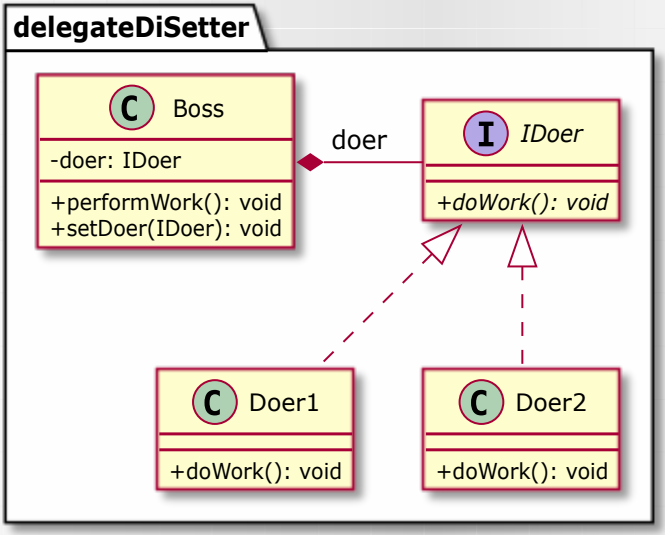
via Constructor

Listing: Boss.java

```java
package delegateDiConstructor;

public class Boss {

  private IDoer doer;

  public Boss(IDoer doer) throws Exception {
    if(doer == null)
      throw new Exception("Doer must not be null!");
    this.doer = doer;
  }

  public void performWork() {
    doer.doWork();
  }


}
```

Dependency Injection

via Setter



Dependency Injection

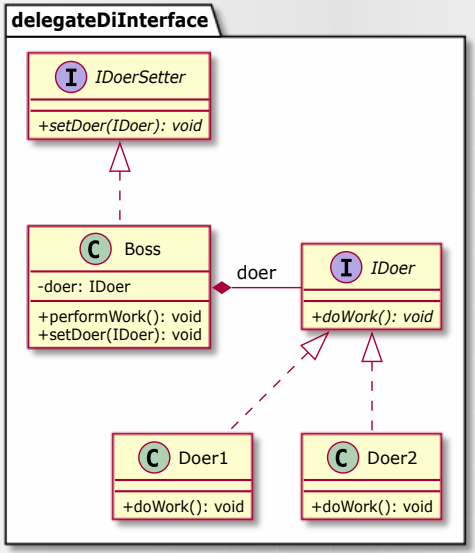
via Setter

Listing: Boss.java

```
1  package delegateDiSetter;
2
3  public class Boss {
4
5    private IDoer doer;
6
7    public void performWork() {
8      doer.doWork();//Delegate
9    }
10
11   public void setDoer(IDoer doer) throws Exception {
12     if(doer == null)
13       throw new Exception("Doer must not be null!");
14     this.doer = doer;
15   }
16
17 }
```

Dependency Injection

via Interface



Politechnika Wrocławska

Dependency Injection

via Interface

Listing: IDoerSetter.java

```
1 package delegateDiInterface;
2
3 public interface IDoerSetter {
4   public void setDoer(IDoer doer) throws Exception;
5 }
```

Dependency Injection

via Interface

Listing: Boss.java

```
1  package delegateDiInterface;
2
3  public class Boss implements IDoerSetter {
4
5    private IDoer doer;
6
7    public void performWork() {
8      doer.doWork();//Delegate
9    }
10   @Override
11   public void setDoer(IDoer doer) throws Exception {
12     if(doer == null)
13       throw new Exception("Doer must not be null!");
14     this.doer = doer;
15   }
16
17 }
```

Politechnika Wrocławska

Wzorce Projektowe

dr inż. Paweł Trajdos

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

5 lutego 2023