Uniwersytet Wrocławski

Rafał Lewandków

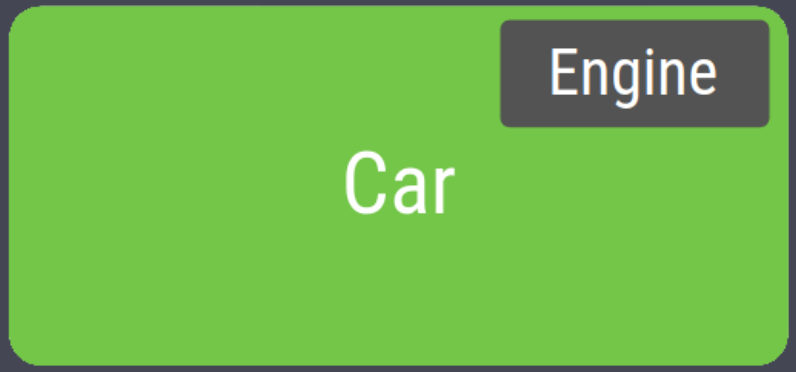
PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

WYKŁAD 12

- Dagger

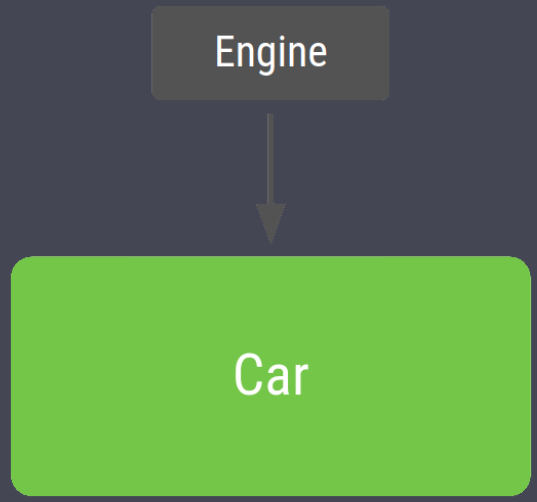
Dagger

```
class Car {

    private Engine engine = new Engine();

    public void start() {
        engine.start();
    }
}


class MyApp {
    public static void main(String[] args) {
        Car car = new Car();
        car.start();
    }
}
```



Dagger

```
class Car {

    private final Engine engine;

    public Car(Engine engine) {
        this.engine = engine;
    }

    public void start() {
        engine.start();
    }
}


class MyApp {
    public static void main(String[] args) {
        Engine engine = new Engine();
        Car car = new Car(engine);
        car.start();
    }
}
```



Uniwersytet Wrocławski

Dagger – constructor injection

```
class Computer {}
class Case {}
class CPU {}
class GPU {}
class Motherboard {}
class PowerSupply {}
```

```
class Computer (
    private val case: Case,
    private val gpu: GPU,
    private val cpu: CPU,
    private val motherboard: Motherboard,
    private val powerSupply: PowerSupply
        ) {
    fun work(): String{
        return "working"
    }
}
```

Uniwersytet Wrocławski

Dagger – constructor injection

```
@Component
interface ComputerComponent {
    fun getComputer(): Computer
}
```

```
class Computer @Inject constructor (
    private val case: Case,
    private val gpu: GPU,
    private val cpu: CPU,
    private val motherboard: Motherboard,
    private val powerSupply: PowerSupply
        ) {
    fun work(): String{
        return "working"
    }
}
```

```
class Case @Inject constructor()
class CPU @Inject constructor()
class GPU @Inject constructor()
class Motherboard @Inject constructor()
class PowerSupply @Inject constructor()
```

Uniwersytet Wrocławski

Dagger – constructor injection

```
class Computer @Inject constructor (
    private val case: Case,
    private val gpu: GPU,
    private val cpu: CPU,
    private val motherboard: Motherboard,
    private val powerSupply: PowerSupply
        ) {
    fun work(): String{
        return "working"
    }
}
```

```
class Case @Inject constructor()
class CPU @Inject constructor()
class GPU @Inject constructor()
class Motherboard @Inject constructor()
class PowerSupply @Inject constructor()
```

```
val component = DaggerComputerComponent.create()
computer = component.getComputer()
```

Uniwersytet Wrocławski

Dagger – field injection

```
@Component
interface ComputerComponent {
    fun inject(activity: MainActivity)
}
```

```
@Inject
lateinit var computer: Computer
```

```
val component = DaggerComputerComponent.create()
component.inject(this)
```

Uniwersytet Wrocławski

Dagger – method injection

```
class Monitor @Inject constructor() {
    fun setListener(computer: Computer): String{
        return "monitor connected"
    }
}
```

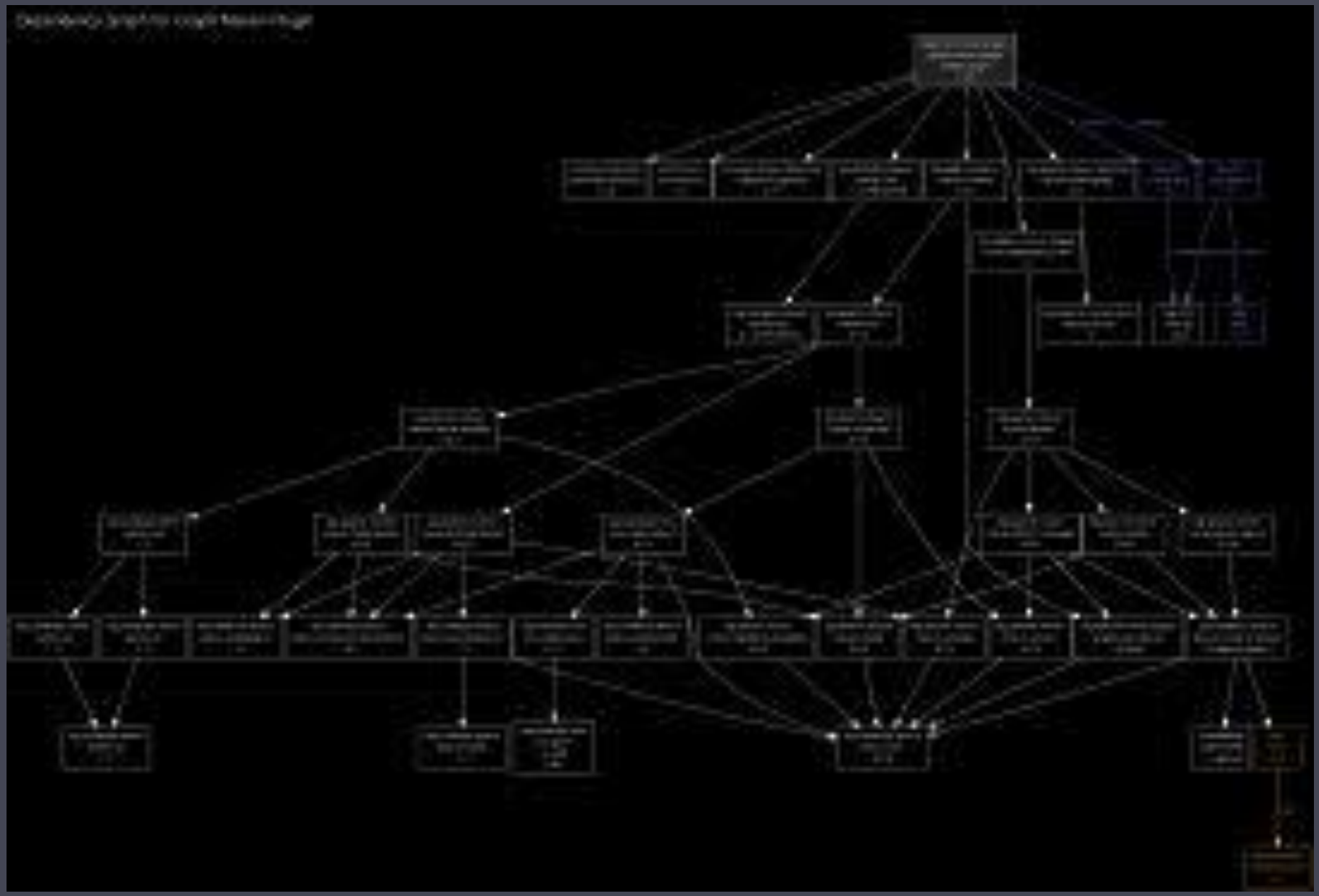
```
class Computer @Inject constructor (
    private val case: Case,
    private val gpu: GPU,
    private val cpu: CPU,
    private val motherboard: Motherboard,
    private val powerSupply: PowerSupply
        ) {

    var text: String = " "

    fun work(): String{
        return "working"
    }

    @Inject
    fun monitor(monitor: Monitor){
        text = monitor.setComputer(this)
    }
}
```

Dagger

- najpopularniejsze jest wstrzykiwanie przez konstruktor
- jeżeli w klasie mamy wszystkie trzy typy wstrzyknięć, kolejność wykonania jest następująca
  - konstruktor
  - pole
  - metoda
- rzadko wykorzystuje się więcej niż jeden sposób
- wykorzystanie wszystkich trzech jest niespotykane
- jednym z niewielu zastosowań wstrzykiwania przez metodę jest sytuacja w której musimy przekazać instancję jako argument

Uniwersytet Wrocławski

Dagger



Dagger

```
@Module
public class APIModule {

    private static final String BASE_URL = "https://raw.githubusercontent.com";

    @Provides
    public PhysicistAPI providePhysicistAPI(){
        return new Retrofit.Builder()
                .baseUrl(BASE_URL)
                .addConverterFactory(GsonConverterFactory.create())
                .addCallAdapterFactory(RxJava3CallAdapterFactory.create())
                .build()
                .create(PhysicistAPI.class);
    }
}
```

Dagger

```
public class PhysicistService {

    @Inject
    PhysicistAPI api;

    public Single<List<Physicist>> getPhysicist(){
        return api.getPhysicists();
    }
}
```

Dagger

```
@Component(modules = {APIModule.class})
public interface APIComponent {
    void inject(PhysicistService service);
}
```

Dagger

```
public class PhysicistService {

    @Inject
    PhysicistAPI api;

    public PhysicistService (){
        DaggerAPIComponent.create().inject(this);
    }
```

Dagger

```java
@Module
public class APIModule {

    private static final String BASE_URL = "https://raw.githubusercontent.com";

    @Provides
    public PhysicistAPI providePhysicistAPI(){
        return new Retrofit.Builder()
                .baseUrl(BASE_URL)
                .addConverterFactory(GsonConverterFactory.create())
                .addCallAdapterFactory(RxJava3CallAdapterFactory.create())
                .build()
                .create(PhysicistAPI.class);
    }
}
```

```java
@Component(modules = {APIModule.class})
public interface APIComponent {
    void inject(PhysicistService service);
}
```

```java
public class PhysicistService {

    @Inject
    PhysicistAPI api;

    public PhysicistService (){
        DaggerAPIComponent.create().inject(this);
    }
```