Guice

Jak ułatwić sobie życie z zależnościami

Mateusz Sobczak

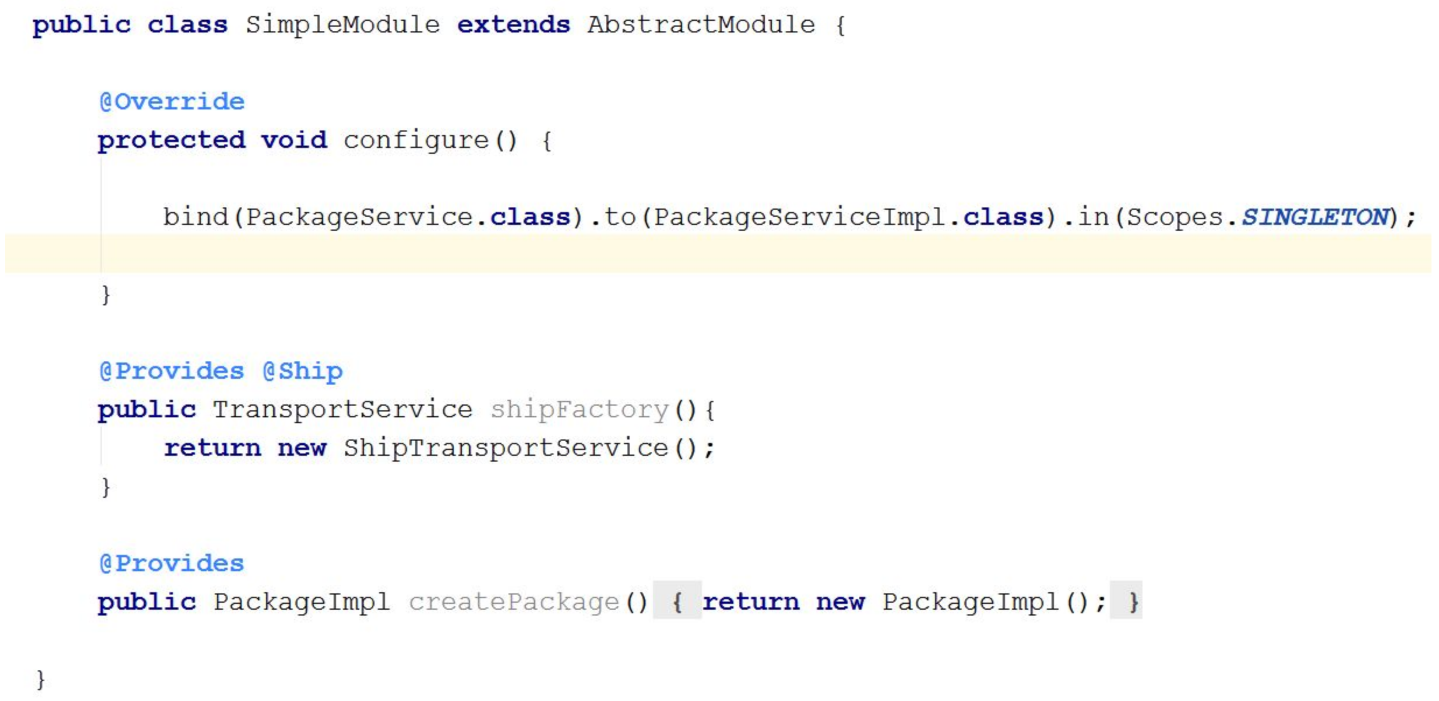
GitHub, Twitter: @zdzisiekcom

https://github.com/zdzisiekcom/guice-scope

Wstrzykiwanie w Guice w wielkim skrócie

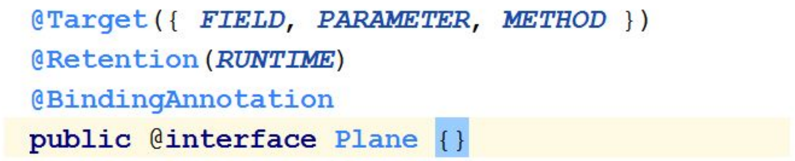
- Konfiguracja kontekstu w modułach
- Adnotacje do rozróżnienia implementacji w konfiguracji
- Scope-y
- Provider-y i ich wstrzykiwanie
  - Wiele instancji obiektów
  - Lazy loading
  - Różne scop-y w obiekcie

Konfiguracja w modulach

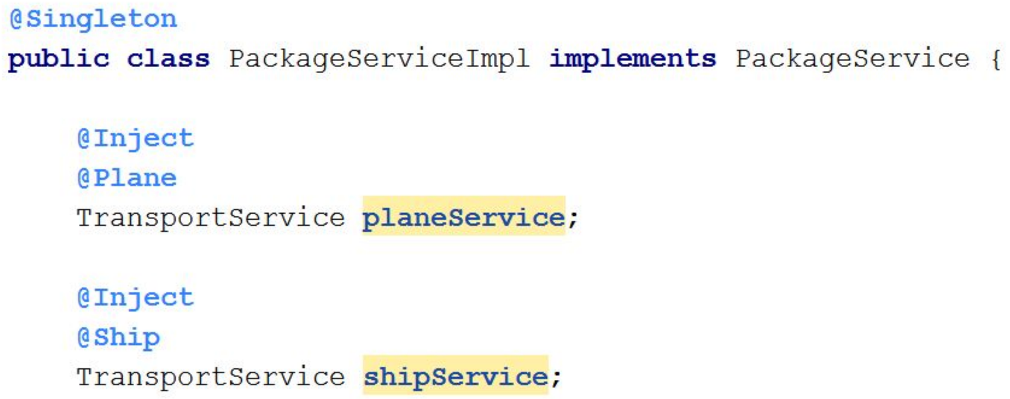


```
public class SimpleModule extends AbstractModule {

    @Override
    protected void configure() {

        bind(PackageService.class).to(PackageServiceImpl.class).in(Scopes.SINGLETON);

    }

    @Provides @Ship
    public TransportService shipFactory(){
        return new ShipTransportService();
    }

    @Provides
    public PackageImpl createPackage() { return new PackageImpl(); }

}
```

Adnotacje do róznych implementacji

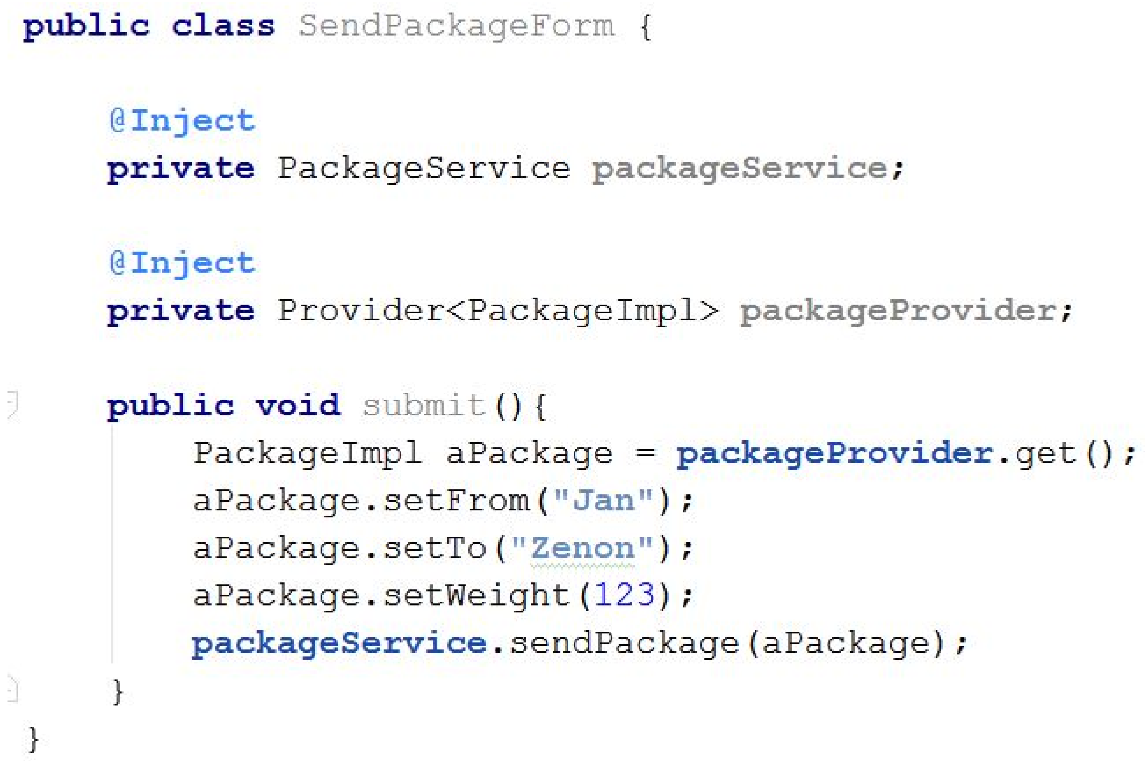


```java
@Target({ FIELD, PARAMETER, METHOD })
@Retention(RUNTIME)
@BindingAnnotation
public @interface Plane {}
```



```java
@Singleton
public class PackageServiceImpl implements PackageService {

    @Inject
    @Plane
    TransportService planeService;

    @Inject
    @Ship
    TransportService shipService;
```

Wstrzykiwanie providerów

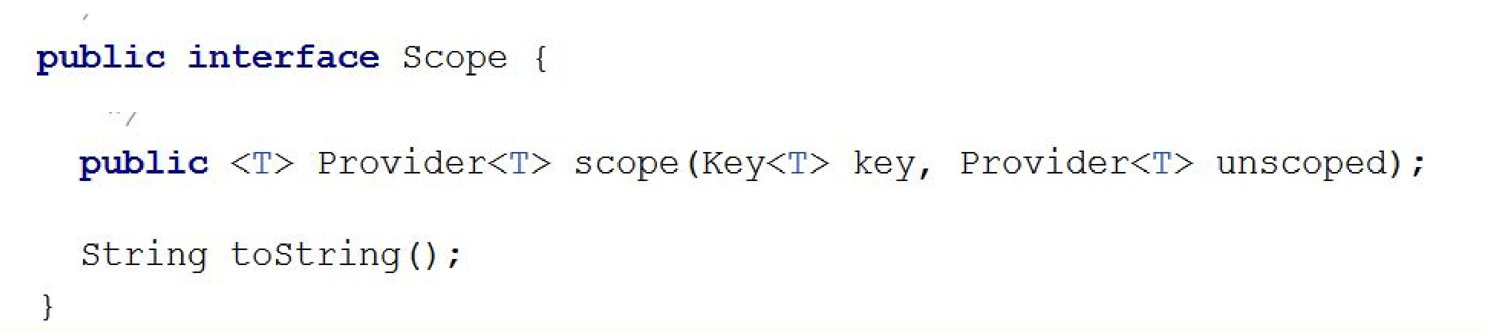


```java
public class SendPackageForm {

    @Inject
    private PackageService packageService;

    @Inject
    private Provider<PackageImpl> packageProvider;

    public void submit(){
        PackageImpl aPackage = packageProvider.get();
        aPackage.setFrom("Jan");
        aPackage.setTo("Zenon");
        aPackage.setWeight(123);
        packageService.sendPackage(aPackage);
    }
}
```

Wlasny zakres wstrzykiwanych zaleznosci

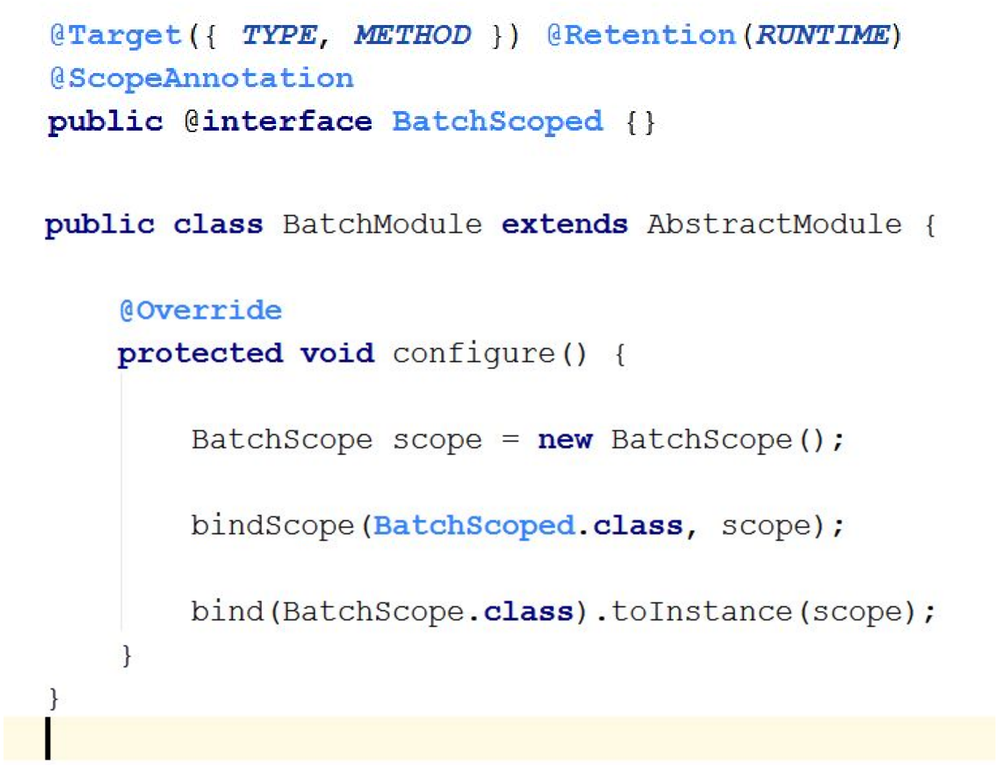
- Dlaczego używać własnego Scope-a?
- Jak go stworzyć?
- Kontrolowanie cyklu życia
- Przykłady użycia
    - Batch Scope
    - Request/Session scope
    - Window Scope
    - Hourly Scope

Batch Scope



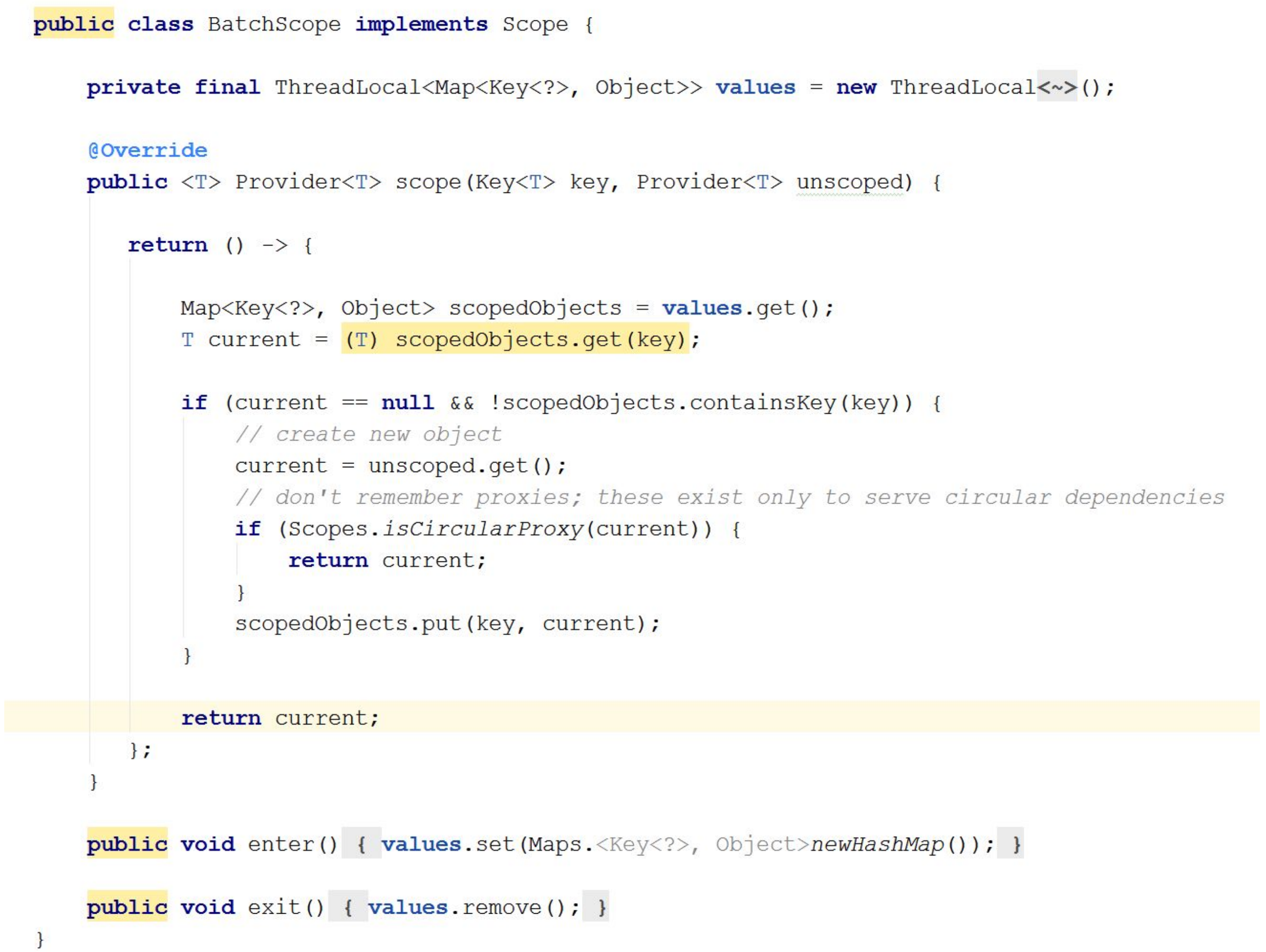
```
   /
public interface Scope {

    */

  public <T> Provider<T> scope(Key<T> key, Provider<T> unscoped);

  String toString();
}
```

Batch Scope



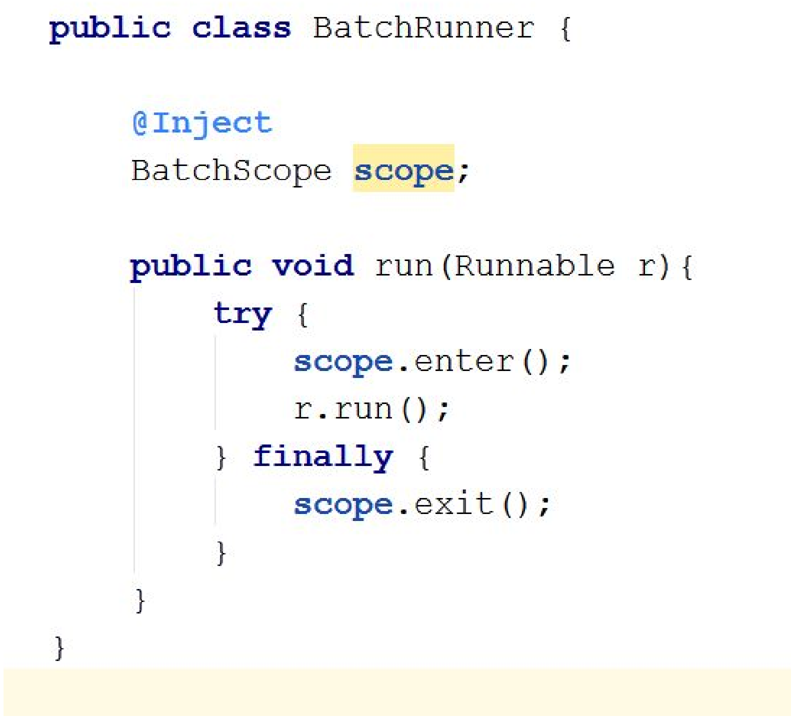
```java
@Target({ TYPE, METHOD }) @Retention(RUNTIME)
@ScopeAnnotation
public @interface BatchScoped {}

public class BatchModule extends AbstractModule {

    @Override
    protected void configure() {

        BatchScope scope = new BatchScope();

        bindScope(BatchScoped.class, scope);

        bind(BatchScope.class).toInstance(scope);
    }
}
```

Batch Scope



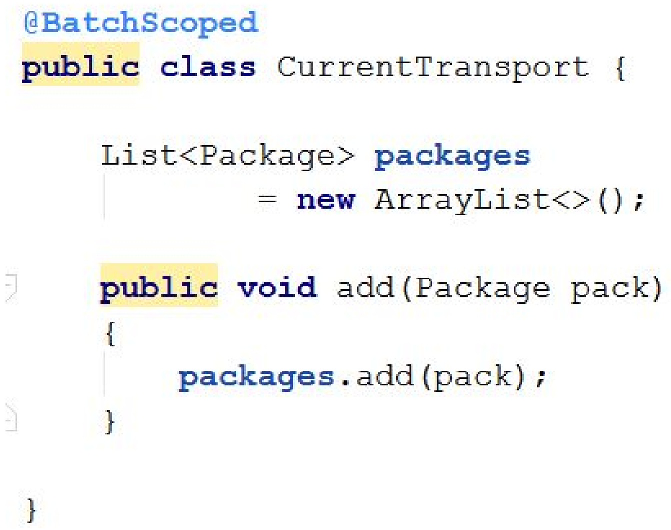
```java
public class BatchScope implements Scope {

    private final ThreadLocal<Map<Key<?>, Object>> values = new ThreadLocal<~>();

    @Override
    public <T> Provider<T> scope(Key<T> key, Provider<T> unscoped) {

        return () -> {

            Map<Key<?>, Object> scopedObjects = values.get();
            T current = (T) scopedObjects.get(key);

            if (current == null && !scopedObjects.containsKey(key)) {
                // create new object
                current = unscoped.get();
                // don't remember proxies; these exist only to serve circular dependencies
                if (Scopes.isCircularProxy(current)) {
                    return current;
                }
                scopedObjects.put(key, current);
            }

            return current;
        };
    }

    public void enter() { values.set(Maps.<Key<?>, Object>newHashMap()); }

    public void exit() { values.remove(); }
}
```

Batch Scope - kontroler

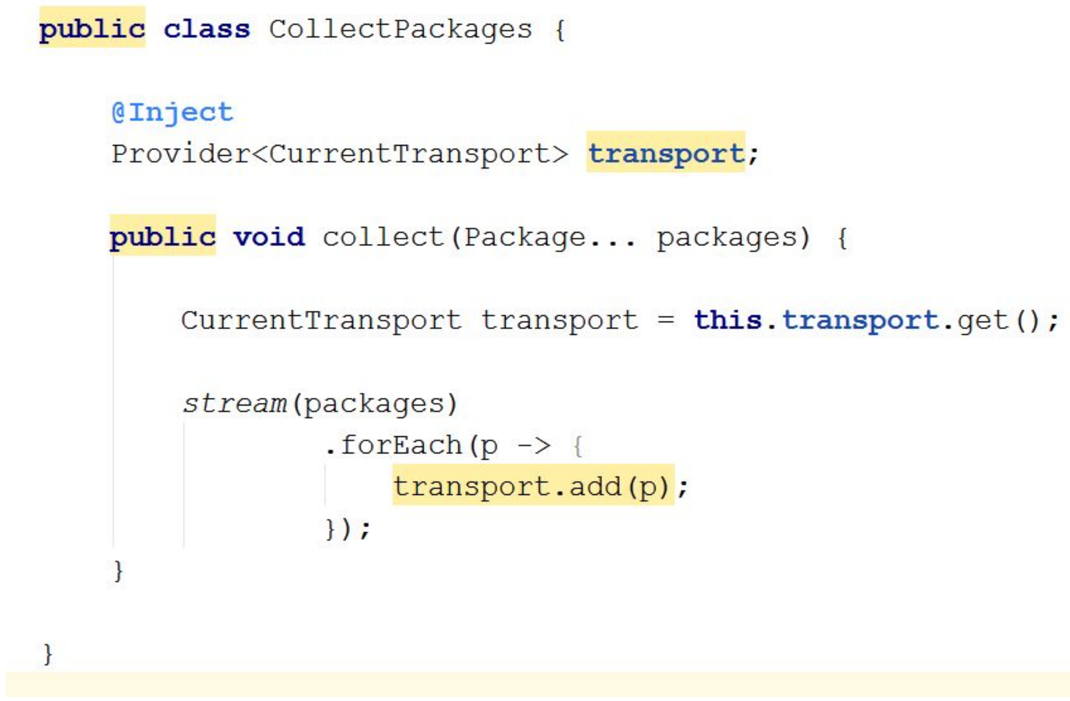


```
public class BatchRunner {

    @Inject
    BatchScope scope;

    public void run(Runnable r){
        try {
            scope.enter();
            r.run();
        } finally {
            scope.exit();
        }
    }
}
```

Batch Scope - uzycie

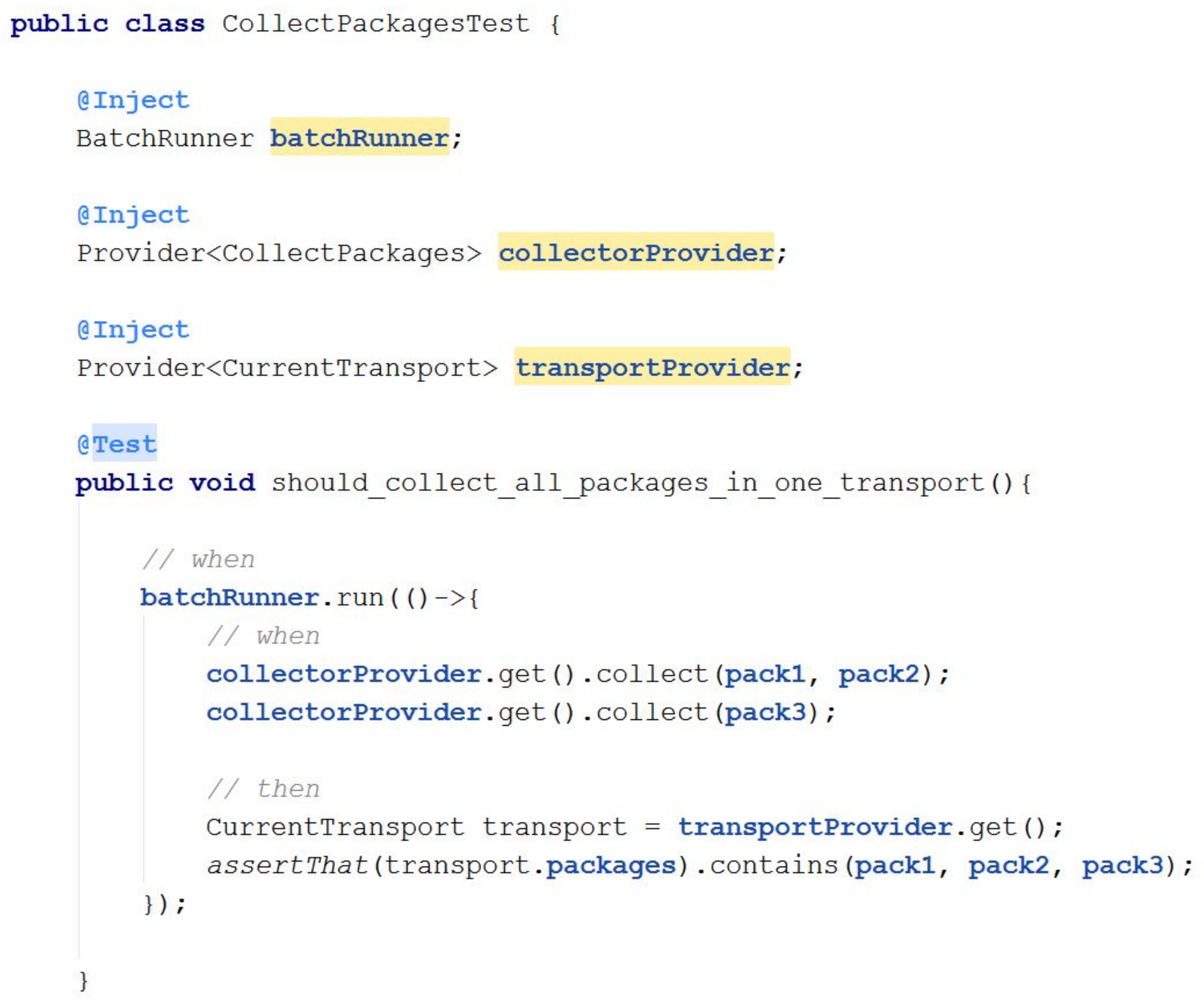


```java
@BatchScoped
public class CurrentTransport {

    List<Package> packages
            = new ArrayList<>();

    public void add(Package pack)
    {
        packages.add(pack);
    }

}
```



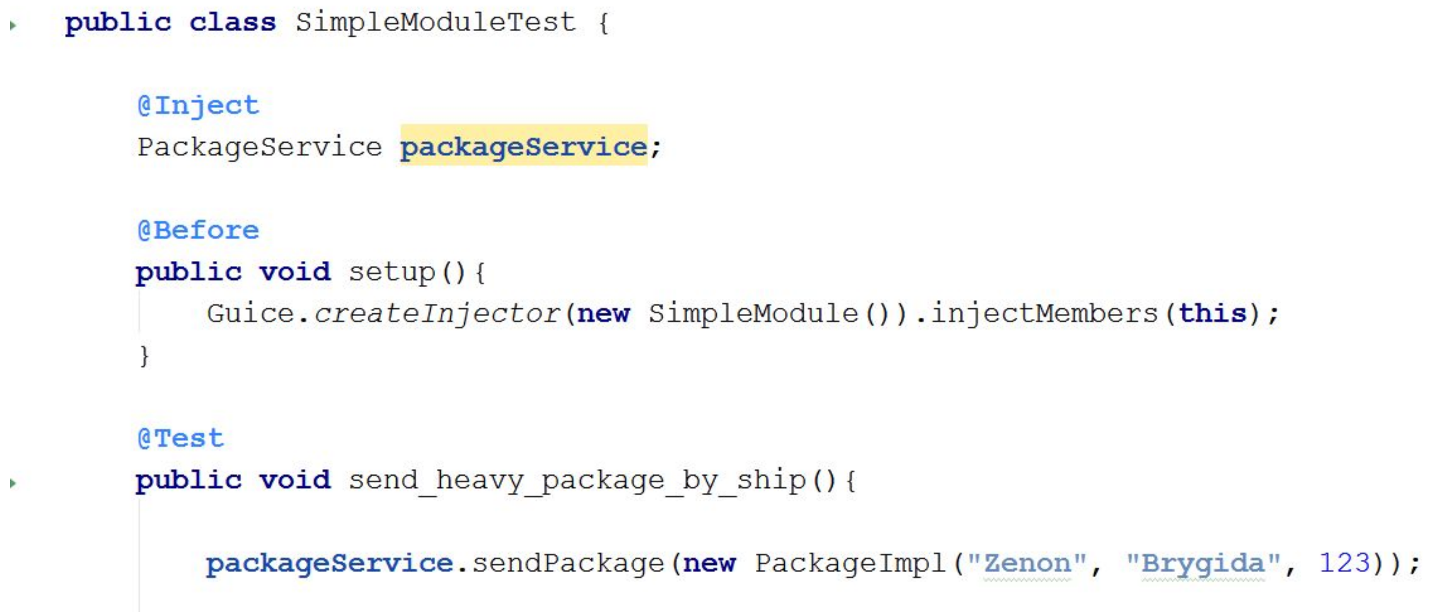
```java
public class CollectPackages {

    @Inject
    Provider<CurrentTransport> transport;

    public void collect(Package... packages) {

        CurrentTransport transport = this.transport.get();

        stream(packages)
                .forEach(p -> {
                    transport.add(p);
                });
    }

}
```

Batch Scope



```java
public class CollectPackagesTest {

    @Inject
    BatchRunner batchRunner;

    @Inject
    Provider<CollectPackages> collectorProvider;

    @Inject
    Provider<CurrentTransport> transportProvider;

    @Test
    public void should_collect_all_packages_in_one_transport(){

        // when
        batchRunner.run(()->{
            // when
            collectorProvider.get().collect(pack1, pack2);
            collectorProvider.get().collect(pack3);

            // then
            CurrentTransport transport = transportProvider.get();
            assertThat(transport.packages).contains(pack1, pack2, pack3);
        });

    }
```

Guice w testach

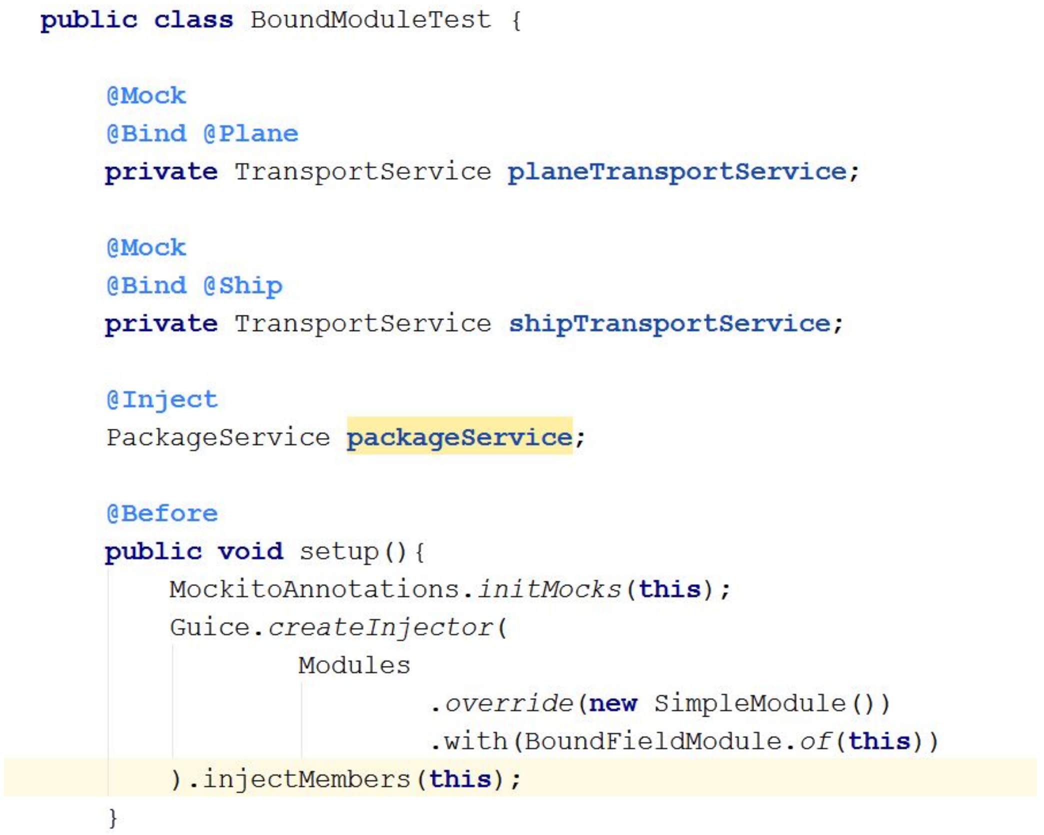


```java
public class SimpleModuleTest {

    @Inject
    PackageService packageService;

    @Before
    public void setup(){
        Guice.createInjector(new SimpleModule()).injectMembers(this);
    }

    @Test
    public void send_heavy_package_by_ship(){

        packageService.sendPackage(new PackageImpl("Zenon", "Brygida", 123));
```

Nadpisywanie modułów

```java
public class OverrideModuleTest {

    @Mock private TransportService shipTransportService;

    @Inject
    PackageService packageService;

    @Before
    public void setup(){
        MockitoAnnotations.initMocks(this);
        Guice.createInjector(
                Modules
                        .override(new SimpleModule())
                        .with(new AbstractModule() {
                            @Override
                            protected void configure() {

                                bind(TransportService.class).annotatedWith(Ship.class)
                                        .toInstance(shipTransportService);

                            }
                        })
        ).injectMembers(this);
    }
```

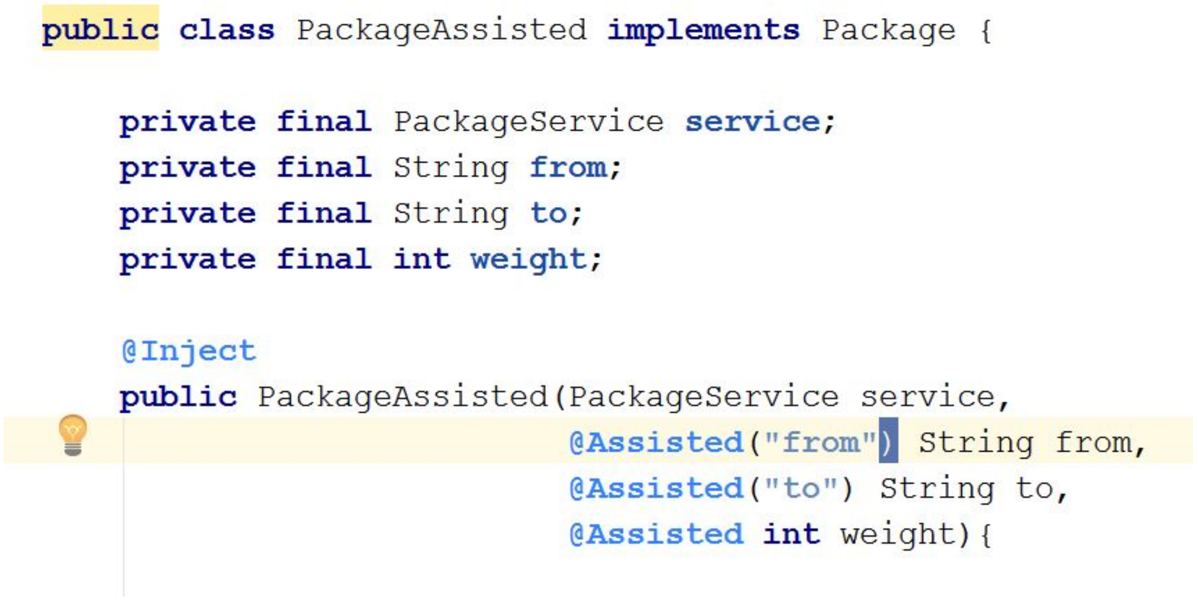
BoundFieldsModule



```java
public class BoundModuleTest {

    @Mock
    @Bind @Plane
    private TransportService planeTransportService;

    @Mock
    @Bind @Ship
    private TransportService shipTransportService;

    @Inject
    PackageService packageService;

    @Before
    public void setup(){
        MockitoAnnotations.initMocks(this);
        Guice.createInjector(
                Modules
                        .override(new SimpleModule())
                        .with(BoundFieldModule.of(this))
        ).injectMembers(this);
    }
```

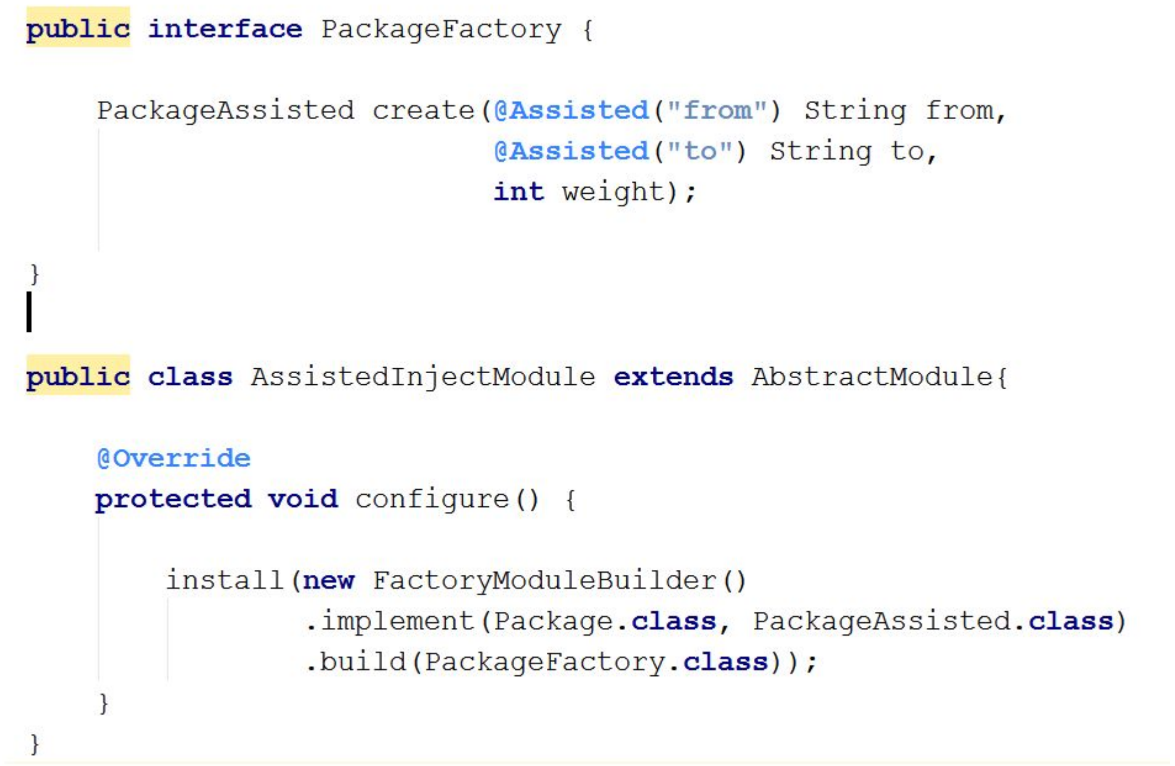
Tworzenie ze wspomaganiem

- AssistedInject do automatycznego tworzenia wspomaganych przez guice-a factory



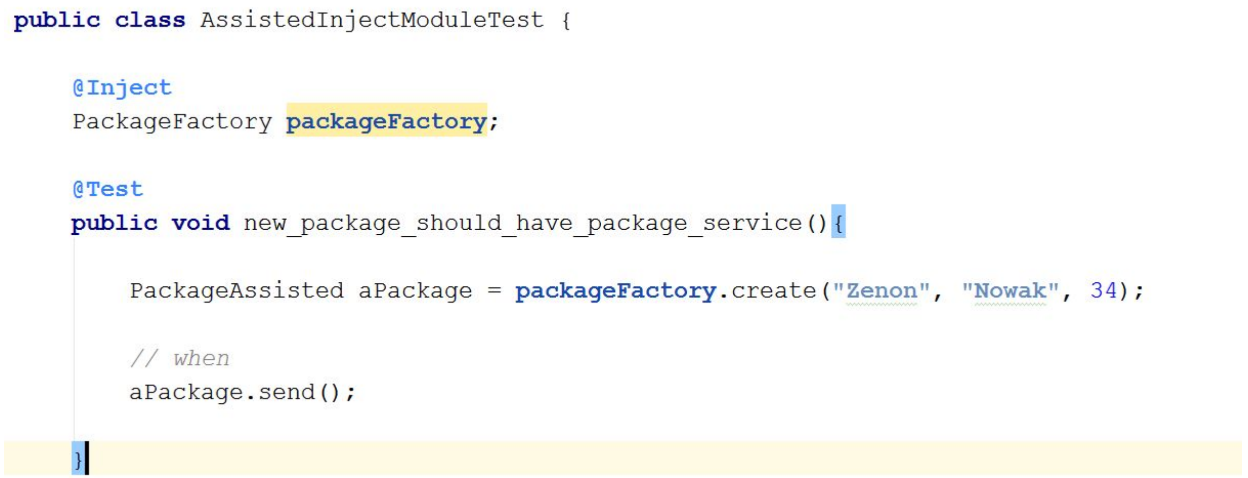
```java
public class PackageAssisted implements Package {

    private final PackageService service;
    private final String from;
    private final String to;
    private final int weight;

    @Inject
    public PackageAssisted(PackageService service,
                           @Assisted("from") String from,
                           @Assisted("to") String to,
                           @Assisted int weight){
```

Assisted Inject



```java
public interface PackageFactory {

    PackageAssisted create(@Assisted("from") String from,
                           @Assisted("to") String to,
                           int weight);

}

public class AssistedInjectModule extends AbstractModule{

    @Override
    protected void configure() {

        install(new FactoryModuleBuilder()
                .implement(Package.class, PackageAssisted.class)
                .build(PackageFactory.class));
    }
}
```

Assisted Inject



```
public class AssistedInjectModuleTest {

    @Inject
    PackageFactory packageFactory;

    @Test
    public void new_package_should_have_package_service(){

        PackageAssisted aPackage = packageFactory.create("Zenon", "Nowak", 34);

        // when
        aPackage.send();

    }
```

Inne aspekty guic-a

- Jak używać aspektów w guice

Inne aspekty Guic-a

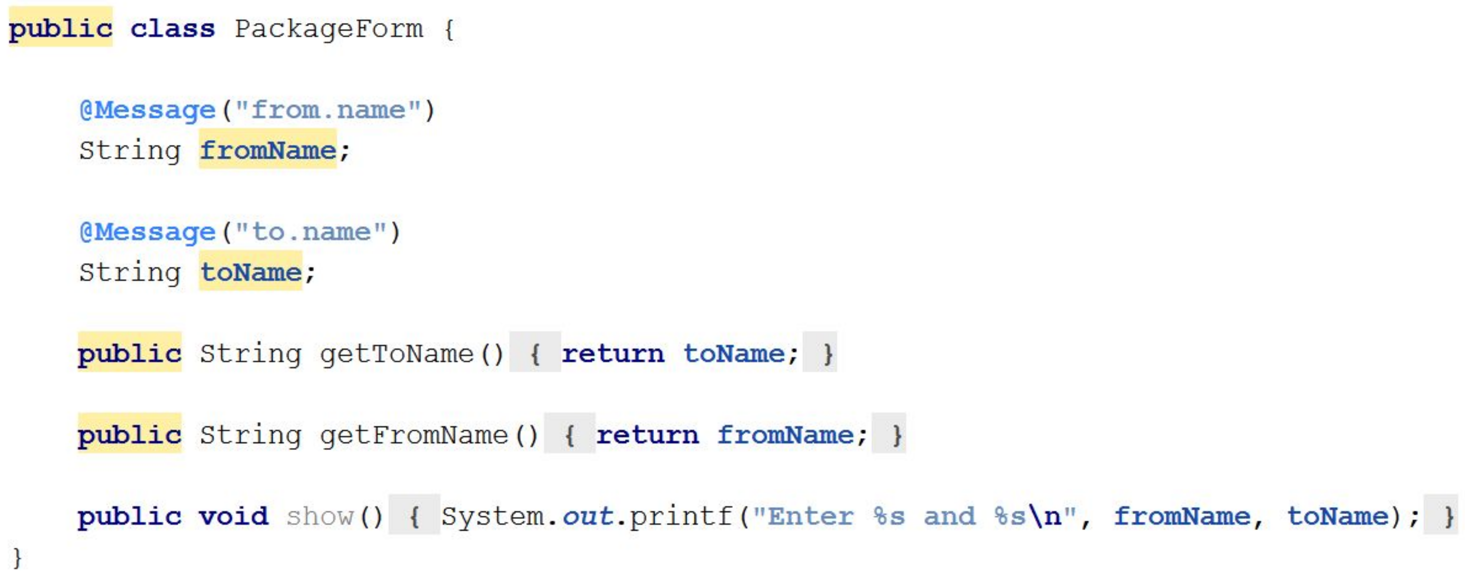
```java
public class WindowScopeModule extends AbstractModule {


    @Override
    protected void configure() {

        WindowScopeScope scope = new WindowScopeScope();

        bindInterceptor(
                Matchers.annotatedWith(WindowContext.class),
                Matchers.annotatedWith(WindowContextDispose.class),
                new ContextDisposer(scope));

    }

}
```

Inne aspekty Guic-a

```java
public class ContextDisposer implements MethodInterceptor {

    private final WindowScopeScope scope;

    public ContextDisposer(WindowScopeScope scope) { this.scope = scope; }

    @Override
    public Object invoke(MethodInvocation methodInvocation) throws Throwable {
        try {
            return methodInvocation.proceed();
        } finally {
            scope.clear(methodInvocation.getThis());
        }
    }
}
```

Adnotacje do wstrzykiwania

- Kiedy mogą pomóc
- Jak stworzyć własną



```java
public class PackageForm {

    @Message("from.name")
    String fromName;

    @Message("to.name")
    String toName;

    public String getToName() { return toName; }

    public String getFromName() { return fromName; }

    public void show() { System.out.printf("Enter %s and %s\n", fromName, toName); }
}
```

Adnotacje do wstrzykiwania

```java
@BindingAnnotation
@Target({ FIELD }) @Retention(RUNTIME)
public @interface Message {
    String value();
}


bindListener(Matchers.any(), new TypeListener()
{
    @Override
    public <I> void hear(TypeLiteral<I> type, TypeEncounter<I> encounter)
    {
        Class<?> clazz = type.getRawType();
        while (clazz != null) {
            for (Field field : clazz.getDeclaredFields()) {
                if (field.isAnnotationPresent(Message.class)) {
                    Message prop = field.getAnnotation(Message.class);
                    encounter.register(new MessageInjector<>(messageProvider, prop, field));
                }
            }
            clazz = clazz.getSuperclass();
        }
    }
});
```

Inne przydatne rozszerzenia

- Multibindings
- Servlet
- Persist
- Grapher
- JNDI
- OSGi
- Mycila - ClosableInjector, JSR-250, Services