```
W oryginalnym
    static void classesCanBeGeneric() {
        var withInteger = GenericWrapper.create(7);
Integer intValue = withInteger.getValue();
        var withString = GenericWrapper.create("Hello");
        String stringValue = withString.getValue();
Nie dziala
```

Jak najedziemy to mamy blad Required type: Integer Cast expression to 'java.lang.Integer' Alt+Shift+Enter More actions... Alt+Enter GenericWrapper withInteger = GenericWrapper.create(7) lab4

Klasa GenericWrapper operuje na Object, zamiast na typie generycznym <T>

```
lass GenericWrapper {
  Object value:
  static GenericWrapper create(Object value) {
      return new GenericWrapper(value):
  GenericWrapper(Object value) {
  Object getValue() {
```

class GenericWrapper<T> { 4 usages private T value; 2 usages private GenericWrapper(T value) { 1usage this.value = value; static <T> GenericWrapper<T> create(T value) { 2 usages return new GenericWrapper<>(value); public T getValue() { 2 usages

GenericWrapper<T> oznacza, że klasa jest generyczna.

T oznacza typ określony w czasie tworzenia obiektu. static <T> GenericWrapper<T> create(T value) {

Przed nazwa oznacza ze funckja jest generyczna, jak jest statyczna bo nie ma dostepu do T

```
static void methodsCanBeGeneric() { 1usage
        Integer anInt = 10;
        Float aFloat = 10f;
       Double aDouble = 10.0;
        Integer resultingInt = NumberInspection.inspectNumberAndReturnTheSameType(anInt);
        Float resultingFloat = NumberInspection.inspectNumb
                                                                   Required type:
                                                                                 Integer
        Double resultingDouble = NumberInspection.inspectNu
                                                                  Cast expression to 'java.lang.Integer' Alt+Shift+Enter More actions... Alt+Enter
class NumberInspection { 3 usages
                                                                  @ efs.task.generics.NumberInspection
                                                                  static Number inspectNumberAndReturnTheSameType(
    @NotNull > Number number
    static Number inspectNumberAndReturnTheSameType(Number
        System.out.println("Double value of the number of t
                                                                                                                                          ue());
                                                                  lab4
        return number;
```

```
class NumberInspection { 3 usages
static <T extends Number> T inspectNumberAndReturnTheSameType(T number) { 3 usages

System.out.println("Double value of the number of type " + number.getClass().getSimpleName() + " is: " + number.doubleValue());
return number;
}
}
```

 $static < \textbf{T} \ extends \ Number > \textbf{T} \ inspectNumber And Return The Same Type (\textbf{T} \ number) \ \{$

<T extends Number> to dodajey bo kompilator nie wie ze T przed nazwa to ta zmienna generyczna

T nuber to co zwracamy

Wyniki 7 Hello Double value of the number of type Integer is: 10.0 Double value of the number of type Float is: 10.0 Double value of the number of type Double is: 10.0 10 10.0 Process finished with exit code 0

Podsumowanie

Dzięki zastosowaniu **typów generycznych**, metody i klasy mogą działać z **dowolnymi typami**, ale nadal zachowują bezpieczeństwo typów. Nie trzeba rzytoowac bo kompilator wie jakieo typu sa obiekty, Dzieki temu kod jest bardziej uniwersalny