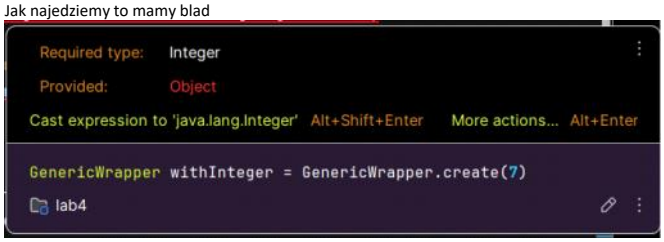
4

poniedziałek, 17 marca 2025 12:06

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

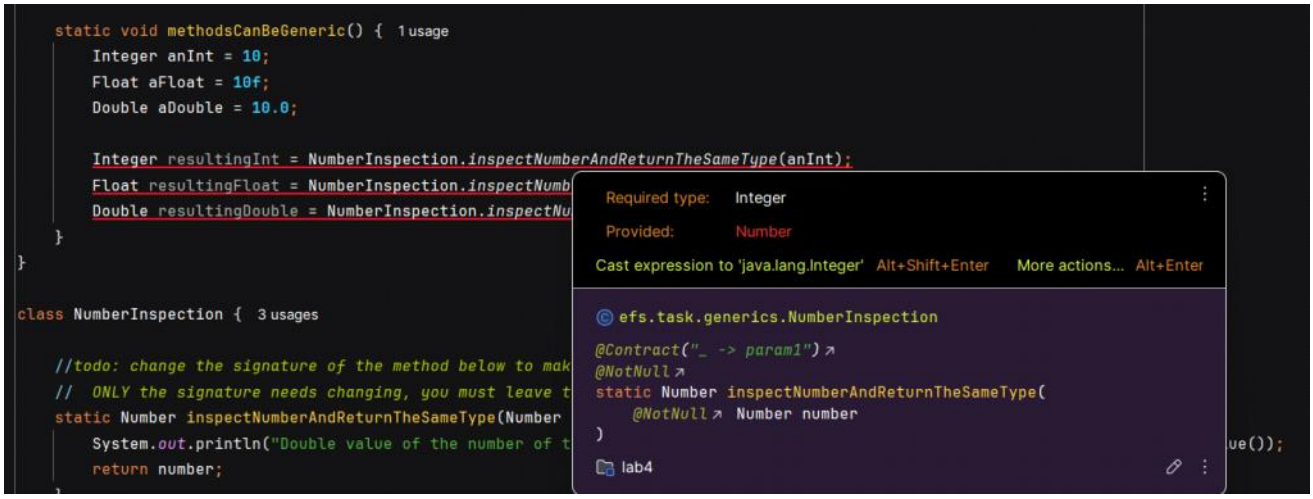


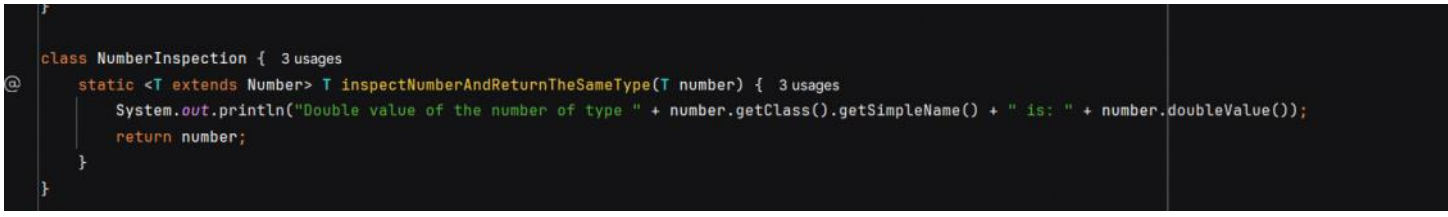
Klasa GenericWrapper operuje na Object, zamiast na typie generycznym <T>

```
class GenericWrapper {

    Object value;

    static GenericWrapper create(Object value) {
        return new GenericWrapper(value);
    }

    GenericWrapper(Object value) {
        this.value = value;
    }

    Object getValue() {
        return value;
    }
}
```

```
class GenericWrapper<T> {  4 usages
    private T value;  2 usages

    private GenericWrapper(T value) {  1 usage
        this.value = value;
    }

    static <T> GenericWrapper<T> create(T value) {  2 usages
        return new GenericWrapper<>(value);
    }

    public T getValue() {  2 usages
        return value;
    }
}
```

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 <T extends Number> T inspectNumberAndReturnTheSameType(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
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