KLASA OPTIONAL
TYP WYLICZENIOWY (ENUM)

- Klasa Optional vs null check
- Typ wyliczeniowy (Enum)

Wykład częściowo oparty na materiałach A. Jaskot

NULL CHECK I NULLPOINTEREXCEPTION

```
String input = ... // external input

System.out.println("Length: " + input.length());
```

```
Length: 3
```

```
Exception in thread "main" java.lang.NullPointerException Create breakpoint : Cannot invoke "String.length()" because "input" is null
    at Main.main(Main.java:11)
```

NULL CHECK I NULLPOINTEREXCEPTION

```
String input = ... // external input

System.out.println("Length: " + input.length());
```

```
Length: 3
```

```
Exception in thread "main" java.lang.NullPointerException Create breakpoint : Cannot invoke "String.length()" because "input" is null
    at Main.main(Main.java:11)
```

```
String input = ... // external input

if (input != null) {
    int length = input.length();
    System.out.println("Length: " + input.length());
} else {
    System.out.println("Input is null");
}
```

KLASA OPTIONAL<T>

- opakowuje wartość, która może (ale nie zawsze będzie) pusta
- ma za zadanie reprezentować brak wartości (pusty obiekt) w sytuacjach, w których wykorzystanie nulla może prowadzić do błędów, zwłaszcza NullPointerException

```
String input = ... // external input

Optional<String> emptyOptional
    = Optional.empty();

Optional<String> optionalInput
    = Optional.of(input);

Optional<String> optionalInput2
    = Optional.ofNullable(input);
```

- metoda *empty()* tworzy pusty Optional
- metoda *of()* tworzy Optional z podanej jako parameter wartości; rzuca NullPointerException, jeżeli ta wartość jest nullem
- metoda *ofNullable()* działa podobnie do *of()*, ale tworzy pusty Optional, jeżeli podana wartość jest nullem

KLASA OPTIONAL<T>

```
String input = ... // external input
Optional<String> optionalInput
    = Optional.ofNullable(input);

if (optionalInput.isPresent()) {
    // some code dependent on value present
}

if (optionalInput.isEmpty()) {
    // some code dependent on value empty
}
```

- metody *isPresent()* i *isEmpty()* sprawdzają, czy Optional posiada wartość / czy jest pusty

```
String input = ... // external input
Optional<String> optionalInput
    = Optional.ofNullable(input);

if (optionalInput.isPresent()) {
    System.out.println(optionalInput.get());
}
```

- metoda *get()* zwraca wartość z obiektu Optional; rzuca NoSuchElementException, jeżeli Optional jest pusty

KLASA OPTIONAL<T>

```
String input = "zzz";
Optional<String> optionalInput
    = Optional.ofNullable(input);

System.out.println(optionalInput.orElse("xxx"));
```

```
zzz
```

```
String input = null;
Optional<String> optionalInput
    = Optional.ofNullable(input);

System.out.println(optionalInput.orElse("xxx"));
```

```
xxx
```

- metoda *orElse()* zwraca wartość z obiektu Optional lub wartość domyślną, jeżeli Optional jest pusty

KLASA OPTIONAL<T>

```
String input = ... // external input
Optional<String> optionalInput
    = Optional.ofNullable(input);

System.out.println(optionalInput
    .orElseGet(() -> generateRandomString()));
```

- metoda *orElseGet()* działa podobnie do *orElse()*, ale zamiast wartości domyślnej przyjmuje interfejs funkcyjny Supplier

```
String input = ... // external input
Optional<String> optionalInput
    = Optional.ofNullable(input);

System.out.println(optionalInput
    .orElseThrow(IllegalArgumentException::new));
```

- metoda orElseThrow() z kolei przyjmuje interfejs funkcyjny Supplier generujący wyjątek

KLASA OPTIONAL A STRUMIENIE

- metody *findFirst()*, *findAny()*, *min()*, *max()* i *reduce()*, wykorzystywane w operacjach kończących strumień, zwracają obiekty typu Optional

```
List<String> words
    = Arrays.asList("XXX", "YYY", "ZZZ", null, "AAA");

Optional<String> result = words.stream()
    .filter(name -> name.startsWith("C"))
    .findFirst();

result.ifPresentOrElse(
    value -> System.out.println
        ("Word starting with 'C': " + value),
    () -> System.out.println("No matching name found.")
);
```

KLASA OPTIONAL A STRUMIENIE

- metody klasy Optional mogą być używane podczas operacji na strumieniach do wykluczenia lub znalezienia pustych wartości

```
List<Optional<String>> optionalValues = Arrays.asList(
    Optional.of("XXX"),
    Optional.empty(),
    Optional.of("YYY"),
    Optional.of("ZZZ"),
    Optional.empty()
);

List<String> nonEmptyValues = optionalValues
    .stream()
    .filter(Optional::isPresent)
    .map(Optional::get)
    .collect(Collectors.toList());

System.out.println
    ("Non-empty values: " + nonEmptyValues);
```

TYP WYLICZENIOWY (ENUM)

- szczególny typ klasy deklarowany za pomocą słowa kluczowego *enum*
- służy do określania predefiniowanych zestawów wartości stałych
- kompilowany do klasy finalnej rozszerzającej klasę Enum

```
enum Level {
    LOW,
    MEDIUM,
    HIGH
}
```

```
enum Status {
    INPROGRESS,
    SUBMITTED,
    ACCEPTED
}
```

- wartości w typie enum wymienia się po przecinku wielkimi literami

TYP WYLICZENIOWY (ENUM)

```
public class Assignment {
    private String author;
    private String status;
    private int grade;

    public Assignment(String author,
        String status, int grade) {
        this.author = author;
        this.status = status;
        this.grade = grade;
    }
}
```

```
Assignment a1
    = new Assignment("XXX", "inprogress", 0);
Assignment a2
    = new Assignment("YYY", "submitted", 0);
Assignment a3
    = new Assignment("ZZZ", "SUBMITTED", 0);
Assignment a4
    = new Assignment("AAA", "sent", 0);
```

TYP WYLICZENIOWY (ENUM)

```
public class Assignment {
    private String author;
    private String status;
    private int grade;

    public Assignment(String author,
        String status, int grade) {
        this.author = author;
        this.status = status;
        this.grade = grade;
    }
}
```

```
public class AssigmnentStatus {

    public static final String INPROGRESS = "inProgress";
    public static final String SUBMITTED = "submitted";
    public static final String ACCEPTED = "accepted";

}
```

```
Assignment a1 = new Assignment
   ("XXX", AssignmentStatus.INPROGRESS, 0);
Assignment a2 = new Assignment
   ("YYY", AssignmentStatus.SUBMITTED, 0);
Assignment a3 = new Assignment
   ("ZZZ", AssignmentStatus.SUBMITTED, 0);

Assignment a4 = new Assignment
   ("AAA", "sent", 0);
```

TYP WYLICZENIOWY (ENUM)

```
public class Assignment {
    private String author;
    private AssigmnentStatus status;
    private int grade;

    public Assignment(String author,
        AssigmnentStatus status, int grade) {
        this.author = author;
        this.status = status;
        this.grade = grade;
    }
}
```

```
public class AssigmnentStatus {
    INPROGRESS,
    SUBMITTED,
    ACCEPTED
}
```

```
Assignment a1 = new Assignment
   ("XXX", AssignmentStatus.INPROGRESS, 0);
Assignment a2 = new Assignment
   ("YYY", AssignmentStatus.SUBMITTED, 0);
Assignment a3 = new Assignment
   ("ZZZ", AssignmentStatus.SUBMITTED, 0);

Assignment a4 = new Assignment
   ("AAA", "sent", 0);
```

TYP WYLICZENIOWY (ENUM)

```
if (a1.getStatus() == AssignmentStatus.ACCEPTED) {
    System.out.println("Assignment complete");
}
```

- wartości typu enum można porównywać za pomoca operatora ==

```
switch (a1.getStatus()) {
    case INPROGRESS:
        System.out.println("Assignment overdue");
        break;
    case SUBMITTED:
        System.out.println("Grading in progress");
        break;
    case ACCEPTED:
        System.out.println("Assignment complete");
        break;
}
```

- wartości typu enum można używać w instrukcji warunkowej switch

TYP WYLICZENIOWY (ENUM)

```
System.out.println("Index: " + a1.getStatus().ordinal());
System.out.println("Name: " + a1.getStatus().name());
```

```
Index: 0
Name: INPROGRESS
```

- metoda *ordinal()* zwraca indeks danej wartości (kolejność wartości ma znaczenie)
- metoda *name()* zwraca nazwę danej wartości

```
AssignmentStatus[] values = AssignmentStatus.values();
for (int i = 0; i < values.length; i++) {
          System.out.println(values[i]);
}
```

```
INPROGRESS
SUBMITTED
ACCEPTED
```

- metoda *values()* zwraca wszystkie dostępne wartości w postaci tablicy

```
AssignmentStatus status =
    AssignmentStatus.valueOf("SUBMITTED");
```

- metoda *valueOf()* konwertuje stringa na odpowiadającą mu wartość

TYP WYLICZENIOWY (ENUM)

```
enum DeliveryMethod {
    STANDARD(5), EXPEDITED(2), NEXT_DAY(1);

    private final int deliveryTime;

    DeliveryMethod(int deliveryTime) {
        this.deliveryTime = deliveryTime;
    }

    public int getDeliveryTime() {
        return deliveryTime;
    }
}
```

- enum może posiadać pola (najlepiej finalne)
- enum może posiadać dodatkowe konstruktory, ale tylko prywatne
- enum może posiadać metody, zarówno statyczne, jak instancji

```
enum BloodType {
    A_POSITIVE, A_NEGATIVE, B_POSITIVE, B_NEGATIVE,
    AB_POSITIVE, AB_NEGATIVE, O_POSITIVE, O_NEGATIVE;

    public boolean isRhPositive() {
        return this == A_POSITIVE || this == B_POSITIVE || this == AB_POSITIVE || this == O_POSITIVE;
    }
}
```