Zadanie 1.

Utworzyć nowy projekt i dodać zależności JUnit.
Utworzyć pakiet **util**. W pakiecie tym należy utworzyć klasę **ObjectsUtil** oraz zaimplementować metody o następujących sygnaturach:

```
public static boolean issOddNumber(int number)
public static boolean isValidPercent(double percent)
public static boolean containsDigits(String string)
public static boolean extendsWithNumberClass(Class<?> clazz)
```

Metoda **isOddNumber** weryfikuje, czy przekazana jako argument wywołania liczba jest **liczbą nieparzystą**.

Metoda **isValidPercent** weryfikuje, czy przekazana jako argument wywołania **liczba jest większa lub równa 0 i jednocześnie mniejsza lub równa 100**.

Metoda **containsDigits** weryfikuje, czy przekazany jako argument wywołania łańcuch znaków zawiera cyfry.

Metoda **extendsWithNumberClass** weryfikuje, czy klasa przekazana jako argument wywołania rozszerza klasę **java.lang.Number**. Podpowiedź. Skorzystać z metody **isAssignableFrom** zdefiniowanej w klasie **java.lang.Class**.

Kolejno, należy utworzyć klasę testową (w tym samym pakiecie co klasa **ObjectsUtil**) o nazwie **ObjectsUtilTest** (**Alt + Insert**, **Test…**, **OK**) Należy zaimplementować metody testowe weryfikujące poprawność zaimplementowanych metod w klasie **ObjectsUtil**.

a) metoda: **void shouldReturnTrueForOddNumber(int number)**
parametryzowana wartościami: **1, 3, 5, -3, 15, Integer.MAX_VALUE**
b) metoda: **void shouldReturnTrueForValidPercent(double value)**
parametryzowana wartościami: **0, 0.0, 0.00, 54, 98.76, 100, 100.**
c) metoda: **void shouldReturnTrueForStringContainingDigits(String value)**
parametryzowana wartościami: **"raz1", "123", "0", "bank999"**
d) metoda: **void shouldReturnFalseForStringNotContainingDigits(String value)**
parametryzowana wartościami: **"", "one", "raz", "test", "sampleValue"**
e) metoda: **void shouldReturnTrueForClassExtendingNumber(Class<?> clazz)**
parametryzowana wartościami: **Integer.class, Long.class, Float.class, Double.class**

W tym celu należy wykorzystać adnotację **@ParametrizedTest** oraz odpowiednio sparametryzowaną adnotację **@ValueSource**.

Zadanie 2.

A)

Utworzyć pakiet **service**. W pakiecie, tym należy utworzyć interfejs funkcyjny:

```
@FunctionalInterface
public interface SecretCodeGenerator {

    String generate(Integer codeLength);
}
```

Utworzyć pakiet **service.enumeration**. W pakiecie tym należy utworzyć **enum implementujące dany interfejs** w postaci:

```
import lombok.RequiredArgsConstructor;
import org.apache.commons.lang3.RandomStringUtils;
import service.SecretCodeGenerator;

@RequiredArgsConstructor
public enum SecretCodeAlphabetEnum implements SecretCodeGenerator {

    DIGITS("0123456789"),
    ALPHABET("abcdefghijkmnpqrstuvwxyzABCDEFGHJKLMNPQRSTUVWXYZ"),
    ALPHABET_LOWERCASE("abcdefghijkmnpqrstuvwxyz"),
    ALPHABET_UPPERCASE("ABCDEFGHJKLMNPQRSTUVWXYZ"),

ALPHABET_DIGITS("abcdefghijkmnpqrstuvwxyzABCDEFGHIJKLMNPQRSTUVWXYZ23456789"),
    ALPHABET_LOWERCASE_DIGITS("abcdefghijkmnpqrstuvwxyz23456789"),
    ALPHABET_UPPERCASE_DIGITS("ABCDEFGHJKLMNPQRSTUVWXYZ23456789");

    private final String domain;

    @Override
    public String generate(Integer codeLength) {
        return RandomStringUtils.random(codeLength, domain);
    }
}
```

Wykorzystując adnotację **@EnumSource** utworzyć metodę testową:
void shouldGenerateCodeOfSpecifiedLength(SecretCodeAlphabetEnum secretCodeAlphabetEnum)

Metoda ta powinna weryfikować (dla każdej z wartości **enum**) prawidłowość wygenerowania sekretnego kodu o zakładanej (np. 10) liczbie znaków.

B)

Kolejno, w pakiecie **util** utworzyć klasę **CalendarUtil** i zaimplementować metody o następujących sygnaturach:

```
public static boolean monthIsBetweenOneAndTwelve(Month month)
public static boolean monthIs30DaysLong(Month month)
public static boolean monthIs31DaysLong(Month month)
public static boolean monthNameEndsWithBer(Month month)
```

Wykorzystując odpowiednio parametryzowaną adnotację ***@EnumSource*** napisać metody testowe.

Metoda **void shouldReturnTrueForAllMonths(Month month)**
uruchamiana dla wszystkich wartości **enum Month**.

Metoda **void shouldReturnTrueFor30DaysLongMonths(Month month)**
uruchamiana dla wartości **"APRIL", "JUNE", "SEPTEMBER", "NOVEMBER"**.

Metoda **void shouldReturnTrueFor31DaysLongMonths(Month month)**
uruchamiana dla wszystkich wartości **enum za wyjątkiem**:
"FEBRUARY", "APRIL", "JUNE", "SEPTEMBER", "NOVEMBER"

Metoda **void shouldReturnTrueForMonthsEndingWithBer(Month month)**
uruchamiana dla wartości kończących się frazą: **„BER”**.
Podpowiedź. Wykorzystać tryb: **EnumSource.Mode.MATCH_ANY** oraz frazę: „**.*BER**”.

Zadanie 3.
W klasie **ObjectsUtil** zaimplementować metody o następujących sygnaturach

```
public static String toUpperCase(String value)
public static Integer square(int number)
```

Wykorzystując odpowiednio parametryzowaną adnotację **@CsvSource** utworzyć metody testowe.

Metoda **void shouldGenerateValidUpperCaseValue(String value, String expectedValue)**
uruchamiana dla wartości: **"raz,RAZ", "dWa,DWA", "TRZY,TRZY", ","**

Metoda **void shouldSquarePassedNumber(int value, int expectedValue)**
uruchamiana dla wartości: **"1:1", "-2:4", "4:16", "0:0"**

Zadanie 4.
Należy utworzyć katalog **resources** w katalogu **test**.
W katalogu tym, należy utworzyć plik o nazwie **temperatures.csv**.

Postać pliku:

```
kelvin,celsius
0,-273
273,0
100,-173
```

Utworzyć klasę **TemperatureUtil** (w pakiecie **util***)* oraz zaimplementować metodę o sygnaturze:

```
public static int celsiusToKelvin(int celsiusTemperature)
```

Podpowiedź. **T(Celsjusz) = T(Kelwin) + 273**.
Utworzyć klasę testową o nazwie **TemperatureUtilTest**.

Wykorzystując odpowiednio parametryzowaną adnotację **@CsvFileSource** utworzyć metodę testową.

Metoda **void shouldConvertCelsiusTemperatureToKelvinTemperature(int expectedKelvinTemperature, int celsiusTemperature)**
uruchamiana dla pliku o nazwie **temperatures.csv**. Proszę zwrócić uwagę na **separator wartości** oraz **konieczność pominięcia pierwszej linii**.

Zadanie 5.

A)

W klasie **ObjectsUtil** zaimplementować metodę:

```
public static boolean isEvenNumber(int number)
```

W klasie testowej **ObjectsUtilTest** zaimplementować metodę statyczną o sygnaturze:

```
private static Stream<Integer> provideEvenNumbers()
```

zwracającą **skończony strumień liczb parzystych**. W tym celu można wykorzystać metody takie jak: **Stream.of, Stream.iterate, Stream.generate** lub utworzyć strumień na podstawie odpowiednio **zainicjalizowanej kolekcji (np. listy)**.

Wykorzystując odpowiednio parametryzowaną adnotację **@MethodSource** utworzyć metodę testową: **void shouldReturnTrueForEvenNumbers(Integer number)**.

B)

Utworzyć w pakiecie **model** klasę **Rectangle**:

```
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
public class Rectangle {

    private Integer width;

    private Integer height;

    public Integer getArea() {
        if (width == null || height == null) {
            return null;
        }

        return width * height;
    }
}
```

W odpowiednim pakiecie, należy utworzyć klasę testową **RectangleTest**, a następnie metodę testową: **void shouldCalculateAreaOfRectangle(Rectangle rectangle, Integer expectedArea)** parametryzowaną odpowiednią **metodą zwracającą strumień argumentów**.

Metoda testowa powinna być wywołana dla następujących obiektów typu **Rectangle**:

```
new Rectangle()
new Rectangle(1, 1)
new Rectangle(5, 4)
new Rectangle(5, 0)
new Rectangle(null, 4)
new Rectangle(4, null)
```

C)

Utworzyć pakiet **util.params** (**w katalogu testów**).
W pakiecie tym zdefiniować klasę: **DaysInfoParamProviders**, a w niej metodę **statyczną** zwracającą strumień dni tygodnia z informacją, czy jest to dzień roboczy

Przykładowa implementacja:

```
public static Stream<Arguments> provideDaysInfo() {
    return Stream.of(
            Arguments.of(DayOfWeek.MONDAY, true),
            Arguments.of(DayOfWeek.FRIDAY, true),
            Arguments.of(DayOfWeek.SUNDAY, false),
            Arguments.of(null, false));
}
```

W klasie **CalendarUtil** zaimplementować metodę:

```
public static boolean isWeekDay(DayOfWeek dayOfWeek)
```

Następnie, odpowiednio parametryzując adnotację **@MethodSource** (**klasa#metoda**) zaimplementować metodę testową wykorzystującą strumień argumentów zdefiniowany w oparciu o metodę **provideDaysInfo()**.

Metoda testowa: **void shouldInformAboutWeekDays(DayOfWeek dayOfWeek, boolean isWeekDay)**

Zadanie 6.
Utworzyć klasę **NumberWithParityArgumentsProvider** implementującą interfejs **ArgumentsProvider**.
Zaimplementować metodę:

```
public Stream<? extends Arguments> provideArguments(ExtensionContext context)
```

zwracającą **strumień argumentów, z informacją o konkretnej liczbie i jej parzystości**.

Wykorzystując odpowiednio parametryzowaną adnotację **@ArgumentsSource**
utworzyć w klasie testowej **ObejctsUtilTest** metodę testową.
Metoda testowa: **void shouldValidateParityOfNumber(int number, boolean isOddNumber)**.

Zadanie 7.
W klasie **ObjectUtil** zaimplementować metodę:

```
public static boolean isNullOrBlankString(String value)
```

Następnie, wykorzystując adnotacje:
@NullSource, **@EmptySource**, **@NullAndEmptySource** napisać odpowiednie metody testowe do danej metody.

Metody:
void shouldReturnTrueForNullInput(String value)
void shouldReturnTrueForEmptyString(String value)
void shouldReturnTrueForNullAndEmptySource(String value)

Zadanie 8*.
W pakiecie **model** utworzyć klasę **Person**.

```
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class Person {

    private String firstName;

    private String middleName;

    private String lastName;

    public String fullName() {
        if (StringUtils.isBlank(middleName)) {
            return String.format("%s %s", firstName, lastName);
        }

        return String.format("%s %s %s", firstName, middleName, lastName);
    }
}
```

Utworzyć klasę testową **PersonTest**.
Utworzyć dwie metody testowe (jedną działającą bezpośrednio na obiekcie **ArgumentAccessor**, drugą działającą w oparciu o adnotację **AggregateWith**) testujące metodę **fullName()** zdefiniowaną w klasie **Person**.

Zadanie 9.

A)

Napisać w klasie **ObjectsUtilTest** metodę testową:
void shouldReturnTrueForNullEmptyAndBlankSource(String value)

parametryzując ją adnotacją **@NullAndEmptySource** oraz przekazując dodatkowo wartości:
" ", "\t ", "\n\n\t ".

B)

Napisać w klasie **TemperatureUtilTest** metodę testową:
void shouldConvertCelsiusTemperatureToKelvinTemperatureCompoundCase
która będzie parametryzowana danymi zawartymi w pliku **temperatures.csv**
oraz wartościami: **"300:27", "73:-200"**.

Zadanie 10.

W klasie **ObjectsUtil** zaimplementować metody:

```
public static Integer divide(Integer a, Integer b) {
    return a / b;
}

public static double sqrt(double value) {
    if (value < 0) {
        throw new IllegalArgumentException("Cannot calculate root of negative
value!");
    }

    return Math.sqrt(value);
}
```

Zaimplementować metody testowe:
public void shouldThrowArithmeticExceptionWhenDivideByZero
public void shouldThrowIllegalArgumentExceptionWhenCalculateSqrtOfNegativeNumber()

wykorzystując odpowiednie konstrukcje służące obsłudze wyjątków, zamieszczone na prezentacji: **assertThrows()** (grupa metod przeciążonych), **assertThatExceptionOfType()**, **assertThatThrownBy()**, **assertThatIllegalArgumentException()**.