# 1. Projekt oks-02

Základní informace:

- **Účel:** příprava základních jednotkových testů
- **Kostra:** `oks-data-02.zip`
- **Odevzdávaný soubor:** `oks-02.jar`

Zadání:

- nad poskytnutou entitní třídou `OsobniCislo.java`, do které jsou uměle zaneseny chyby, napište JUnit testy, které prokazatelně tyto chyby odhalí

Popis vstupních dat:

- stáhněte si soubor `oks-data-02.zip` a rozbalte jej
  - všechny soubory jsou v kódování UTF-8 bez BOM
- v adresáři `src/oks02` se nachází zdrojové `.java` soubory
  - adresář obsahuje následující soubory:
    - `OsobniCislo.java` - třída, nad kterou budete psát JUnit testy
      - tento soubor budete importovat do IntelliJ
      - je trochu jiný, než byl v projektu OKS-01
    - `Konstanty.java` - třída konstant
      - tento soubor budete importovat do IntelliJ, ale nebudete jej ani testovat, ani měnit
    - `TypStudia.java` - výčtový typ
      - tento soubor budete importovat do IntelliJ, ale nebudete jej ani testovat, ani měnit
    - `Hlavni.java` - triviální hlavní třída, která ukazuje (nějakou) funkčnost třídy `OsobniCislo.java`
- v adresáři `lib` se nacházejí JAR knihovny JUnit 5
  - použijete tři z nich v případě, že IntelliJ nepřipojí JUnit 5 automaticky (viz dále tento text a případně `oks-04.pdf` str.6 a str.11)
- v adresáři `kontrola` se nachází kontrolní program a příslušné knihovny
  - adresář obsahuje následující adresáře a soubory:
    - `oks02` - adresář s `.class` soubory testovaných tříd a soubory `Kontrola_Prj_02.class` a `Kontrola_Prj_02$Cislo.class`, které slouží pro kontrolu splnění zadání
    - `kontrola.bat` - překlad a spuštění kontrolního programu

Postup řešení:

- v IntelliJ založte nový projekt `oks-prj-02`
  - nastavte celému projektu kódování UTF-8

    **File / Settings / Editor / File encodings / Project encoding: UTF-8**
  - v adresáři `src` založte balík `oks02`
  - přetáhněte soubory `src/oks02/OsobniCislo.java`, `src/oks02/Konstanty.java` a `src/oks02/TypStudia.java` do adresáře balíku `oks02`
  - přetáhněte soubor `src/oks02/Hlavni.java` do adresáře balíku `oks02` a kontrolně jej spusťte
    - jeho výpis je pro vás jen informativní
- v IntelliJ otevřete třídu `OsobniCislo` a pozorně si ji prohlédněte
  - kontrakt popsaný v JavaDoc dokumentaci v této třídě a též v příloze tohoto zadání (viz níže) je závazná specifikace, jak by se třída a její jednotlivé metody měly chovat
  - pro zvýšení testovatelnosti mají všechny atributy přístupové právo `public` a budete je v JUnit testech přímo využívat

    ### Note

    Dávat atributům přístupové právo `public` pro lepší testovatelnost je v reálném případě většinou zbytečné. Tam se totiž můžeme spolehnout na správnou funkci getrů a setrů, které bychom využili místo přímého přístupu k atributům.
    - stejně tak jsou i všechny metody `public`
  - `toString()` je inspirována studentskými programy tak, “aby to na první pohled nějak fungovalo” - tuto metodu nebudete testovat
    - stejně tak nebudete testovat metodu `private void naplnAtributy()`, protože ve svém těle volá několik dalších metod
  - všechny zbývající metody obsahují jednu **nebo více** chyb, které musíte JUnit testy prokázat
    - některé chyby jsou vidět na první pohled, jiné jsou skryté

      ### Warning

      V žádném případě neměňte cokoliv ve třídě `OsobniCislo`. To není váš úkol! Váš úkol je pomocí testů chyby odhalit a prokázat.
  - chyby jsou i v metodách `getXY()`, což není běžné (normálně se getry a setry netestují)
    - protože se na jejich bezchybnou funkčnost nemůžete spolehnout, musíte před testováním getru nastavit přímým přístupem příslušný atribut, např.

```
private OsobniCislo oc;

@Test
void getFakulta() {
  oc.fakulta = "A";
```

```
    assertEquals("A", oc.getFakulta());
  }
```

### Note

Tentýž postup použijete i pro testy `void` metod, které nastavují jednotlivé atributy a nejsou k nim getry, např. `zpracujNepovinne()` .

- pokud budete potřebovat otestovat některou metodu více testy, rozlište tyto testovací metody pořadovými čísly (včetně podtržítka) na konci jejich názvu, např.:

```
@Test
void zpracujPrijmeni_1() {
  ...
@Test
void zpracujPrijmeni_2() {
  ...
```

- všechny metody `zpracujXY()` je třeba otestovat jak se správným skutečným parametrem, tak i s hodnotou `null`, např.:

```
@Test
void zpracujPrijmeni_1() {
  oc.zpracujPrijmeni("Novák");
  assertEquals(...
}

@Test
void zpracujPrijmeni_2() {
  oc.zpracujPrijmeni(null);
  assertEquals(...
}
```

- nezapomeňte na testování výjimky u `zpracujRokNastupu()`
- aby testy byly co nejjednodušší, je výhodné využít pro počáteční inicializaci (vytvoření instance `OsobniCislo`) metodu:

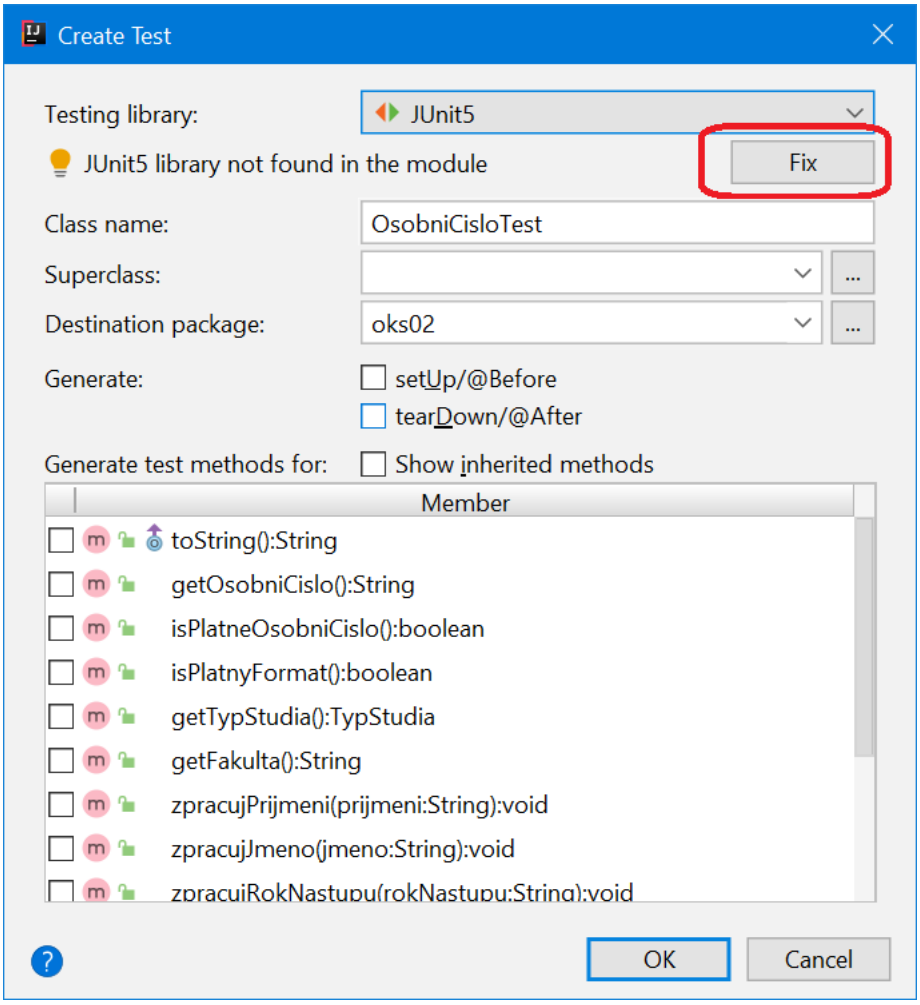
```
@BeforeEach
public void setUp() {
```

- testy mají typicky dvě řádky příkazů - viz výše příklad
  - ♦ někdy mají pouze jeden příkaz, někdy až tři
  - ♦ v testech nejsou nikde příkazy `if`, `for`, `switch` apod.

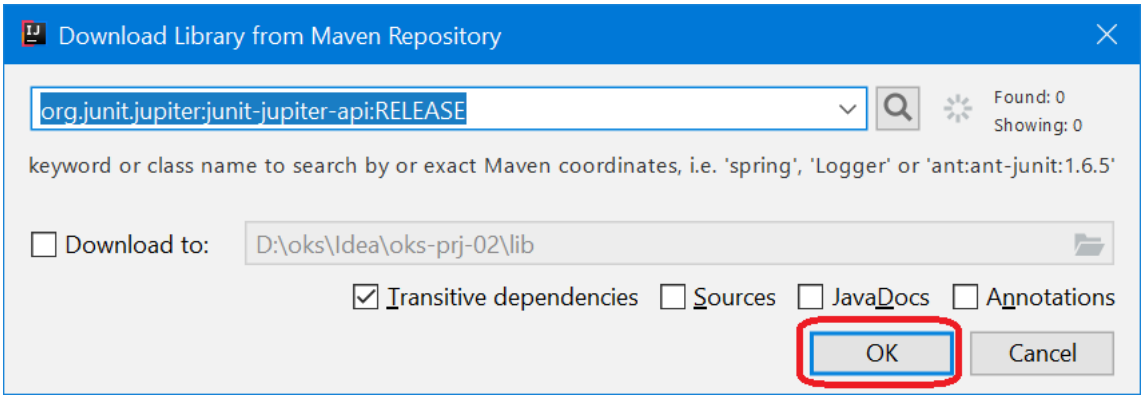
■ v IntelliJ označte projekt `oks-prj-02` a příkazem **File / New / Directory** vytvořte adresář `test`

■ v IntelliJ umístěte kurzor kamkoliv do třídy `OsobniCislo` a z kontextové nabídky použijte **Generate... / Test...**

- zvolte Junit 5 a případně nechte připojit knihovnu

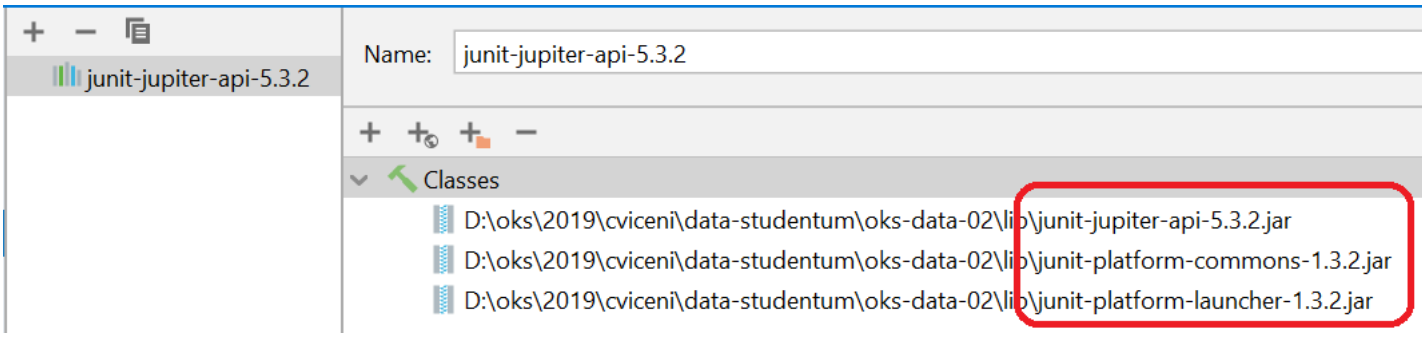



a následně

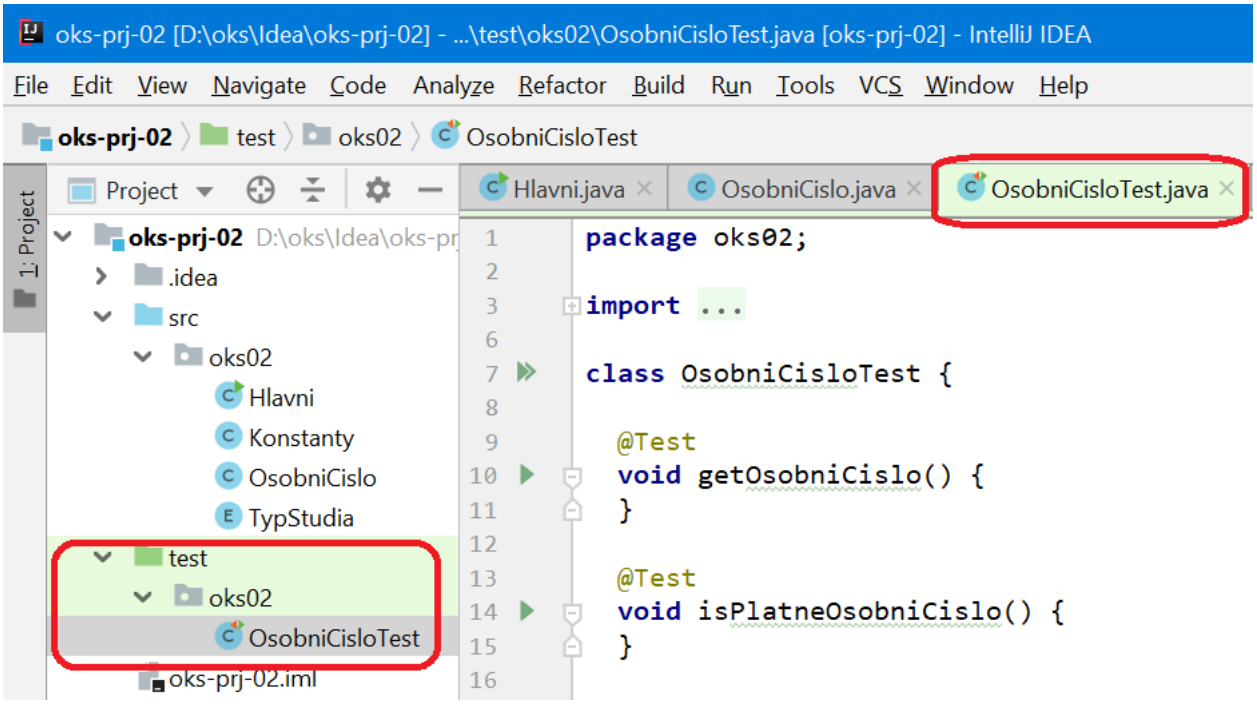



## Note

Pokud není tato možnost nabídnuta nebo selže, připojte tři knihovny JUnit 5 z adresáře `lib` přímo (viz `oks-04.pdf` str.6)




- nechte si vygenerovat kostry jednotlivých testů pro všechny metody, kromě `toString()`
- měl by vzniknout `test/oks02/OsobniCisloTest`




- podle pokynů výše pište těla jednotlivých testů a ihned je zkoušejte spustit - stačí kliknout na zelenou šipku vlevo u testované metody
  - můžete napsat kolik chcete testů, které neselžou - ty se nepočítají
  - počítají se pouze testy, které selžou, tj. odhalí standardním způsobem pomocí aserce chybu

Kontrola úplnosti řešení:

- adresář `test` z IntelliJ překopírujte někam do svého pomocného adresáře, např. `D:\zzz`
  - dále sem překopírujte obsah adresářů `kontrola` a `lib`
- spusťte soubor `kontrola.bat` nebo z něj použijte příslušné příkazy
  - Pozor - pokud pracujete v Linuxu, bude zřejmě nutné v nastavení *classpath* změnit oddělovač souborů z ;

```
-cp apiguardian-api-1.0.0.jar;junit-jupiter-api-5.3.2.jar ...
```

na :

```
-cp apiguardian-api-1.0.0.jar:junit-jupiter-api-5.3.2.jar ...
```

■ uvidíte pravděpodobně výpis typu:

```
Pocet spustenych testu: 20<br />
Pocet ignorovanych testu: 0<br />
Pocet testu, ktere selhaly: 18<br />
Vypis analyzy testu: <br />
Testovaci metoda: getOsobniCislo_4() je pravdepodobne neucinna<br />
Testovana metoda: isPlatneOsobniCislo() neni pravdepodobne zcela pokryta ▶
testem. Pridejte minimalne jeste 1 test(y).<br />
Testovana metoda: getOsobniCislo() neni pravdepodobne zcela pokryta testem. ▶
Pridejte minimalne jeste 1 test(y).<br />
```

■ z analýzy vyplývá:

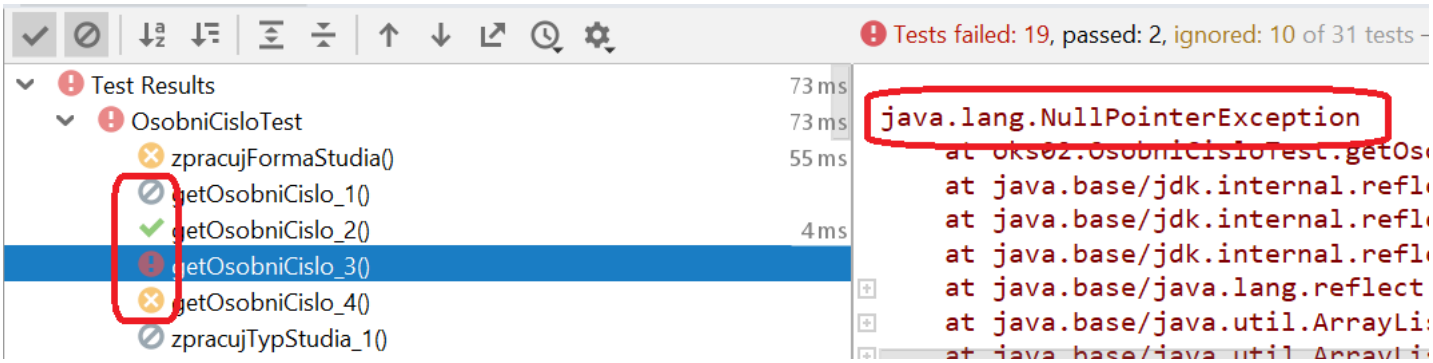
- `Testovaci metoda: getOsobniCislo_4() je pravdepodobne neucinna`

  metoda `getOsobniCislo_4()` sice selhala, ale k selhání nedošlo tím, že by se testovala metoda `OsobniCislo.getOsobniCislo()`, ale jiným způsobem, např. že byl v těle testu příkaz `fail()`

- `Testovana metoda: isPlatneOsobniCislo() neni pravdepodobne zcela pokryta testem. Přidejte minimálně ještě 1 test(y).`

  metoda `isPlatneOsobniCislo()` vyžaduje pokrytí více než jedním testem - je třeba přidat minimálně ještě jeden test

- může se vám při spouštění z IntelliJ zdát, že je vše v pořádku, protože testy přece selhávají; je ale rozdíl mezi:
  - *Failure* - kdy test selže, protože selže metoda `assertXY()` - to je správné (očekávané) chování
  - *Error* - kdy test selže, protože je v něm nějaká programová chyba, která vyhodí výjimku (nejčastěji `NullPointerException`)
    - typický případ je, kdy v těle testu voláme nějakou instanční metodu, ale v metodě `setUp()` - nebo někde jinde - nebyla instance vytvořena




■ testy přidáváte a měníte tak dlouho, dokud budou ve `Vypis analyzy testu:` hlášeny nějaké problémy

Příprava souborů k odevzdání:

■ v adresáři, ve kterém byla prováděna kontrola, použijte příkaz:

```
jar cMf oks-02.jar test/oks02/OsobniCisloTest.java
```

- výsledkem bude soubor `oks-02.jar`, který budete odevzdávat
  - opravdu není spustitelný a obsahuje pouze soubor `OsobniCisloTest.java`
  - jeho velikost by měla být do 2 KB - pokud je větší, je v `.jar` něco navíc
- po úspěšném odevzdání na Portále, proklikněte výsledné OK a, prosím, vyplňte v tabulce časovou náročnost této úlohy

# 1.1. Příloha - specifikace osobního čísla

**Note**

Ve skutečnosti má konstrukce osobního čísla více možností, než je dále uváděno. Ale tyto možnosti jsou buď zastaralé nebo jsou využívány minoritně a my je nebudeme brát v úvahu.

## 1.1.1. Povolené znaky

- jako znaky jsou povoleny jen
  - dekadické číslice 0, 1, 2, ..., 9
  - vybraná velká písmena anglické abecedy - viz dále části osobního čísla

## 1.1.2. Části osobního čísla

- osobní číslo na ZČU se sestává z pěti (šesti) částí - `FrrTppppfN`, např. `A12B0987P`, kde:

### 1.1.2.1. `F` = Fakulta

- jedno písmeno, které určuje fakultu
- povoleny jsou znaky:
  - `A` = FAV - Fakulta aplikovaných věd
  - `E` = FEL - Fakulta elektrotechnická
  - `S` = FST - Fakulta strojní
  - `P` = FPE - Fakulta pedagogická
  - `K` = FEK - Fakulta ekonomická
  - `F` = FF - Fakulta filosofická
  - `R` = FPR - Fakulta právnická
  - `Z` = FZS - Fakulta zdravotnických studií
  - `D` = FDU - Fakulta designu a umění Ladislava Sutnara

### 1.1.2.2. `rr` = rok nástupu na studia

- poslední dvojčíslí roku nástupu na studia

- vždy dvě číslice včetně případné první nuly

- hodnota 00 je povolena a udává rok 2000

### 1.1.2.3. `T` = typ studia

- jedno písmeno, které určuje typ studia

- povoleny jsou znaky

  - `B` = bakalářský

  - `N` = navazující

  - `P` = doktorský

  - `M` = magisterský (jen některé fakulty, např. FPR)

### 1.1.2.4. `pppp` = pořadové číslo

- pořadové číslo studenta

  - studenti jsou v prvním kole přijímacího řízení řazeni podle abecedy, takže pořadové číslo většinou odpovídá i pořadí jejich příjmení

    - může se stát, že student, který nastoupil na fakultu později, má vyšší pořadové číslo, než by odpovídalo abecednímu pořadí jeho příjmení a jména

    - nelze tedy činit žádné závěry ohledně příjmení studenta vyplývající z jeho pořadového čísla

- je vždy čtyřmístné s nevýznamovými nulami, začíná od `0001`

  - číslování nemusí být spojité

- mohou se mezi sebou míchat studenti různé formy studia (např. prezenční a kombinovaná)

- studenti různých typů (např. bakalářský a navazující) studia mají vždy svoji skupinu očíslovanou od `0001`

### 1.1.2.5. `f` = forma studia

- jedno písmeno, které určuje formu studia

- povoleny jsou znaky

  - `P` = prezenční

  - `K` = kombinovaná

  - `D` = distanční

### 1.1.2.6. `N` = nepovinné

- jedno nebo více písmen, které určují doplňující informace

- nejčastěji se používá písmeno `I` jako označení mezinárodních (*international*) studentů, např. `A14B0001PI`