Proszę napisać program, który:

1. generuje **klucz publiczny** (e, n) i **prywatny** (d, n) RSA, wychodząc od dwóch liczb pierwszych: $p = 397$ p=397, $q = 103$ q=103 (3 p.),
2. koduje wyraz podany jako parametr z wykorzystaniem algorytmu RSA (3 p.),
3. odszyfrowuje zakodowany wcześniej wyraz (dla testu) (3 p.).

Realizacja zadania oceniana jest na 9 punktów. 1 punkt można uzyskać rozkodowując losowe hasło, które znajduje się w osobnej aktywności na UPEL.

Zadanie dodatkowe (nieoceniane, do realizacji gdy ktoś skończy przed czasem lub do realizacji w domu dla chętnych):
Proszę dopisać możliwość wyboru trybu działania programu (kodowanie/dekodowanie) oraz dodać możliwość deszyfrowania wartości podanych jako argumenty wywołania programu, np.:
java RSA code Sierpien
java RSA decode 10613 39378 9444 5109 9197 39378 9444 16086

Generowanie kluczy
Należy obliczyć kolejno:

- $n = p \cdot q$ n=p·q
- $\phi(n) = (p-1)(q-1)$ ϕ(n)=(p−1)(q−1)
- e e - niewielka liczba nieparzysta względnie pierwsza z $\phi(n)$ ϕ(n). Proszę ją obliczyć jako najmniejszą nieparzystą liczbę spełniającą $NWD(e, \phi(n))$ NWD(e,ϕ(n)). Przypominam algorytm Euklidesa (lub można wykorzystać odpowiednią metodę klasy `BigInteger`):

```
Euclid(a, b)
1. while b ≠ 0
2.       do temp ← a mod b
3.          a ← b
4.          b ← temp
5. return a
```

- d d - multiplikatywna odwrotność $e \mod \phi(n)$ emodϕ(n), tj. liczba naturalna spełniająca równanie: $(d \cdot e) \mod \phi(n) = 1$ (d·e)modϕ(n)=1. Możemy ją znaleźć dzięki rozszerzonemu algorytmowi Euklidesa (wywołanemu dla argumentów e e oraz $\phi(n)$ ϕ(n)):

```
extendedEuclid(a, b)
1.  x0 ← 1
2.  x ← 0
3.  b0 ← b
4.  while b ≠ 0
5.        do q ← ⌊a/b⌋
6.           temp ← x
7.           x ← x0 - q · x
8.           x0 ← temp
9.           temp ← a mod b
10.          a ← b
11.          b ← temp
12. if x0 < 0
13.    then x0 ← x0 + b0
14. return x0
```

Kodowanie

- Przed szyfrowaniem: każdą z wejściowych liter konwertujemy na liczbę (u nas najprościej: zgodnie z kodowaniem UTF-16 -- wystarczy zwykłe rzutowanie na typ całkowity).

- Szyfrowanie przy pomocy klucza publicznego: dla każdej liczby t (≡ litery) wykonujemy $c = t^e \mod n$ c=temodn.
- Uwaga na wyjście poza zakres! Przydatny algorytm szybkiego potęgowania z dzieleniem modulo (albo lepiej wykorzystać klasę `BigInteger` i odpowiednią metodę):

```
powMod(t, e, n)
1. y ← 1
2. for k ← m-1 downto 0 // m = liczba bitów liczby e
3.     do y ← y · y mod n
4.        if e_k == 1 // e_k = k-ty bit liczby e (na pozycji k)
5.           then y ← y · t mod n
6. return y
```

Dekodowanie

- Deszyfrowanie przy pomocy klucza prywatnego: dla każdej zakodowanej liczby c wykonujemy: $c^d \mod n$ cdmodn. Wynik powinien zgadzać się z wejściową liczbą (literą) tt.