1.
Implementacja własnego ładowacza klas (zgodnie z używanym od wersji JDK 1.2 modelem delegacji) odbywa się poprzez dziedziczenie z `ClassLoader` oraz

a) przysłonięcie metody `findClass()`, odpowiedzialnej za załadowanie kodu bajtowego klasy oraz zdefiniowanie klasy – do czego wykorzystywana jest metoda `defineClass()`
b) przesłonięcie metody `loadClass()`, odpowiedzialnej za załadowanie kodu bajtowego klasy oraz zdefiniowanie klasy– do czego wykorzystywana jest metoda `defineClass()`
c) przesłonięcie metody `forName()`, odpowiedzialnej za załadowanie kodu bajtowego klasy oraz zdefiniowanie klasy– do czego wykorzystywana jest metoda `defineClass()`
d) przysłonięcie metody `findClass()`, odpowiedzialnej za załadowanie kodu bajtowego klasy oraz zdefiniowanie klasy – do czego wykorzystywane są metody `loadClass()` i `defineClass()`

5.
Do czego służy adnotacja `@FunctionalInterface`?

a) Pozwala określić metodę, które będzie użyta w pierwszej kolejności spośród metod interfejsu funkcjonalnego
b) Umożliwia wygenerowanie błędu kompilacji jeśli interfejs nie spełnia warunków bycia interfejsem funkcjonalnym
c) Służy do definiowania skrótowych nazw metod
d) Pozwala, by dany interfejs mógł być użyty jako interfejs funkcjonalny

10.
Co może zawierać interfejs zadeklarowany w Java 9 i wyżej?

a) Implementacje prywatnych statycznych metod
b) Implementacje dowolnych publicznych metod
c) Tylko deklaracje metod (jeśli nie są to metody domyślne)
d) Deklaracje prywatnych interfejsów

26.
W której linii poniższego kodu kompilator wskaże błąd?

```
interface I1 {
      default int m() { return 0;}
}

class D1 {}
class D2 extends D1 {}
class D3 extends D2 {}

class C<T extends D2 & I1 > {          // 1
      public T m(T t) {                 // 2
            D3 d = (D3) t;              // 3
            return (t.m() > 0) ? t : d;// 4
      }
}
```

a) 1
b) 2
c) 3
d) 4