1.
Jaki będzie wynik próby skompilowania i uruchomienia poniższego fragmentu kodu?

```
int a = 220;
int b = 202;
int c = 200;
int d = 020;
int e = 002;
int sum = a + b + c + d + e;
System.out.println("Sum = " + sum);
```

a) Na ekranie pojawi się: `Sum = 644`
b) Na ekranie pojawi się: `Sum = 640`
c) Na ekranie pojawi się: `Sum = 222`
d) Kompilacja tego fragmentu zakończy się błędem

2.
Które z poniższych nie jest słowem kluczowym języka Java?

a) `default`
b) `implement`
c) `imports`
d) `volatile`

3.
Co można powiedzieć o poniższych wyrażeniach?

```
double d1 = 200.0;        // 1
double d2 = 2E+2;         // 2
double d3 = 0.02E4;       // 3
double d4 = 20000e-2;     // 4
```

a) Wszystkie wyrażenia są poprawne. Każda z zadeklarowanych zmiennych przyjmie wartość `200.0`
b) Poprawne jest tylko wyrażenie `1`. Kompilacja pozostałych wyrażeń zakończy się błędem
c) Błąd kompilacji pojawi się w wyrażeniu `3`
d) Błąd kompilacji pojawi się w wyrażeniu `4`

4.
Co można powiedzieć o poniższych wyrażeniach?

```
float f2 = "1"; // 1
float f1 = 1F;  // 2
float f3 = 1.0; // 3
float f4 = 1.0d;// 4
```

a) Wyrażenie 1 jest poprawne
b) Wyrażenie 2 jest poprawne
c) Wyrażenie 3 jest poprawne
d) Wszystkie wyrażenia są błędne

5.
Co można powiedzieć o poniższym kodzie?

```
public class A {
    int i = 10;
    volatile public long l = 0; // 1
    long getLong() {
        volatile long l = 10;     // 2
        return l;
    }
    volatile int getInt() {      // 3
        return i;
    }
}
```

a) Jego kompilacja przebiegnie bezbłędnie
b) Podczas jego kompilacji wystąpi błąd w liniach 1 i 2
c) Podczas jego kompilacji wystąpi błąd w liniach 2 i 3
d) Podczas jego kompilacji wystąpi błąd w liniach 1, 2 i 3

6.
Co można powiedzieć o poniższym kodzie?

```
public class A {
    static <T> void fromArrayToArray(T[] a1, T[] a2) {
        for (int i=0; i< a1.length; i++) {
            a2[i] = a1[i];
        }
    }

    public static void main(String args[]) {
        Float[] af = new Float[10];
        Number[] an = new Number[10];
        fromArrayToArray(an,af);
    }

}
```

a) Jego kompilacja i uruchomienie medoty `main()` przebiegnie bez błędów
b) Jego kompilacja zakończy się błędem
c) Jego kompilacja powiedzie się, jednak po uruchomieniu metody `main()` wyrzucony zostanie wyjątek `java.lang.ArrayStoreException`
d) Jego kompilacja powiedzie się, jednak po uruchomieniu metody `main()` wyrzucony zostanie wyjątek `java.lang.NullPointerException`

7.
Co można powiedzieć o poniższym kodzie?

```
import java.util.ArrayList;
import java.util.Collection;

public class A {
        static <T> void fromColToCol(Collection<T> c1, Collection<T> c2) {
          for (T o : c1) {
            c2.add(o);
        }
    }

    public static void main(String args[]) {
        Collection<Number> cn = new ArrayList<Number>();
        Collection<Float> cf = new ArrayList<Float>();
        fromColToCol(cf, cn);
    }
}
```

a) Jego kompilacja i uruchomienie medoty `main()` przebiegnie bez błędów
b) Jego kompilacja zakończy się błędem
c) Jego kompilacja powiedzie się, jednak po uruchomieniu metody `main()` zostanie wyrzucony wyjątek `java.lang.ClassCastException`
d) Jego kompilacja powiedzie się, jednak po uruchomieniu metody `main()` zostanie wyrzucony wyjątek `java.lang.NullPointerException`

8.
Co można powiedzieć o poniższym kodzie?

```
public class A {
    public static <T> boolean areEqual(B<T> t1, B<T> t2) {
        return ((Object)t1.get()).equals((Object)t2.get());
    }

    public static void main(String args[]) {
        var b1 = new B<Double>();  b1.set(Double.valueOf("1")); // 1
        B<Double> b2 = new B<>();  b2.set(1.0);                 // 2
        boolean areEqual = areEqual(b1, b2);                    // 3
        System.out.println(areEqual);
    }
}
class B<T> {
    private T object;
    public T get() { return this.object; }
    public void set(T object) { this.object = object;}
}
```

a) Jego kompilacja zakończy się błędem w którejś z linii `1` lub `2`
b) Jego kompilacja zakończy się błędem w linii `3`
c) Po jego kompilacji i uruchomieniu metody `main()` na ekranie pojawi się napis `false`
d) Po jego kompilacji i uruchomieniu metody `main()` na ekranie pojawi się napis `true`

9.
Co można powiedzieć o poniższym kodzie?

```
import java.util.Properties;

public class A {
    public static void main(String args[]) {
        Properties p = System.getProperties();
        System.out.println("a"+p.getProperty("path.separator")+"b");
    }
}
```

a) Jego kompilacja powiedzie się, jednak po uruchomieniu metody `main()` zostanie wyrzucony wyjątek `java.util.NoSuchElementException`
b) Po jego kompilacji i uruchomieniu metody `main()` na ekranie pojawi się napis `anullb`
c) Po jego kompilacji i uruchomieniu metody `main()` na ekranie pojawi się napis `a/b` lub `a\b` (zależnie od systemu operacyjnego)
d) Po jego kompilacja i uruchomieniu metody `main()` na ekranie pojawi się znak `a;b` lub `a:b` (zależnie od systemu operacyjnego)

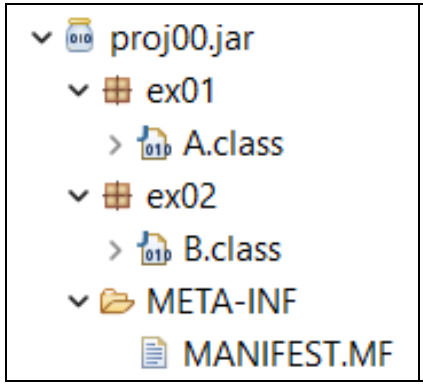
10.
Co można powiedzieć o poniższym kodzie?

```
interface I{
    default void m() {     }
}
interface J{
    void m();
}

public class A implements I, J { // 1
    @Override
    public void m() {            // 2
        I.super.m();             // 3
    }
    public static void main(String[] args) {
        new A().m();
    }
}
```

a) Jego kompilacja zakończy się błędem w linii `1` (konflikt nazw metod w implementowanych interfejsach)
b) Jego kompilacja zakończy się błędem w linii `2` (zły modyfikator dostępu)
c) Jego kompilacja zakończy się błędem w linii `3` (źle wywołano metodę domyślną)
d) Jego kompilacja i uruchomienie medoty `main()` przebiegnie bez błędów

11.

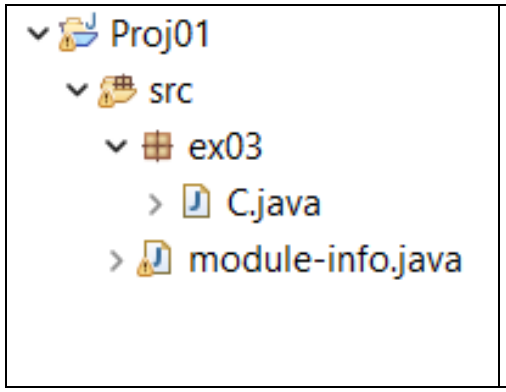
Niech `proj00.jar` będzie niemodułowym plikiem `jar`, zawierającym kod bajtowy następujących klas:




```
package ex01;
import ex02.B;

public class A {
  public void m(B b) {}
}
```

```
package ex02;

public class B {
}
```

Niech `Proj01` będzie modułowym projektem o strukturze jak niżej:




```
package ex03;
import ex01.A;
import ex02.B;

public class C {
  public void m() {
      new A().m(new B()); }}
```

```
module proj01 {
// 2
}
```

`Proj01` będzie można skompilować, jeżeli:

a) `proj00.jar` trafi na ścieżkę klas projektu, a w `module-info.java` w linii `2` pojawi się
   `requires proj00;`

b) `proj00.jar` trafi na ścieżkę modułów projektu, a w `module-info.java` w linii `2` pojawi się linia
   `requires proj00;`

c) `proj00.jar` trafi na ścieżkę klas tego projektu

d) nie da się nic zrobić, kompilowanie modułowego projektu, w którym wykorzystuje się klasy z biblioteki niemodułowej nie jest możliwe

12.

Niech `projekt02.jar` będzie modułowym plikiem `jar`, zawierającym kod bajtowy następujących klas:

projekt02.jar
ex01
A.class
ex02
B.class
module-info.class
META-INF
	`package ex01;`
	`import ex02.B;`
	
	`public class A {`
	`  public void m(B b) {}`
	`}`
		`package ex02;`
		
		`public class B {`
		`}`
			`module proj02 {`
			`// 1`
			`}`

Niech `Proj03` będzie modułowym projektem o strukturze jak niżej:

Proj03
src
ex03
C.java
module-info.java
	`package ex03;`
	`import ex01.A;`
	`import ex02.B;`
	
	`public class C {`
	`  public void m(){ new A().m(new B());}}`
		`module proj03 {`
		`// 2`
		`}`

`Proj03` będzie można skompilować, jeżeli:

a) `projekt02.jar` trafi na jego ścieżkę klas, zaś w linii `2` pojawi się: `requires proj02;`

b) `projekt02.jar` trafi na jego ścieżkę modułów, zaś w linii `2` pojawi się: `requires projekt02;`

c) `projekt02.jar` powstanie po wstawieniu w linii `1` wyrażenia: `exports ex01, ex02;`
oraz plik ten trafi na ścieżkę modułów projektu, zaś w linii `2` pojawi się: `requires proj02;`

d) `projekt02.jar` powstanie po wstawieniu w linii `1` wyrażeń: `exports ex01; exports ex02;`
oraz plik ten trafi na ścieżkę modułów projektu, zaś w linii `2` pojawi się: `requires proj02;`

13.
Co można powiedzieć o poniższym kodzie?

```
package ex00;
interface I {
    void m(int i, D d, C c);
}
```

```
package ex00;
public class C {
    public void mD(int i, D d) {}
}
```

```
package ex00;
public class D {
    public void mC(int i, C c) {}
}
```

```
package ex00;
public class A {
    public void method(I i) {
        i.m(0, null, null); // 0
    }
    public static void main(String[]
args) {
        A a = new A();
        a.method(D::mC);     // 1
        a.method(C::mD);     // 2
    }
}
```

a) podczas kompilacji tego kodu wystąpi błąd w liniach 0, 1, 2
b) podczas kompilacji tego kodu wystąpi błąd w liniach 1, 2
c) podczas kompilacji tego kodu wystąpi błąd w linii 2
d) kod ten skompiluje się poprawnie

14.

Jaki efekt przyniesie próba uruchomienia kolejno klas `A` i `B` na różnych konsolach (zakładając, że są w jednym projekcie, a deklaracje pakietów, importów oraz zawartość `module-info.java` były poprawne)?

`public class A {`
`public static void main(String[] args) throws Exception`
`{`
`  Registry reg = LocateRegistry.createRegistry(3000);`
`  reg.rebind("Server",`
`  (I) UnicastRemoteObject.exportObject(`
`  (I) i -> i.n(), 0));`
`  }}`
	`public interface I`
	`extends Remote {`
	`void m(J j) throws`
	`RemoteException;`
	`}`

`public class B {`
`public static void main(String[] args) throws Exception`
`{`
`I i = (I) LocateRegistry.getRegistry("localhost", 3000)`
`  .lookup("Server");`
`i.m((J) UnicastRemoteObject.exportObject(`
`  (J) () -> System.out.println("1"), 0));`
`i.m((J) () -> System.out.println("2"));`
`System.exit(0);`
`  }}`
	`public interface J`
	`extends Remote {`
	`void n() throws`
	`RemoteException;`
	`}`

a) konsola `A` pozostanie pusta, a na konsoli `B` zostanie wypisane `1` a potem wyrzucony zostanie wyjątek

b) na konsoli `A` zostanie wypisane `2`, a na konsoli `B` zostanie wypisane `1`

c) konsola `A` pozostanie pusta, a na konsoli `B` zostanie wypisane `1` oraz `2`,

d) klas nie da się uruchomić, bo w zademonstrowany sposób nie można używać wyrażeń lambda

15.
Co można powiedzieć o poniższym kodzie (zakładając, że wcześniej dokonano wymaganych importów)?

```
List<?> li = new ArrayList<>();
```

a) Jego kompilacja zakończy się błędem
b) Jego kompilacja powiedzie się, po takiej deklaracji będzie można wywołać komendę
`li.add(0);`
c) Jego kompilacja powiedzie się, po takiej deklaracji będzie można wywołać komendę
`li.addAll(Arrays.asList(new Integer[] {1,2,3}));`
d) Jego kompilacja powiedzie się, po takiej deklaracji będzie można wywołać komendę
`li.get(0);`

16.
Co można powiedzieć o poniższym kodzie?

```
class B {
}
class C extends B {
}
interface I {
    public B m();
}
interface K extends I {
    public C m();
}
```

```
public class A {
    public C m() {
        return null;
    }
    public static void method(K i) {
        System.out.println(i.m());
    }
    public static void main(String[] args) {
        method((K) new A()); // 1
    }
```

a) Jego kompilacja zakończy się błędem z uwagi na konflikt nazw metod w deklaracji interfejsów
b) Jego kompilacja zakończy się błędem z uwagi na niedopuszczalne rzutowanie w linii `1`
c) Jego kompilacja przebiegnie bezbłędnie, ale po uruchomieniu metody `main()` zostanie wyrzucony wyjątek
d) Jego kompilacja i uruchomienie przebiegnie bez przeszkód

17.
Co można powiedzieć o poniższym kodzie?

```
interface I1 {
}
interface I2 {
    int m2();
}
public class A implements I1, I2 {                // 1
    public int m2() {
        return 2; }

    public static void main(String[] args) {
          Object o = new A();
          System.out.println(((I1&I2) o).m2()); // 2
    }
}
```

a) Jego kompilacja zakończy się błędem w linii 1
b) Jego kompilacja zakończy się błędem w linii 2
c) Jego kompilacja powiedzie się, jednak po uruchomieniu metody `main()` zostanie wyrzucony wyjątek
d) Jego kompilacja i uruchomienie metody `main()` przebiegną poprawnie. Na ekranie zostanie wypisane 2

18.
Co można powiedzieć o poniższym kodzie?

```
interface I1 {
    int m();
}
interface I2 {
    int m();
}
public class A {
    public int m() {
        return 0;}

    public static void main(String[] args) {
         A a = new A();                          // 1
         System.out.println(((I1&I2)a).m()); // 2
    }
}
```

a) Jego kompilacja zakończy się błędem w linii 1
b) Jego kompilacja zakończy się błędem w linii 2
c) Jego kompilacja powiedzie się, a po uruchomieniu metody `main()` zostanie wyrzucony wyjątek w linii 1
d) Jego kompilacja powiedzie się, a po uruchomieniu metody `main()` zostanie wyrzucony wyjątek w linii 2

19.
Co można powiedzieć o poniższym kodzie umieszczonym w jednym pliku?

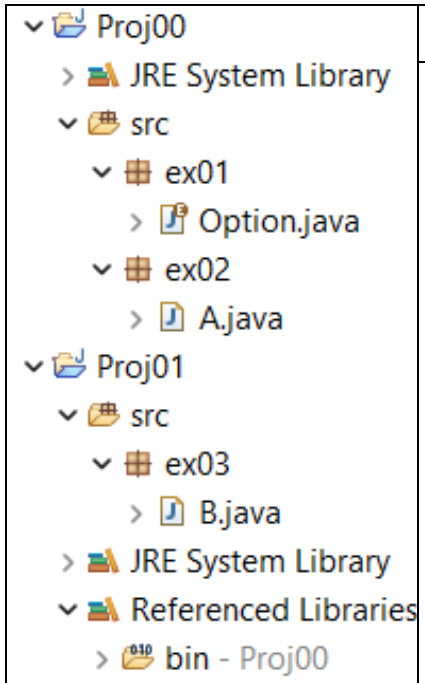
```
@FunctionalInterface
interface J1 extends I<String, Integer> {}

@FunctionalInterface
interface J2 extends I<Integer, Integer> {}

interface I<T, N extends Number> {
    void m(T arg);
    void m(N arg);
}
```

a) Jego kompilacja zakończy się błędem (interfejs `J1` nie może być zadeklarowany jako funkcyjny)
b) Jego kompilacja zakończy się błędem (interfejs `J2` nie może być zadeklarowany jako funkcyjny)
c) Jego kompilacja zakończy się błędem (deklaracja interfejsu `I` powinna być przed deklaracjami `J1` i `J2`)
d) Jest to poprawny kod

20.

Niech w `Proj00` i `Proj01` będą standardowymi projektami, a klasy `Proj00` będą widoczne w ścieżce klas `Proj01`. Co można powiedzieć o poniższym scenariuszu (zakładając, że deklaracje pakietów, importów oraz ścieżka do pliku są poprawne):

1. Skompilowano `Proj00` i uruchomiono `main()` z klasy `A` (powstał plik `obj.dat`)
2. Do typu wyliczeniowego `Option` dodano nową stałą ***POOR*** i przekompilowano go na nowo
3. Skompilowano `Proj01` i uruchomiono `main()` z klasy `B` (odczytano plik `obj.dat`)

<table>
<tr><td rowspan="2"> 
</td><td>

```
public enum Option { RICH }
```

</td><td></td></tr>
<tr><td>

```
public class A {
public static void
main(String[] a) { try (
var fos = new
FileOutputStream("../obj.dat")
;
var oos = new
ObjectOutputStream(fos)) {
oos.writeObject(Option.RICH);
    } catch (Exception e) {
ex.printStackTrace();}
}}
```

</td><td>

```
public class B {
public static void
main(String[] a) { try (
var fos = new
FileInputStream("../obj.dat")
;
var ois = new
ObjectInputStream(fos)) {
Object o = ois.readObject();
    } catch (Exception e) {
ex.printStackTrace();}
}}
```

</td></tr>
</table>

a) Wszystkie kroki scenariusza wykonają się bez problemu

b) Scenariusz przerwie się w kroku 1 wyrzuceniem wyjątku `java.io.NotSerializableException`

c) Scenariusz przerwie się w kroku 3 wyrzuceniem wyjątku `java.io.InvalidObjectException`

d) Żaden krok scenariusz nie wykona się, bo źle zaimplementowano operacje na plikach

21.
Co można powiedzieć o poniższym kodzie?

```java
package ex01;
public abstract class A {
    public int i;

    public void setI(int i) {
        this.i = i;
    }
    public A createA(int i){
        return this(10);   // 1
    }
    protected A(int i) { // 2
        setI(i);
    }
}
```

```java
package ex02;

import ex01.A;

public class B extends A {
    B(int i) {
        super(i); // 3
    }
}
```

a) Jest to niepoprawny kod (zła implementacja metody fabryki w linii 1)
b) Jest to niepoprawny kod (zła deklaracja konstruktora w linii 2)
c) Jest to niepoprawny kod (złe wywołanie konstruktora klasy nadrzędnej w linii 3)
d) Jest to poprawny kod

22.
Co można powiedzieć o poniższym kodzie ?

```
public class A {
    public void run() {
        new Thread(() -> {
            while (true) {
                synchronized (A.class) {
                    try { wait(); } catch (InterruptedException e) {}
                    System.out.println("1");
            }}}).start();

        new Thread(() -> {
            while (true) {
                synchronized (A.class) {
                    notify(); }
            }}).start();
    }
    public static void main(String[] args) {
        new A().run();
    }
}
```

a) Jego kompilacja zakończy się błędem
b) Kod skompiluje się poprawnie. Jednak po jego uruchomieniu wyrzucony zostanie wyjątek `java.lang.IllegalMonitorStateException`
c) Kod skompiluje się i uruchomi poprawnie. Na ekranie w nieskończoność będzie wypisywane `1`
d) Kod skompiluje się i uruchomi poprawnie. Mimo działania programu na ekranie nic się nie pojawi

23.
Co można powiedzieć o poniższym kodzie?

```
public class A {
    public void run() {
        Thread t = new Thread(() -> {
            while (true) {
                synchronized (this) {
                    try { wait();
                    } catch (InterruptedException e) {
                        System.out.println("1"); }
                    System.out.println("2");
            }}});
        t.start();
        new Thread(() -> {
            while (true) {
                t.interrupt();
            }}).start();
    }
    public static void main(String[] args) {
        new A().run();   }
}
```

a) Kod skompiluje się i uruchomi poprawnie. Na ekranie naprzemiennie bez końca pojawiać się będą `1` i `2`
b) Kod skompiluje się i uruchomi poprawnie. Na ekranie pojawia się `1` i `2` po czym program skończy działanie
c) Kod skompiluje się poprawnie. Jednak po jego uruchomieniu zostanie wyrzucony wyjątek `java.lang.IllegalMonitorStateException`
d) Jego kompilacja zakończy się błędem

24.

Które zestawienie oddaje poprawnie metody klasy `java.lang.Object`?

a)

- Object()
- clone() : Object
- equals(Object) : boolean
- finalize() : void
- getClass() : Class<?>
- hashCode() : int
- notify() : void
- notifyAll() : void
- toString() : String
- wait() : void
- wait(long) : void
- wait(long, int) : void

b)

- Object()
- clone() : Object
- equals(Object) : boolean
- finalize() : void
- getClass() : Class<?>
- hashCode() : int
- notify() : void
- notifyAll() : void
- toString() : String
- wait() : void
- wait(long) : void
- wait(long, int) : void

c)

- Object()
- clone() : Object
- equals(Object) : boolean
- finalize() : void
- getClass() : Class<?>
- hashCode() : int
- notify() : void
- notifyAll() : void
- toString() : String
- wait() : void
- wait(long) : void
- wait(long, int) : void

d)

- Object()
- clone() : Object
- equals(Object) : boolean
- finalize() : void
- getClass() : Class<?>
- hashCode() : int
- notify() : void
- notifyAll() : void
- toString() : String
- wait() : void
- wait(long) : void

Legenda:

A – **abstract** N – **native** F – **final** C – constructor S – **static**	**public** **private** **package** **protected**	**@Deprecated** **@Deprecated** **@Deprecated** **@Deprecated**

25. Która deklaracja metody parametryzowanej typem jest poprawna?

a) `public <T> void m1(<T extends Number>[] t) { }`
b) `public void m2<T>(<T super Number>[] t) { }`
c) `public <T extends Number> void m3(T[] t) { }`
d) `<T> public void m4(T[] t) { }`

26.
Co można powiedzieć o poniższym kodzie?

```
public static void main(String[] args) {
    int i = 0;

    et: i++;
    do {
     for(; i<10; i++) {
       if (i%3 == 0) {
           System.out.print(i+" ");
           break et;
       }
     }
    } while (i<3);
}
```

a) po jego poprawnej kompilacji i uruchomieniu na ekranie pojawiać się będzie w nieskończoność: `0 0 0 ...`
b) po jego poprawnej kompilacji i uruchomieniu na ekranie pojawi się: `3 6 9`
c) jego kompilacja zakończy się błędem
d) jego kompilacja powiedzie się, jednak przy próbie uruchomienia zostanie wyrzucony wyjątek

27.
Która z opcji JVM pozwala wyświetlić na ekranie informacje o optymalizacji dokonanej przez JIT?

a) `-XX:+PrintJITOptimization`
b) `-XX:+PrintOptimization`
c) `-XX:+PrintCompilation`
d) `-XX:+TraceOptimization`

28.
Co można powiedzieć o poniższym kodzie?

```
public class A {

    public A() throws Exception {
    }
    public A(int i){        // 1
    }
    public static void main(String[] args) {
        new A(10);          // 2
    }
}
```

a) Jego kompilacja zakończy się błędem w linii `1`
b) Jego kompilacja zakończy się błędem w linii `2`
c) Kompilacja tego kodu i uruchomienie metody `main()` przebiegnie bez błędu
d) Kompilacja tego kodu powiedzie się, ale po uruchomieniu metody `main()` zostanie wyrzucony wyjątek

29.
Co można powiedzieć o poniższym kodzie?

```
class B extends A {
    { i = 2; }          // 1
}
public class A {
    { i = 1; }          // 2
    public int i = 0;

    public static void main(String[] args) {
        System.out.print(new A().i + " " + new B().i);
    }
}
```

a) Jego kompilacja zakończy się błędem w linii `1`
b) Jego kompilacja zakończy się błędem w linii `2`
c) Jego kompilacja powiedzie się, a po uruchomieniu metody main() na ekranie pojawi się: `0 2`
d) Jego kompilacja powiedzie się, a po uruchomieniu metody main() na ekranie pojawi się: `1 2`

30.
Co można powiedzieć o poniższym kodzie?

```
@FunctionalInterface        // 1
interface I {
    default void n() {  }
    J m();
}

interface J {
     void n();
     static void k() {};
}
public class A implements I {
     public J m() {          // 2
       return this::n;       // 3
     }
}
```

a) Jego kompilacja powiedzie się
b) Jego kompilacja zakończy się błędem w linii `1` (`I` nie może być zadeklarowany jako interfejs funkcyjny)
c) Jego kompilacja zakończy się błędem w linii `2` (jeśli z metody zwracany jest interfejs, to musi on być interfejsem funkcyjnym)
d) Jego kompilacja zakończy się błędem w linii `3` (nie można w ten sposób zwracać referencji do metody domyślnej interfejsu `I` dostarczanego przez instancję klasy `A`)