1.
Które z poniższych deklaracji modułów umieszczone w odpowiednich plikach `module-info.java` są poprawne (pomijając fakt, że mamy tu puste nawiasy `{}`):

a) `module simpleLibrary10 {}`

b) `module pl.edu.pwr.simplelibrary-1.0 {}`

c) `module simpleLibrary-1.0-SNAPSHOT {}`

d) `module simple-library_01 {}`

2.
Co można powiedzieć o poniższym kodzie (dla przejrzystości pominięto wymagane importy)?

```
public interface I extends Remote {
    record R (int x) {}                          // 1
    public R getR() throws RemoteException;
    public void setR(R r) throws RemoteException;
}
```

a) Jest to poprawny kod.
b) Jest to niepoprawny kod. Aby kod był poprawny linię `1` należałoby zastąpić linią:
```
    public record R (int x) {}
```
c) Jest to niepoprawny kod. Aby kod był poprawny linię `1` należałoby zastąpić linią:
```
    record R (int x) implements Serializable {}
```
d) Jest to niepoprawny kod. Nie można w interfejsie deklarować metod korzystających z typu, który jest zadeklarowany wewnątrz tego interfejsu.

3.
Co można powiedzieć o poniższym kodzie (dla przejrzystości pominięto wymagane importy)?

```
public interface MyInterface extends Remote {
    enum R { A, B, C}                          // 1
    public R getR() throws RemoteException;
    public void setR(R r) throws RemoteException;
}
```

a) Jest to poprawny kod.
b) Jest to niepoprawny kod. Aby kod był poprawny linię `1` należałoby zastąpić linią:

```
public enum R { A, B, C}
```

c) Jest to niepoprawny kod. Aby kod był poprawny linię `1` należałoby zastąpić linią:

```
enum R implements Serializable { A, B, C}
```

d) Jest to niepoprawny kod. Nie można w interfejsie deklarować metod korzystających z typu, który jest zadeklarowany wewnątrz tego interfejsu.

4.
Co można powiedzieć o poniższym kodzie (jeden plik źródłowy)?

```
class C implements B.I {
    private void m() {      // 1
        System.out.println("B.m()"); }
}
class B {
    protected interface I {
        public void m(); }
}
public class A {
    public static void main(String[] args) {
        new C().m();        // 2
        }
}
```

a) Jest to poprawny kod.
b) Jest to niepoprawny kod. Będzie poprawny, jeśli modyfikator dostępu w linii `1` będzie różny od **`private`**.
c) Jest to niepoprawny kod. Deklaracja klasy `B` powinna być przed deklaracją klasy `C`.
d) Jest to niepoprawny kod, jednak sama składnia wyrażenia występującego w linii `2` jest poprawna.

5.
Jeśli jakaś modułowa aplikacja (w sensie JPMS) korzysta z modułu `library` znajdującego się w ścieżce modułów i udostępniającego pakiety z definicjami interfejsów rmi bez żadnych zależności przechodnich, to aby dało się skorzystać w tej aplikacji z tych interfejsów (i tę aplikację uruchomić):

a) wystarczy, że w `module-info.java` aplikacji pojawi się linijka:

```
requires library;
```

b) w `module-info.java` aplikacji muszą pojawić się linijki:

```
requires library;
requires java.rmi;
```

c) w `module-info.java` aplikacji musi pojawić się linijka:

```
requires library, java.rmi;
```

d) w `module-info.java` aplikacji musi pojawić się linijka z nazwą pakietu zawierającego klasę z metodą `main()`, np.

```
exports ex;
```

6.
Załóżmy, że mamy dwa standardowe projekty Java (nazwijmy je P1 i P2). W obu projektach jest zdefiniowana klasa `B` w domyślnym pakiecie, z potencjalnie różnymi wartościami w miejsu wielokropka.

```
class B implements Serializable {
   private static final long serialVersionUID = ...; // 1
}
```

Co się stanie przy próbie uruchomienia fragmentu kodu z P1, a następnie fragmentu kodu z P2?

P1	P2

`try(var s = new ServerSocket(2000)) {`
`  while (true) {`
`  var sc = s.accept();`
`  System.out.println(new`
`  ObjectInputStream(`
`  s.accept().getInputStream())`
`  .readObject());`
`  sc.close(); }`
`} catch (Exception e) {`
`  e.printStackTrace(); }`
	`try(var s = new Socket("localhost",`
	`2000)){`
	`  new ObjectOutputStream(`
	`  s.getOutputStream())`
	`  .writeObject(new B());`
	`  } catch (Exception e) {`
	`  e.printStackTrace();}`

a) Komunikacja zakończy się wyrzuceniem wyjątku po stronie P1 (`java.io.InvalidClassException`) jeśli w klasach `B` w miejscu wielokropka (linia `1`) będą różne wartości.
b) Jeśli z obu klas `B` zostanie usunięta linia `1`, to komunikacja przebiegnie poprawnie.
c) Komunikacja zakończy się wyrzuceniem wyjątku po stronie P1 (`java.io.InvalidClassException`) niezależnie od tego, co zostanie wstawione w miejce wielokropka (linia `1`).
d) Komunikacja zakończy się wyrzuceniem wyjątku po obu stronach.

7.
Jaki będzie wynik uruchomienia klasy `A`, jeśli została ona zadeklarowana wraz z interfejsem `I` jak niżej?

```
interface I {
  default A new(int i) {
     return new A(2*i);
  }
}
public class A implements I {
  public int i;
  public A(int i) {
    this.i = i;
  }
  public static void main(String[] args) {
    A a = new A(2).new(1);
    System.out.println(a.i);
  }
}
```

a) Na ekranie zostanie wypisane `2` .
b) Na ekranie zostanie wypisane `4` .
c) Kompilacja klasy `A` zakończy się błędem.
d) Uruchomienie klasy `A` zakończy się wyrzuceniem wyjątku.

8.
Co moża powiedzieć o poniższym kodzie (jeden plik źródłowy)?

```
interface I {
    public static void m() { n();}
    public default void n() {}
}
public class A implements I {
    public static void main(String[] args) {
    }
}
```

a) Jest to poprawny kod.
b) Jest to niepoprawny kod. Błędy występują w kodzie klasy `A`.
c) Jest to niepoprawny kod. Błędy występują w kodzie interfejsu `I`.
d) Choć jest to poprawny kod, uruchomienie klasy `A` zakończy się wyrzuceniem wyjątku.

9.
Co można powiedzieć o poniższym kodzie (jeden plik źródłowy, dla przejrzystości pominięto wymagane importy)?

```
class MyException extends Exception {
    MyException(String message) {
    super(message);
}
}
interface I {
    public void m() throws MyException;
}
public class A implements I {
    public void m() throws Exception {
    }
}
```

a) Jest to poprawny kod.
b) Jest to niepoprawny kod. Błąd występuje w klasie `MyException`
c) Jest to niepoprawny kod. Błąd występuje w klasie `A` (w metodzie `m()` źle zadeklarowano typ wyrzucanego wyjątku)
d) Jest to niepoprawny kod. Błąd występuje w klasie `A` (w metodzie `m()` choć zadeklarowano typ wyrzucanego wyjątku, to jednak nigdzie go nie zgłoszono)

10.
Co można powiedzieć o poniższym kodzie (jeden plik źródłowy)?

```
interface I {
    public default void m() throws Exception {}
}
abstract class B {
    public abstract void m();
}
public class A extends B implements I {
    public void m() {
    }
}
```

a) Jest to poprawny kod.
b) Jest to niepoprawny kod. Jego kompilacja zakończy się błędem z uwagi na konflikt nazw metod.
c) Jest to niepoprawny kod. Jego kompilacja zakończy się błędem z uwagi na brak klauzuli **`throws`** przy metodzie `m()` w klasie `A`
d) Jest to niepoprawny kod. Jego kompilacja zakończy się błędem z uwagi na niepoprawnie zdefiniowany interfejs `I`

11.
Co można powiedzieć o poniższym kodzie (jeden plik źródłowy)?

```
interface I {
    public default void m() throws RuntimeException {}
}
public abstract class A implements I {
    public void m() {
        I.m();
    }
}
```

a) Jego kompilacja zakończy się błędem z uwagi na niepoprawnie zdefiniowany interfejs `I`
b) Jego kompilacja zakończy się błędem z uwagi na nieobsłużony wyjątek w metodzie `A.m()`.
c) Jego kompilacja zakończy się błędem z uwagi na niedopuszczale w tym kontekście użycie deklaracji **`abstract`** przy deklaracji klasy `A`.
d) Jego kompilacja zakończy się błędem z uwagi na próbę wywołania metody `I.m()` jakby to była metoda statyczna, a nie instancyjna.

12.
Co można powiedzieć o poniższym kodzie?

```
public enum E {
    X;                                      // 1;
    E(int i ){ System.out.println(i); } // 2;
    public static void main(String[] args) { }
}
```

a) Jego kompilacja zakończy się błędem. Typ wyliczeniowy nie może mieć tylko jednego elementu wyliczeniowego (linia `1`)
b) Jego kompilacj zakończy się błędem. Typ wyliczeniowy nie może mieć metody `main()` .
c) Jest to poprawny kod
d) Jego kompilacja zakończy się błędem. Jeśli już jest zadeklarowany jeden konstruktor z argumentem (linia `2`), to każdy element wyliczeniowy powinien być zadeklarowany z jakimś argumentem (np. `X(1)`).

13.
Co można powiedzieć o poniższym kodzie (jeden plik źródłowy)?

```
enum E {
    X(1), Y(2);       // 1
    private int i;
    E(int i){ this.i = i; }
    { System.out.println(Integer.toString(i) + " "); } // 2
}
public class A {
    public static void main(String[] args) {
        E e = E.X;    // 3
    }
}
```

a) Jest to poprawny kod. Po uruchomieniu klasy `A` na ekranie wypisane zostanie `1 2`
b) Jest to poprawny kod. Po uruchomieniu klasy `A` na ekranie wypisane zostanie `0 0`
c) Jest to niepoprawny kod. Błąd jest w przypisaniu w linii `3` (zabrakło atrybutu w nawiasach po prawej stronie przypisania).
d) Jest to niepoprawny kod. Typ wyliczeniowy nie może mieć instancyjnego bloku inicjalizacji (linia `2`) ani deklaracji elementów wyliczeniowych z nawiasami (linia `1`).

14.
Co można powiedzieć o poniższym kodzie (jeden plik źródłowy)?

```
class B {}
class C extends B {}
class D extends C {}

public class A<T> {
    private T t;
    public T getT() { return t; }
    public void setT(T t) { this.t = t; }

    public void m(A<? extends C> x, A<? super C> y) {
        B b = x.getT();  // 1
        x.setT(new B()); // 2
        D c = y.getT();  // 3
        y.setT(new D()); // 4
    }
}
```

a) Jest to niepoprawny kod. Błędy występują w liniach 1 i 2
b) Jest to niepoprawny kod. Błędy występują w liniach 3 i 4
c) Jest to niepoprawny kod. Błędy występują w liniach 1 i 4
d) Jest to niepoprawny kod. Błędy występują w liniach 2 i 3

15.
Co można powiedzieć o poniższym kodzie?

```
public class A<T> {
    A<?>[] tab1 = new A<?>[5];              // 1
    A<String>[] tab2 = new A<String>[5]; // 2
    T[] tab3 = new T[10];                   // 3
}
```

a) Jest to poprawny kod.
b) Jest to niepoprawny kod, ale błędu nie ma w linii `1`.
c) Jest to niepoprawny kod, ale błędu nie ma w linii `2`.
d) Jest to niepoprawny kod, ale błędu nie ma w linii `3`.

16.
Co można powiedzieć o poniższym kodzie (dla przejrzystości pominięto wymagane importy)?

```
public class A {
    public static void main(String[] args) {
        float min = Arrays.stream(new xxx[]{})  // 1
                  .min()                        // 2
                  .orElse(0);                   // 3
        System.out.println(min);
    }
}
```

a) Jeśli w miejsce **xxx** w linii 1 wstawiony zostanie typ **int**, to kompilacja tego kodu powiedzie się i po uruchomieniu klasy A na ekranie zostanie wypisaniem 0.
b) Jeśli w miejsce **xxx** w linii 1 wstawiony zostanie typ **int**, to kompilacja tego kodu zakończy się błędem w linii 3.
c) Jeśli w miejsce **xxx** w linii 1 wstawiony zostanie typ **float**, to kompilacja tego kodu zakończy się błędem wskazanym w tej linii
d) Jeśli w miejsce **xxx** w linii 1 wstawiony zostanie typ **char**, to kompilacja tego kodu zakończy się błędem w linii 2.

17.
Co można powiedzieć o poniższym kodzie (dla przejrzystości pominięto wymagane importy)?

```
@author("Johny")
public class A {
    /**
    * @input i input parameter
    * @input a input parameter
    * @output  intentionally null
    */
    public String m(int i, A a) {
        return null;
    }
}
```

a) Kod zawiera blok komentarza wykorzystywany przez `javadoc`, ale brak w nim adnotacji rozpoznawanych przez to narzędzie.
b) Kod zawiera blok komentarza wykorzystywany przez `javadoc`, w bloku tym występuje przynajmniej jedna adnotacja rozpoznawana przez to narzędzie.
c) Widoczna w kodzie adnotacja `author` nie jest adnotacją wykorzystywaną przez `javadoc`.
d) Kompilacja tego kodu kompilatorem `javac` zakończy się błędem.

18.
Co można powiedzieć o poniższym kodzie (dla przejrzystości pominięto wymagane importy)?

```
interface I {
    static void exclamate(String in) {
        System.out.print(in+"! ");
    }
}
public class A {
    public static void main(String[] args) {
        Arrays.asList("A", "B", "C")
        .forEach(I::exclamate);
    }
}
```

a) Jest to niepoprawny kod. Niepoprawnie zadeklarowano interfejs `I` .
b) Jest to niepoprawny kod. Zły jest atrybut przekazany do metody `forEach()` .
c) Jest to poprawny kod. Po uruchomieniu klasy `A` na ekranie pojawi się: `A! B! C!` .
d) Jest to poprawny kod. Po uruchomieniu klasy `A` na ekranie pojawi się: `C! B! A!` .

19.
Co można powiedzieć o poniższym kodzie (dla przejrzystości pominięto wymagane importy)?

```
public class A extends Thread {
    public A(String name){ super(name); }
    public synchronized void m() {
        while(true) {
            System.out.println(this.getName());
        }
    }
    public void run() { m(); }
    public static void main(String[] args) {
        Arrays.stream(new A[]{new A("1"),new A("2"),new A("3")})
                    .forEach(x -> x.start());
    }
}
```

a) Jest to poprawny kod. Po uruchomieniu klasy A na ekranie będzie wypisywana w nieskończoność cyfra 1
b) Jest to poprawny kod. Po uruchomieniu klasy A na ekranie będą wypisywane w nieskończoność przeplatające się sekwencje każdej z cyfr 1, 2, 3 .
c) Jest to niepoprawny kod. Źle zadeklarowano konstruktor klasy `A` .
d) Jest to niepoprawny kod. Błąd występuje w metodzie `main()`.

20.
Jaki będzie wynik uruchomienia następującego polecenie z linii komend: `javac -d classes *.java` jeśli w bieżącym katalogu są jedynie pliki `A.java` i `B.java` o zawartości jak niżej?

A.java	B.java
`package ex01;` `public class A { }`	`package ex01.ex02;` `public class B { }`

a) Na ekranie pojawi się informacja o błędzie: `error: invalid flag: -p.`
b) W bieżącym katalogu pojawią się pliki `A.class` i `B.class`
c) Pojawią się pliki w następujących ścieżkach: `.\classes\ex01\A.class` oraz `.\classes\ex01\ex02\B.class`
d) Na ekranie pojawi się informacja o błędzie: `System nie może odnaleźć określonej ścieżki`

21.
Jaki będzie wynik uruchomienia następującego polecenie z linii komend: `java A.java 1 2 3`
jeśli w bieżącym katalogu znajduje się plik `A.java` o zawartości jak niżej

```
import java.util.Arrays;
public class A {
    public static void main(String... args) {
        if(args.length>0) {
            System.out.print(args[0]);
            A.main(Arrays.copyOfRange(args, 1, args.length));
        }
    }
}
```

a) Na ekranie pojawi się informacja o błędzie:
`Error: Could not find or load main class A.java`
b) Na ekranie pojawi się informacja o błędzie:
`Error: Could not find or load main class A.class`
c) Na ekranie pojawi się informacja o błędzie:
`A.java:3: error: varargs notation not allowed on receiver parameter`
d) Na ekranie pojawi się napis:
`123`

22.
Co można powiedzieć o poniższym kodzie?

```
public class A {
    public static void main(String[] args) {
        int i = 10;
        do {
            System.out.println(i);
            if(i % 3)
                continue;
            i --;
        } while(true);
    }
}
```

a) Jest to niepoprawny kod. Jego kompilacja nie powiedzie się.
b) Jest to poprawny kod. Po uruchomieniu klasy `A` na ekranie w nieskończoność będzie wypisywane `10`.
c) Jest to poprawny kod. Po uruchomieniu klasy `A` na ekranie po jednokrotnym wypisaniu `10` w nieskończoność wypisywane będzie `9`.
d) Jest to poprawny kod. Po uruchomieniu klasy `A` wypisane zostanie `10` i program skończy swoje działanie.

23.
Kiedy i jak tworzy się i uwidacznia główną scenę (`Stage`) w aplikacji JavaFX?

a) W wyniku działania metody `start()`, w której uruchamia się konstruktor klasy `Stage`
b) W wyniku działania metody `launch()` (ta metoda klasy `Application` tworzy `Stage` i przekazuje referencję do `Stage` do metody `start()`, która finalnie pokazuje elementy na ekranie)
c) W wyniku uruchomienia konstruktora klasy `Application` (instancja klasy `Application` reprezentuje okienko na ekranie).
d) Niezależnie od umiejscowienia polecenia zaraz po uruchomieniu konstruktora klasy `Stage` pojawi się okienko na ekranie.

24.
Który z poniższych przykładów odpowiada zawartości pliku `module-info.java` aplikacji napisanej z wykorzystaniem JavaFX, której kod umieszczono w pakiecie `mypackage`?

a)
```
module mymodule {
  requires javafx.controls;
  requires javafx.fxml;
  requires javafx.graphics;
  requires javafx.fxml;
}
```

b)
```
module mymodule {
  requires javafx.graphics;
  requires javafx.fxml;
  opens mypackage to
javafx.controls, javafx.fxml;
}
```

c)
```
module mymodule {
  requires javafx.fxml;
  opens mypackage to
javafx.graphics, javafx.controls,
javafx.fxml;
}
```

d)
```
module mymodule {
  requires javafx.controls;
  requires javafx.fxml;
  opens mypackage to
javafx.graphics, javafx.fxml;
}
```

25.
Co można powiedzieć o poniższym kodzie?

```
class B extends A {
    public int i = ++super.i;
}
public class A {
    public int i = 2;
    public static void main(String[] args) {
        System.out.print(new B().i + " " + new A().i);
    }
}
```

a) Jest to niepoprawny kod. Jego kompilacja nie powiedzie się.
b) Jest to poprawny kod. Po uruchomieniu klasy `A` na ekranie zostanie wypisane `2 2` .
c) Jest to poprawny kod. Po uruchomieniu klasy `A` na ekranie zostanie wypisane `3 3` .
d) Jest to poprawny kod. Po uruchomieniu klasy `A` na ekranie zostanie wypisane `3 2` .

26.
Co można powiedzieć o poniższym kodzie?

```
public class A implements Cloneable {
    int i = 1;
public static void main(String[] args)throws CloneNotSupportedException {
    A a1 = new A(), a2 = (A) a1.clone();
    System.out.println(a1.i + " " + a2.i);
    }
}
```

a) Jest to niepoprawny kod. Jego kompilacja zakończy się błędem z uwagi na brak implementacji metody `clone()`.
b) Jest to poprawny kod. Po uruchomieniu klasy `A` wyrzucony zostanie wyjątek `java.lang.CloneNotSupportedException` .
c) Jest to poprawny kod. Po uruchomieniu klasy `A` na ekranie wypisane zostanie `1 1` .
d) Jest to poprawny kod. Po uruchomieniu klasy `A` na ekranie wypisane zostanie `1 0` .

27.
Co można powiedzieć o poniższym kodzie?

```
class B extends A {
    { i = 10; }          // 1
}
public class A {
    public final int i; // 2
    public static void main(String[] args) {
        B b = new B();   // 3
        A a = new A();   // 4
    }
}
```

a) Kompilacja tego kodu zakończy się zgłoszeniem błędu w linii `1` i linii `2`
b) Kompilacja tego kodu zakończy się zgłoszeniem błędu w linii `3` i linii `4`
c) Kod skompiluje się poprawnie, a uruchomienie klasy `A` przebiegnie bez przeszków.
d) Kod skompiluje się poprawnie, a po uruchomieniu klasy `A` zostanie zgłoszony wyjątek.

28.
Co można powiedzieć o poniższym kodzie?

```
interface I {
    default void m() {
        this.n();                          // 1
    }
    default void n() {
        System.out.println("I.n()"); }
}
public class A implements I {
    public void n() {                      // 2
        System.out.println("A.n()"); }
    public static void main(String[] args) {
        I i = new A();
        i.m();
    }
}
```

a) Kompilacja tego kodu zakończy się zgłoszeniem błędu w linii `1`
b) Kompilacja tego kodu zakończy się zgłoszeniem błędu w linii `2`
c) Kod skompiluje się poprawnie, a po uruchomieniu klasy `A` na ekranie pojawi się `A.n()`
d) Kod skompiluje się poprawnie, a po uruchomieniu klasy `A` na ekranie pojawi się `I.n()`

29.
Co można powiedzieć o poniższym kodzie?

```
interface I {
    static void n() {
        System.out.print("I.n() ");
    }
    void m();
}
public abstract class A implements I {
    public static void main(String[] args) {
        I i = new A() {     // 1
                    public void m() {
                        System.out.print("A.n() ");
                    }
                };
        i.m();              // 2
        i = I::n;           // 3
        i.m();              // 4
    }
}
```

a) Jest to niepoprawny kod. Jego kompilacja nie powiedzie się z uwagi na błędy w liniach `1` i `2`
b) Jest to niepoprawny kod. Jego kompilacja nie powiedzie się z uwagi na błędy w liniach `3` i `4`
c) Kod skompiluje się poprawnie, a po uruchomieniu klasy `A` zgłoszony zostanie wyjątek
d) Kod skompiluje się poprawnie, a po uruchomieniu klasy `A` na ekranie pojawi się `A.n()I.n()`

30.
Które polecenie korzystające z narzędzia dostępnego w JDK pozwala na dekompilację/dezasemblowanie klasy `A.class`

a) `javac -d A.class`
b) `javap -p A.class`
c) `java  -d A.class`
d) `decompile A.class`