1.
Co można powiedzieć o następującym kodzie:

```
XXX o  = new YYY(){
    // ... tutaj odpowiednia implementacja
};
```

a) Jeśli `XXX` jest jakąś klasą różną od `Object`, to `YYY` nie musi być klasą
b) Jeśli `XXX` jest interfejsem, to `YYY` musi być interfejsem
c) Jeśli `XXX` jest typem `Object`, to `YYY` może być klasą bądź interfejsem
d) Niezależnie od tego, czym będą `XXX` oraz `YYY`, kod będzie niepoprawny

2.
Co można powiedzieć o poniższym kodzie?

```
interface I { }
class A implements I { }
class B extends A { }
class C{
    public static void main(String[] args) {
       A o  = new B(){};
       System.out.print( (o instanceof A) ? "A": "X");
       System.out.print( (o instanceof B) ? "B": "X");
       System.out.print( (o instanceof I) ? "I": "X");
    }
}
```

a) Jest to niepoprawny kod
b) Jest to poprawny kod. Po uruchomieniu klasy `C` na ekranie pojawi się `AXX`
c) Jest to poprawny kod. Po uruchomieniu klasy `C` na ekranie pojawi się `BXX`
d) Jest to poprawny kod. Po uruchomieniu klasy `C` na ekranie pojawi się `ABI`

3.
Co można powiedzieć o poniższym kodzie?

```
interface I {
    void m(I i);                // 1
}
class C{
    void k(I i){ }
    static void l(I i){ }
    public static void main(String[] args) {
        I i = (new C())::k; // 2
        i.m(C::l);              // 3
    }
}
```

a) Jest to poprawny kod. Uruchomienie klasy `C` przebiegnie bez przeszkód
b) Jest to niepoprawny kod. Błąd występuje w linii `1`
c) Jest to niepoprawny kod. Błąd występuje w linii `2`
d) Jest to niepoprawny kod. Błąd występuje w linii `3`

4.
Co można powiedzieć o poniższym kodzie?

```
class C {
    public static void main(String[] args) {
        int a = 10;
        int b = 0;
        try {
            System.out.println(a / b);
        } catch (RuntimeException e) {
            System.out.print(e+ " > ");
        }
        System.out.println("The end");
    }
}
```

a) Jest to niepoprawny kod. Źle zadeklarowano przechwytywanie wyjątku
b) Kod skompiluje się poprawnie, a po uruchomienu klasy `C` na ekranie pojawi się napis
`The end`
c) Kod skompiluje się poprawnie, a po uruchomienu klasy `C` na ekranie pojawi się napis
`java.lang.ArithmeticException: / by zero > The end`
d) Kod skompiluje się poprawnie, a po uruchomienu klasy `C` na ekranie pojawi się napis
`java.lang.RuntimeException: / by zero > The end`

5.
Co można powiedzieć o poniższym kodzie?

```
class A<U> {
    U a;
}

class C {
    static <U extends Number> A<U> m() {
        return new A<>();
    }

    public static void main(String[] args) {
        var a1 = C.<Double>m();
         a1.a = 20;
    }
}
```

a) Kod ten zawiera błąd w metodzie `m()`
b) Kod ten zawiera błąd w metodzie `main()`
c) Kod skompiluje się poprawnie, ale uruchomienie klasy `C` zakończy się wyrzuceniem wyjątku
d) Kod skompiluje się poprawnie, a uruchomienie klasy `C` powiedzie się

6.
W której linii poniższego kodu należy wstawić deklarację

```
String name = "";
```

aby program skompilował się oraz uruchomił poprawnie (dla przejrzystości pominięto wymagane importy)?

```
public class A {
        // 1
    void start() {
        // 2
        IntStream.rangeClosed(1, 2)
                .forEach(i -> {
                    name +=Integer.toString(i);
                    System.out.println(name);});
    }
    public static void main(String[] args) {
        // 3
      new A().start();
    }
}
```

a) w linii 1
b) w linii 2
c) w linii 3
d) w żadnej z powyższych, gdyż kod ten zawiera błędy

7.
Jaki będzie wynik uruchomienia klasy `A` (dla przejrzystości pominięto wymagane importy)?

```
public class A {
    void start() {
        AtomicReference<String> name = new AtomicReference<>("");
        IntStream.rangeClosed(1,2)
                .forEach(i ->
                    new Thread(() -> {
                        System.out.print(name.updateAndGet(
                                n -> n + Integer.toString(i)));
                    }).start());
    }
    public static void main(String[] args) {
        new A().start();
    }
}
```

a) Na ekranie zostanie wypisane `2 21` lub `1 12`
b) Na ekranie zostanie wypisane `2 21` lub `21 2` lub `12 1` lub `1 12`
c) Kompilacja klasy `A` zakończy się błędem z uwagi na złe miejsce zadeklarowania `name`
d) Uruchomienie klasy `A` zakończy się wyrzuceniem wyjątku.

8.
Co moża powiedzieć o poniższym kodzie?

```
@FunctionalInterface
interface I1 {
    String m(String string);
    default String n(){ return "I1"; }
}
@FunctionalInterface
interface I2 {
    String m(String string);
    default String n() { return "I2"; }
}
@FunctionalInterface
interface I3 extends I1, I2 {
    // 1
}
```

a) Jest to poprawny kod. Klasa implementująca interfejs `I3` musi dostarczyć ciała dla metody `m()`
b) Jest to poprawny kod. W klasie z tego samego pakietu co interfejsy będzie można napisać
```
String s = ((I2) I3).n();
```
c) Jest to niepoprawny kod. Intefejsy `I1` oraz `I2` mają po dwie zadeklarowane metody, a przecież intefejsy funkcjonalne mogą posiadać tylko jedną zadeklarowaną metody
d) Jest to niepoprawny kod. Będzie poprawny po wstawieniu w linii `1` deklaracji
```
default String n() { return "I3"; }
```

9.
Co można powiedzieć o poniższym kodzie?

```
public class A {
    private String name = null;

    public A(String name) { this.name = name; }

    public final String getName() { return name; }

    public static void main(String[] args) {
        String s = new String("A");
        s = new A("B").getName();
        s = new String("C");
    }
}
```

a) Kod ten skompiluje się poprawnie. Jednak próba uruchomienia klasy `A` zakończy się wyrzuceniem wyjątku
b) Kod ten skompiluje się poprawnie. Próba uruchomienia klasy `A` powiedzie się
c) Jest to niepoprawny kod. Konstruktor klasy `A` ma zły modyfikator dostępu
d) Jest to niepoprawny kod. Wartość zwrócona z metody finalnej zostaje nadpisana

10.
Co można powiedzieć o poniższym kodzie?

```
public class A {
 public static <U> U m(Class<U> type, String s){
   U out=null;
   try {
    if(type==Integer.class) out = (U) Integer.valueOf(Integer.parseInt(s));
    if(type==Double.class) out = (U) Double.valueOf(Double.parseDouble(s));
   } catch (NumberFormatException e){ out = null; }
   return out;
  }
 public static void main(String[] args) {
   System.out.println(A.<Double>m(Double.class,"10.2")); // 1
   System.out.println(A.m(Double.class,"10.2"));         // 2
   int i = A.m(Integer.class,"10.2");
 }
}
```

a) Jest to poprawny kod. Po uruchomieniu klasy `A` program zakończy się bez błędu
b) Po poprawnej kompilacji i uruchomieniu klasy `A` wyrzucony zostanie wyjątek
c) Jest to niepoprawny kod. Błąd występuje w linii `1`
d) Jest to niepoprawny kod. Błąd występuje w linii `2`

11.
Co można powiedzieć o poniższym kodzie?

```
interface I { void m(int i); }
interface J { void m(double d); }

public class A {
    void n(I i)  {  i.m(0);}
    void n(J j)  {  j.m(0.0);}

    public static void main(String[] args) {
        A a = new A();
        a.n((I) x -> { });     // 1
        a.n((int x) -> { });   // 2
        a.n(x -> { });         // 3
    }
}
```

a) Jest to niepoprawny kod. Błąd występuje w linii `1`
b) Jest to niepoprawny kod. Błąd występuje w linii `2`
c) Jest to niepoprawny kod. Błąd występuje w linii `3`
d) Jest to poprawny kod. Jego kompilacja i uruchomienie klasy `A` powiedzie się

12.
Co można powiedzieć o poniższym kodzie?

```
class EA extends Exception{}
class EAA extends EA{}

public class A {
    public static void main(String[] args) {
        try {
            throw new EAA();
        } catch (EA e) { System.out.println("1:"+e); }
          catch (Exception e) { System.out.println("2:"+e); }
    }
}
```

a) Jego kompilacja zakończy się błędem.
b) Po poprawnej kompilacji i uruchomieniu klasy `A` na ekranie pojawi się `1:ex.EAA`
c) Po poprawnej kompilacji i uruchomieniu klasy `A` na ekranie pojawi się `1:ex.EA`
d) Po poprawnej kompilacji i uruchomieniu klasy `A` na ekranie pojawi się `2:ex.EAA`

13.
Co można powiedzieć o poniższym kodzie?

```
interface I{ default void m(){} }          // 1
record R1(int x, int y) implements I{ } // 2

public class A implements I{
    public static void main(String[] args) {
        new R1(1, 2).m();                  // 3
    }
}
```

a) Jest to niepoprawny kod. Błąd występuje w linii 1
b) Jest to niepoprawny kod. Błąd występuje w linii 2
c) Jest to niepoprawny kod. Błąd występuje w linii 3
d) Jest to poprawny kod.

14.
Co można powiedzieć o poniższym kodzie?

```
interface I{ static void m(){} }      // 1
record R1(int x, int y) extends A{ } // 2

public class A implements I{
    public static void main(String[] args) {
        new A().m();                  // 3
    }
}
```

a) Jest to niepoprawny kod. Błąd występuje w linii 1
b) Jest to niepoprawny kod. Błędy występują w liniach 1 i 3
c) Jest to niepoprawny kod. Błędy występują w liniach 2 i 3
d) Jest to poprawny kod.

15.
Jeśli powstać ma aplikacja napisana z wykorzystaniem JavaFX, to które z wymienionych niżej warunków najlepiej oddają wymagania co do zawartości module-info?

```
module ... {
    requires javafx.controls; // 1
    requires javafx.fxml;     // 2

    opens ... to javafx.fxml; // 3
    exports ...;              // 4
}
```

a) w module-info muszą wystąpić wszystkie linie
b) w module-info muszą wystąpić będą linie `1`, `4`
c) w module-info muszą wystąpić będą linie `1`, `2`, `3`
d) w module-info musi wystąpić jedynie linia `1`

16.
Która z metod pozwala sprawdzić, czy `Selector` ma jakiś `Channel` gotowy do operacji I/O w Java NIO?

a) `select()`
b) `isReady()`
c) `poll()`
d) `accept()`

17.
Która z metod służy do wiązania kanału z selektorem w Java NIO?

a) `channel.register(selector, ops)`
b) `selector.register(channel, ops)`
c) `channel.attach(selector, ops)`
d) `selector.attach(channel, ops)`

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

```
interface I {
    static void exclamate(String in) {
        System.out.print(in.charAt(in.length())+"! ");
    }
}
public class A {
    public static void main(String[] args) {
        Arrays.asList("X", "Y", "Z")
                .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: `X! Y! Z!`
d) Jest to poprawny kod. Jednak po uruchomieniu klasy `A` wyrzucony zostanie wyjątek

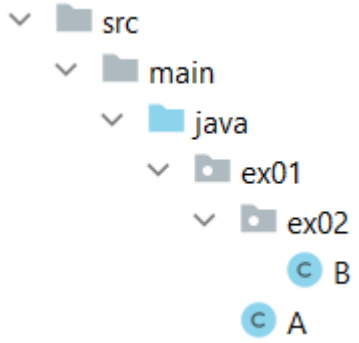
19.
Co można powiedzieć o poniższej konstrukcji (gdzie `xxx` i `yyy` to miejsce na odpowiedni kod)?

```
try(xxx){
     yyy
}
```

a) taka konstrukcja jest niepoprawna (brakuje bloku `catch`)
b) taka konstrukcja jest niepoprawna (nie można po `try` umieszczać kodu w nawiasach okrągłych)
c) to poprawna konstrukcja (w miejscu `xxx` można jedynie deklarować i inicjalizować obiekty klas implementujących interfejs `java.lang.AutoCloseable`, zaś w miejscu `yyy` można tych obiektów użyć)
d) to poprawna konstrukcja (w miejscu `xxx` można wstawić dowolny kod, który zainicjalizuje zmienne wykorzystywane w miejscu `yyy`)

20. Niech źródła klas `A` oraz `B` o podanym kodzie będą umieszczone w katalogach odpowiadających ich pakietom, w projekcie mavenowym o strukturze katalogów jak na rysunku poniżej.

A.java	B.java	Struktura katalogów

`package ex01;`
 	
`public class A {`
`}`
	`package ex01.ex02;`
	`import ex01.A;`
	
	`public class B {`
	`    A a = new A();`
	`}`
		

Jaki będzie efekt wykonania poniższych komend z poziomu katalogu wskazanego przed znakiem zachęty?

```
main> javac .\java\ex01\*.java
main> javac .\java\ex01\ex02\*.java
```

a) Obie komendy wykonają się poprawnie. Pliki `A.class` i `B.class` pojawią się odpowiedno w katalogach, w których mieściły się `A.java` i `B.java`
b) Pierwsza komenda wykona się poprawnie. Uruchomienie drugiej spowoduje wypisanie informacji o błędzie:
`error: cannot find symbol`
c) Uruchomienie pierwszej, jaki i drugiej komendy spowoduje wypisanie informacji o błędzie:
`error: Invalid filename`
d) Obie komendy wykonają się poprawnie. Pliki `A.class` i `B.class` pojawią się w bieżącym katalogu

21.
Jaki będzie wynik uruchomienia klasy o kodzie jak niżej?

```
public class A {
    static Object o = new Object();
    static boolean empty = true;

    public void cons()  {
        synchronized (o) {
            try{ while (empty) wait(); } catch (InterruptedException e) {}
            empty = true; notify(); }
    }
    public void prod() {
        synchronized (o) {
            try{ while (!empty) wait();} catch (InterruptedException e) {}
            empty = false; notify(); }
    }
    public static void main(String[] args) {
        A a = new A();
        new Thread(a::cons).start(); // 1
        new Thread(a::prod).start(); // 2
    }
}
```

a) Wyrzucony zostanie wyjątek `java.lang.IllegalMonitorStateException`
b) Zaczną działać dwa niekończące się wątki
c) Wątek z linii `2` zakończy swoje działanie, wątek z linii `1` będzie działał bez końca
d) Nie będzie żadnych wyników, bo kod ten nie skompiluje się poprawnie

22.
Co można powiedzieć o poniższym kodzie?

```
public class A {
    public static void main(String[] args) {
        double d = 10.2;
        float f = (float) (d/2.0);   // 1
        int i = (int)f--;            // 2
        System.out.println(i);
    }
}
```

a) Jest to niepoprawny kod. Błąd występuje w linii `1`
b) Jest to niepoprawny kod. Błąd występuje w linii `2`
c) Jest to poprawny kod. Po uruchomieniu klasy `A` na ekranie pojawi się `4`
d) Jest to poprawny kod. Po uruchomieniu klasy `A` na ekranie pojawi się `5`

23.
Co można powiedzieć o próbie uruchomienia poniższego kodu przy włączonych asercjach?

```
public class A {
    public static void main(String [] args) {
        int i = 1;
        assert (i > 1) ? "failed" : "passed" ;
        System.out.println("the end");
    }
}
```

a) Nie można uruchomić klasy A, bo jej kod zawiera błędy
b) Po uruchomieniu klasy `A` na ekranie pojawi się napis the end
c) Po uruchomieniu klasy `A` wyrzucony zostanie wyjątek `java.lang.AssertionError`, a na ekranie pojawi się napis `the end`
d) Po uruchomieniu klasy `A` wyrzucony zostanie wyjątek `java.lang.AssertionError` z wiadomością `failed`

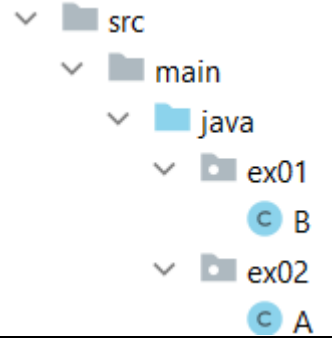
24.
Co można powiedzieć o poniższym kodzie?

```
public class A {
    public static void main(String [] args) {
        Object[] ao = new Object[]{"A", 1, 2.2}; // 1
        for(var o : ao) {                        // 2
            System.out.print(o);                 // 3
        }
    }
}
```

a) Jego kompilacja zakończy się błędem wykazanym w linii `1`
b) Jego kompilacja zakończy się błędem wykazanym w linii `2`
c) Jego kompilacja zakończy się błędem wykazanym w linii `3`
d) Po kompilacji i uruchomieniu na ekranie pojawi się `A12.2`

25.
Co można powiedzieć o poniższym kodzie?

A.java	B.java	Struktura katalogów

`package ex02;`
`import ex01.B;`
`public class A extends B {`
`}`
	`package ex01;`
	`public class B{`
	`  public B() { this(0); }`
	`  B(int x){}`
	`}`
		

a) Jego kompilacja nie powiedzie się. Błąd występuje w klasie `B`
b) Jego kompilacja nie powiedzie się. Błąd występuje w klasie `A`
c) Jest to poprawny kod. Obiekty klas `A` i `B` mogą być tworzone w klasach z pakietów innych niż `ex01` i `ex02`
d) Jest to poprawny kod. Ale obiektu klasy `A` nie da się utworzyć w klasach z pakietów innych niż `ex01` i `ex02`

26.
Co można powiedzieć o poniższym kodzie?

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

a) Jest to niepoprawny kod. Jego kompilacja zakończy się błędem
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 `2 2`

27.
Co można powiedzieć o poniższym kodzie (zakładamy, że zadeklarowano wszystkie wymagane importy)?

```
package ex01;
import ...
public class A  {
    public interface I extends Remote {
         void show(String s) throws RemoteException; }
    public static void main(String[] args) throws RemoteException {
        Registry r = LocateRegistry.createRegistry(1000);
        r.rebind("A", UnicastRemoteObject.exportObject(
             (I) s -> System.out.println(s),0));   }
}
```

```
package ex02;
import ...
public class B {
    public static void main(String[] args)
                      throws RemoteException, NotBoundException{
        Registry r = LocateRegistry.getRegistry(1000);
        r.lookup("A").show("hello");  }
}
```

a) Po uruchomieniu klasy `A`, a potem klasy `B` w osobnych konsolach, na konsoli klasy `A` pojawi się `hello`
b) Kompilacja tego kodu zakończy się zgłoszeniem błędu w klasie `A`
c) Kompilacja tego kodu zakończy się zgłoszeniem błędu w klasie `B`
d) Kompilacja obu klas się uda, jednak tylko klasę `A` da się uruchomić poprawnie

28.
Co można powiedzieć o poniższym kodzie (zakładamy, że zadeklarowano wszystkie wymagane importy)?

```
public class A {
    public static void main(String[] args)    {
        try( Socket s = new Socket("localhost", 5000)){ // 1
            s.setDoOutput(true);                        // 2
            OutputStream out = s.getOutputStream();
        } catch (Exception e) {
        }
    }
}
```

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, jednak podczas uruchomienia zgłoszony zostanie wyjątek
d) Kod skompiluje się i uruchomi bez błędu

29.
Co można powiedzieć o poniższym kodzie?

```
public class A {
    private static int i=1;
    private int id = ++i;
    private A(){ id--; }

    public void m(A a) {
        System.out.print("1");
        if(id>2) return;
        a.m(new A());
        System.out.print("0");
    }

    public static void main(String[] args) {
        new A().m(new A());
    }
}
```

a) Po jego uruchomieniu zgłoszony zostanie wyjątek `StackOverflowError`
b) Jego kompilacja nie powiedzie się (nie można utworzyć obiektu klasy `A` w metodzie `main`)
c) Po jego uruchomieniu na konsoli zostanie wypisane: `11100`
d) Po jego uruchomieniu na konsoli zostanie wypisane: `1111000`

30.
Co można powiedzieć o poniższym kodzie?

```
public class A {
    private void m(){
        private class B{     // 1
            void m(){}       // 2
        }
        new B().m();         // 3
    }

    public static void main(String[] args) {
        new A().m();
    }
}
```

a) Jest to niepoprawny kod. Jego kompilacja zakończy się błędem w linii `1`
b) Jest to niepoprawny kod. Jego kompilacja zakończy się błędem w linii `2`
c) Jest to niepoprawny kod. Jego kompilacja zakończy się błędem w linii `3`
d) Jest to poprawny kod. Jego kompilacja i uruchomienie powiedzie się