1.
Załóżmy, że poniższy kod

```
package examples;
public class A {

    public static void main(String[] args) {
        System.out.println("OK");
    }
}
```

umieszczono w pliku `A.java` w następującej strukturze katalogów

```
.
└───src
    └───examples
            A.java
```

Które z poleceń spowodują wypisanie na ekranie napisu `OK`?

a) `java .\src\examples\A.java`
b) `javac -d .\bin .\src\examples\A.java && java -p .\bin\ examples.A`
c) `javac .\src\examples\*.java && java -cp .\bin\ examples.A`
d) `javac .\src\examples\A.java`

2.

Załóżmy, że plik `m.txt` ma zawartość jak niżej, zaś plik `A.class` to wynik kompilacji poniższego kodu

`Manifest-Version: 1.0`
`Main-Class: examples.A`
`Created-By: Anonymous`
	`package examples;`
	`public class A {`
	
	`  public static void main(String[] args) {`
	`  System.out.println("OK");`
	`  }`
	`}`

Załóżmy, że pliki te umieszczono w następującej strukturze katalogów:

```
.
│   m.txt
│
└───bin
    └───examples
            A.class
```

Które z poleceń pozwolą wygenerować wykonywalny jar?

a) `jar cfm a.jar m.txt -C .\bin .`
b) `jar cmf m.txt a.jar .\bin .`
c) `jar cfM a.jar m.txt -C .\bin .`
d) Żadne, bo zawartość pliku `m.txt` nie jest poprawnym manifestem

3.
Które ze zdań są prawdziwe?

a) Wykonywalny jar musi zawierać tylko jedną klasę z metodą `main()`
b) Wykonywalny jar to archiwum zip posiadające manifest umieszczony w podkatalogu `META` tego archiwum
c) Wykonywalny jar to archiwum zip, posiadające manifest umieszczony w katalogu głównym tego archiwum
d) Wykonywalny jar nie może zawierać pliku manifestu o dowolnej nazwie

4.
Do czego służy opcja `-d` kompilatora javac?

a) Pozwala wyświetlić szczegółowe informacje o użyciu lub nadpisaniu elementów klasy uznanych za przestarzałe (`deprecated`)
b) Pozwala wskazać ścieżkę modułów
c) Pozwala wskazać miejsce, do którego mają trafić wyniki kompilacji
d) Nie stosuje się takiej opcji

5.

Załóżmy, że istnieje projekt o następującej strukturze, zawierający pliki `A.java` oraz `module-info.java` o zawartości jak niżej

test01
JRE System Library [JavaSE-11]
src
(default package)
A.java
module-info.java
mymodule
	`public class A {`
	`}`
		`module mymodule {`
		`  exports default; // 1`
		`}`

Przy takich warunkach które ze stwierdzeń jest prawdziwe?

a) Jest to poprawnie skonfigurowany modułowy projekt biblioteki, jednak po kompilacji i wyeksportowaniu do modułowego pliku jar nie będzie można z niego skorzystać w innych programach

b) Jest to poprawnie skonfigurowany modułowy projekt biblioteki, a po kompilacji i wyeksportowaniu do modułowego pliku jar będzie można z niego korzystać w innych programach

c) Aby to był poprawnie skonfigurowany modułowy projekt biblioteki w linii 1 powinno być: `exports A;`

d) Jest to niepoprawnie skonfigurowany modułowy projekt biblioteki, niezależnie od tego, co wpisane zostanie w linii 1

6.
Co można powiedzieć o poniższym kodzie?

```
interface I{
    <T> void m(T a);
}
public class A implements I{
    void m(Integer  a) {}
}
```

a) Jest to poprawny kod
b) Jest to niepoprawny kod (źle sparametryzowano typem metodę `m` w interfejsie `I`)
c) Jest to niepoprawny kod (źle zaimplementowano metodę `m` interfejsu `I` w klasie `A`)
d) Jest to niepoprawny kod (nie można umieszczać deklaracji interfejsu oraz klasy w tym samym źródle kodu)

7.
Co można powiedzieć o poniższym kodzie?

```
interface I<T>{
     void m(T a);
}
public class A implements I<String>, I<Long>{
    @Override
    public void m(String a) {
    }
    public void m(Integer a) {
    }
}
```

a) Jest to niepoprawny kod (źle zadeklarowano interfejs `I`)
b) Jest to niepoprawny kod (źle zadeklarowano typ argumentu w metodzie `m(Integer a)` zaimplementowanej w klasie `A`)
c) Jest to niepoprawny kod (aby kod był poprawny wystarczy usunąć `I<Long>` z listy interfejsów implementowanych przez klasę `A`)
d) Jest to poprawny kod

8.
Co można powiedzieć o poniższym kodzie?

```
public class A {
    public enum B {
        enum C {
            A, B, C;
        }
    }
}
```

a) Jest to niepoprawny kod (nie można zagnieżdżać publicznego typu wyliczeniowego w klasie publicznej)
b) Jest to niepoprawny kod (nie można zagnieżdżać jednego typu wyliczeniowego w drugim)
c) Jest to niepoprawny kod (nie można nadawać takich samych nazw elementom typu wyliczeniowego jak nazwy klas)
d) Jest to poprawny kod

9.
Co można powiedzieć o poniższym kodzie
(przy założeniu, że wszystkie wymagane importy zostały prawidłowo zadeklarowane)?

```
public class A {
   void m() throws Exception {
       throw new Exception();
   }

   public void n() throws Exception {
       try {
           m();
       } catch(Exception e) {
           }
       }
}
```

a) Jest to poprawny kod
b) Jest to niepoprawny kod (źle utworzono wyjątek w metodzie `m()`)
c) Jest to niepoprawny kod (metoda `n()` nie powinna być zadeklarowana z klauzulą `throws`)
d) Jest to niepoprawny kod (metoda `n()` używa metody `m()` choć nie ma do niej dostępu)

10.
Co można powiedzieć o poniższym kodzie
(przy założeniu, że wszystkie wymagane importy zostały prawidłowo zadeklarowane)?

```
public class A implements Comparable<A> {
    static int i;
    private int id;

    public A() { id = i++;}
    @Override
    public int compareTo(A a) {
        return a.id - id;
    }

    public static void main(String[] args) {
        A a1 = new A(), a2 = new A(), a3 = new A();
        var ts = new TreeSet<A>();
        ts.add(a2); ts.add(a1); ts.add(a3);
        for(A a : ts)
          System.out.print(a.id+" ");
    }
}
```

a) Jest to poprawny kod. Po jego wykonaniu na ekranie pojawi się: `0 1 2`
b) Jest to poprawny kod. Po jego wykonaniu na ekranie pojawi się: `2 1 0`
c) Jest to poprawny kod. Po jego wykonaniu na ekranie pojawi się: `1 0 2`
d) Jest to niepoprawny kod

11.
Co można powiedzieć o poniższym kodzie
(przy założeniu, że wszystkie importy zostały prawidłowo zadeklarowane)?

```
public class A {

    public static void main(String[] args) {
        Thread t = new Thread(new Runnable() {
            public synchronized void run() {
                try {
                    if (Thread.interrupted()) {
                        System.out.print("2");
                    }
                    wait();
                } catch (InterruptedException e) {
                    System.out.print("1");
                }
            }
        });
        t.start();
        t.interrupt();
    }
}
```

a) Jest to poprawny kod. Po każdym jego uruchomieniu na ekranie pojawi się `1` (program skończy się)
b) Jest to poprawny kod. Po każdym jego uruchomieniu na ekranie pojawi się `2` (program nie skończy się)
c) Jest to niepoprawny kod. Po jego uruchomieniu na ekranie pojawi się `1` albo `2` (program skończy się albo nie)
d) Jest to niepoprawny kod

12.
Co można powiedzieć o poniższym kodzie
(przy założeniu, że wszystkie importy zostały prawidłowo zadeklarowane)?

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

        t.start();
        t.interrupt();
    }
}
```

a) Jest to poprawny kod. Po każdym jego uruchomieniu na ekranie pojawi się `1` (program nie skończy się)
b) Jest to poprawny kod. Po każdym jego uruchomieniu na ekranie pojawi się `12` (program skończy się)
c) Jest to poprawny kod. Po jego uruchomieniu na ekranie pojawi się `1` lub `12` (program nie skończy się lub skończy się)
d) Jest to niepoprawny kod

13.
Co można powiedzieć o poniższym kodzie
(przy założeniu, że wszystkie importy zostały prawidłowo zadeklarowane)?

```
public class A {
    public static Integer lock, monitor;

    public static void main(String[] args) {
        Thread t1 = new Thread(() -> {
         synchronized (monitor){
             try { monitor.wait();
            } catch (InterruptedException e) {}}
         });

        Thread t2 = new Thread(() -> {
         synchronized (lock){
                   monitor.notify(); }
         });
        t1.run();
        t2.run();
    }
}
```

a) Kod skompiluje się, po uruchomieniu wyrzucony zostanie wyjątek: `IllegalMonitorStateException`
b) Kod skompiluje się, po uruchomieniu wyrzucony zostanie wyjątek: `NullPointerException`
c) Kod skompiluje się, po uruchomieniu program będzie działał w nieskończoność
d) Podczas kompilacji kodu pojawią się błędy

14.
Co można powiedzieć o poniższym kodzie
(przy założeniu, że wszystkie importy zostały prawidłowo zadeklarowane)?

```
public class A {

    public static void main(String[] args)    {

        try( Socket s = new Socket("localhost", 2000)){ // 1
          s.setDoOutput(false);                          // 2
          OutputStream out = s.getOutputStream();
        } catch (Exception e) {
        }
    }
}
```

a) Jest to niepoprawny kod. Błąd występuje w linii `1`
b) Jest to niepoprawny kod. W linii `2` powinno być napisane: `s.setDoOutput(true);`
c) Jest to niepoprawny kod. Linia `2` jest zbędna
d) Kod skompiluje się bez błędu, choć gniazda jawnie nie zamknięto i użyto w bloku `catch` bazowego typu wyjątku

15.
Co można powiedzieć o poniższym kodzie kodzie
(przy założeniu, że wszystkie importy zostały prawidłowo zadeklarowane,
a jego uruchomienie odbywa się z opcją `-Djava.security.manager`
przekazaną wirtualnej maszynie)?

```
public class A {
    public static void main(String[] args)    {
        SecurityManager sm = new SecurityManager();
        System.setSecurityManager(sm);
    }
}
```

a) Jego kompilacja się nie powiedzie. Zapomniano w nim obsłużyć wyjątki
b) Jego kompilacja się nie powiedzie. Zapomniano w nim zadeklarować zezwolenia
c) Jest to poprawny kod. Jego uruchomienie zakończy się bez błędów
d) Jest to poprawny kod. Jednak próba jego uruchomienia zakończy się wyrzuceniem wyjątku `AccessControlException`

16.
Co można powiedzieć o poniższych kodach (przy założeniu, że wszystkie importy zostały prawidłowo zadeklarowane, `module-info.java` jest poprawny i nie użyto żadnych parametrów JVM)?

```java
interface I extends Remote {
  public default void m() throws RemoteException {
     System.out.println( this.
             getClass().getTypeName()+".m()"); }
  }
public class A extends UnicastRemoteObject
  implements I {
  private static final long serialVersionUID = 1L;

  protected A() throws RemoteException {
    super(); }

  public static void main(String[] args)
   throws RemoteException, AlreadyBoundException {
    LocateRegistry.createRegistry(3000).
       bind("A", new A());
  }
}
```

```java
public class B {
  public static void main(String[] args)
  throws RemoteException, NotBoundException {

  Registry reg =
           LocateRegistry.getRegistry(3000);
  ((I) reg.lookup("A")).m();
  }
}
```

a) Kod po lewej nie skompiluje się (nie wolno było użyć `this` w domyślnej metodzie interfejsu `I`)
b) Kod po lewej nie skompiluje się (w klasie `A` brak zaimplementowanej metody interfejsu `I`)
c) Po kompilacji i uruchomieniu kolejno klas `A` i `B` w konsoli klasy `A` pojawi się `example.A.m()`
d) Po kompilacji i uruchomieniu kolejno klas `A` i `B` w konsoli klasy `B` pojawi się `example.A.m()`

17.
Co można powiedzieć o o poniższym kodzie kodzie?

```
interface I {
    void m(I i);   // 1
}

public class A implements I{

    public static void main(String[] args) {
        A a = new A();
        a.m((var i) ->{ System.out.println("A.m()");}); // 2
    }

    @Override
    public void m(I i) {
       i.m(null);
    }
}
```

a) Jego kompilacja się nie powiedzie. Typ atrybutu metody interfejsu nie może być tym interfejsem (linia `1`)
b) Jego kompilacja się nie powiedzie. W wyrażeniu lambda źle zastosowano słowo `var` (linia `2`)
c) Po kompilacji i uruchomieniu klasy A zgłoszony zostanie wyjątek `StackOverflowError`
d) Po kompilacji i uruchomieniu klasy A na konsoli zostanie wypisane `A.m()`

18.
Co można powiedzieć o o poniższym kodzie 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>3) return;
      a.m(new A());
      System.out.print("0");
    }

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

a) Po jego uruchomieniu na konsoli zostanie wypisane: `111110000`
b) Po jego uruchomieniu na konsoli zostanie wypisane: `11111100000`
c) Po jego uruchomieniu zgłoszony zostanie wyjątek `StackOverflowError`
d) Jego kompilacja nie powiedzie się (nie można utworzyć obiektu klasy `A` w metodzie `main`)

19.
Co można powiedzieć o poniższym kodzie?

```
public class A {
     public static void m(Object ... o) {
     }

    public static void main(String[] args) {
        m("A", 12, new A());
    }
}
```

a) Ten kod skompiluje się i uruchomi poprawnie
b) Jego kompilacja nie powiedzie się (niepoprawna deklaracja metody `m()`)
c) Jego kompilacja nie powiedzie się (niepoprawne użycie metody `m()`)
d) Ten kod skompiluje się poprawnie, ale po jego uruchomieniu wyrzucony zostanie wyjątek

20.
Co można powiedzieć o poniższym kodzie
(przy założeniu, że wszystkie wymagane importy zostały prawidłowo zadeklarowane)?

```
public class A {

    public static void main(String[] args) {
        List<Object> l = new LinkedList<>(Arrays.asList(1,2,3)); // 1
        Iterator<Object> i = l.iterator(l.size());  // 2
        while (i.hasPrevious()) {
            System.out.print(i.previous());
        }
    }
}
```

a) Jest to poprawny kod
b) Jest to niepoprawny kod (zły atrybut konstruktora listy powiązanej w linii `1`)
c) Jest to niepoprawny kod (złe parametryzowanie typem iteratora w linii `2`)
d) Jest to niepoprawny kod (użyto niewłaściwy iterator)