1. Co można powiedzieć o poniższym kodzie (zakładając, że znajduje się on w jednym pliku `A.java`)?

```
public class A {
     public int i;
     {
        i++;
         System.out.print(i);
    }
    public static void main(String[] args) {
        A a1 = new A(), a2 = new A();
    }
}
```

a) Jego kompilacja nie powiedzie się (kod wykonywalny może pojawić się jedynie w ciele metody)
b) Jego kompilacja nie powiedzie się (kod wykonywalny można umieścić w bloku poza ciałem metody, ale blok ten musi być blokiem statycznym)
c) Po kompilacji i wykonaniu metody `main` na ekranie pojawi się: `1`
d) Po kompilacji i wykonaniu metody `main` na ekranie pojawi się: `11`

2. Co można powiedzieć o poniższym kodzie (zakładając, że znajduje się on w jednym pliku `A.java`)?

```
public class A {
    public static int i;
    static {
       i++;
        System.out.print(i);
    }
    public static void main(String[] args) {
       A a1 = new A(), a2 = new A();
    }
}
```

a) Jego kompilacja nie powiedzie się (kod wykonywalny może pojawić się jedynie w ciele metody)
b) Po kompilacji i wykonaniu metody `main` na ekranie pojawi się: `1`
c) Po kompilacji i wykonaniu metody `main` na ekranie pojawi się: `11`
d) Po kompilacji i wykonaniu metody `main` na ekranie pojawi się: `12`

3. Co można powiedzieć o poniższym kodzie (zakładając, że znajduje się on w jednym pliku `A.java`)?

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

a) Jego kompilacja nie powiedzie się (kod wykonywalny może pojawić się jedynie w ciele metody)
b) Po kompilacji i wykonaniu metody `main` na ekranie pojawi się: `0`
c) Po kompilacji i wykonaniu metody `main` na ekranie pojawi się: `01`
d) Po kompilacji i wykonaniu metody `main` na ekranie pojawi się: `12`

4. Co można powiedzieć o poniższym kodzie (zakładając, że znajduje się on w jednym pliku `A.java`)?

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

a) Jego kompilacja nie powiedzie się (kod wykonywalny może pojawić się jedynie w ciele metody)
b) Jego kompilacja nie powiedzie się (kod wykonywalny można umieścić w bloku, ale blok ten musi być blokiem statycznym)
c) Po kompilacji i wykonaniu metody `main` na ekranie pojawi się: `1`
d) Po kompilacji i wykonaniu metody `main` na ekranie pojawi się: `2`

5. Niech klasy `A` i `B` będą zdefiniowane w jednym pliku jak niżej. Co można o nich powiedzieć?

```
public abstract class A {
  A(){ m();}                        // 1
  public abstract void m();
  public static void main(String... args){
      A a = new B();                // 2
  }
}

class B extends A{
  void m(){                   // 3
     System.out.println("B.m()");}
}
```

a) Wystąpi błąd kompilacji w linii `1`
b) Wystąpi błąd kompilacji w linii `2`
c) Wystąpi błąd kompilacji w linii `3`
d) Po kompilacji i uruchomieniu metody `main` klasy `A` na ekranie pojawi się `B.m()`

6. Co można powiedzieć o źródłach kodu interfejsu `I` oraz klasy `A` (zapisanych w różnych plikach)?

```
package pakiecik;

public interface I {
  void m(int i);
}
```

```
import pakiecik.I;

public class A implements I {
    public static void main(String[] args) {
     A.m(1);                                  // 1
    }
    public static void m(int i) {     // 2
    }
}
```

a) Wystąpi błąd kompilacji w linii `1` (wywołanie metody instancyjnej bez utworzenia obiektu)
b) Wystąpi błąd kompilacji w linii `2` (niepotrzebne słówko `static`)
c) Po poprawnej kompilacji metoda `main` zadziała poprawnie
d) Po poprawnej kompilacji próba uruchomienia metody `main` zakończy się wyrzuceniem wyjątku

7. Co można powiedzieć o poniższym kodzie?

```
Integer i = 65536;  // 1
Long l = (long) 0;      // 2
Float f = new Float("20"); // 3
Double d = 40;    // 4
```

a) Niepoprawna deklaracja w linii 1
b) Niepoprawna deklaracja w linii 2
c) Niepoprawna deklaracja w linii 3
d) Niepoprawna deklaracja w linii 4

8. Co można powiedzieć o źródłach kodu klasy `B` oraz klasy `A` (zapisanych w różnych plikach)?

```
package pakiecik;

public class B {
    public static void main(String[] args) {
      System.out.println((new A()).a);
    }
}
```

```
public class A{
    public Integer a;
}
```

a) Wystąpi błąd podczas kompilacji klasy `B`
b) Wystąpi błąd podczas kompilacji klasy `A`
c) Kompilacja obu klas powiedzie się, jednak uruchomienie metody `main` zakończy się wyrzuceniem wyjątku
d) Kompilacja obu klas powiedzie się, a uruchomienie metody `main` zakończy się wypisaniem `null`

9. Co można powiedzieć o poniższym kodzie?

```
int i,j = 0;
et2: for(i =0; i<3; i++){
    et1: do
    {
        j++;
        System.out.print(i+ " " + j + ",");
        if (i%2 ==0)
            continue et2;
        else break et1;

    } while (j<3);
    j=0;
}
```

a) Jego kompilacja i uruchomienie powiedzie się. Na ekranie zostanie wypisane `0 1,1 2,2 1,`
b) Jego kompilacja i uruchomienie powiedzie się. Na ekranie zostanie wypisane `0 1,1 2,2 3,`
c) Jego kompilacja i uruchomienie powiedzie się. Na ekranie zostanie wypisane `0 1,1 2,2 4,`
d) Podczas jego kompilacji wystąpi błąd

10. Co można powiedzieć o poniższym kodzie?

```
Integer i,j;
j = new Integer(20);     // 1
i = j;
j ++;                    // 2
System.out.println(i);
```

a) Jego kompilacja zakończy się błędem w linii `1`
b) Jego kompilacja zakończy się błędem w linii `2`
c) Jego kompilacja i uruchomienie powiedzie się. Na ekranie zostanie wypisane `20`
d) Jego kompilacja i uruchomienie powiedzie się. Na ekranie zostanie wypisane `21`

11. Co można powiedzieć o poniższych kodach źródłowych klasy `A` i interfejsu `I`?

```
class A implements pakiecik.I {
    public static void main(String[] args) {
        new A().m(10);
    }

    @Override
    public void m(int i) {
        System.out.println(i);
    }
}
```

```
package pakiecik;

public interface I {
    public void m(int i)
            throws Exception;
}
```

a) Kompilacja kodów powiedzie się, a po uruchomieniu metody `main` na ekranie zostanie wypisane `10`
b) Kompilacja kodu klasy `A` nie powiedzie się (w implementacji metody `m()` brak klauzuli **`throws`**)
c) Kompilacja kodu klasy `A` nie powiedzie się (w implementacji metody `main()` brakuje bloku `try/catch`)
d) Kompilacja kodu klasy `A` nie powiedzie się (zły modyfikator dostępu w implementacji metody `m`)

12. Co można powiedzieć o poniższym kodzie źródłowym Java 1.8 (zamieszczonym w 3 różnych plikach)?

```
public interface I {
  default public void m(){
     System.out.println();
  }
}
```

```
public interface J {
   public void m();
}
```

```
public class A implements I,J{
   public static void main(String[] args){
   }
}
```

a) Jego kompilacja i uruchomienie metody `main` powiedzie się
b) Jego kompilacja zakończy się błędem (w interfejsach nie można implementować metod)
c) Jego kompilacja zakończy się błędem (w klasie `A` brak implementacji metody `m()`)
d) Jego kompilacja zakończy się błędem (klasa nie może implementować interfejsów z metodami o takich samych sygnaturach)

13. Co można powiedzieć o poniższym kodzie źródłowym Java 1.8 (zamieszczonym w 2 różnych plikach)?

```
package inny;
import pakiecik.*;

public class A implements I{
    @Override
     void m(int i){
    }
}
```

```
package pakiecik;

public interface I {
    void m(int i);
}
```

a) Jego kompilacja powiedzie się
b) Jego kompilacja zakończy się błędem (metoda `m()` w klasie `A` powinna być publiczna)
c) Jego kompilacja zakończy się błędem (błędnie zadeklarowany import w klasie `A`)
d) Jego kompilacja zakończy się błędem (interfejs `I` dostarcza metody `m()` z dostępem pakietowym, a ponieważ klasa `A` jest z innego pakietu mamy tu niezgodność)

14. Co można powiedzieć o poniższym kodzie?

```
class E extends Exception{}

public class A {
    public static void m() throws E, Exception { }

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

a) Jego kompilacja i uruchomienie metody `main` powiedzie się
b) Jego kompilacja zakończy się błędem (brak bloku `try/catch`)
c) Jego kompilacja zakończy się błędem (brak deklaracji importu klasy `Exception`)
d) Jego kompilacja zakończy się błędem (bo metoda `m()` nie wyrzuca zadeklarowanych wyjątków)

15. Co można powiedzieć o poniższym kodzie?

```
class B extends A {}

public class A {
   public static void n(A[] a){} // 1

   public static void main(String[] args) {
    B[] aB = new B[2];
    A[] aA = new A[2];
    n(aB);                    // 2
    n(aA);                    // 3
   }
}
```

a) Jego kompilacja i uruchomienie metody `main` powiedzie się
b) Jego kompilacja zakończy się błędem w linii `1`
c) Jego kompilacja zakończy się błędem w linii `2`
d) Jego kompilacja zakończy się błędem w linii `3`

16. Co można powiedzieć o poniższym kodzie (zakładając, że wykonano wszystkie konieczne importy)?

```
import java.util.*;

class B extends A {}

public class A {
   static void n(Collection<A> a){}  //1

   public static void main(String[] args) {
     Collection<B> cB = new LinkedList<B>();
     Collection<A> cA = new LinkedList<A>();
     n(cB);                              // 2
     n(cA);                              // 3
    }
}
```

a) Jego kompilacja i uruchomienie metody `main` powiedzie się
b) Jego kompilacja zakończy się błędem w linii `1`
c) Jego kompilacja zakończy się błędem w linii `2`
d) Jego kompilacja zakończy się błędem w linii `3`

17. Co można powiedzieć o poniższym kodzie?

```
public class A {
    enum E {
        RAZ(1), DWA(2);              // 1
        final int i;
        E(int i) { this.i = i; }
        E() { this.i = 0; }

    }

    public static void main(String args[]) {
        E[] aE = { E.RAZ, E.RAZ }; // 2
        for (E e : aE)
            switch (e) { case RAZ: break; case DWA: break; } // 3
    }
}
```

a) Jego kompilacja i uruchomienie powiedzie się
b) Jego kompilacja zakończy się błędem w linii 1
c) Jego kompilacja zakończy się błędem w linii 2
d) Jego kompilacja zakończy się błędem w linii 3

18. Co można powiedzieć o poniższym kodzie?

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;

public class A{

    public static void main(String[] args){
        Integer[] ti = {3,1,2};
        ArrayList<Integer> al = new ArrayList<Integer>(Arrays.asList(ti));
        al.sort((a1, a2) -> a2-a1);
        Collections.sort(al,(a1, a2) -> a1-a2);
        System.out.println(al);
    }

}
```

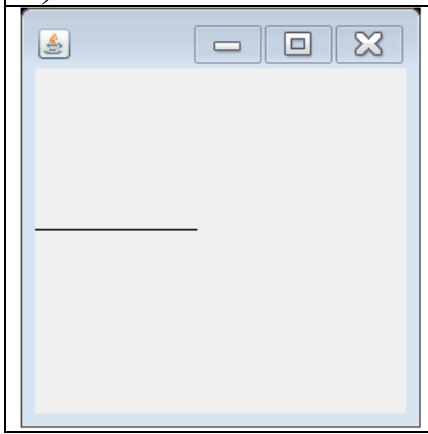
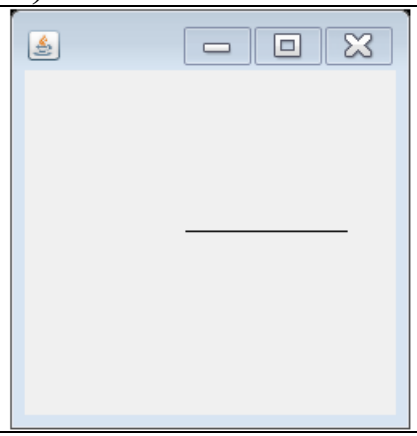
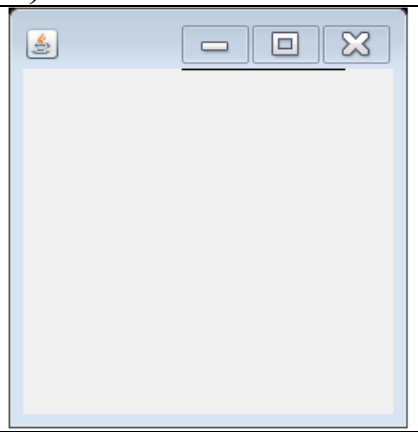
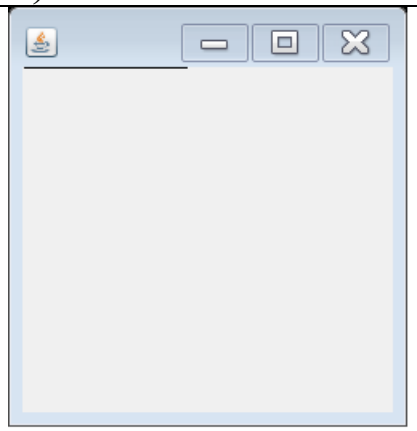
a) Jego kompilacja powiedzie się, a po uruchomieniu metody `main` na ekranie pojawi się `[1, 2, 3]`
b) Jego kompilacja powiedzie się, a po uruchomieniu metody `main` na ekranie pojawi się `[3, 2, 1]`
c) Jego kompilacja zakończy się błędem (zły parametr konstruktora `ArrayList`)
d) Jego kompilacja zakończy się błędem (złe parametry metody `sort`)

19. Jak wygląda ramka, na którą wstawiono panel o metodzie `paintComponent` jak niżej?

```
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D) g;

    AffineTransform saveAT = g2d.getTransform();
    AffineTransform t = new AffineTransform();

    t.scale(1, -1);
    t.translate(100,-100);
    g2d.setTransform(t);
    g2d.drawLine(-100, 0, 0, 0);
    g2d.setTransform(saveAT);
}
```

a)	b)	c)	d)
			

20. Co można powiedzieć o poniższym kodzie?

```
JButton btnNewButton = new JButton("<font color=red>AAA</font>");
```

a) Jest to deklaracja przycisku, na którym pojawi się napis `<font color=red>AAA</font>`
b) Jest to deklaracja przycisku, na którym pojawi się napis `AAA`
c) Jest to deklaracja przycisku, na którym pojawi się napis AAA
d) Jest to niepoprawna deklaracja przycisku

21. Co można powiedzieć o skutkach kompilacji i wykonania poniższego fragmentu kodu

```
B[] ta = {new B(),new B()};
JList list = new JList(ta);
contentPane.add(list, BorderLayout.WEST);
```

przy klasie `B` zdefiniowanej jak niżej

```
class B{
    String a ="B";
}
```

i założeniu, że wszystkie importy są wykonane oraz że miejsce wystąpienia tego fragmentu kodu jest właściwe

a) Kompilacja tego fragmentu kodu zakończy się błędem
b) Na ekranie pojawi się lista z dwoma wierszami (w każdym będzie literka `B`)
c) Na ekranie pojawi się lista z dwoma wierszami (w każdym różny ciąg znaków postaci `B@10a32dce`)
d) Na ekranie pojawi się pusta lista

22. Co można powiedzieć o poniższym kodzie?

```
class R extends Thread {
    Thread t;
    R() {
        t = new Thread(this);        // 1
        t.start(); }
    public void run() { try {
        sleep(100);
    } catch (InterruptedException e) {   }
    }
}

public class A {
    public static void main(String args[])
                throws InterruptedException { // 2
        R r = new R();
        r.join();                  // 3
    }
}
```

a) Jego uruchomienie i kompilacja przebiegną poprawnie
b) Jego kompilacja nie uda się. Błąd wystąpi w linii 1
c) Jego kompilacja nie uda się ze względu na błędną deklarację w linii 2
d) Jego kompilacja nie uda się ze względu na złe polecenie w linii 3

23. Co można powiedzieć o poniższym kodzie?

```
public class A {
    static boolean koniec = false;

    synchronized public void m(){
     while(!koniec) try { wait();  } catch (InterruptedException e) { }}

    synchronized public void n(){ notify();}

    public static void main(String args[]) {
        Thread t1,t2;
        A a = new A();
        t1 = new Thread(new Runnable(){
            public void run() {  a.m(); }});
        t2 = new Thread(new Runnable(){
            public void run() { koniec = true;  a.n(); }});
        t2.start(); t1.start();
    }}
```

a) Jego kompilacja i uruchomienie powiedzie się. Oba wątki zakończą swoje działanie
b) Jego kompilacja i uruchomienie powiedzie się. Wątek `t1` nigdy nie skończy swojego działania
c) Jego kompilacja i uruchomienie powiedzie się. Aby wątek `t1` skończył swoje działanie należy zamienić kolejność uruchamiania wątków: `t1.start(); t2.start();`
d) Jego kompilacja zakończy się błędem

24. Co można powiedzieć o poniższym kodzie?

```
class B extends A{}                    // 1
class A {}
class D extends C implements I{} // 2
class C extends A implements I{} // 3
interface I{}
```

a) Jego kompilacja powiedzie się
b) Jego kompilacja zakończy się błędem w linii 1
c) Jego kompilacja zakończy się błędem w linii 2
d) Jego kompilacja zakończy się błędem w linii 3

25. Co można powiedzieć o poniższym kodzie?

```
class C {
    static class B {
        static final int x = 10; // 1
    }
}

public class A {
    public static void main(String[] args) {
        C c = new C();          // 2
        C.B obj1 = c.new B(); // 3
    }
}
```

a) Jego kompilacja i uruchomienie powiedzie się
b) Podczas kompilacji wystąpi błąd w linii 1
c) Podczas kompilacji wystąpi błąd w linii 2
d) Podczas kompilacji wystąpi błąd w linii 3

26. Co można powiedzieć o poniższej metodzie (zakładając, że wszystkie konieczne importy zostały dokonane)?

```
public void run() {
    String command;
    Socket so = null;
    BufferedReader instream = null;
    while (true) {
        try {
            so = new ServerSocket(2000).accept(); // 1
            instream = new BufferedReader(
                           new InputStreamReader(so.getInputStream())); // 2
            while(true){
                command = instream.readLine();
                System.out.println(command);
            }
        } catch (Exception ex) {
            instream.close();so.close();          // 3
        }
    }
}
```

a) Jej kompilacja powiedzie się
b) Jej kompilacja zakończy się błędem (źle zadeklarowany strumień w linii 2)
c) Jej kompilacja zakończy się błędem (nieobsłużony wyjątek w linii 3)
d) Jej kompilacja zakończy się błędem (zły parametr konstruktora `ServerSocket` w linii 1)

27. Niech klasa A będzie zdefiniowana jak niżej. Co można o niej powiedzieć (zakładając, że wszystkie konieczne importy zostały wykonane)?

```
class A {
    public class B {
        private int i;
    }
    public static void main(String[] args) {
     Socket so;
     try {
        so = new Socket("192.168.1.200", 2000); // 1
        ObjectInputStream out = new ObjectInputStream(so.getInputStream());
        B b = (B) out.readObject(); // 2
        System.out.println(b.i); // 3
     } catch (Exception e) {}
    }
}
```

a) Jej kompilacja powiedzie się
b) Jej kompilacja zakończy się błędem w linii 1
c) Jej kompilacja zakończy się błędem w linii 2
d) Jej kompilacja zakończy się błędem w linii 3

28. Co można powiedzieć o poniższym kodzie (przy założeniu, że odpowiednie importy zostały wykonane)?

```
interface I extends Remote {
    void m() throws RemoteException; // 1
}
class MI implements I {
    public void m() { System.out.println("m()");}
}
class A {
    public static void main(String[] args) {
        Registry reg;
        String nazwa = "MI";
        try {
            reg = LocateRegistry.createRegistry(2000);
            MI mi = new MI();
            reg.rebind(nazwa, mi); // 2
        } catch (RemoteException e) { e.printStackTrace();}
}}
```

a) Po kompilacji i uruchomieniu program będzie oczekiwał na przychodzące żądania
b) Po kompilacji i uruchomieniu program natychmiast zakończy działanie
c) Podczas kompilacji wystąpi błąd w linii 1
d) Podczas kompilacji wystąpi błąd w linii 2

29. Co będzie wynikiem wykonania poniższego kodu?

```
URL obj = new URL("http://www.pwr.edu.pl/index.dhtml");
System.out.print(obj.getHost());
```

a) pwr.edu.pl
b) 156.17.193.247
c) www.pwr.edu.pl
d) http://www.pwr.edu.pl/index.dhtml

30. Co można powiedzieć o poniższym fragmencie kodu (przy założeniu, że wszystkie konieczne importy zostały wykonane)?

```
String ftpurl = "ftp://user:password@server.com/projekty/plik.zip";
try {
            URL url = new URL(ftpUrl);                          // 1
            URLConnection conn = url.openConnection();          // 2
            InputStream inputStream = conn.getInputStream();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
```

a) kod ten skompiluje się i wykona poprawnie
b) kod ten skompiluje się poprawnie, ale podczas próby jego wykonania wystąpi błąd (nieobsługiwany protokół FTP)
c) podczas jego kompilacji wystąpi błąd w linii `1`
d) podczas jego kompilacji wystąpi błąd w linii `2`