1. Co można powiedzieć o poniższym kodzie?

```
public class A  {
    void m(int a) { }
    int m(String s){return Integer.parseInt(s);}
    String m(int a){ return Integer.toString(a);}
}
```

a) Jego kompilacja powiedzie się
b) Jego kompilacja nie powiedzie się (nieobsłużony wyjątek `NumberFormatException`)
c) Jego kompilacja nie powiedzie się (zła konwersja z `int` do `String`)
d) Jego kompilacja nie powiedzie się (złe przeciążenie metody `m`)

2. Co można powiedzieć o poniższym kodzie (zamieszczonym w dwóch osobnych plikach)?

```
package inny;
public class A {
   private void k() {}
           void l() {}
   public void m() {}
          void n() {}
}
```

```
package inny;

public class B {
   B(){
      A a = new A(); // 1
      a.l();  // 2
      a.n();  // 3
   }
}
```

a) Jego kompilacja powiedzie się
b) Jego kompilacja nie powiedzie się (wystąpi błąd w linii 1)
c) Jego kompilacja nie powiedzie się (wystąpi błąd w linii 2)
d) Jego kompilacja nie powiedzie się (wystąpi błąd w linii 3)

3. Co można powiedzieć o poniższym kodzie (zamieszczonym w dwóch osobnych plikach)?

```
package pakiecik;

public class A {
    private int i;

    public void m(A a) {
          a.i++;  // 1

    }
}
```

```
package inny;

import pakiecik.A;

public class B extends A{
   public static void main(String[] args){
       B b1 = new B(), b2 = new B(); // 2
       b1.m(b2);                     // 3

   }
}
```

a) Jego kompilacja i wykonanie metody `main` powiedzie się
b) Jego kompilacja nie powiedzie się (wystąpi błąd w linii `1`)
c) Jego kompilacja nie powiedzie się (wystąpi błąd w linii `2`)
d) Jego kompilacja nie powiedzie się (wystąpi błąd w linii `3`)

4. Co można powiedzieć o poniższym kodzie (zamieszczonym w dwóch osobnych plikach)?

```
package pakiecik;

public class A {
    protected int i;

    public int m(A a) {
      a.i++; return i; // 1

    }
}
```

```
package inny;

import pakiecik.A;

public class B extends A{
   public static void main(String[] args){
       B b1 = new B(), b2 = new B(); // 2
       b1.i =b1.m(b2);               // 3

   }
}
```

a) Jego kompilacja i wykonanie metody `main` powiedzie się
b) Jego kompilacja nie powiedzie się (wystąpi błąd w linii `1`)
c) Jego kompilacja nie powiedzie się (wystąpi błąd w linii `2`)
d) Jego kompilacja nie powiedzie się (wystąpi błąd w linii `3`)

5. Co można o powiedzieć o poniższym kodzie (zamieszczonym w trzech osobnych plikach)?

```
public class A {
    private A(){} // 1
}
```

```
public class B extends A {
    public B(){   } // 2
}
```

```
public class C {
    public static void main(String[] args) {
        B b = new B();  // 3
}}
```

a) Jego kompilacja powiedzie się
b) Wystąpi błąd kompilacji w linii 1
c) Wystąpi błąd kompilacji w linii 2
d) Wystąpi błąd kompilacji w linii 3

6. Co można o powiedzieć o poniższym kodzie?

```
public class A {
    private class A {}

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

a) Jego kompilacja powiedzie się
b) Wystąpi błąd kompilacji (nie można deklarować wewnętrznych klas o takiej samej nazwie jak klasa zewn.)
c) Wystąpi błąd kompilacji (nie można deklarować wewnętrznych klas jako prywatnych)
d) Wystąpi błąd kompilacji (nie można odwoływać się do typu klasy wewnętrznej instancyjnej będąc wewnątrz jakiejś statycznej metody).

7. Co można powiedzieć o poniższym kodzie?

```
public class A {
    private void m(){                  // 1
        private final class B {}   // 2
    }

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

a) Jego kompilacja powiedzie się
b) Wystąpi błąd kompilacji w linii 1
c) Wystąpi błąd kompilacji w linii 2
d) Wystąpi błąd kompilacji w linii 3

8. Co można powiedzieć o poniższym kodzie?

```
public class A {
    private int i;

    default void m() {
    }

    void m(int i) {
        this.i = i;
    }
}
```

a) Jego kompilacja powiedzie się
b) Wystąpi błąd kompilacji (zły dostęp do zmiennej prywatnej)
c) Wystąpi błąd kompilacji (nie można umieszczać w klasie dwóch metod o tej samej nazwie)
d) Wystąpi błąd kompilacji (implementacja metody domyślnej może nastąpić tylko w interfejsie, a nie w klasie)

9. Co można powiedzieć o poniższym kodzie?

```
public class A {
    private String s;

    String getS(){ return s;}

     A(String s){ this.s=s; }

    public static void main(String[] args){
        A a = new A("1");
        switch(a.getS()){
         case "1": System.out.print("1,"); break;
         case "2": System.out.print("2,"); break;
        }

    }
}
```

a) Jego kompilacja i uruchomienie powiedzie się. Na ekranie zostanie wypisane `1,`
b) Jego kompilacja i uruchomienie powiedzie się. Na ekranie zostanie wypisane `1,2,`
c) Wystąpi błąd kompilacji (brak bezaurgumentowego konstruktora)
d) Wystąpi błąd kompilacji (zła instrukcja `switch`)

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 poniższym kodzie?

```
interface I{
    public void n();
}

public class A {

    public void m(){}

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

a) Jego kompilacja powiedzie się
b) Wystąpi błąd kompilacji w linii 1
c) Wystąpi błąd kompilacji w linii 2
d) Wystąpi błąd kompilacji w linii 3

12. Co można powiedzieć o poniższym kodzie?

```
import java.util.Set;
import java.util.HashSet;

public class A {
    private String s;
    @Override
    public boolean equals(Object obj) {
     if (! ((A)obj).equals(s)) return false;
     return true;
    }

    public A(String s) {this.s=s;}
    public static void main(String[] args){
        Set<A> z = new HashSet<>();
        z.add(new A("1"));
        z.add(new A("1"));
        System.out.println(z.size());
    }
}
```

a) Jego kompilacja i uruchomienie metody `main` powiedzie się. Na ekranie pojawi się `1`
b) Jego kompilacja i uruchomienie metody `main` powiedzie się. Na ekranie pojawi się `2`
c) Jego kompilacja powiedzie się, jednak podczas uruchomienia pojawi się wyjątek
d) Jego kompilacja zakończy się błędem

13. Co można powiedzieć o poniższym kodzie?

```
package pakiecik;

import java.util.Set;
import java.util.TreeSet;

public class A {

    public static void main(String[] args){
        Set<A> z = new TreeSet<A>();
        z.add(new A());
        z.add(new A());
        System.out.println(z.size());
    }
}
```

a) Jego kompilacja i uruchomienie metody `main` powiedzie się. Na ekranie pojawi się `1`
b) Jego kompilacja i uruchomienie metody `main` powiedzie się. Na ekranie pojawi się `2`
c) Jego kompilacja powiedzie się, jednak podczas uruchomienia pojawi się wyjątek
d) Jego kompilacja zakończy się błędem

14. Co można powiedzieć o poniższym kodzie?

```
package pakiecik;

class E extends Exception{}

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

    public static void main(String[] args) throws Exception {
        m();
    }
}
```

a) Jego kompilacja i uruchomienie metody `main` powiedzie się
b) Jego kompilacja zakończy się błędem (brak wyjątku `E` w klauzuli **`throws`**)
c) Jego kompilacja zakończy się błędem (brak deklaracji importu klasy `Exception`)
d) Jego kompilacja zakończy się błędem (bo w ciele metody `m()` brak wyrzucenia 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 a = new A();
    n(ab);                    // 2
    n(a,a);                   // 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)?

```
public class A<T> {                                  // 1
    Collection<T> c = new LinkedList<T>(); // 2

    public static void main(String[] args) {
        (new A<Integer>()).c.add(10);  // 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<T> {                                    // 1
    public <T,U> U m(U u, T t){                        // 2
      if(u.equals(t)) return u;
       else return null;}

    public static void main(String[] args) {
       System.out.println((new A<Integer>()).m("10", "10")); // 3
    }
}
```

a) Jego kompilacja i uruchomienie metody `main` powiedzie się (na ekranie wypisane zostanie `10`)
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?

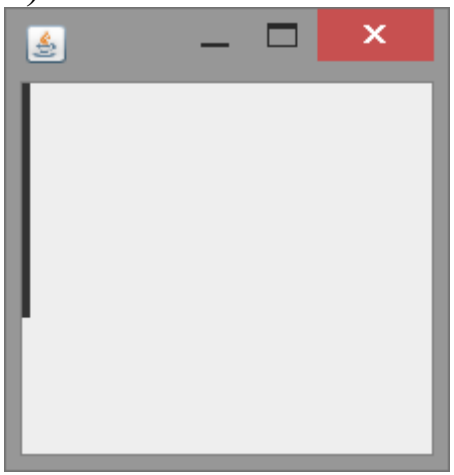
```
public class A<T> { // 1

     public static T t;      // 2

    public static void main(String[] args){
        A<Integer> a = new A<>();  a.t++;  // 3
    }
}
```

a) Jego kompilacja i uruchomieniu 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`

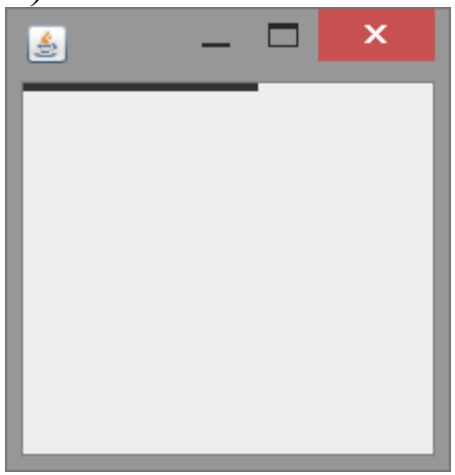
19. Jak wygląda ramka, na którą wstawiono panel o metodzie `paintComponent` jak niżej?

```
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2d = (Graphics2D) g;
    AffineTransform saveAT = g2d.getTransform();
    AffineTransform t = new AffineTransform();

    t.translate(2,2);
    t.scale(4, -4);
    t.rotate(Math.PI/4);
    g2d.setTransform(t);
    g2d.drawLine(0, 0, 20, -20);
    g2d.setTransform(saveAT);
}
```

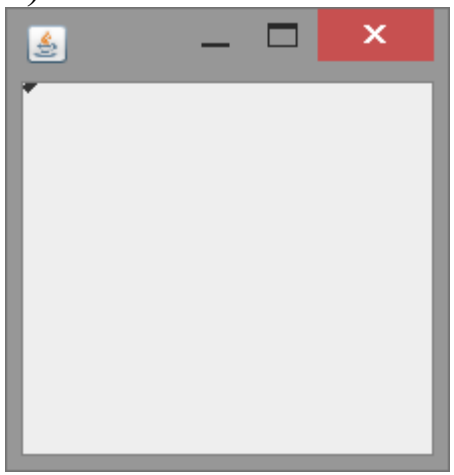
a)



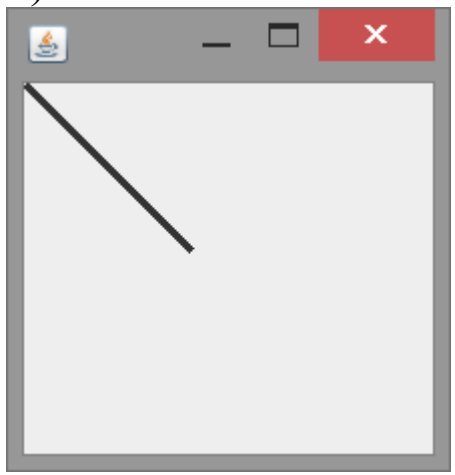
b)



c)



d)

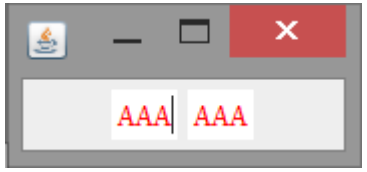


20. Jak wygląda panel, na którym wstawiono komponenty instrukcjami jak niżej?

```
setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
JTextArea ta = new JTextArea();
ta.setText("<html><body><font color=\"red\">AAA</font></body></html>");
add(ta);

JTextPane tp = new JTextPane();
tp.setText("<html><body><font color=\"red\">AAA</font></body></html>");
add(tp);
```

a) Panel się nie pojawi, bo w powyższym kodzie jest błąd (źle zredagowany ciąg znaków w metodzie `setText`)

b)



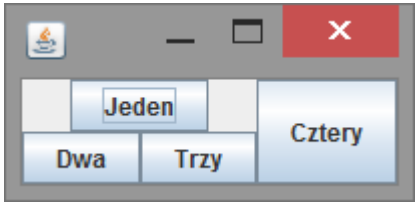
c)

d)

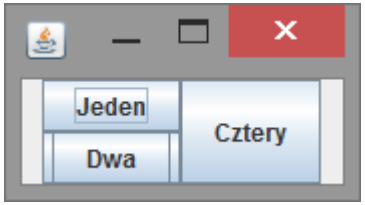
21. Jak wygląda ramka, na którą wstawiono panel skonstruowany metodami jak niżej?

```
setLayout(new GridBagLayout());
GridBagConstraints c = new GridBagConstraints();
JButton btn1 = new JButton("Jeden");
c.fill = GridBagConstraints.HORIZONTAL;
c.gridwidth = 2;   c.gridx = 1; c.gridy = 0;
add(btn1, c);

c = new GridBagConstraints();
JButton btn2 = new JButton("Dwa");
                   c.gridx = 1; c.gridy = 1;
add(btn2, c);

c = new GridBagConstraints();
JButton btn3 = new JButton("Trzy");
c.fill = GridBagConstraints.HORIZONTAL;
                   c.gridx = 2; c.gridy = 1;
add(btn3, c);

c = new GridBagConstraints();
JButton btn4 = new JButton("Cztery");
c.fill = GridBagConstraints.VERTICAL;
c.gridheight = 2; c.gridx = 3; c.gridy = 0;
add(btn4, c);
```

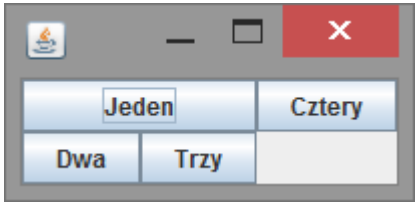
a)



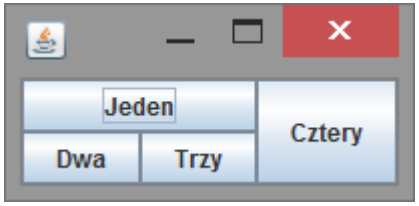
b)



c)



d)



22. Co można powiedzieć o poniższym kodzie?

```
class R extends Thread {
    R(Runnable t) {
        new Thread(t).start();                          // 1
    }
    public void run() {
        try {
            join();
        } catch (InterruptedException e) {}
    }
}
public class A {
    public static void main(String args[]) {
        R r = new R(() -> System.out.println("T")); // 2
        r.start();
    }
}
```

a) Jego kompilacja nie uda się. Błąd wystąpi w linii `1` lub linii `2` (tj. w którejś z nich)
b) Jego kompilacja i uruchomienie metody `main` powiedzie się: na ekranie zostanie wypisanie `T`, po czym program zakończy swojego działanie.
c) Jego kompilacja i uruchomienie metody `main` powiedzie się: na ekranie zostanie wypisanie `T`, jednak program nie zakończy swojego działania.
d) Jego kompilacja i uruchomienie metody `main` powiedzie się: na ekranie nic się nie pojawi, a program zakończy swoje działanie.

23. Co można powiedzieć o poniższym fragmencie kodu przy danych definicjach klas `B`, `Prod`, `Cons`?

```
B b = new B();
Prod p = new Prod(b);
Cons c = new Cons(b);
c.start();  p.start();
```

```
class B { double pole = 12.0;}

class Prod extends Thread {
    private B b = null;
    Prod(B b) {this.b = b; }

    public void run() {
        while(true){
        synchronized(b.pole){
        b.pole = Math.random();
        }}
    }}

class Cons extends Thread {
    private B b = null;
    Cons(B b) {this.b = b;}

    public void run() {
        while(true){
        synchronized(b.pole){
            System.out.println(b.pole);
        }}
    }}
```

a) Jego kompilacja i uruchomienie powiedzie się. Powstaną dwa wątki, które będą zmieniać i odczytywać wartość `b.pole` wzajemnie się przy tym wykluczając (iteracje w pętli `while` będą wykonywać się na zmianę, raz jednego wątku, raz drugiego)

b) Jego kompilacja i uruchomienie powiedzie się. Powstaną dwa wątki, które będą zmieniać i odczytywać wartość `b.pole` wzajemnie się przy tym wykluczając (iteracje w pętli `while` będą mogły powtórzyć się kilka razy zanim dojdzie do wykluczenia)

c) Jego kompilacja i uruchomienie powiedzie się. Powstaną dwa wątki, które zakleszczą się (program wypisze kilka wartości, a potem „zawiesi się”).

d) Jego kompilacja zakończy się błędem

24. Co można powiedzieć o poniższym kodzie?

```
import java.util.Random;

class B {
    public static String s = "Ala";
}

class W extends Thread {
    private static Random r = new Random();
    W(String s) {super(s);}                          // 1
    public void run() {
        while (true) {
            synchronized (B.s) {                     // 2
                B.s = this.getName()+                // 3
                        Long.toString(Math.abs(r.nextLong()), 36);
                System.out.println(B.s);
        }}}
}

public class A {
    public static void main(String[] s) {
       new W("A").start();  new W("B").start();
    }
}
```

a) Jego kompilacja i wykonanie 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`

25. Co można powiedzieć o poniższym kodzie?

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

public class A {
    public static void main(String[] args) {
        C.B obj = new C().new B(); // 2
        int i = obj.x + 2;          // 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 klasie `A` (zakładając, że wszystkie konieczne importy zostały dokonane)?

```
public class A {
static BufferedReader instream;
static Socket s1, s2;

public static void main(String[] args) throws UnknownHostException, IOException {

    new Thread(() -> {
        try { s1 = new ServerSocket(2000).accept();
              instream = new BufferedReader(new
                                    InputStreamReader(s1.getInputStream()));
              System.out.println(instream.readLine());
        } catch (Exception ex) {
            System.out.println("Exception");
        }}).start();
    s2 = new Socket("localhost", 2000);
    s2.close();
    }
}
```

a) Jej kompilacja i uruchomienie metody `main` powiedzie się. Program zakończy się, a na ekranie pojawi się `Exception`
b) Jej kompilacja i uruchomienie metody `main` powiedzie się. Program zakończy się, a na ekranie pojawi się `null`
c) Jej kompilacja powiedzie się. Po uruchomieniu metody `main` ekran pozostanie pusty, a program nie skończy swojego działania.
d) Jej kompilacja zakończy się błędem

27. Co można powiedzieć o klasie `A` (zakładając, że wszystkie konieczne importy zostały dokonane)?

```
class A {
    public static void main(String[] args) {
     Socket so;
     try {
        so = new Socket("192.168.75.94", 2000); // 1
        ObjectInputStream out = new ObjectInputStream(so.getInputStream());
        Integer i = out.readObject(); // 2
        System.out.println(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 i klasy znajdują się w osobnych plikach)?

```
public interface I extends Remote {
 default void m() throws RemoteException { System.out.println("A"); }  // 1
}

public class A extends UnicastRemoteObject implements I {        // 2
 public static void main(String[] args) {
    try {
      LocateRegistry.createRegistry(1099).rebind("A", new A()); // 3
    } catch (RemoteException e) {
 }}}

public class B {
 public static void main(String[] args) {
    try {
        ((I) LocateRegistry.getRegistry(1099).lookup("A")).m(); // 4
        } catch (RemoteException | NotBoundException e) {
}}}
```

a) Jego kompilacja i uruchomienie powiedzie się, jeśli najpierw zostanie uruchomiona metoda `main` klasy `A`, a potem metoda `main` klasy `B`. Tylko wtedy zdalnie uruchomiona zostanie metoda `m()`.
b) Podczas kompilacji wystąpi błąd w linii `1`
c) Podczas kompilacji wystąpi błąd w linii `2`
d) Podczas kompilacji wystąpi błąd w liii `3` lub `4` (tzn. którejś z nich)

29. Co można powiedzieć o poniższym kodzie (przy założeniu, że odpowiednie importy zostały wykonane i klasy znajdują się w osobnych plikach)?

```
interface I extends Remote {
     default void m() throws RemoteException { System.out.println("A"); }  // 1
    }

public class A implements I, Serializable {                              // 2
    public static void main(String[] args) {
        try {
          LocateRegistry.createRegistry(1099).rebind("A", new A()); // 3
          Thread.sleep(50000);
        } catch (RemoteException|InterruptedException e) {
     }}}

public class B {
 public static void main(String[] args) {
    try {
        ((A) LocateRegistry.getRegistry().lookup("A")).m(); // 4
        } catch (RemoteException | NotBoundException e) {
}}}
```

a) Jego kompilacja i uruchomienie powiedzie się, jeśli najpierw zostanie uruchomiona metoda `main` klasy `A`, a potem metoda `main` klasy `B`. Jednak uruchomienie metody `m()` będzie lokalne.
b) Podczas kompilacji wystąpi błąd w linii `1`
c) Podczas kompilacji wystąpi błąd w linii `2`
d) Podczas kompilacji wystąpi błąd w liii `3` lub `4` (tzn. którejś z nich)

30. Zakładając, że w jakiejś aplikacji poprawnie wykonała się poniższa linia kodu (`namiastka` jest obiektem pozwalającym odpalić zdalnie zaimplementowane gdzieś metody interfejsu `I`)

```
LocateRegistry.createRegistry(1099).rebind("A", namiastka); // 1
```

to co można powiedzieć o poniższym kodzie (przy założeniu, że odpowiednie importy zostały wykonane i klasy znajdują się w osobnych plikach)?

```
public interface I extends Remote {
     void m() throws RemoteException;
    }

public class B {
     public static void main(String[] args) {
        try {
            ((I) Naming.lookup("rmi://localhost:1099/A")).m(); // 2
            } catch (RemoteException | MalformedURLException | NotBoundException e) {
    }}}
```

a) Jego kompilacja i uruchomienie powiedzie się (metoda `m` zostanie wywołana zdalnie)
b) Jego kompilacja powiedzie się, jednak po uruchomieniu metody `main` klasy `B` nie dojdzie do zdalnego wywołania metody `m` (do bloku `catch` przekazany zostanie wyjątek)
c) Aby kompilacja i zdalne wywołanie metody `m` podczas uruchomienia metod `main` klasy `B` powiodło się kod w linii `2` musi zostać zastąpiony poniższym kodem

```
((I) Naming.lookup("A")).m();
```

d) Jego kompilacja i zdalne wywołanie metody `m` w metodzie `main` powiedzie się, jeśli kod w linii `1` zostanie zastąpiony poniższym kodem

```
LocateRegistry.createRegistry(1099).rebind("rmi://localhost:1099/A", namiastka);
```