Zadania III rok informatyki inzynierskiej

1. Wyrażenia, instrukcje, metody
2. Klasy i obiekty
3. Obsługa wyjątków

4 Strumienie

5 Kontenery

6 Wielowątkowość

7 Programowanie sieciowe

Zadanie 1.1

Okna dialogowe są wygodnym sposobem interakcji programu z użytkownikiem. Kilka specjalizowanych rodzajów dialogów pozwala na wyświetlanie komunikatów, wprowadzanie danych czy uzyskiwanie potwierdzeń wykonywanych czynności od użytkownika. Do wyświetlania dialogów służą statyczne metody `showXXXDialog` z klasy `JOptionPane` pakietu `javax.swing`. Jest ich dość dużo, ze względu na bogate możliwości konfiguracji wyglądu i zachowań dialogów.

Napisz program, który:

1. za pomocą okna dialogowego pobierze od użytkownika łańcuch znakowy,
2. zamieni w nim małe litery na wielkie,
3. wyświetli wynik w dialogowym oknie informacyjnym.

Wskazówki

- Do wyświetlania okna dialogowego pozwalającego na wprowadzenie danych służy przeciążona, statyczna metoda `showInputDialog()` z klasy `JOptionPane`.
- Do zamiany małych liter na wielkie służy metoda `toUpperCase()` z klasy `String`. Zwraca ona nowy obiekt (nie modyfikuje źródła)!
- Do wyświetlenia okna informacyjnego służy statyczna, przeciążona metoda klasy `JOptionPane` o nazwie `showMessageDialog()`.

- Spośród wielu wariantów metod przeciążonych należy wybrać te o najmniejszej liczbie parametrów. Wykorzystują one domyślne wartości dla konfigurowalnych cech dialogów (np. ikona, tytuł).
- Programy korzystające z okien biblioteki `Swing` (w szczególności korzystające z dialogów wyświetlanych przez metody klasy `JOptionPane`) należy kończyć wywołując metodę `System.exit(int)`.

Zadanie 1.2.

Korzystając z klasy `BigInteger` napisać program, który wyznacza silnię z podanej liczby całkowitej n nawet dla dużych **n** (rzędu kilkuset).

Zadanie 1.3.

Napisać program sumujący liczby nieparzyste z przedziału od 1 do n, gdzie n - podaje użytkownik na starcie programu. Program powinien zakończyć sumowanie na liczbie n, gdy liczba n jest nieparzysta lub na liczbie n – 1, gdy liczba n jest parzysta.

Zadanie 1.4.

Operatory bitowe pozwalają traktować zmienne typów całkowitoliczbowych jak zestawy bitów i wykonywać na nich operacje. Oprócz bitowych odpowiedników operatorów logicznych (alternatywa, koniunkcja, negacja) dostępne są również operatory przesunięcia. Ich działanie polega na przesunięciu całego zestawu bitów o zadaną pozycję w lewo lub w prawo (część bitów zostanie utrącona, a wolne miejsca wypełnione 0 lub 1). Operatory bitowe stosuje się rzadko, głównie do kodowania w zwarty sposób binarnych informacji w postaci tzw. flag, które można badać za pomocą tzw. masek (są to zmienne typów całkowitoliczbowych z określonymi bitami ustalonymi na 1, a pozostałymi na 0). Chcemy mieć metody, które zmieniają liczbę na napis ją reprezentujący w zadanym systemie liczenia (binarny, ósemkowy, szesnastkowy). Aby uprościć zadanie, ograniczymy się do liczb nieujemnych. Zatem należy zaimplementować trzy metody pobierające jako argument liczbę całkowitą typu `int` i zwracającą łańcuch znakowy (obiekt klasy `String`) będący:

1. binarną reprezentacją argumentu,
2. ósemkową reprezentacją argumentu,
3. szesnastkową reprezentacją argumentu.

Wskazówki

• Można skorzystać z operatorów bitowych przesunięcia w prawo i koniunkcji.

• Można skorzystać z operatorów dzielenia oraz reszty z dzielenia (/, %).

Zadanie 2.1

Zaprojektuj klasę `Rational`, reprezentującą liczby wymierne jako pary liczb całkowitych (licznik i mianownik), wraz z podstawowymi działaniami arytmetycznymi i porównaniem. W klasie powinny znaleźć się nastę-pujące metody publiczne (oprócz konstruktora):

1. dodawanie: `Rational add(Rational arg);`
2. mnożenie: `Rational mul(Rational arg);`
3. odejmowanie: `Rational sub(Rational arg);`
4. dzielenie: `Rational div(Rational arg);`
5. równość: `boolean equals(Rational arg);`
6. porównanie: `int compareTo(Rational arg);`
7. tekstowa reprezentacja liczby: `String toString().`

Metody 1–4 powinny zwracać jako rezultat referencję do nowego obiektu klasy `Rational`, będącego wynikiem operacji wykonanej na argumencie `arg` i `this`. Metoda 5. ma porównywać obiekty klasy `Rational` na podstawie wartości liczb, np. 1/2 = 2/4. Metoda 6. ma działać podobnie, jak odpowiadająca jej metoda `compareTo(Object o)` z interfejsu

`java.lang.Comparable`:

- Jeśli `this` jest równe `arg`, to zwraca **0**.
- Jeśli `this` jest mniejsze od `arg`, to zwraca `-1`.
- Jeśli this jest większe niż arg, to zwraca 1.

Metoda 7. ma zwracać łańcuch znakowy opisujący ten obiekt. Na przykład może to być napis postaci 1/2 lub -1/1.

Zadanie 2.2.

Napisz klasę opisującą równanie kwadratowe o postaci $y = ax^2 + bx + c$. Współczynniki a, b i c powinny być prywatne. Zdefiniuj następujące publiczne funkcje składowe:

- nadającą wartości współczynnikom,
- obliczającą y dla podanego x,
- wyznaczającą liczbę pierwiastków.

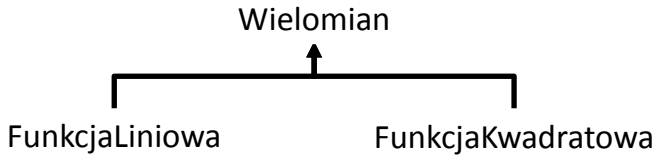
Potrzebne wzory:

- delta: $d = b^2 - 4ac$,
- liczba pierwiastków:

  p =0 : d < 0, 1 : d = 0, 2 : d > 0.

Zadanie 2.3.

Zdefiniuj poniższą hierarchię klas:



tak, aby w wyniku wykonania programu:

```
public class Zadanie {
      public static void main(String[] args) {
      Wielomian w[] = new Wielomian[3];
      w[0] = new FunkcjaLiniowa(2, 1); // 2x + 1
      w[1] = new FunkcjaKwadratowa(1, -2, 2); // x*x - 2x + 2
      w[2] = new FunkcjaKwadratowa(1, 0, -1); // x*x - 1
      for (int i=0; i<3; i++) {
            w[i].wypiszMiejscaZerowe();
            }
      }
}
```

na ekranie pojawił się wynik:

```
-0.5
brak
-1 1
```

Wskazówka

- Wielomian może być klasą abstrakcyjną lub nawet interfejsem.

Zadanie 3.1

Poniższy program:

```
class Kolejka {
      static final int N = 5;
      private Object[] tab;
      private int pocz, zaost, lbel;
            public Kolejka() {
```

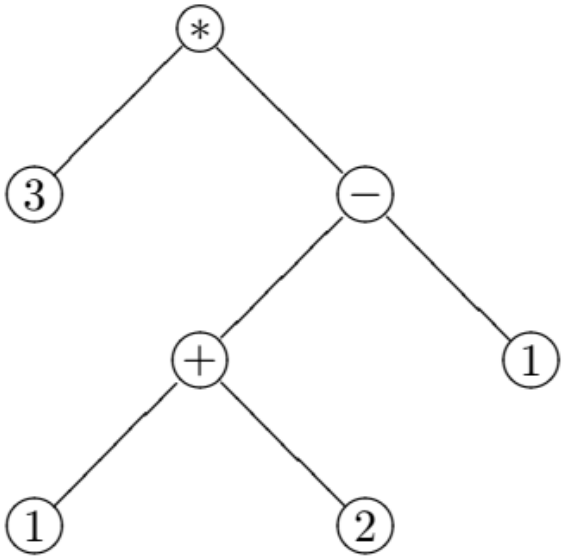
```
            pocz=0; zaost=0; lbel=0;
            tab = new Object[N];
      }

      void doKolejki(Object el) {
            tab[zaost] = el;
            zaost = (zaost+1) % N;
            ++lbel;
      }

      Object zKolejki() {
            int ind = pocz;
            pocz = (pocz+1) % N;
            --lbel;
            return tab[ind];
      }
}

public class Zadanie {
      public static void main(String[] args) {
            Kolejka k = new Kolejka();
            k.doKolejki(new Integer(7));
            k.doKolejki(new String("Ala ma kota"));
            k.doKolejki(new Double(3.14));
            for (int i=1; i<=3; ++i)
                  System.out.println((k.zKolejki()).toString());
      }
}
```

zmodyfikuj tak, aby w funkcji main można było przechwytywać wyjątki przepełnienia (próba dodania, gdy liczba elementów w kolejce wynosi N) i niedomiaru kolejki (próba pobrania elementu kolejki, gdy liczba elementów w kolejce wynosi 0):

```
      public static void main(String[] args) {
            Kolejka k = new Kolejka();
            try {
                  k.doKolejki(new Integer(7));
                  k.doKolejki(new String("Ala ma kota"));
                  k.doKolejki(new Double(3.14));
                  for (int i=1; i<=4; ++i)
                  System.out.println((k.zKolejki()).toString());
            }
                  catch (Przepelnienie e) {
                  System.out.println("Przepełniona kolejka!");
            }
                  catch (Niedomiar e) {
                  System.out.println("Pusta kolejka!");
            }
      }
```

Zadanie 3.2.

Wiadomo, że wyrażenie arytmetyczne może być reprezentowane za pomocą drzewa binarnego. Wierzchołki wewnętrzne takiego drzewa reprezentują działanie, natomiast liście reprezentują stałe. Na przykład wyrażeniu $(3 * ((1 + 2) - 1))$ odpowiada następujące drzewo:

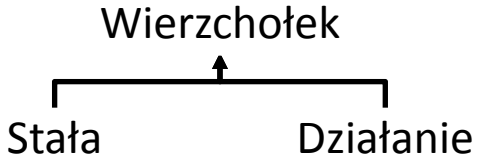


W poniższym programie:

```
abstract class Wierzcholek {
      Wierzcholek lewy, prawy;
      public abstract int wartosc();
}
class Stala extends Wierzcholek {
      private int wart;
      public Stala(int x) {
            wart = x;
      }
      public int wartosc() {
            return wart;
      }
}
class Dzialanie extends Wierzcholek {
      private char op; // operator +, -, / lub *
      public Dzialanie(char znak) {
            op = znak;
      }
      public void dodajLewyArg(Wierzcholek arg) {
            lewy = arg;
      }
      public void dodajPrawyArg(Wierzcholek arg) {
            prawy = arg;
      }
      public int wartosc() {
            switch (op) {
                  case '+': return lewy.wartosc() + prawy.wartosc();
                  case '-': return lewy.wartosc() - prawy.wartosc();
                  case '/': return lewy.wartosc() / prawy.wartosc();
                  case '*': return lewy.wartosc() * prawy.wartosc();
```

```
            }
            return 0;
      }
}
class Wyrazenie {
      private Wierzcholek korzen;
      private Wierzcholek utworzDrzewo(String w, int p, int q) {
            if (p == q)
                  return new Stala(Character.digit(w.charAt(p), 10));
            else {
                  int i = p+1, nawiasy = 0;
                  while ( (nawiasy != 0) || (w.charAt(i) == '(') ||
                  (w.charAt(i) == ')') || (Character.isDigit(w.charAt(i))))
                  {
                        if (w.charAt(i) == '(') ++nawiasy;
                        if (w.charAt(i) == ')') --nawiasy;
                        ++i;
                  }
                  Dzialanie nowy = new Dzialanie(w.charAt(i));
                  nowy.dodajLewyArg(utworzDrzewo(w, p+1, i-1));
                  nowy.dodajPrawyArg(utworzDrzewo(w, i+1, q-1));
                  return nowy;
            }
      }
      public Wyrazenie(String w) {
            korzen = utworzDrzewo(w, 0, w.length()-1);
      }
      public int oblicz() {
            return korzen.wartosc();
      }
}
public class Zadanie {
            public static void main(String[] args) {
            Wyrazenie wyr = new Wyrazenie("(3*((1+2)-1))");
            System.out.println("" + wyr.oblicz());
      }
}
```

stworzono hierarchię klas:



a następnie zaimplementowano klasę `Wyrazenie`, której metoda `oblicz()` zwraca wartość podanego wyrażenia. Zakładamy, że konstruktor akceptuje wyrażenia arytmetyczne skonstruowane zgodnie z gramatyką:

```
<wyrażenie> ::= (<wyrażenie><działanie><wyrażenie>)
<wyrażenie> ::= <stała>
<działanie> ::= + | - | / | *
<stała> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Uzupełnij przytoczony program o obsługę następujących wyjątków:

- dzielenie przez zero,
- niepoprawnie skonstruowane wyrażenie.

Zadanie 4.1.

Napisz prosty edytor tekstowy, w którym będzie możliwość zapisywania tekstu do pliku w jednym z wybranych standardów kodowania znaków: UTF-8, ISO-8859-2 lub windows-1250.

Wskazówki:

- Skorzystaj z klas OutputStreamWriter oraz FileOutputStream.
- Łańcuchami reprezentującymi wymienione standardy kodowania znaków są: UTF8, ISO8859 2 oraz Cp1250.

Zadanie 4.2.

Napisz program kompresujący plik do formatu GZIP oraz program rozpakowujący plik GZIP.

Wskazówka

- Skorzystaj z klas GZIPOutputStream oraz GZIPInputStream.

Zadanie 5.1.

Dostosuj poniższą klasę:

```
class Wspolrzedna {
      private int x, y;
      public Wspolrzedna(int _x, int _y) {
            x = _x;
            y = _y;
      }
      public String toString() {
            return "(" + x + ", " + y + ")";
      }
}
```

do wymagań stawianych wobec elementów dodawanych do kontenera `TreeSet`, tak aby w wyniku wykonania programu:

```
import java.util.*;
//Tu wstaw zmodyfikowaną klasę przechowującą współrzędne punktu
public class Zadanie {
	private static void wypiszElementy(TreeSet zbior) {
		Iterator it = zbior.iterator();
		while (it.hasNext()) {
			System.out.println((it.next()).toString());
		}
	}
	public static void main(String[] args) {
		TreeSet zbior = new TreeSet();
		zbior.add( new Wspolrzedna(2, 3) );
		zbior.add( new Wspolrzedna(-3, 0) );
		zbior.add( new Wspolrzedna(-1, 2) );
		zbior.add( new Wspolrzedna(-1, 2) );
		zbior.add( new Wspolrzedna(-3, -2) );
		wypiszElementy(zbior);
	}
}
```

punkty zbioru `TreeSet` były wyświetlone na ekranie w kolejności leksykograficznej (czyli (−3, −2), (−3, 0), (−1, 2), (2, 3)).

Wskazówka

- Klasa `Wspolrzedna` powinna implementować interfejs `Comparable`.

Zadanie 5.2.

Dostosuj klasę `Wspolrzedna` z poprzedniego zadania do wymagań stawianych wobec elementów dodawanych do kontenera `HashMap`. Wówczas w wyniku wykonania programu:

```
import java.util.*;
//Tu wstaw zmodyfikowaną klasę przechowującą współrzędne punktu
public class Zadanie {
	public static void main(String[] args) {
	HashMap mapa = new HashMap();
	mapa.put(new Wspolrzedna(2, 3), new String("czerwony"));
	mapa.put(new Wspolrzedna(-3, 0), new String("czarny"));
	mapa.put(new Wspolrzedna(-1, 2), new String("czerwony"));
	mapa.put(new Wspolrzedna(2, -1), new String("czarny"));
	Wspolrzedna w = new Wspolrzedna(-1, 2);
	System.out.println("Punkt " + w.toString()
	+ " ma kolor " + mapa.get(w));
	}
}
```

na ekranie zostanie wyświetlony tekst:

```
Punkt (-1, 2) ma kolor czerwony
```

Wskazówka

- Klasa `Wspolrzedna` powinna przesłonić metody `hashCode` oraz `equals`.

Zadanie 5.3

W poniższej klasie Graf:

```
import java.util.*;
class Graf {
      private int n; // liczba wierzchołków, V = {0,1,...,n-1}
      private LinkedList[] tab; // tablica wierzchołków połączo-
                                // nych z danym wierzcholkiem
      public Graf(String lan) {
            StringTokenizer st = new StringTokenizer(lan, "() ,");
            n = Integer.parseInt(st.nextToken());
            tab = new LinkedList[n];
            for (int i=0; i<n; ++i)
            tab[i] = new LinkedList();
            while (st.hasMoreTokens()) {
                  tab[Integer.parseInt(st.nextToken())].add(
                  new Integer(st.nextToken()));
            }
      }
      public String toString() {
      ...
      }
}
public class Zadanie {
      public static void main(String[] args) {
            Graf g = new Graf("4, (0,1), (1,2), (3,0), (1,3)");
            System.out.println(g.toString());
      }
}
```

zdefiniuj metodę `toString` w taki sposób, aby graf był przedstawiany jako tablica ciągów wierzchołków połączonych z kolejnymi wierzchołkami grafu skierowanego:

```
0: 1
1: 2 3
2:
3: 0
```

Wskazówki

• W celu wielokrotnego dołączania łańcucha (lub liczby) na końcu innego łańcucha najlepiej skorzystać z klasy **`StringBuffer`** i jej metody **`append`**.

- Przejście do nowego wiersza realizujemy dołączając do łańcucha sekwencję sterującą "\n".

Zadanie 6.1

W poniższym programie użytkownik ma możliwość wprowadzania tekstu do okienka. Działający w programie wątek zamienia we wpisywanym tekście wystąpienie znaku klamry otwierającej na słowo **`begin`** oraz znaku klamry zamykającej na słowo **`end`**.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class Zamieniacz extends Thread {
      JTextArea okno;
      volatile boolean zakonczyc;
      public Zamieniacz(JTextArea comp) {
            okno = comp;
            zakonczyc = false;
      }
      public void run() {
            while (! zakonczyc) {
            try {
                  String tekst = okno.getText();
                  int indeks = tekst.indexOf("{");
                  if (indeks >= 0) {
                        okno.replaceRange("begin", indeks, indeks+1);
                        okno.setCaretPosition(tekst.length()+4);
                  }
                  else {
                        indeks = tekst.indexOf("}");
                        if (indeks >=0) {
                              okno.replaceRange("end", indeks,
                              indeks+1);
                              okno.setCaretPosition(tekst.length()+2);
                        }
                  }
                  sleep(2000);
            }
            catch (Exception e) {}
            }
      }
}

public class NewJFrame extends JFrame {
      public NewJFrame() {
            initComponents();
            setSize(350, 250);
            watek = new Zamieniacz(jTextArea1);
            watek.start();
      }
      private void initComponents() {
      jScrollPane1 = new JScrollPane();
      jTextArea1 = new JTextArea();
      setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
      addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent evt) {
                  formWindowClosing(evt);
            }
            });
            jTextArea1.setPreferredSize(new Dimension(300, 200));
            jScrollPane1.setViewportView(jTextArea1);
            getContentPane().add(jScrollPane1, BorderLayout.CENTER);
            pack();
      }
      private void formWindowClosing(WindowEvent evt) {
            watek.zakonczyc = true;
            watek = null;
      }
      public static void main(String args[]) {
```

```
            EventQueue.invokeLater(new Runnable() {
                  public void run() {
                  new NewJFrame().setVisible(true);
                  }
            });
      }
      private JScrollPane jScrollPane1;
      private JTextArea jTextArea1;
      private Zamieniacz watek;
}
```

Napisz oraz dodaj do programu wątek sprawdzający co 10 sekund, czy użytkownik wprowadził do okienka tekstowego słowo niecenzuralne (np. *„cholera”*) i informujący o tym fakcie za pomocą odpowiedniego komunikatu przekazanego do metody **`JOptionPane.showMessageDialog().`**

Zadanie 6.2.

W poniższym programie zdefiniowano klasę **`KolejkaKomunikatow`**, do której może odwoływać się kilka wątków naraz.

```
import java.util.*;
class KolejkaKomunikatow {
      Vector kolejka = new Vector();
      public synchronized void wyslij(Object ob) {
            kolejka.addElement(ob);
      }
      public synchronized Object odbierz() {
            if (kolejka.size() == 0) return null;
            Object ob = kolejka.firstElement();
            kolejka.removeElementAt(0);
            return ob;
      }
}
class Watek extends Thread {
      private KolejkaKomunikatow koko;
      private int istart;
      public Watek(KolejkaKomunikatow kk, int pocz) {
            koko = kk;
            istart = pocz;
      }
      public void run() {
            try {
                  for (int i=istart; i<=10; i+=2) {
                        koko.wyslij(new Integer(i));
                        Thread.sleep(50);
                  }
            }
            catch (InterruptedException e) {};
      }
}
public class Zadanie {
      public static void main(String args[]) {
            KolejkaKomunikatow k = new KolejkaKomunikatow();
            Watek w1 = new Watek(k, 1);
            Watek w2 = new Watek(k, 2);
```

```
            w1.start();
            w2.start();
            try {
                  w1.join();
                  14 Zbiór zadań do przedmiotu programowanie
                  obiektowe
                  w2.join();
            }
            catch (InterruptedException e) {};
            Object ob = k.odbierz();
            while (k != null) {
                  System.out.println(((Integer)ob).toString());
                  ob = k.odbierz();
            }
      }
}
```

Zaimplementuj w podobny sposób odwoływanie się przez wątki do klasy **`HashMap`** (klucz może być obiektem klasy **`String`**, a wartość obiektem klasy **`Integer`**).

Zadanie 7.1

Poniższy program łączy się z podanym (jako parametr wywołania) „daytime" serwerem na porcie 13, a następnie odczytuje komunikat wysyłany przez serwer.

```
import java.net.*;
import java.io.*;
public class Zadanie {
      public static void main(String[] args) {
            String nazwahosta;
                  if (args.length > 0) {
                  nazwahosta = args[0];
            }
            else {
                  nazwahosta = "time-a.nist.gov";
            }
            try {
                  Socket gniazdo = new Socket(nazwahosta, 13);
                  InputStream strumien = gniazdo.getInputStream();
                  BufferedReader bufor = new BufferedReader(
                  new InputStreamReader(strumien));
                  String wiersz = "";
                  while (wiersz != null) {
                        System.out.println(wiersz);
                        wiersz = bufor.readLine();
                  }
            }
            catch (UnknownHostException e) {
                  System.err.println(e);
            }
            catch (IOException e) {
                  System.err.println(e);
            }
      }
}
```

Listę wybranych serwerów podających aktualną datę i czas przedstawiono poniżej:

- time-a.nist.gov
- time-b.nist.gov
- time-nw.nist.gov
- time.windows.com

Po połączeniu się z jednym z nich przez port 37 wysyła on 32 bity reprezentujące liczbę sekund, które upłynęły od północy 1 stycznia 1900 r. Napisz program odczytujący tę liczbę.

Wskazówki

- Skorzystaj z metody **`read()`** klasy **`InputStream`**.
- Cztery bajty zamień na liczbę typu long za pomocą operatorów **`<<`** oraz **`|`**.

Zadanie 7.2

Rozważmy następujący program-serwer.

```
import java.net.*;
import java.io.*;

class Gracz implements Runnable {
    private int plansza[][];
    // liczba >= 100 to mina, 0, 1, ..., 8 -- ile wokól min
    private boolean klikniete[][];
    private PrintWriter out;
    private BufferedReader in;
    private Socket polaczenie;
    private void InicjujPlansze() {
        plansza = new int[11][11];
        klikniete = new boolean[11][11];
        int w, k, licznik;
        licznik = 0;
        while (licznik < 10) {
        w = (int)(Math.random()*9) + 1;
        k = (int)(Math.random()*9) + 1;
        if (plansza[w][k] < 100) {
        ++licznik;
        plansza[w][k] = 100;
        ++plansza[w-1][k-1];
        ++plansza[w-1][k];
        ++plansza[w-1][k+1];
        ++plansza[w][k-1];
        ++plansza[w][k+1];
        ++plansza[w+1][k-1];
        ++plansza[w+1][k];
        ++plansza[w+1][k+1];
        }
        }
    }
    public Gracz(Socket polaczenie) {
        InicjujPlansze();
```

```
    this.polaczenie = polaczenie;
    try {
    out = new PrintWriter(polaczenie.getOutputStream(), true);
    in = new BufferedReader(
    new InputStreamReader(polaczenie.getInputStream()));
    }
    catch (IOException e) {
    System.out.println(e.toString());
    }
}
public void run() {
    int w, k, odkryte, liczba;
    String wsp, odp;
    boolean koniec = false;
    odkryte=0;
    try {
        out.println("OK.");
        while ((!koniec) && (odkryte<71)) {
            wsp = in.readLine();
            if (wsp == null) koniec = true;
            else {
                try {
                    liczba = Integer.parseInt(wsp);
                    w = (int)((liczba-1)/9) + 1;
                    k = (liczba-1) % 9 + 1;
                }
                catch (NumberFormatException e) {
                    w = 200;
                    k = 200;
                }
                if ((w>=1) && (w<=9) && (k>=1) && (k<=9)) {
                    if (plansza[w][k] >= 100) {
                        out.println("bum");
                        koniec = true;
                    }
                    else {
                        out.println(Integer.toString(plansza[w][k]));
                        if (!klikniete[w][k]) ++odkryte;
                    }
                    klikniete[w][k] = true;
                }
            }
        }
    }
    catch (IOException e) {
        System.out.println(e.toString());
    }
    finally {
        try {
        polaczenie.close();
        }
        catch (IOException e) {}
        }
    }
}

public class Serwer {
    public static void main(String[] args) {
    ServerSocket server;
    try {
    server = new ServerSocket(9696);
```

```
            while(true) {
            Socket polaczenie = server.accept();
            Thread t = new Thread(
            new Gracz(polaczenie));
            t.start();
            }
            }
            catch (IOException e) {
            System.out.println(e.toString());
            }
        }
    }
```

Jest to program, który dla każdego klienta, który się z nim połączy generuje pole minowe znane z windowsowej gry „Saper". Numerację pól tego pola przedstawiono na poniższym rysunku:

```
 1  2  3  4  5  6  7  8  9
10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27
28 29 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45
46 47 48 49 50 51 52 53 54
55 56 57 58 59 60 61 62 63
64 65 66 67 68 69 70 71 72
73 74 75 76 77 78 79 80 81
```

Serwer na tym polu rozmieszcza losowo 10 min. Na każdym polu wolnym od miny umieszcza liczbę od 0 do 8 określającą ile min jest wokół niego. Napisz klienta, który:

1. połączy się z serwerem przez port 9696;
2. odbierze wiersz tekstu (słowo „OK.");

w pętli, do momentu „odkrycia" wszystkich 71 wolnych pól lub natrafienia na minę (serwer wtedy odpowiada „bum"), będzie przesyłał serwerowi liczbę x (jako String) podaną przez użytkownika i odbierał od serwera liczbę (również jako String) określającą ile jest min wokół pola x.