Języki i metody programowania Java
Lab3 – wyjątki

https://docs.oracle.com/javase/tutorial/

http://zofia.kruczkiewicz.staff.iiar.pwr.wroc.pl/wyklady/pojava/javazk5_1.pdf

Zofia Kruczkiewicz

1. Należy dokonać zmiany kodu programu typu kalkulator (zad3 z lab1) uzupełnioną o operacje mnożenia i odejmowania na postać obiektową.

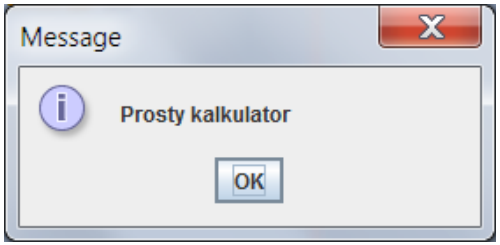
Należy zdefiniować następujące metody:

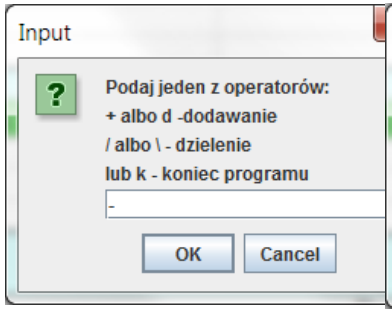
A. do wprowadzanie danych
B. do wykonania obliczeń: 4 metody
C. do wyboru obliczeń: 1 metoda
D. do wyświetlania danych: 1 metoda

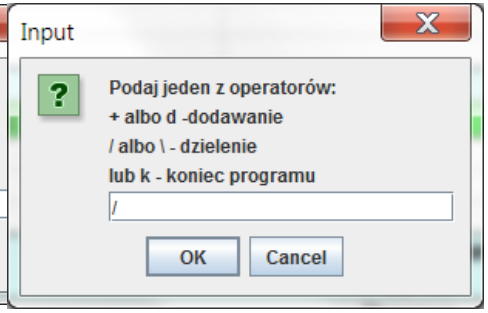
```
package instrukcje2;
import javax.swing.JOptionPane;

public class Instrukcje2 {
   public static void main(String[] args) {
      char op;
      int a, b, c;
      String s, wynik;
      JOptionPane.showMessageDialog(null, "Prosty kalkulator\n");
      do {
         s = JOptionPane.showInputDialog(null,
               "Podaj jeden z operatorów:\n"
               + "+ albo d - dodawanie\n"
               + "/ albo \\ - dzielenie\n"
               + "lub k - koniec programu");
         op = s.charAt(0);
         if (op == 'k')
            break;                          //wyjście z pętli do while
         s = JOptionPane.showInputDialog(null, "Podaj lewy argument");
         a = Integer.parseInt(s);
         s = JOptionPane.showInputDialog(null, "Podaj  prawy argument");
         b = Integer.parseInt(s);
```

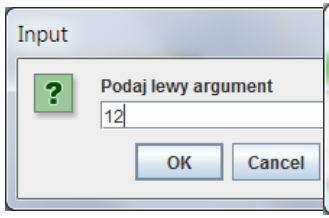
Zmiana operacji na operacje całkowitoliczbowe

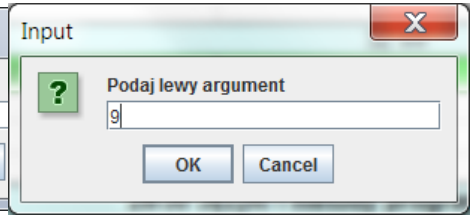
Zmiana operacji na operacje całkowitoliczbowe

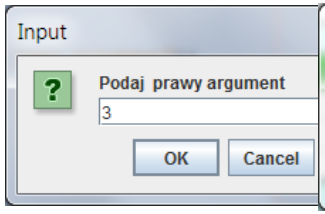


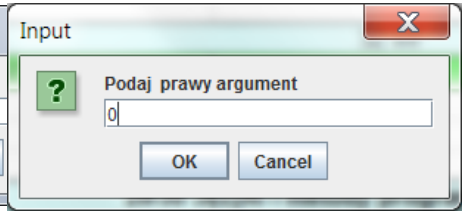




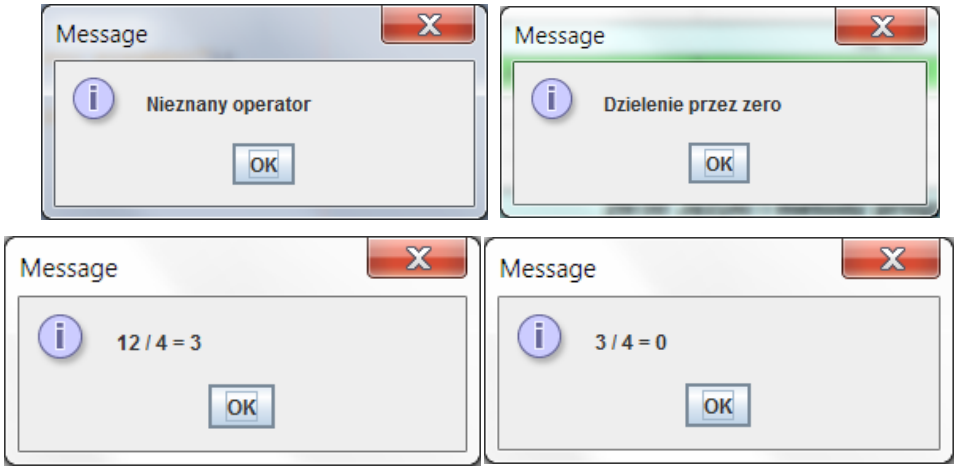








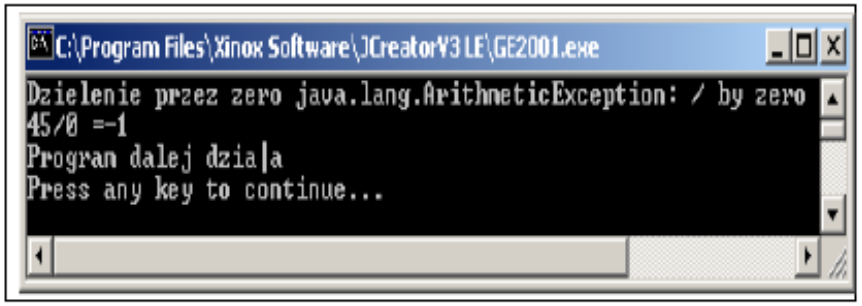
```
switch (op) {
            case '+':                          //brak instrukcji break powoduje przejście do kolejnego case
            case 'd':
               c = a + b;
               wynik = a + " " + "+" + " " + b + " = " + c;
               break;                          //break powoduje zakończenie instrukcji switch, gdy op='+' lub op ='d'
            case '/':
            case '\\':
               if (b != 0) {
                  c = a / b;
                  wynik = a + " " + op + " " + b + " = " + c;
               } else {
                  wynik = "Dzielenie przez zero";
               }
               break;                          //break powoduje zakończenie instrukcji switch, gdy op='\' lub op ='/'
            default:
               wynik = "Nieznany operator";
         }
         JOptionPane.showMessageDialog(null, wynik);
      } while (true);
      wynik = "Koniec programu";
      JOptionPane.showMessageDialog(null, wynik);
      System.exit(0);
   }
}
```



2. Należy wykonać kopię programu z p.1. W tym programie należy zastosować obsługę wyjątku wg przykładu podanego poniżej dla operacji dzielenia (metoda z p.B)

2) Przechwytywanie wyjątków przez program – kontynuowanie programu po obsłudze wyjątku – blok try... catch

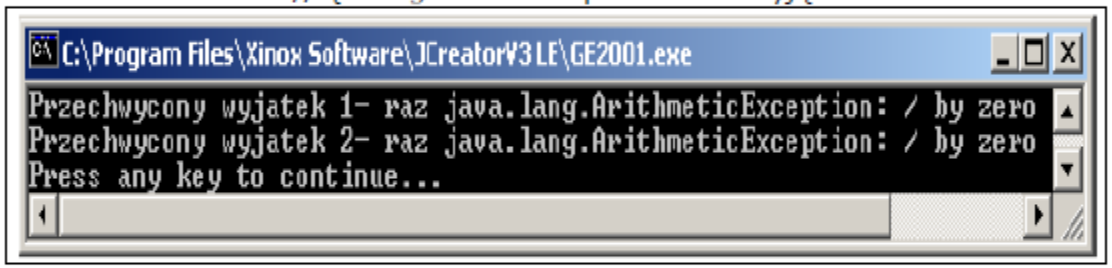
```
class Wyjatek_2              //plik Proba_2.java
{  int x;
   Wyjatek_2(int x_)
     {    x = x_;   }
   int iloraz()
     {    int p = -1;
          try
            {   p=45/x;   }                  //możliwość generowania wyjątku od dzielenia przez 0
          catch( ArithmeticException e)      //przechwycenie wyjątku
           {   System.out.println("Dzielenie przez zero "+e); }
          return p;                          //kontynuacja programu
     }
int podaj_x()
     {    return x; }
}

public class Proba_2
{
 public static void main(String ags[])
 { Wyjatek_2 w1=new Wyjatek_2(0);
   System.out.println("45/"+w1.podaj_x()+" ="+w1.iloraz());  // wystąpienie i obsługa wyjątku
   System.out.println("Program dalej działa");
 }}
```



3. Należy wykonać kopię programu z p.1. Należy zastosować obsługę wyjątku wg przykładu podanego poniżej dla operacji dzielenia, opartą na ponownym ręcznym generowaniu wyjątku (metoda z p. B i metoda z p.C).

5) Ponowne generowanie wyjątku („ręczne") - kontynuowanie programu po obsłudze wyjątku

```
class Wyjatek_5                 //plik Proba_5.java
{ static void odwrotnosc (int a)
  {   try
      { int b=1/a;  }                       //automatyczne wywołanie wyjątku, gdy a=0
      catch (ArithmeticException e)         //przechwycenie wyjątku od dzielenia przez 0
       {  System.out.println("Przechwycony wyjatek 1- raz "+e);
          throw e;                          //ręczne generowanie powtórzenia wyjątku
       }
  }
}
```




```
public class Proba_5
{
   public static void main(String ags[])
      {   try
           { Wyjatek_5.odwrotnosc(0); }//zagnieżdżona obsługa wyjątku -  wymuszona obsługa wyjątku - throw
             catch (ArithmeticException e)
           {  System.out.println("Przechwycony wyjatek 2- raz "+e);}
      }
}
```

Autor: Zofia Kruczkiewicz, Programowanie obiektowe – Java, wykład 6 10

4. Należy wykonać kopię programu z p.1. Należy zastosować obsługę wyjątku wg przykładu podanego poniżej dla operacji dzielenia, opartą na ręcznym generowaniu wyjątku (metoda z p. B i metoda z p.C).

6) Generowanie wyjątku („ręczne") - kontynuowanie programu po obsłudze wyjątku

klauzula throw *Wystapienie_klasy_pochodnej_Throwable*

```
class Wyjatek_6                  //plik Proba_6.java
{  static void odwrotnosc (int a)
    { try
            { if (a>1)
              throw new ArithmeticException("Generowanie wyjatku"); }// ręczne generowanie wyjątku
      catch (ArithmeticException e)
            {    System.out.println("Przechwycony wyjatek 1- raz "+e);
                 throw e;
            }    //ręczne generowanie powtórzenia wyjątku

  }
}
```


```
public class Proba_6
{
  public static void main(String ags[])
    {  try
       {  Wyjatek_6.odwrotnosc(2);   }      //zagnieżdżona obsługa wyjątku - wymuszona obsługa wyjątku - throw
       catch (ArithmeticException e)
            {    System.out.println("Przechwycony wyjatek 2- raz "+e);}
    }
 }
```

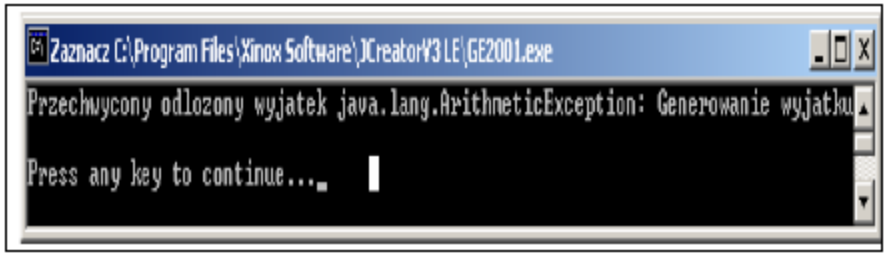
Autor: Zofia Kruczkiewicz, Programowanie obiektowe – Java, wykład 6 11

5. Należy wykonać kopię programu z p.1. Należy zastosować obsługę wyjątku wg przykładu podanego poniżej dla operacji dzielenia, opartą na zastosowaniu klauzuli **throws**. W metodzie dokonującej obliczenia powinna wystąpić pierwsza obsługa wyjątku (p.B), a ponownie obsłużona w metodzie z p.C.

7) Przekazanie obsługi wyjątku do innej części programu – klauzula throws

typ nazwa metody (lista_parametrów) **throws** *lista_wyjątków*

```
class Wyjatek_7                  //plik Proba_7.java
 { static void odwrotnosc (int a )  throws Exception
    { if (a>1)
       throw new ArithmeticException ("Generowanie wyjatku"); }
 }

public class Proba_7
  { public static void main(String ags[])
    { try
      {  Wyjatek_7.odwrotnosc(2); }
     catch (Exception e)
     { System.out.println("Przechwycony odlozony wyjatek "+e); }
    }
  }
```



Obowiązkowa obsługa wyjątków w miejscu wywołania metody *odwrotnosc()* dotyczy grupy wyjątków bezpośrednich (*explicite exception* użytych w klauzuli **throws**. Typ wyjątku w bloku **catch** musi być albo identycznej klasy użytej w **throws** lub klasy, od której dziedziczy klasa wyjątku użyta w **throws**.

Zasada ta nie dotyczy wyjątków typu pośredniego (*implicit exception*) , czyli:

- Error (np. OutOfMemoryError)
- RuntimeException (np. ArithmeticException)

oraz dziedziczących od tych klas.

Z tej zasady wynika, że klasy bazowe dla obu typów wyjątków np. *Exception*, są wyjątkami bezpośrednimi

Autor: Zofia Kruczkiewicz, Programowanie obiektowe – Java, wykład 6 12

6. Należy wykonać kopię programu z p.1. Należy zastosować obsługę wyjątku wg przykładu podanego poniżej dla operacji mnożenia, wykorzystując zagnieżdżone bloki generujące wyjątki (generowanie wyjątku w metodzie dokonujące obliczenia B, a obsłużone w metodzie prezentującej wynik wywołanej metody obliczeniowej D.

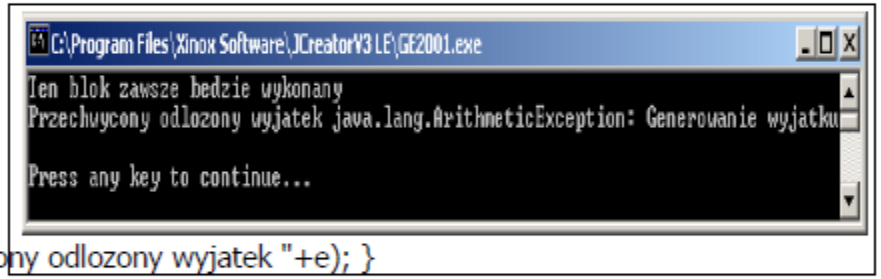
8) Wyjątki generowane w bloku zagnieżdżonym w bloku try, mogą być obsłużone w jego bloku catch

```
class Wyjatek_7_1            //plik Proba_7.java
 {
     static void odwrotnosc (int a)
       { if (a>1)
          throw new ArithmeticException("Generowanie wyjatku");
       }
      static void oblicz(int b)
       {  odwrotnosc(b); }

  }

public class Proba_7_1
 {
public static void main(String ags[])
 {   try
     {
       Wyjatek_7_1.oblicz(2);
     }
     catch (Exception e)
      {    System.out.println("Przechwycony odlozony wyjatek "+e);}
 }
}
```

7. Należy wykonać kopię programu z p.1. Należy zastosować obsługę wyjątku wg przykładu podanego poniżej dla operacji dzielenia, opartą na zastosowaniu klauzuli **throws** z zastosowaniem klauzuli **finally** – metody B i C

9) Wykonanie wskazanej części metody po bloku try po wystąpieniu w niej wyjątku lub przy braku jego wystąpienia - klauzula finally

```
class Wyjatek_8                    //plik Proba_8.java
  {   static void odwrotnosc (int a) throws Exception
        { try
           {  if (a>1)
              throw new ArithmeticException("Generowanie wyjatku");   }
           finally  // (zamiast catch) wykonanie instrukcji po wystąpieniu wyjątku lub bez wystąpienia wyjątku
             {  System.out.println("Ten blok zawsze bedzie wykonany");          }
        }
  }
public class Proba_8
{ public static void main(String args[])
    { try
         { Wyjatek_8.odwrotnosc(2);  }
      catch (Exception e)
        {   System.out.println("Przechwycony odlozony wyjatek "+e); }
    }
}
```




```
public class Proba_8
{ public static void main(String args[])
    { try
         { Wyjatek_8.odwrotnosc(1);  }
      catch (Exception e)
        {   System.out.println("Przechwycony odlozony wyjatek "+e); }
    }}
```

Autor: Zofia Kruczkiewicz, Programowanie obiektowe – Java, wykład 6 14