

# KLASY, INTERFEJSY, ITP

## ZAGADNIENIA:

- Klasy, modyfikatory dostępu, pakiety.
- Zmienne i metody statyczne.
- Klasy abstrakcyjne, dziedziczenie.
- Interfejsy.
- Komentarze i javadoc,



# DWIE KLASY

```
public class Klasa1{  
    public void metoda1(){  
        ...  
    }  
    ...  
}
```

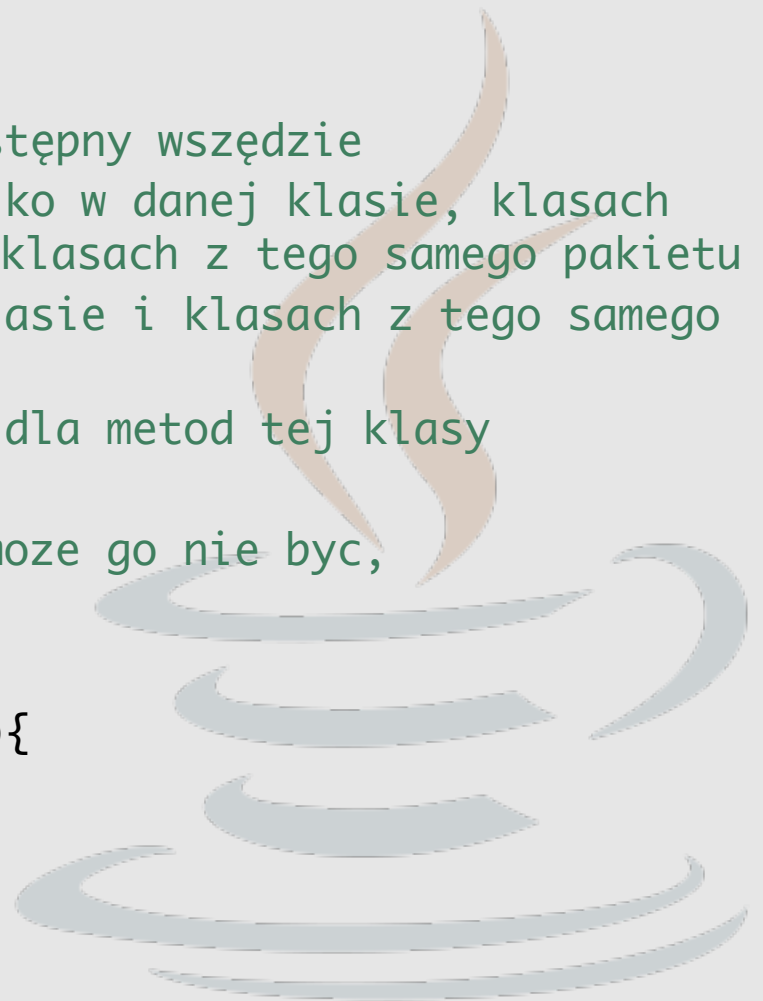
```
...  
}
```

```
public class Klasa2{  
    ...  
    public void metoda2(){  
        Klasa1 k1;  
        k1 = new Klasa1(...);  
        k1.metoda1();  
    }  
}
```



# KLASY

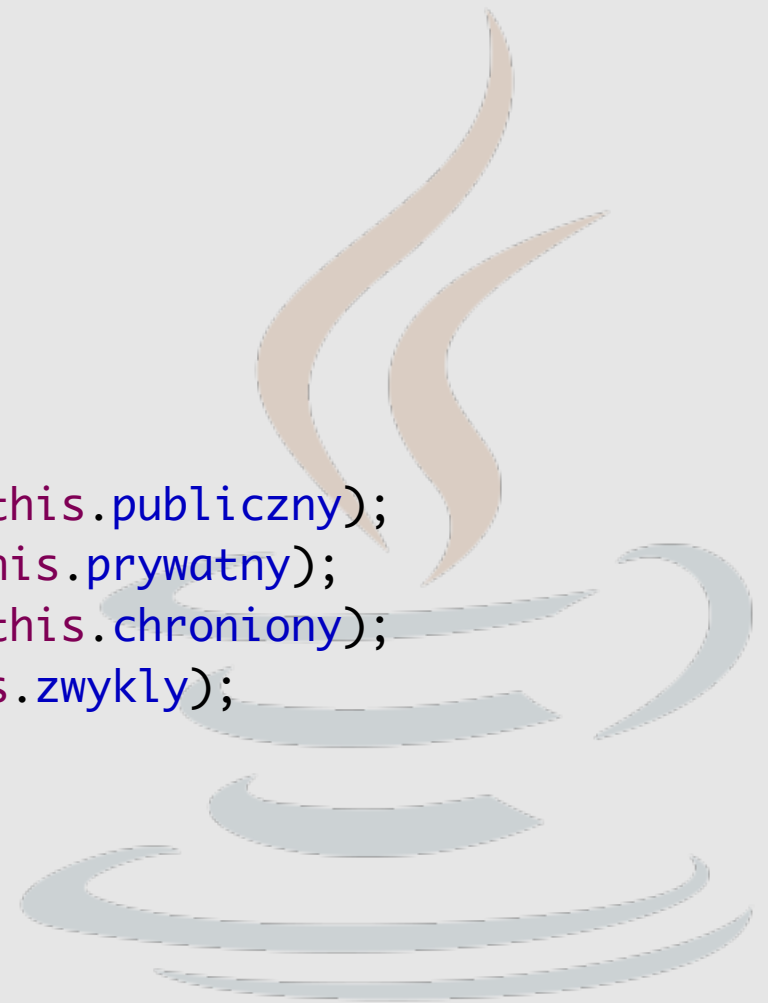
```
package pakiet.podpakiet;
public class Klasa {
    public int publiczny; // public - dostępny wszędzie
    protected int chroniony; // dostępny tylko w danej klasie, klasach
    // potomnych i klasach z tego samego pakietu
    int zwykly; // dostępny tylko w danej klasie i klasach z tego samego
    // pakietu
    private int prywatny; // dostępny tylko dla metod tej klasy
    protected Klasa(){
        // konstruktor może nic nie robić, może go nie być,
        // nie musi być publiczny
    }
    public Klasa(int a, int b, int c, int d){
        this.publiczny = a;
        this.prywatny = b;
        this.chroniony = c;
        this.zwykly = d;
    }
}
```



# KLASY

```
public void set(){  
    this.publiczny = 7;  
    this.prywatny = 13;  
    this.chroniony = 27;  
    this.zwykly = 11;  
}
```

```
public void print(){  
    System.out.println("publiczny: " + this.publiczny);  
    System.out.println("prywatny: " + this.prywatny);  
    System.out.println("chroniony: " + this.chroniony);  
    System.out.println("zwykly: " + this.zwykly);  
    System.out.println();  
}
```



# KLASY

**Access Levels**

<b>Modifier</b>	<b>Class</b>	<b>Package</b>	<b>Subclass</b>	<b>World</b>
<code>public</code>	Y	Y	Y	Y
<code>protected</code>	Y	Y	Y	N
<i>no modifier</i>	Y	Y	N	N
<code>private</code>	Y	N	N	N

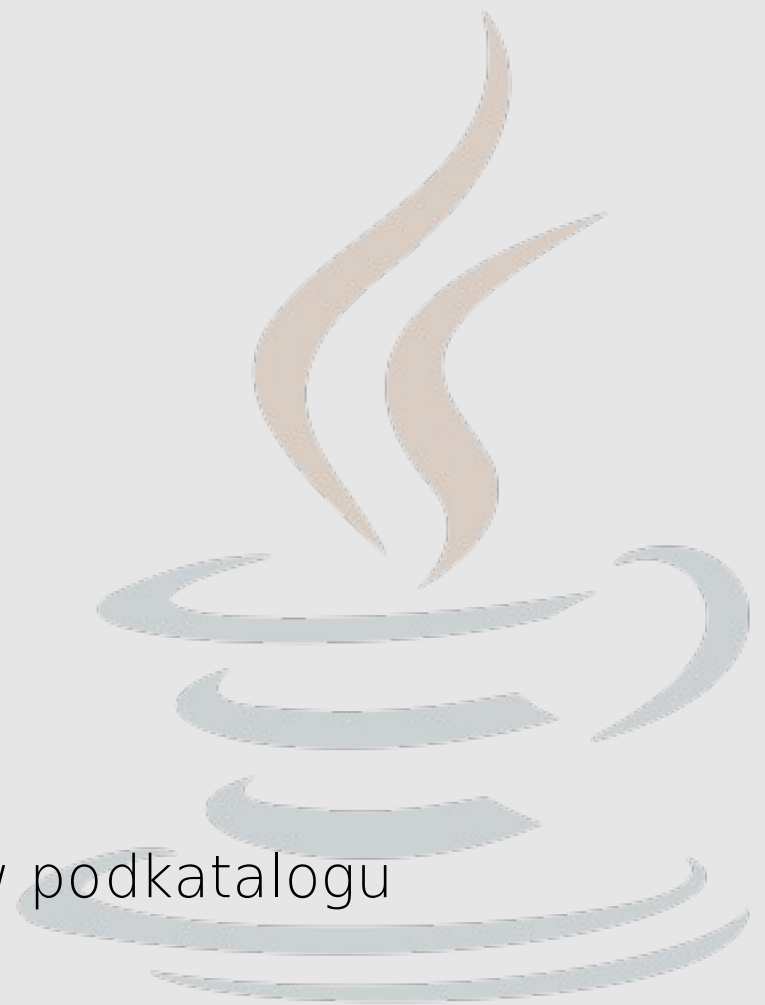
# KLASY

```
public static void main(String args[]){
    Klasa k1 = new Klasa();
    k1.print();
    k1.set();
    k1.print();
    Klasa k2 = new Klasa(1,2,3,4);
    k2.print();
}
}
```

## URUCHOMIENIE:

java pakiet.podpakiet.Klasa

plik `Klasa.class` musi sie znajdowac w podkatalogu  
`./pakiet/podpakiet/`



# PAKIETY

Klasy można grupować w pakiety. Nazwa pakietu, do którego należy klasa jest podana w pliku definiującym klasę:

```
package pakiet.podpakiet;
```

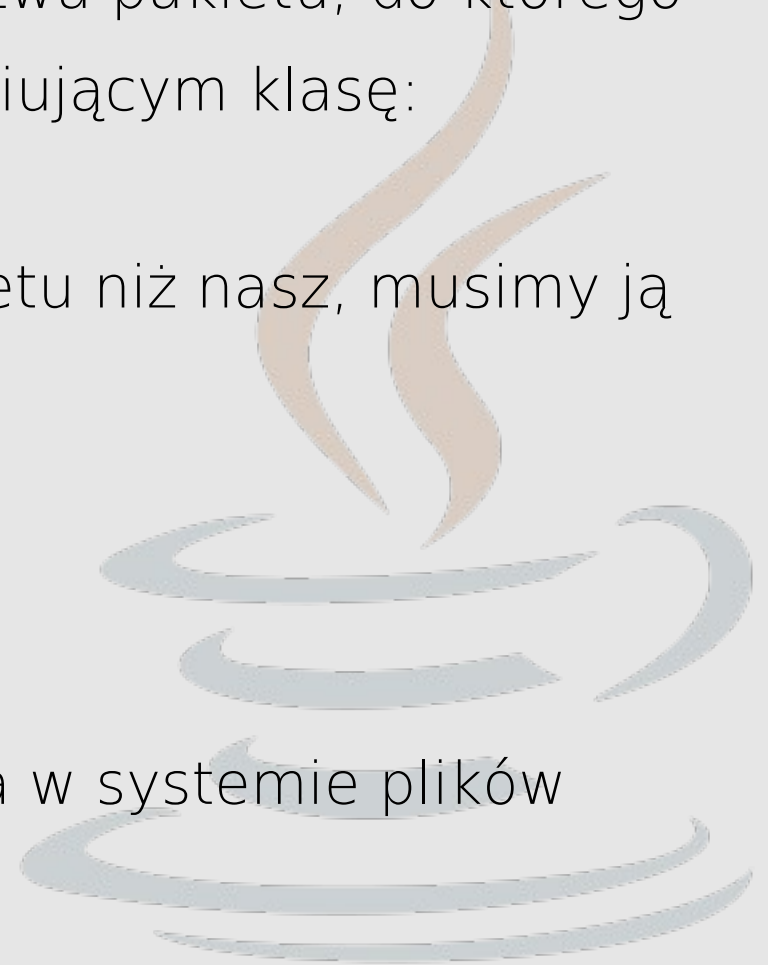
Jeśli chcemy użyć klasy z innego pakietu niż nasz, musimy ją uprzednio zaimportować:

```
import pakiet.podpakiet.Klasa;
```

lub

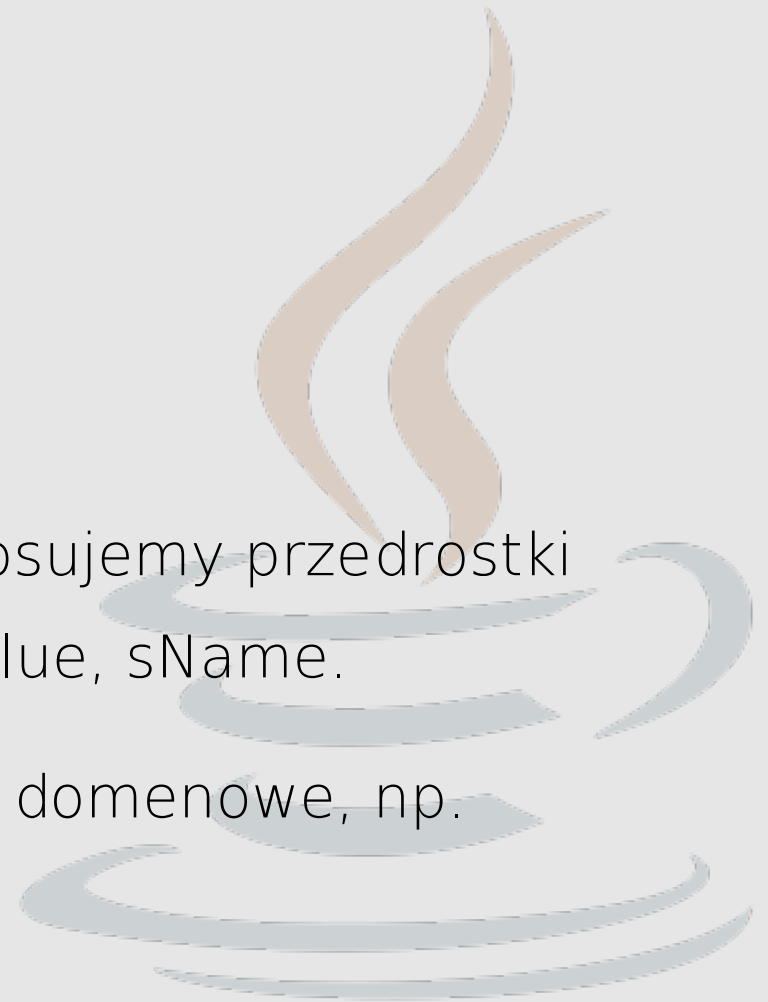
```
import pakiet.podpakiet.*;
```

hierarchia pakietów jest odwzorowana w systemie plików w hierarchii katalogów.



# KLASY - KONWENCJE

- nazwa pakietu: z **małej** litery,
- nazwa klasy: z **DUŻEJ** litery,
- nazwa atrybutu: z **małej** litery,
- nazwa metody: z **małej** litery,
- nazwa zmiennej: z **małej** litery, stosujemy przedrostki określające typ zmiennych, np. `iValue`, `sName`.
- nazwy pakietów – odwrotne nazwy domenowe, np.  
`pl.edu.uj.fais.java.wyklad2`





# ZMIENNE I METODY STATYCZNE

```
public class Klasa1{
    public static void metoda1(){
        ...
    }
}
public class Klasa2{
    ...
    public void metoda2(){
        Klasa1.metoda1();
    }
}
```

Atrybuty i metody statyczne są związane z klasą a nie z jej instancjami (obiektami). Metody statyczne nie mogą więc bezpośrednio wywoływać zwykłych metod lub korzystać ze zwykłych atrybutów, gdyż one są określone wyłącznie w kontekście obiektów.

# ZMIENNE I METODY STATYCZNE

```
public class Klasa1{  
...  
    static{ // jawna inicjalizacja zmiennych statycznych  
        ...  
    }  
...  
}
```

Atrybuty statyczne są inicjalizowane bezpośrednio po załadowaniu klasy przez JVM.

Atrybuty zwykłe (niestatyczne) są inicjowane w momencie utworzenia obiektu (wywołania konstruktora). Zadeklarowane a niezainicjowane atrybuty są ustawiane na  $\emptyset$  lub `null`.

# ZMIENNE I METODY STATYCZNE

```
public class OrderTest {  
  
    static{  
        System.out.println("static");  
    }  
  
    public OrderTest(){  
        System.out.println("constructor");  
    }  
  
    public static void main(String[] args){  
        System.out.println("main: begin");  
        OrderTest o;  
        System.out.println("main: middle");  
        o = new OrderTest();  
        System.out.println("main: end");  
    }  
}
```



# KLASY ABSTRAKCYJNE

```
public abstract class AbstractClass {  
    ...  
    public abstract int doSomething();  
    public int doSomethingElse(){  
        ...  
    }  
    ...  
}
```

```
public class SpecificClass extends AbstractClass{  
    public int doSomething(){  
        ...  
    }  
}
```

Klasa abstrakcyjna to klasa, której jedna z metod jest abstrakcyjna.  
Nie można bezpośrednio tworzyć instancji klasy abstrakcyjnej.

# DZIEDZICZENIE

```
public class AnyClass extends AnotherClass{  
    ...  
}
```

Klasa może mieć tylko jednego, bezpośredniego rodzica (inaczej niż w C++). Jeśli klasa nie posiada rodzica, dziedziczy automatycznie po klasie Object (java.lang.Object). W związku z tym instancja dowolnej klasy jest obiektem (instancją klasy Object).

# INTERFEJSY

```
public interface InterfaceExample {
    public void method1();
    public int method2(double i);
    public AnotherInterface method3(AnyClass ac, AnyInterface ai);
}

public class ImplementationClass implements InterfaceExample, OtherIf {
    public void method1(){
        ...
    }
    ...
}
```

Interfejsy posiadają hierarchię dziedziczenia, jednak jest ona zupełnie niezależna od hierarchii klas.

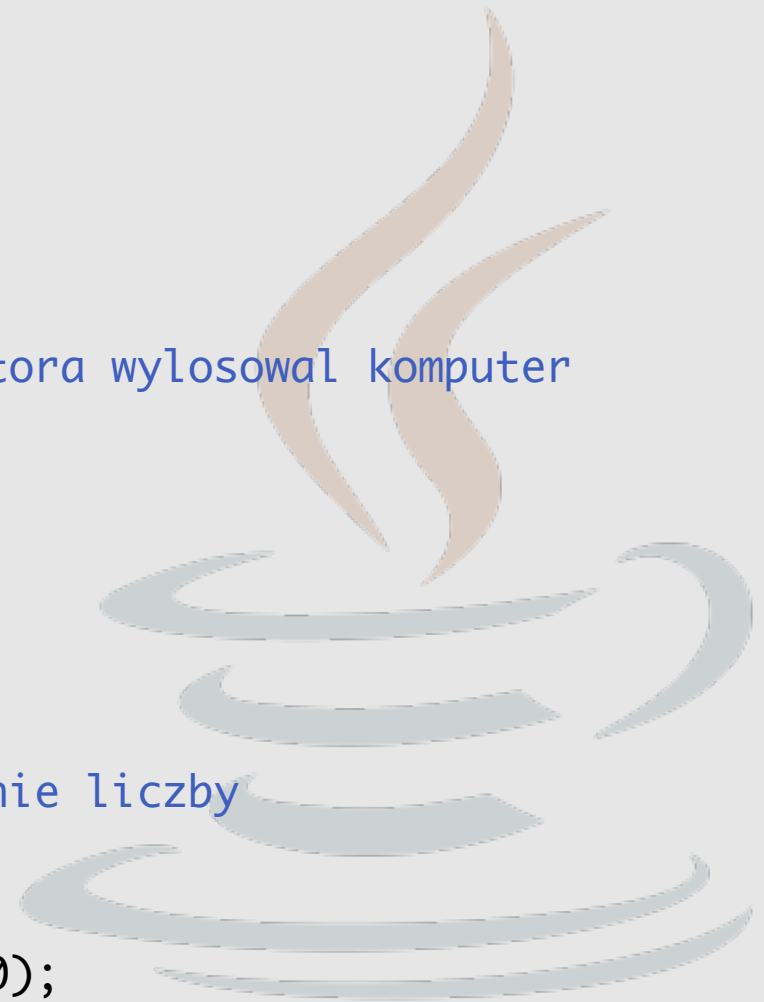
# JAVADOC

## TryAndCheck.java

```
import java.io.IOException;

/**
 * Klasa umożliwiająca zgadywanie liczby, która wylosował komputer
 * @author Kubus Puchatek
 */
public class TryAndCheck {
    private int number;

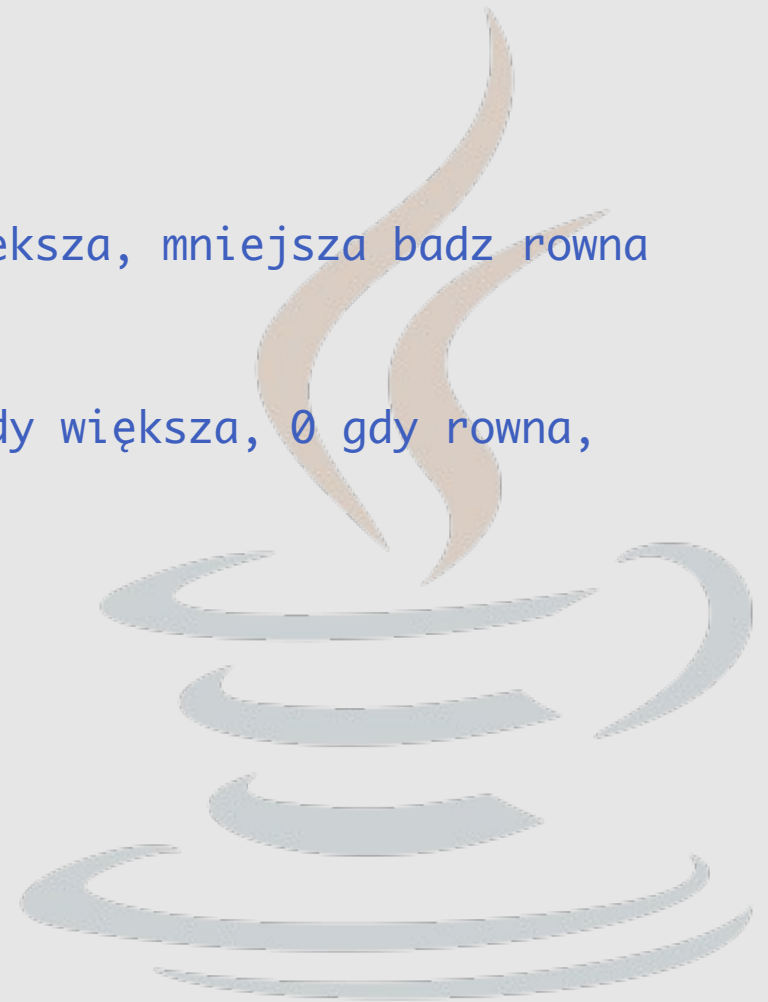
    /**
     * konstruktor, w nim odbywa sie losowanie liczby
     */
    public TryAndCheck(){
        this.number = (int)(Math.random()*10);
    }
}
```



# JAVADOC

## TryAndCheck.java (c.d.)

```
/**
 * sprawdza, czy podana wartosc jest wieksza, mniejsza badz rowna
 * wylosowanej liczbie
 * @param iv
 * @return -1 gdy iv jest mniejsza, 1 gdy wieksza, 0 gdy rowna,
 */
public byte check(int iv){
    if (iv<this.number) return -1;
    if (iv>this.number) return +1;
    return 0;
}
```



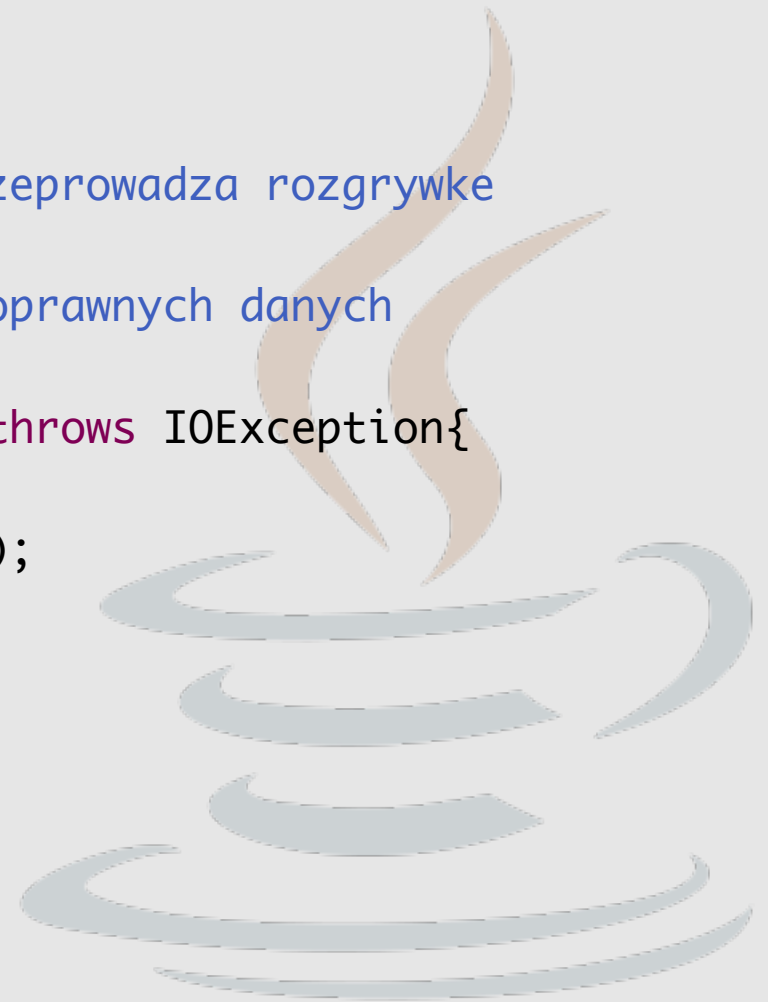


# JAVADOC

## TryAndCheck.java (c.d.)

```
/**
 * metoda uruchamiana automatycznie. Przeprowadza rozgrywkę
 * @param args nieobsługiwane
 * @throws IOException w przypadku niepoprawnych danych
 */
public static void main(String[] args) throws IOException{

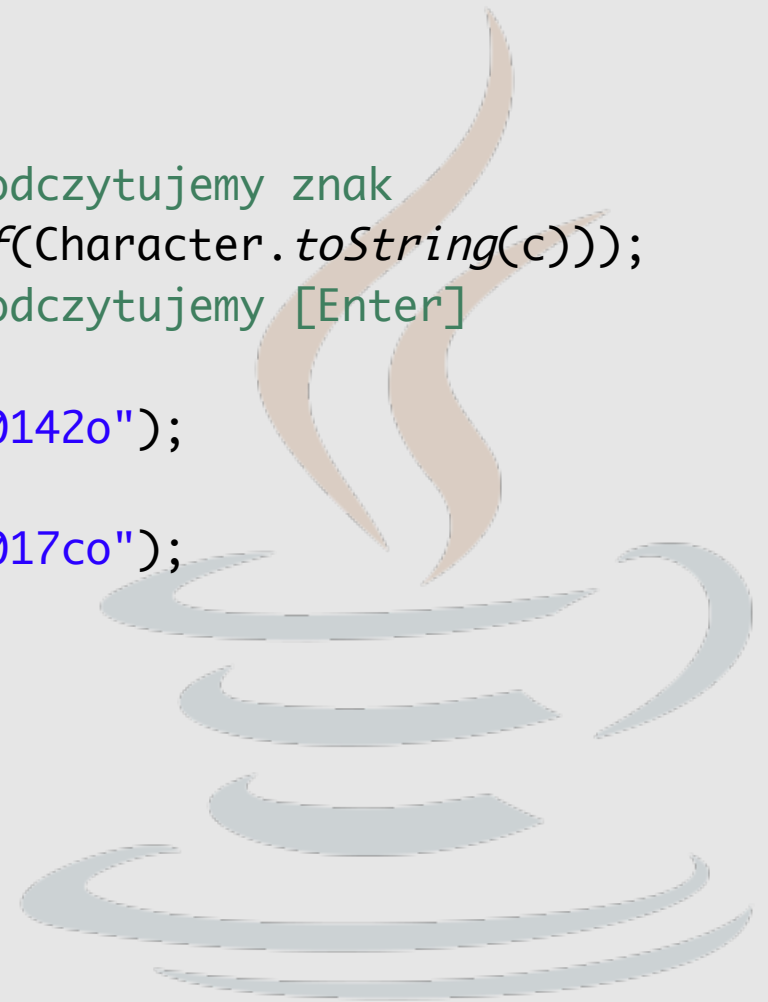
    TryAndCheck play = new TryAndCheck();
    int res;
    char c;
```



# JAVADOC

TryAndCheck.java (c.d.)

```
do{
    c = (char)System.in.read(); // odczytujemy znak
    res = play.check(Integer.valueOf(Character.toString(c)));
    c = (char)System.in.read(); // odczytujemy [Enter]
    if(res<0)
        System.out.println("Za ma\u0142o");
    if(res>0)
        System.out.println("Za du\u017co");
}while(res!=0);
System.out.println("Gratulacje");
}
}
```



# JAVADOC

AUTOMATYCZNE GENEROWANIE DOKUMENTACJI

javadoc TryAndCheck.java

The screenshot shows a web browser window titled "TryAndCheck" displaying the Javadoc documentation for the `TryAndCheck` class. The browser's address bar shows the file path: `file:///Users/ciesla/Documents/workspace/java-01/src/console/index.html`. The page content includes:

- All Classes**: A sidebar with a link to `TryAndCheck`.
- Class Hierarchy**: A diagram showing `java.lang.Object` as the superclass of `console.TryAndCheck`.
- Class Declaration**:

```
public class TryAndCheck
extends java.lang.Object
```
- Description**: "Klasa umożliwiaiąca zgadywanie liczby, która wylosował komputer"
- Constructor Summary**:
  - `TryAndCheck()`: konstruktor, w nim odbywa się losowanie liczby
- Method Summary**:

byte	<code>check(int iv)</code>	sprawdza, czy podana wartość jest większa, mniejsza bądź równa wylosowanej liczbie
static void	<code>main(java.lang.String[] args)</code>	metoda uruchamiana automatycznie.
- Methods inherited from class java.lang.Object**: `clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`
- Constructor Detail**:

```
TryAndCheck
public TryAndCheck()
konstruktor, w nim odbywa się losowanie liczby
```

# ĆWICZENIA

- Proszę zaimplementować klasę reprezentującą liczbę zespoloną, posiadającą metody umożliwiające dodawanie, mnożenie i dzielenie takich liczb,
- Proszę zaimplementować klasę abstrakcyjną reprezentującą funkcję jednej zmiennej posiadającą metodę abstrakcyjną **double getValue(double x)** oraz zwykłą metodę **double findMinimum(double a, double b)**, która znajduje lokalne minimum funkcji w przedziale [a,b]

DZIĘKUJĘ ZA UWAGĘ