Faculty of Electronics and Information Technology
WARSAW UNIVERSITY OF TECHNOLOGY

Programowanie Aplikacyjne (PAP)

Java - wprowadzenie

Julian Myrcha
Institute of Computer Science
26.02.2025

- Java jest nowoczesnym językiem ogólnego przeznaczenia, w pierwszej trójce popularności

- Java jest nowoczesnym językiem ogólnego przeznaczenia, w pierwszej trójce popularności
- Java jest kompilowana do kodu pośredniego, który jest następnie kompilowany na maszynie na której jest wykonywany

- Java jest nowoczesnym językiem ogólnego przeznaczenia, w pierwszej trójce popularności
- Java jest kompilowana do kodu pośredniego, który jest następnie kompilowany na maszynie na której jest wykonywany
- Szerokie wykorzytanie w korporacjach

- Java jest nowoczesnym językiem ogólnego przeznaczenia, w pierwszej trójce popularności
- Java jest kompilowana do kodu pośredniego, który jest następnie kompilowany na maszynie na której jest wykonywany
- Szerokie wykorzytanie w korporacjach
- obecnie właścicielem patentów jest firma Oracle

- J2SE 1.2 December 8, 1998
- J2SE 1.3 May 8, 2000
- J2SE 1.4 February 6, 2002
- J2SE 5.0 September 30, 2004
- Java SE 6 December 11, 2006
- Java SE 7 July 28, 2011
- **Java SE 8 March 18, 2014 LTS**
- Java SE 9 September 21, 2017
- Java SE 10 March 20, 2018
- **Java SE 11 September 25, 2018 LTS**
- Java SE 12 March 19, 2019
- Java SE 13 September 17, 2019
- Java SE 14 March 17, 2020
- Java SE 15 September 15, 2020
- Java SE 16 March 15, 2021
- **Java SE 17 September 15, 2021 LTS**
- Java SE 18 March 15, 2022
- Java SE 19 September 15, 2022
- Java SE 20 March 15, 2023
- **Java SE 21 September 15, 2023 LTS**
- Java SE 22 March 15, 2023
- Java SE 23 September 15, 2024

Java to nie tylko Java SE (Standard Edition)

- Jakarta EE, dawniej Java Platform, Enterprise Edition (Java EE) i Java 2 Platform, Enterprise Edition (J2EE)
- rozszerza Java SE o specyfikacje funkcji korporacyjnych, takich jak przetwarzanie rozproszone i usługi internetowe.

Historia:

- J2EE 1.2 (1999)
- J2EE 1.4 (2003)
- Java EE 5 (2006)
- Java EE 6 (2009)
- Java EE 7 (2013)
- Java EE 8 (2017)
- Jakarta EE 8 (2019) - całkowicie zgodne z Java EE 8
- Jakarta EE 9 (2020) - zmiana przestrzeni javax.* to jakarta.*

- Eclipse - popularne darmowe IDE do wszystkiego
  - kilkadziesiąt wersji, każda to zbiór wielu wtyczek
  - możliwość samodzielnego złożenia IDE z wtyczek

- Eclipse - popularne darmowe IDE do wszystkiego
  - kilkadziesiąt wersji, każda to zbiór wielu wtyczek
  - możliwość samodzielnego złożenia IDE z wtyczek
- IntelliJ IDEA - ide popularne w korporacjach
  - IntelliJ IDEA Ultimate - wersja pełna, płatna (ale z domeny *.edu można utworzyć roczne darmowe konto)
  - IntelliJ IDEA Community

- Eclipse - popularne darmowe IDE do wszystkiego
  - kilkadziesiąt wersji, każda to zbiór wielu wtyczek
  - możliwość samodzielnego złożenia IDE z wtyczek
- IntelliJ IDEA - ide popularne w korporacjach
  - IntelliJ IDEA Ultimate - wersja pełna, płatna (ale z domeny *.edu można utworzyć roczne darmowe konto)
  - IntelliJ IDEA Community
- Visual Studio Code
  - darmowy, lekki, dobrze zaprojektowany
  - Java dodana za pomocą wtyczek

- Eclipse - popularne darmowe IDE do wszystkiego
  - kilkadziesiąt wersji, każda to zbiór wielu wtyczek
  - możliwość samodzielnego złożenia IDE z wtyczek
- IntelliJ IDEA - ide popularne w korporacjach
  - IntelliJ IDEA Ultimate - wersja pełna, płatna (ale z domeny *.edu można utworzyć roczne darmowe konto)
  - IntelliJ IDEA Community
- Visual Studio Code
  - darmowy, lekki, dobrze zaprojektowany
  - Java dodana za pomocą wtyczek
- Netbeans
  - popularne bardziej do nauki niż w przemyśle
  - zaawansowany (np. wizualna edycja Swinga)

podstawowe pojęcia:

Java Virtual Machine (np. Oraclowy HotSpot) - wykonuje kod pośredni

JRE - zawiera JVM, instalujemy gdy chcemy wykonywać programy (ale nie zawiera kompilatorów i debuggera)

JDK - zawiera JVM, instalujemy gdy chcemy tworzyć i wykonywać programy

Java SE - Java Platform Standard Edition (do aplikacji Desktop)

Java EE (Jakarta EE) - Java Platform Enterprise Edition (do aplikacji korporacyjnych)

- Wykonywany w maszynie pośredniej, standardowe środowisko (biblioteki) daje łatwą przenośność między architekturami.

- Wykonywany w maszynie pośredniej, standardowe środowisko (biblioteki) daje łatwą przenośność między architekturami.
- Obiektowy - brak globalnych zmiennych i metod

- Wykonywany w maszynie pośredniej, standardowe środowisko (biblioteki) daje łatwą przenośność między architekturami.
- Obiektowy - brak globalnych zmiennych i metod
- Wszystkie klasy dziedziczą z klasy Object

- Wykonywany w maszynie pośredniej, standardowe środowisko (biblioteki) daje łatwą przenośność między architekturami.
- Obiektowy - brak globalnych zmiennych i metod
- Wszystkie klasy dziedziczą z klasy Object
- Jednodnokrotne dziedziczenie

- Wykonywany w maszynie pośredniej, standardowe środowisko (biblioteki) daje łatwą przenośność między architekturami.
- Obiektowy - brak globalnych zmiennych i metod
- Wszystkie klasy dziedziczą z klasy Object
- Jednodnokrotne dziedziczenie
- Pojęcie interfejsu - definicja funkcjonalności ale bez implemntacji(*)

- Wykonywany w maszynie pośredniej, standardowe środowisko (biblioteki) daje łatwą przenośność między architekturami.
- Obiektowy - brak globalnych zmiennych i metod
- Wszystkie klasy dziedziczą z klasy Object
- Jednodnokrotne dziedziczenie
- Pojęcie interfejsu - definicja funkcjonalności ale bez implemntacji(*)
- Obiekty zawsze są referencją - czyli trzeba je utworzyć za pomocą `new`

- Wykonywany w maszynie pośredniej, standardowe środowisko (biblioteki) daje łatwą przenośność między architekturami.
- Obiektowy - brak globalnych zmiennych i metod
- Wszystkie klasy dziedziczą z klasy Object
- Jednodnokrotne dziedziczenie
- Pojęcie interfejsu - definicja funkcjonalności ale bez implemntacji(*)
- Obiekty zawsze są referencją - czyli trzeba je utworzyć za pomocą new
- Pamięć zarządzana automatycznie - sama się zwalnia (chyba że zrobimy pętlę zależności)

- Wykonywany w maszynie pośredniej, standardowe środowisko (biblioteki) daje łatwą przenośność między architekturami.
- Obiektowy - brak globalnych zmiennych i metod
- Wszystkie klasy dziedziczą z klasy Object
- Jednodnokrotne dziedziczenie
- Pojęcie interfejsu - definicja funkcjonalności ale bez implemntacji(*)
- Obiekty zawsze są referencją - czyli trzeba je utworzyć za pomocą new
- Pamięć zarządzana automatycznie - sama się zwalnia (chyba że zrobimy pętlę zależności)
- Część zasobów trzeba jednak zwolnić samodzielnie

- Wykonywany w maszynie pośredniej, standardowe środowisko (biblioteki) daje łatwą przenośność między architekturami.
- Obiektowy - brak globalnych zmiennych i metod
- Wszystkie klasy dziedziczą z klasy Object
- Jednodnokrotne dziedziczenie
- Pojęcie interfejsu - definicja funkcjonalności ale bez implemntacji(*)
- Obiekty zawsze są referencją - czyli trzeba je utworzyć za pomocą new
- Pamięć zarządzana automatycznie - sama się zwalnia (chyba że zrobimy pętlę zależności)
- Część zasobów trzeba jednak zwolnić samodzielnie
- Struktura pakietów odwzorowana w położeniu plików źródłowych

- Nazwy klas z dużej litery

```
@Entity
public class Person implements Serializable {
  private static final long serialVersionUID = -3741264748388489528L;
  private int id;
  public int getId() { return id; }
  public void setId(int id) { this.id = id; }
  public static void main(String[] args) {
  }
}
```

- Nazwy klas z dużej litery
- Nazwy metod z małej litery

```java
@Entity
public class Person implements Serializable {
  private static final long serialVersionUID = -3741264748388489528L;
  private int id;
  public int getId() { return id; }
  public void setId(int id) { this.id = id; }
  public static void main(String[] args) {
  }
}
```

- Nazwy klas z dużej litery
- Nazwy metod z małej litery
- `public` przed `static`

```java
@Entity
public class Person implements Serializable {
  private static final long serialVersionUID = -3741264748388489528L;
  private int id;
  public int getId() { return id; }
  public void setId(int id) { this.id = id; }
  public static void main(String[] args) {
  }
}
```

- Nazwy klas z dużej litery
- Nazwy metod z małej litery
- `public` przed `static`
- Parametr metody `main()` nazywamy `args`

```java
@Entity
public class Person implements Serializable {
  private static final long serialVersionUID = -3741264748388489528L;
  private int id;
  public int getId() { return id; }
  public void setId(int id) { this.id = id; }
  public static void main(String[] args) {
  }
}
```

- Nazwy klas z dużej litery
- Nazwy metod z małej litery
- `public` przed `static`
- Parametr metody `main()` nazywamy `args`
- Stałe piszemy dużą literą (przeważnie ;-)

```
1 @Entity
2 public class Person implements Serializable {
3   private static final long serialVersionUID = -3741264748388489528L;
4   private int id;
5   public int getId() { return id; }
6   public void setId(int id) { this.id = id; }
7   public static void main(String[] args) {
8   }
9 }
```

- Nazwy klas z dużej litery
- Nazwy metod z małej litery
- `public` przed `static`
- Parametr metody `main()` nazywamy `args`
- Stałe piszemy dużą literą (przeważnie ;-)
- `używamyNotacjiWielbłądziej`

```
1 @Entity
2 public class Person implements Serializable {
3   private static final long serialVersionUID = -3741264748388489528L;
4   private int id;
5   public int getId() { return id; }
6   public void setId(int id) { this.id = id; }
7   public static void main(String[] args) {
8   }
9 }
```

```
+    -    *    /    %    ^    &    |    !    ~
=    +=   -=   *=   /=   %=    ^=    ++    --
==   !=    <    >    &=  >>=   <<=   >=   <=
||   &&    >>    <<    ?:
```

Słowa kluczowe

```
abstract    continue    for         new         switch
assert      default     goto        package     synchronized
boolean     do          if          private     this
break       double      implements  protected   throw
byte        else        import      public      throws
case        enum        instanceof  return      transient
catch       extends     int         short       try
char        final       interface   static      var
class       finally     long        strictfp    void
const       float       native      super       volatile
                                                while
```

literały (wartości)

```
String lecture = "pap";
int age = 99;
```

separatory

```
[ ]  ( )  { }  ,  ;  .  "
```

na przykład:

```
String language = "iipw";
```

```java
package pap;

public class CmdLine {
  public static void main(String[] args) {
    for (String arg : args) {
      System.out.println(arg);
    }
  }
}
```

Aby to przetestować w vs code wybieramy trójkąt i tam create launch.json file

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "java",
      "name": "Debug (Launch) - Current File",
      "request": "launch",
      "mainClass": "${file}"
    },{
      "type": "java",
      "args": "ala ma kota",
      "name": "Debug (Launch)-CmdLine<SimpleSwing_8edcdf2a>",
      "request": "launch",
      "mainClass": "pap.CmdLine",
      "projectName": "SimpleSwing_8edcdf2a"
    },
    ...
  ]
}
```

- Język statycznie typowany - zmienna zawsze przechowuje wartość tego samego typu

- Język statycznie typowany - zmienna zawsze przechowuje wartość tego samego typu
- Mamy 2 grupy typów

  prymitywne - boolean, char, byte, short, int, long, float, double

- Język statycznie typowany - zmienna zawsze przechowuje wartość tego samego typu
- Mamy 2 grupy typów

  prymitywne - boolean, char, byte, short, int, long, float, double

Liczby całkowite:

byte - 8 bitów od -128 do 127

short - 16 bitów od -32,768 do 32,767

char - 16 bitów od 0 do 65,535

int - 32 bitów od -2,147,483,648 do 2,147,483,647

long - 64 bitów od -9,223,372,036,854,775,808 do 9,223,372,036,854,775,807

- Język statycznie typowany - zmienna zawsze przechowuje wartość tego samego typu
- Mamy 2 grupy typów

  prymitywne - boolean, char, byte, short, int, long, float, double

Liczby całkowite:

```
1  package pap;
2  public class IntegerNotation {
3    public static void main(String[] args) {
4      int n1 = 21;      // 21
5      int n2 = 0x21;      // 33 = 2*16+1
6      int n3 = 021;       // 17 = 2*8+1
7      int n4 = 0b1001;    // 9  = 1*8+0*4*0*2+1
8      long a = 23456789123L;
9      System.out.println(a==23_456_789_123L);// true
10   }
11 }
```

- Język statycznie typowany - zmienna zawsze przechowuje wartość tego samego typu
- Mamy 2 grupy typów

  prymitywne - boolean, char, byte, short, int, long, float, double

  referencje - class, interface, array

- Język statycznie typowany - zmienna zawsze przechowuje wartość tego samego typu
- Mamy 2 grupy typów

  prymitywne - boolean, char, byte, short, int, long, float, double

  referencje - class, interface, array

```java
package pap;
import java.math.BigInteger;

public class UnlimitedIntegers {
  public static void main(String[] args) {
    System.out.println(Long.MAX_VALUE); // 9223372036854775807
    BigInteger b = new BigInteger("12345678912");
    BigInteger c = new BigInteger("5635476764765875885888");
    BigInteger a = b.multiply(c);
    System.out.println(a);  // 69573786653836058541518034562560
  }
}
```

- Język statycznie typowany - zmienna zawsze przechowuje wartość tego samego typu
- Mamy 2 grupy typów

  prymitywne - boolean, char, byte, short, int, long, float, double

  referencje - class, interface, array

```
1  package pap;
2  import java.math.BigDecimal; import java.math.MathContext; import java.math.RoundingMode;
3  public class UnlimitedIntegers {
4    public static void main(String[] args) {
5      System.out.println(Double.MAX_VALUE); // 1.7976931348623157E308
6      System.out.println(Long.MAX_VALUE);   // 92233720354775807
7      BigDecimal n = new BigDecimal("695737653851803456256.123");
8      System.out.println(n);        // 695737653851803456256.123
9      n = n.pow(2, new MathContext(300, RoundingMode.DOWN));
10     BigDecimal n1 = n.multiply(new BigDecimal(1), new MathContext(300, RoundingMode.DOWN));
11     BigDecimal n2 = n.multiply(new BigDecimal(1), new MathContext(30, RoundingMode.DOWN));
12     System.out.println(n1);  // 484050882987211884671415493888514849187774.991129
13     System.out.println(n2);  // 4.840508829872118846714154938885E+41
14   }
15 }
```

Zmiennopozycyjne i notacja naukowa

```java
package pap;
import java.math.BigDecimal;
import java.text.DecimalFormat;
public class ScientificNotation {
  public static void main(String[] args) {
    double a = 123f;
    double n = 1.2345E10;
    DecimalFormat dec = new DecimalFormat("#.00");
    System.out.println(dec.format(a));       // 123.00
    System.out.println(dec.format(n));       // 12345000000.00
    BigDecimal bd = new BigDecimal("1.2345e-19");
    System.out.println(bd.toEngineeringString()); // 123.45E-21
    System.out.println(bd.toPlainString());  // 0.000000000000000000012345
  }
}
```

- Java nie łapie wyjątków związanych z przepełnieniem

```java
package pap;

public class Overflow {
  public static void main(String[] args) {
    int a = Integer.MAX_VALUE-1;
    System.out.println(String.format("%h",a));  // 7ffffffe
    a++;
    System.out.println(String.format("%h",a++));  // 7fffffff
    System.out.println(String.format("%h",a++));  // 80000000
    System.out.println(String.format("%h",a++));  // 80000001
    System.out.println(String.format("%h",a++));  // 80000002


  }
}
```

- Java nie łapie wyjątków związanych z przepełnieniem
- Musimy robić arytmetykę ręcznie

```java
package pap;

public class Overflow {
  public static void main(String[] args) {
    int a = Integer.MAX_VALUE-1;
    System.out.println(String.format("%h",a));  // 7ffffffe
    a++;
    System.out.println(String.format("%h",a++));  // 7fffffff
    System.out.println(String.format("%h",a++));  // 80000000
    System.out.println(String.format("%h",a++));  // 80000001
    System.out.println(String.format("%h",a++));  // 80000002
    a = Integer.MAX_VALUE;
    a = Math.addExact(a, 1); // java.lang.ArithmeticException: integer overflow
  }
}
```

- Zmienne przechowują wartości (referencja do obiektu to też wartość)

```
1  package pap;
2
3  public class Variables {
4    public static void main(String[] args) {
5      String city ;
6      String country = "Poland";
7      var zipcode = "00-667" ;    // var dla leniwych -> typ na podstawie inicjalizacji
8      int age, length;
9      city = "Szczecin";
10     System.out.println(city);
11     System.out.println(age);    // blad kompilacji bo zmienna niezainicjalizowana
12     System.out.println(country);
13   }
14 }
```

- Zmienne przechowują wartości (referencja do obiektu to też wartość)
- Kompilator wykrywa błąd, gdy zmienna jest niezainicjalizowana

```
1  package pap;
2
3  public class Variables {
4    public static void main(String[] args) {
5      String city ;
6      String country = "Poland";
7      var zipcode = "00-667" ;    // var dla leniwych -> typ na podstawie inicjalizacji
8      int age, length;
9      city = "Szczecin";
10     System.out.println(city);
11     System.out.println(age);    // blad kompilacji bo zmienna niezainicjalizowana
12     System.out.println(country);
13   }
14 }
```

- Stałe nie mogą zmienić wartości nadanej w czasie inicjalizacji

```
1  package pap;
2
3  public class Constants {
4    public static void main(String[] args) {
5      final int WIDTH = 100;
6      final int HEIGHT = 150;
7      var var = 140;
8      System.out.println(var);
9      var = 150;
10     System.out.println(var);
11     WIDTH = 110;    // blad kompilacji bo stala
12   }
13 }
```

- Stałe nie mogą zmienić wartości nadanej w czasie inicjalizacji
- Tworzymy za pomocą słowa kluczowego `final`

```
1  package pap;
2
3  public class Constants {
4    public static void main(String[] args) {
5      final int WIDTH = 100;
6      final int HEIGHT = 150;
7      var var = 140;
8      System.out.println(var);
9      var = 150;
10     System.out.println(var);
11     WIDTH = 110;    // blad kompilacji bo stala
12   }
13 }
```

```java
package pap;

public class Enumerations {
  enum Days {
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
    FRIDAY, SATURDAY, SUNDAY }
  public static void main(String[] args) {
    Days day = Days.MONDAY;
    if (day == Days.MONDAY)
      System.out.println("It is Monday");  // It is Monday
    System.out.println(day);         // MONDAY
    for (Days d : Days.values())
      System.out.println(d);         // MONDAY
  }                                  // TUESDAY
}                                    // ...
```

możemy nadawać wartości

```
package pap;
public class Enumerations {
  enum Days {
      MONDAY(10),  TUESDAY(20),  WEDNESDAY(30),  THURSDAY(40),
      FRIDAY(50), SATURDAY(60), SUNDAY(70);
    private int value;
    private Days(int value) { this.value = value; }
    public int getValue() { return value; }
    }
  ...
}
```

- typ `boolean` jest typem prostym, typ `Boolean` jest obiektem (boxem) do przechowywania boolean

```
package pap;
public class SBoolean {
  public static void main(String[] args) {
    boolean flaga = true;
    //int value = (int)flaga;   // cannot cast from bolean to int
    Boolean oFlaga = !flaga;
    //System.out.println(flaga.equals(false));
              // cannot invoke equals(boolean) on primitive type
    System.out.println(oFlaga.equals(false));  // true
    flaga = !oFlaga;
    System.out.println(oFlaga?"wartosc true":"wartosc false");  // ternary
    if(flaga == oFlaga)
      System.out.println("rowne");
    else if (flaga) {
      System.out.println(flaga);  // True
  }}
}
```

```java
package pap;

public class Tablice {
  public static void main(String[] args) {
    int[] numbers = new int[5];
    numbers[0] = 10;
    numbers[1] = 20;
    numbers[2] = 30;
    numbers[3] = 40;
    numbers[4] = 50;
    for (int i = 0; i < numbers.length; i++)
      System.out.println(numbers[i]);
  }
}
```

```java
package pap;

public class Tablice {
  public static void main(String[] args) {
    int[] numbers = new int[5];
    numbers[0] = 10;
    numbers[1] = 20;
    numbers[2] = 30;
    numbers[3] = 40;
    numbers[4] = 50;
    for (var v:numbers)
      System.out.println(v);
  }
}
```

```java
package pap;
import java.util.*;

public class Tablice {
  public static void main(String[] args) {
    int[] numbers = new int[5];
    Arrays.fill(numbers, 0, 3, -50);
    for (var v:numbers)
      System.out.println(v); // -50, -50, -50, 0, 0
  }
}
```

```java
package pap;
import java.util.*;

public class Tablice {
  public static void main(String[] args) {
    int[] numbers = new int[] {124,123,122,121,120};
    Arrays.fill(numbers, 1, 3, -50);
    for (var v:numbers)
      System.out.println(v); // 124, -50, -50, -50, 120
  }
}
```

- wielowymiarowe:

to tablica tablic

```
int[][] numbers = {{1, 2, 3, 4, 5}, { 6, 7, 8}, {1} };
System.out.println(numbers[2][0]);      // 1
```

- wielowymiarowe:

to tablica tablic

```
int[][] numbers = {{1, 2, 3, 4, 5}, { 6, 7, 8}, {1} };
System.out.println(numbers[2][0]);      // 1
```

- iteracja po tablicy:

to tablica tablic

```
int[][] numbers = {{1, 2, 3, 4, 5}, { 6, 7, 8}, {1} };
for(var a: numbers)
  for(var b:a)
    System.out.println(b);
```

- w case może być dowolna liczba wartości będących stałymi

```
1  package pap;
2  public class ESample {
3    public static void main(String[] args) {
4      String day = "MONDAY";
5      switch(day) {
6        case "MONDAY", "TUESDAY":
7          System.out.println("nie lubie");
8        case "WEDNESDAY":
9          System.out.println("lepiej");
10         break;
11       default:
12         System.out.println("podoba sie");
13     }
14   }
15 }
```

- w `case` może być dowolna liczba wartości będących stałymi
- jak nie damy `break` to przechodzimy do następnego warunku

```
1  package pap;
2  public class ESample {
3    public static void main(String[] args) {
4      String day = "MONDAY";
5      switch(day) {
6        case "MONDAY", "TUESDAY":
7          System.out.println("nie lubie");
8        case "WEDNESDAY":
9          System.out.println("lepiej");
10         break;
11       default:
12         System.out.println("podoba sie");
13     }
14   }
15 }
```

- w `case` może być dowolna liczba wartości będących stałymi
- jak nie damy `break` to przechodzimy do następnego warunku
- przykład z `enum`

```
1  package pap;
2  enum Day {
3    MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
4    FRIDAY, SATURDAY, SUNDAY
5  }
6  public class ESample {
7    public static void main(String[] args) {
8    Day day = Day.MONDAY;
9    switch(day) {
10     case MONDAY, TUESDAY:
11       System.out.println("nie lubie");
12     case WEDNESDAY:
13       System.out.println("lepiej");
14       break;
15     default:
16       System.out.println("podoba sie");
17   }}
18 }
```

while

```
int i = 5;
while (i < 10) {
  System.out.println(i);
  i++;
}
```

do/while

```
int i = 5;
do {
  System.out.println(i);
  i++;
} while (i < 10);
```

for

```
for (int i = 5; i < 10; i++)
  System.out.println(i);
```

- break przerywa wykonanie pętli i wychodzi

```
for (int i = 0; i < 10; i++) {
  System.out.println(i);
  if (i == 5)
    break;
  System.out.println(i);
}
```

- `break` przerywa wykonanie pętli i wychodzi
- `continue` przerywa wykonanie aktualnej iteracji pętli i przechodzi do następnej iteracji

```java
for (int i = 0; i < 10; i++) {
  System.out.println(i);
  if (i == 5)
    break;
  System.out.println(i);
}
```

```java
for (int i = 0; i < 10; i++) {
  System.out.println(i);
  if (i == 5)
    continue;
  System.out.println(i);
}
```

Klasa Java.Math zawiera wiele ciekawych metod obliczeniowych

Math.max(x,y) - zwraca

- pliki:

  binarne - operacje na 8 bitowych bajtach

  znakowe - operacje na 16 bitowych znakach (`FileReader`, `FileWriter`)
- w pakiecie `java.io`
- standardowe strumienie

  System.in - stąd czytamy

  System.out - tu piszemy wyniki programu

  System.err - tu piszemy komunikaty o błędach

- pliki:

  binarne - operacje na 8 bitowych bajtach

  znakowe - operacje na 16 bitowych znakach (`FileReader`, `FileWriter`)

- w pakiecie `java.io`
- standardowe strumienie

  System.in - stąd czytamy

  System.out - tu piszemy wyniki programu

  System.err - tu piszemy komunikaty o błędach

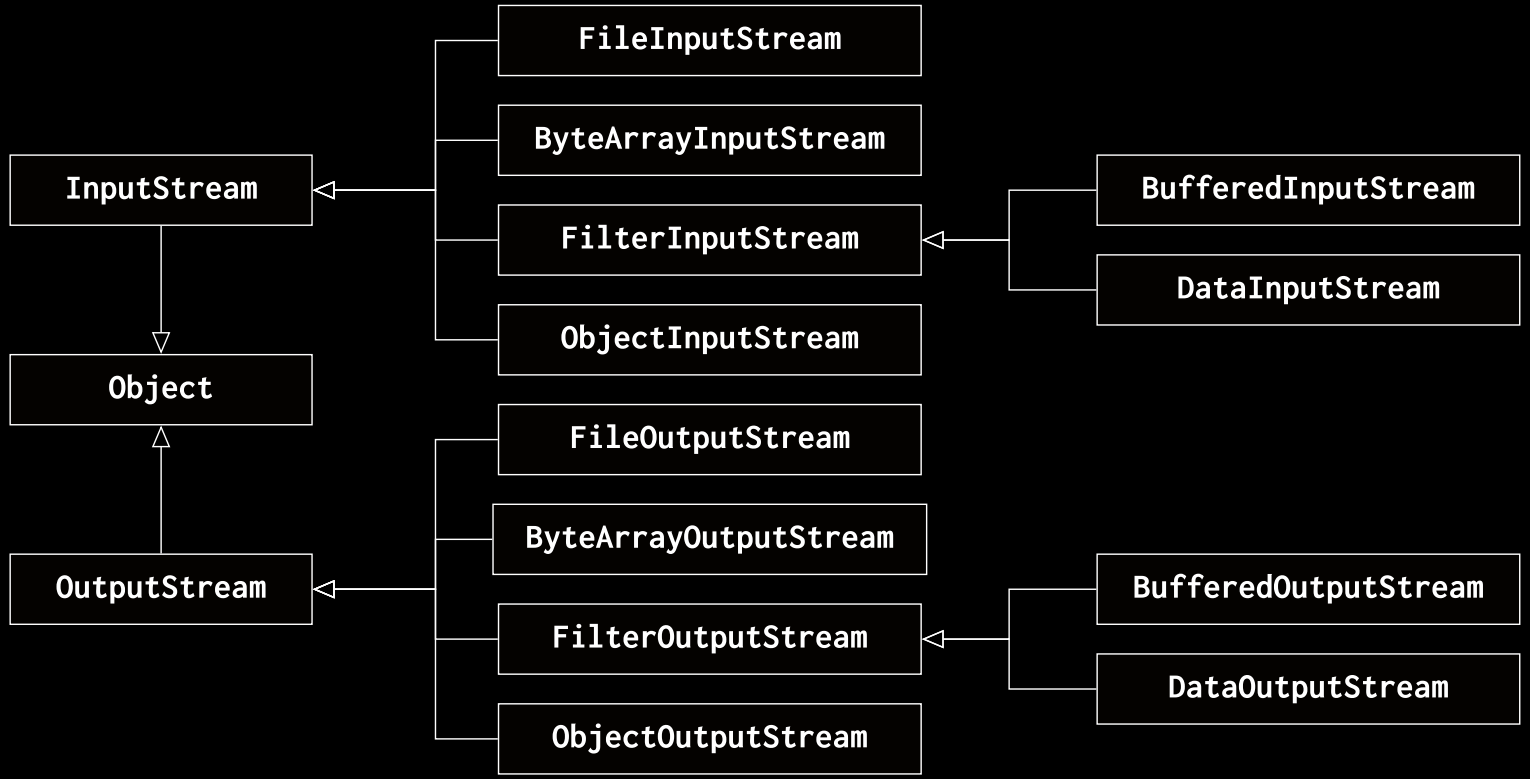
zapisanie błędów w oddzielnym pliku

```
command1 <wejscie.txt        # czytamy z pliku wejscie.txt
command1 2>error.log         # do pliku
command1 2>/dev/null         # wywal
command1 &>wyjscie.txt       # oba w pliku wyjscie.txt
command1  >wyjscie.txt       # oba w pliku wyjscie.txt
command1 1>wyjscie.txt 2>&1  # oba w pliku wyjscie.txt
```

```java
package pap;
import java.io.*;
public class FileCopy {
  public static void main(String args[]) throws IOException {
    FileInputStream in = null;
    FileOutputStream out = null;
    try {
       in = new FileInputStream("sample-input.txt");
       out = new FileOutputStream("sample-output.txt");
       int b;
       while ((b = in.read()) != -1) {
         out.write(b);
       }
    } finally {
      if (in != null)
        in.close();
      if (out != null)
        out.close();
    }
  }
}
```

```java
package pap;
import java.io.*;
public class ConsoleSample {
  public static void main(String args[]) throws IOException {
    InputStreamReader cin = null;

    try {
      cin = new InputStreamReader(System.in);
      System.out.println("kopiuje, litera 'q' konczy.");
      char c;
      do {
        c = (char) cin.read();
        System.out.print(c);
      } while(c != 'q');
    }finally {
     if (cin != null)
       cin.close();
    }
  }
}
```


FileInputStream
ByteArrayInputStream
InputStream
FilterInputStream
BufferedInputStream
DataInputStream
ObjectInputStream
Object
FileOutputStream
ByteArrayOutputStream
OutputStream
BufferedOutputStream
FilterOutputStream
DataOutputStream
ObjectOutputStream

- Klasa `String` odpowiada za ciąg znaków

```
String str1 = "balbinka";
char[] data = {'b','a','l','b','i','n','k','a' };
String str2 = new String(data);
String str3 = "balbinka";
String str4 = new String(str1);
System.out.println(str1);   // balbinka
System.out.println(str2);   // balbinka
System.out.println(str3);   // balbinka
System.out.println(str4);   // balbinka
System.out.println(str1 == str2);      // false
System.out.println(str1 == str3);      // true (to efekt optymalizacji kompilatora)
System.out.println(str1 == str4);      // false
System.out.println(str1.equals(str3)); // true (poprawny sposob porownywania)
```

- Klasa `String` odpowiada za ciąg znaków
- znaki są reprezentowane w UTF-16

```
String str1 = "balbinka";
char[] data = {'b','a','l','b','i','n','k','a' };
String str2 = new String(data);
String str3 = "balbinka";
String str4 = new String(str1);
System.out.println(str1);   // balbinka
System.out.println(str2);   // balbinka
System.out.println(str3);   // balbinka
System.out.println(str4);   // balbinka
System.out.println(str1 == str2);      // false
System.out.println(str1 == str3);      // true (to efekt optymalizacji kompilatora)
System.out.println(str1 == str4);      // false
System.out.println(str1.equals(str3)); // true (poprawny sposob porownywania)
```

- Klasa `String` odpowiada za ciąg znaków
- znaki są reprezentowane w UTF-16
- Stringi są stałymi, zatem wystąpienia takiego samego stringu w Javie mogą być reużywane przez kompilator zwracając referencje do tego samego obiektu

```
String str1 = "balbinka";
char[] data = {'b','a','l','b','i','n','k','a' };
String str2 = new String(data);
String str3 = "balbinka";
String str4 = new String(str1);
System.out.println(str1);   // balbinka
System.out.println(str2);   // balbinka
System.out.println(str3);   // balbinka
System.out.println(str4);   // balbinka
System.out.println(str1 == str2);      // false
System.out.println(str1 == str3);      // true (to efekt optymalizacji kompilatora)
System.out.println(str1 == str4);      // false
System.out.println(str1.equals(str3)); // true (poprawny sposob porownywania)
```

- Klasa `String` odpowiada za ciąg znaków
- znaki są reprezentowane w UTF-16
- Stringi są stałymi, zatem wystąpienia takiego samego stringu w Javie mogą być reużywane przez kompilator zwracając referencje do tego samego obiektu
- Operacje na Stringach:

  charAt(int) - zwraca znak z podanego indeksu

```
1 String str1 = "balbinka";
2 char znak = str1.charAt(3);
3 String s = Character.toString(znak)
4 System.out.println(znak);   // b
5 System.out.println(s);      // b
```

- Klasa `String` odpowiada za ciąg znaków
- znaki są reprezentowane w UTF-16
- Stringi są stałymi, zatem wystąpienia takiego samego stringu w Javie mogą być reużywane przez kompilator zwracając referencje do tego samego obiektu
- Operacje na Stringach:

  charAt(int) - zwraca znak z podanego indeksu

```
1 String str1 = "balbinka";
2 char znak = str1.charAt(3);
3 String s = Character.toString(znak)
4 System.out.println(znak);   // b
5 System.out.println(s);      // b
```

- Możemy użyć notacji podobnej do C

```java
package pap;

public class Strings {
    public static void main(String[] args) {
        String name = "Jeremy Clarkson";
        int age = 60;
        String output = String.format("%s ma %d lata.", name, age);
        System.out.println(output);
        output = String.format("%2$s ma %1$d lata.", age, name);
        System.out.println(output);
        System.out.println(name+" ma "+String.valueOf(age)+" lata.");
        System.out.println(String.format(">%7d<",age));           // >     60<
        System.out.println(String.format(">%7.1f<",(float)age));  // >   60.0<
        System.out.println(String.format(">%-7.1f<",(float)age)); // >60.0   <
        System.out.println(String.format(">%07d<",age));          // >0000060<
        System.out.println(String.format(">%,7d<",100*age));      // >  6,000<
    }
}
```

- Możemy użyć notacji podobnej do C
- Składnia formatowania jest bardzo bogata

```java
package pap;

public class Strings {
    public static void main(String[] args) {
        String name = "Jeremy Clarkson";
        int age = 60;
        String output = String.format("%s ma %d lata.", name, age);
        System.out.println(output);
        output = String.format("%2$s ma %1$d lata.", age, name);
        System.out.println(output);
        System.out.println(name+" ma "+String.valueOf(age)+" lata.");
        System.out.println(String.format(">%7d<",age));           // >     60<
        System.out.println(String.format(">%7.1f<",(float)age));  // >   60.0<
        System.out.println(String.format(">%-7.1f<",(float)age)); // >60.0   <
        System.out.println(String.format(">%07d<",age));          // >0000060<
        System.out.println(String.format(">%,7d<",100*age));      // >  6,000<
    }
}
```

```java
package pap;
import java.util.Scanner;       // bez import musielibysmy pisac java.util.Scanner

public class ReadLine {
    public static void main(String[] args) {
        System.out.print("Podaj swoj wiek:");
        Scanner sc = new Scanner(System.in);
        String age = sc.nextLine();
        System.out.println("Masz " + age+ " lat");
        sc.close();
    }
}

```

a co jak podamy napis balbinka zamiast wieku?

```java
package pap;
import java.util.Scanner;       // bez import musielibysmy pisac java.util.Scanner
public class ReadLine {
    public static void main(String[] args) throws NumberFormatException {
        System.out.print("Podaj swoj wiek:");
        Scanner sc = new Scanner(System.in);
        String s = sc.nextLine();
        int age = Integer.parseInt(s);
        System.out.println("Masz " + age+ " lat");
        sc.close();
    }
}
```

teraz balbinka nie przejdzie, ale program nie jest nadal OK

```java
package pap;
import java.util.Scanner;       // bez import musielibysmy pisac java.util.Scanner
public class ReadLine {
    public static void main(String[] args)  {
        System.out.print("Podaj swoj wiek:");
        Scanner sc = new Scanner(System.in);
        String s = sc.nextLine();
        try {
        int age = Integer.parseInt(s);
        System.out.println("Masz " + age+ " lat");
        } catch (NumberFormatException fException) {
            System.out.println("niepoprawny format wieku");
            // throw fException;
        } finally {
            sc.close();
        }
    }
}
```

teraz balbinka nie przejdzie, i program jest OK

```java
package pap;
public class TrySample {
    public Boolean bramkarz;
    public void pierwszaPolowa() {
        System.out.println("pierwszaPolowa");
    }
    public void drugaPolowa() throws Exception {
        System.out.println("drugaPolowa poczatek");
        if(!bramkarz)
            throw new Exception("stracilismy gola");
        System.out.println("drugaPolowa koniec");
    }
    public void mecz() throws Exception {
        System.out.println("poczatek meczu");
        pierwszaPolowa();
        drugaPolowa();
        System.out.println("koniec meczu");
    }
    public static void main(String[] args)  throws Exception {
        try {
            TrySample sample = new TrySample();
            sample.mecz();
            System.out.println("nic sie nie stalo");
        } catch (Exception fException) {
            System.out.println("halo? gdzie ta pilka?");
        } finally {
            System.out.println("na koniec mowimy do widzenia ...");
        }
    }
}
```

```
poczatek meczu
pierwszaPolowa
drugaPolowa poczatek
na koniec mowimy do widzenia ...
Exception in thread "main" java.lang.NullPointerException:
        Cannot invoke "java.lang.Boolean.booleanValue()" because "this.bramkarz" is null
        at pap.TrySample.drugaPolowa(TrySample.java:10)
        at pap.TrySample.mecz(TrySample.java:17)
        at pap.TrySample.main(TrySample.java:22)
```

- synctactic sugar

```
1  class Foo implements AutoCloseable {
2    public void method() {
3      System.out.println("method");
4    }
5    public void close() {
6      System.out.println("close");
7    }
8  }
9  public class App {
10   public static void main(String[] args) throws Exception {
11     System.out.println("before");
12     try (Foo f = new Foo()) {
13       f.method();
14     }
15
16
17
18     System.out.println("after");
19   }
20 }
```

- synctactic sugar
- jest równoważne

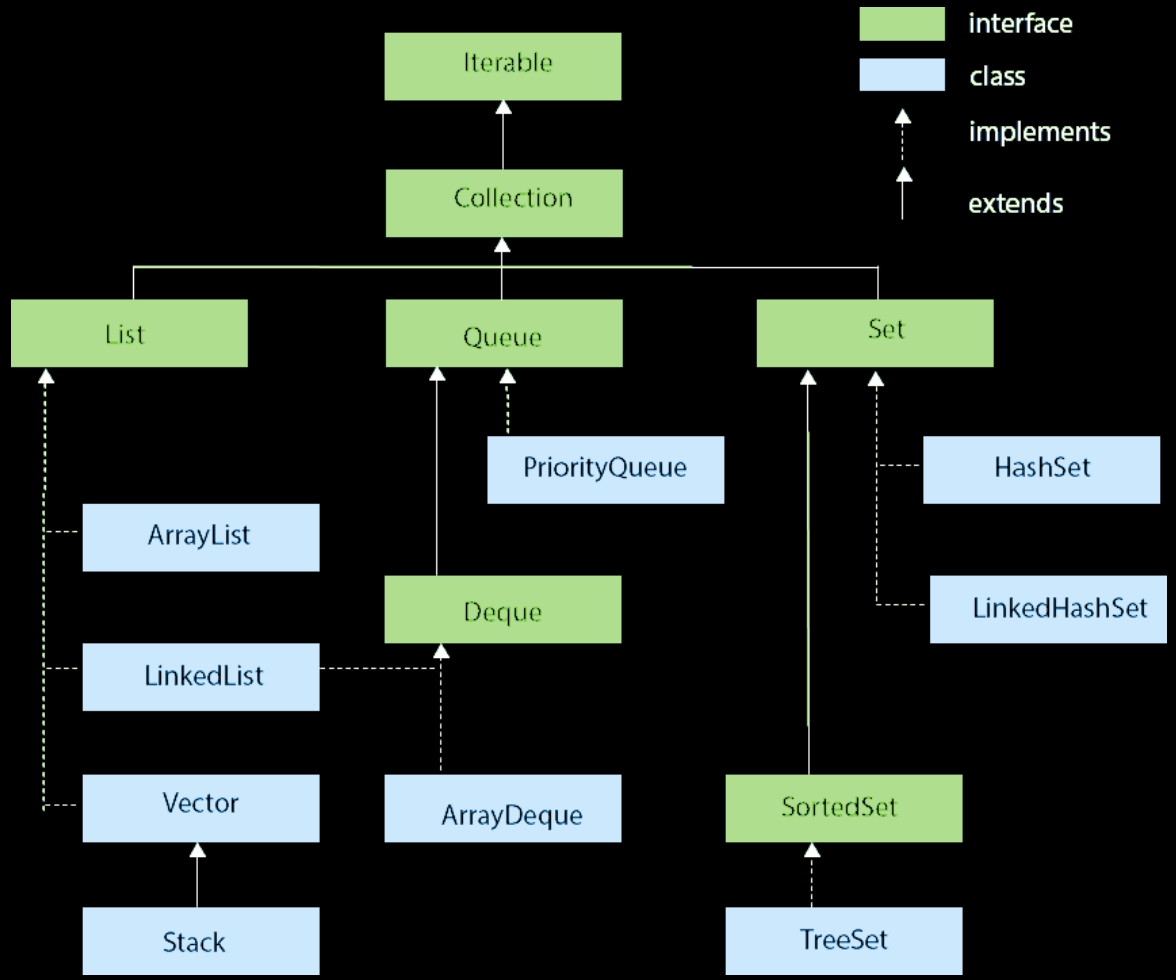
```
class Foo implements AutoCloseable {
  public void method() {
    System.out.println("method");
  }
  public void close() {
    System.out.println("close");
  }
}
public class App {
  public static void main(String[] args) throws Exception {
    System.out.println("before");
    Foo f = new Foo();
    try  {
      f.method();
    } finally {
      f.close();
    }
    System.out.println("after");
  }
}
```

- synctactic sugar
- jest równoważne
- oba drukują:

```
before
method
close
after
```

- synctactic sugar
- jest równoważne
- oba drukują:
- działa gdy przerwanie po alokacji obiektu

```
class Foo implements AutoCloseable {
  public void method() {
    System.out.println("method");
  }
  public void close() {
    System.out.println("close");
  }
}
public class App {
  public static void main(String[] args) throws Exception {
    System.out.println("before");
    Foo f = new Foo();
    try  {
      f.method();
    } finally {
      f.close();
    }
    System.out.println("after");
  }
}
```


interface
class
implements
extends
Iterable
Collection
List
Queue
Set
PriorityQueue
HashSet
ArrayList
Deque
LinkedHashSet
LinkedList
Vector
ArrayDeque
SortedSet
Stack
TreeSet

- Klasa `ArrayList` implementuje dynamiczną tablicę obiektów

```
// Tworzy obiekt ArrayList
java.util.ArrayList<String> cars = new java.util.ArrayList<String>();
cars.add("Syrenka");      // dodaj na koncu
cars.add("Polonez");      // dodaj na koncu
cars.add(1, "Fiat 126p"); // dodaj na pozycji 1,

System.out.println(cars);        // [Syrenka, Fiat 126p, Polonez]
System.out.println(cars.size()); // 3

cars.remove(1);                  // cars.clear() czysci wszystko
System.out.println(cars);        // [Syrenka, Polonez]


```

- Klasa `ArrayList` implementuje dynamiczną tablicę obiektów
  - dla typów prostych (boolean,...) tworzymy kolekcje wrapperów (Boolean, ...)

```
1  // Tworzy obiekt ArrayList
2  java.util.ArrayList<String> cars = new java.util.ArrayList<String>();
3  cars.add("Syrenka");      // dodaj na koncu
4  cars.add("Polonez");      // dodaj na koncu
5  cars.add(1, "Fiat 126p"); // dodaj na pozycji 1,
6
7  System.out.println(cars);        // [Syrenka, Fiat 126p, Polonez]
8  System.out.println(cars.size()); // 3
9
10 cars.remove(1);                  // cars.clear() czysci wszystko
11 System.out.println(cars);        // [Syrenka, Polonez]
12
13
```

- Klasa `ArrayList` implementuje dynamiczną tablicę obiektów
  - dla typów prostych (boolean,...) tworzymy kolekcje wrapperów (Boolean, ...)
- dostęp do elementów za pomocą metod `get` i `set`

```
1  // Tworzy obiekt ArrayList
2  java.util.ArrayList<String> cars = new java.util.ArrayList<String>();
3  cars.add("Syrenka");      // dodaj na koncu
4  cars.add("Polonez");      // dodaj na koncu
5  cars.add(1, "Fiat 126p"); // dodaj na pozycji 1,
6  // cars.set(3, "Cinquecento"); // IndexOutOfBoundsException: Index 3 out of bounds for length 3
7  System.out.println(cars);        // [Syrenka, Fiat 126p, Polonez]
8  System.out.println(cars.size()); // 3
9  System.out.println(cars.get(1)); // Fiat 126p
10 cars.remove(1);                  // cars.clear() czysci wszystko
11 System.out.println(cars);        // [Syrenka, Polonez]
12
13
```

- Klasa `ArrayList` implementuje dynamiczną tablicę obiektów
  - dla typów prostych (boolean,...) tworzymy kolekcje wrapperów (Boolean, ...)
- dostęp do elementów za pomocą metod `get` i `set`
- dostęp do elementów za pomocą metod `indexOf` i `lastIndexOf`

```
// Tworzy obiekt ArrayList
java.util.ArrayList<String> cars = new java.util.ArrayList<String>();
cars.add("Syrenka");      // dodaj na koncu
cars.add("Polonez");      // dodaj na koncu
cars.add(1, "Fiat 126p"); // dodaj na pozycji 1,
// cars.set(3, "Cinquecento"); // IndexOutOfBoundsException: Index 3 out of bounds for length 3
System.out.println(cars);        // [Syrenka, Fiat 126p, Polonez]
System.out.println(cars.size()); // 3
System.out.println(cars.get(1)); // Fiat 126p
cars.remove(1);                  // cars.clear() czysci wszystko
System.out.println(cars);        // [Syrenka, Polonez]
for (String car : cars)
    System.out.println(car);
```

- Klasa `ArrayList` implementuje dynamiczną tablicę obiektów
  - dla typów prostych (boolean,...) tworzymy kolekcje wrapperów (Boolean, ...)
- dostęp do elementów za pomocą metod `get` i `set`
- dostęp do elementów za pomocą metod `indexOf` i `lastIndexOf`
- sortowanie (w miejscu) za pomocą metody `Collections.sort`

```
1  // Tworzy obiekt ArrayList
2  java.util.ArrayList<String> cars = new java.util.ArrayList<String>();
3  cars.add("Syrenka");      // dodaj na koncu
4  cars.add("Polonez");      // dodaj na koncu
5  cars.add(1, "Fiat 126p"); // dodaj na pozycji 1,
6  // cars.set(3, "Cinquecento"); // IndexOutOfBoundsException: Index 3 out of bounds for length 3
7  System.out.println(cars);        // [Syrenka, Fiat 126p, Polonez]
8  System.out.println(cars.size()); // 3
9  System.out.println(cars.get(1)); // Fiat 126p
10 cars.remove(1);                  // cars.clear() czysci wszystko
11 System.out.println(cars);        // [Syrenka, Polonez]
12 java.util.Collections.sort(cars);
13 System.out.println(cars);        // [Polonez, Syrenka]
```

- instrukcją `for`

```
1  package proz;
2  import java.util.*;
3
4  public class ArrayListSample {
5    public static void main(String args[]) {
6      ArrayList<String> list = new ArrayList<String>();// {tworzymy liste
7      list.add("Jeremy Clarkson"); // dodajemy elementy do listy
8      list.add("James May");
9      list.add("Richard Hammond");
10     for(int i=0;i<list.size();i++)
11        System.out.println(list.get(i));
12   }
13 }
```

- instrukcją `for`
- iteratorem

```java
package proz;
import java.util.*;

public class ArrayListSample {
  public static void main(String args[]) {
    ArrayList<String> list = new ArrayList<String>();// {tworzymy liste
    list.add("Jeremy Clarkson"); // dodajemy elementy do listy
    list.add("James May");
    list.add("Richard Hammond");
    // przegladamy liste za pomoca iteratora
    var itr = list.iterator(); // pobranie iteratora
    while (itr.hasNext()) { // sprawdzenie czy cos jeszcze jest
      System.out.println(itr.next());// pobranie elementu i przesuniecie do przodu
    }
  }
}
```

- instrukcją `for`
- iteratorem
- instrukcją `foreach`

```
1  package proz;
2  import java.util.*;
3
4  public class ArrayListSample {
5    public static void main(String args[]) {
6      ArrayList<String> list = new ArrayList<String>();// {tworzymy liste
7      list.add("Jeremy Clarkson"); // dodajemy elementy do listy
8      list.add("James May");
9      list.add("Richard Hammond");
10     for(String person:list) {
11       System.out.println(person);
12     }
13   }
14 }
```

- instrukcją `for`
- iteratorem
- instrukcją `foreach`
- funkcją `forEach`

```
1  package proz;
2  import java.util.*;
3
4  public class ArrayListSample {
5    public static void main(String args[]) {
6      ArrayList<String> list = new ArrayList<String>();// {tworzymy liste
7      list.add("Jeremy Clarkson"); // dodajemy elementy do listy
8      list.add("James May");
9      list.add("Richard Hammond");
10     list.forEach(a->{ // tu uzywamy wyrazenia lambda
11         System.out.println(a);
12     });
13   }
14 }
```

- Klasa `LinkedList` implementuje ten sam interfejs `List` co `ArrayList`

- Klasa `LinkedList` implementuje ten sam interfejs `List` co `ArrayList`
- inna wewnętrzna reprezentacja:

  ArrayList - tablica (automatyczna relokacja, gdy za mała)

  LinkedList - lista dwukierunkowa

- Klasa `LinkedList` implementuje ten sam interfejs `List` co `ArrayList`
- inna wewnętrzna reprezentacja:

  ArrayList - tablica (automatyczna relokacja, gdy za mała)
  - dostęp swobodny do elementów
  - dodawanie/usuwanie elementów na końcu

  LinkedList - lista dwukierunkowa

- Klasa `LinkedList` implementuje ten sam interfejs `List` co `ArrayList`
- inna wewnętrzna reprezentacja:

  ArrayList - tablica (automatyczna relokacja, gdy za mała)
  - dostęp swobodny do elementów
  - dodawanie/usuwanie elementów na końcu

  LinkedList - lista dwukierunkowa
  - dostęp sekwencyjny za pomocą iteratora
  - dodawanie/usuwanie elementów wewnątrz listy (`addFirst`, `addLast`, `removeFirst`, `removeLast`, `getFirst`, `getLast`)

- Klasa `LinkedList` implementuje ten sam interfejs `List` co `ArrayList`

- zwykłe `Arrays.asList()` tworzy listę opartą na tablicy

```java
package pap;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
public class StreamTest {
  public static void main(String[] args) {
    String[] arr = new String[]{"a", "b", "c", "d"};
    List<String> lista = Arrays.asList(arr);
    //lista.add("e");            //blad -lista niemodyfikowalna
    arr[0]="A";                  //zmiana w tablicy zmienia list
    lista.forEach(e->System.out.print(e));
    System.out.println("");      // Abcd:
  }
}
```

- zwykłe `Arrays.asList()` tworzy listę opartą na tablicy
  - która jest niemodyfikowalna

```java
package pap;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
public class StreamTest {
  public static void main(String[] args) {
    String[] arr = new String[]{"a", "b", "c", "d"};
    List<String> lista = Arrays.asList(arr);
    //lista.add("e");            //blad -lista niemodyfikowalna
    arr[0]="A";                  //zmiana w tablicy zmienia list
    lista.forEach(e->System.out.print(e));
    System.out.println("");      // Abcd:
  }
}
```

- zwykłe `Arrays.asList()` tworzy listę opartą na tablicy
  - która jest niemodyfikowalna
  - która pokazuje aktualną (także zmienioną) zawartość tablicy

```
1  package pap;
2  import java.util.ArrayList;
3  import java.util.Arrays;
4  import java.util.List;
5  public class StreamTest {
6    public static void main(String[] args) {
7      String[] arr = new String[]{"a", "b", "c", "d"};
8      List<String> lista = Arrays.asList(arr);
9      //lista.add("e");            //blad -lista niemodyfikowalna
10     arr[0]="A";                  //zmiana w tablicy zmienia list
11     lista.forEach(e->System.out.print(e));
12     System.out.println("");      // Abcd:
13   }
14 }
```

- zwykłe `Arrays.asList()` tworzy listę opartą na tablicy
  - która jest niemodyfikowalna
  - która pokazuje aktualną (także zmienioną) zawartość tablicy
- trzeba z tej listy zrobić kopię (czyli `ArrayList`)

```
1  package pap;
2  import java.util.ArrayList;
3  import java.util.Arrays;
4  import java.util.List;
5  public class StreamTest {
6    public static void main(String[] args) {
7      String[] arr = new String[]{"a", "b", "c", "d"};
8      List<String> lista = new ArrayList<>(Arrays.asList(arr));
9      lista.add("e");              //kopia - mozna modyfikowac
10     arr[0]="A";                  //zmiana w tablicy nie zmienia listy
11     lista.forEach(e->System.out.print(e));
12     System.out.println("");      // abcde:
13   }
14 }
```

- zwykłe `Arrays.asList()` tworzy listę opartą na tablicy
  - która jest niemodyfikowalna
  - która pokazuje aktualną (także zmienioną) zawartość tablicy
- trzeba z tej listy zrobić kopię (czyli `ArrayList`)
  - czyli zmiany w tablicy już nie są widoczne

```
1  package pap;
2  import java.util.ArrayList;
3  import java.util.Arrays;
4  import java.util.List;
5  public class StreamTest {
6    public static void main(String[] args) {
7      String[] arr = new String[]{"a", "b", "c", "d"};
8      List<String> lista = new ArrayList<>(Arrays.asList(arr));
9      lista.add("e");              //kopia - mozna modyfikowac
10     arr[0]="A";                  //zmiana w tablicy nie zmienia listy
11     lista.forEach(e->System.out.print(e));
12     System.out.println("");      // abcde:
13   }
14 }
```

- Nowe API wprowadzono w Java 8

- Nowe API wprowadzono w Java 8
  - Wcześniejsze (`java.util.Date` i `java.util.Calendar`) rozwiązania były kiepskie
    - nieodporne na wątki
    - pracochłonny interfejs dla typowych operacji
    - trudna obsługa stref czasowych

- Nowe API wprowadzono w Java 8
  - Wcześniejsze (`java.util.Date` i `java.util.Calendar`) rozwiązania były kiepskie
  - Wszyscy stosowali bibliotekę JODA (`https://www.joda.org/joda-time/`)

- Nowe API wprowadzono w Java 8
  - Wcześniejsze (`java.util.Date` i `java.util.Calendar`) rozwiązania były kiepskie
  - Wszyscy stosowali bibliotekę JODA (`https://www.joda.org/joda-time/`)
  - Nowe API opracowane przez autora biblioteki JODA

- Nowe API wprowadzono w Java 8
  - Wcześniejsze (`java.util.Date` i `java.util.Calendar`) rozwiązania były kiepskie
  - Wszyscy stosowali bibliotekę JODA (`https://www.joda.org/joda-time/`)
  - Nowe API opracowane przez autora biblioteki JODA
- Klasy

- Nowe API wprowadzono w Java 8
  - Wcześniejsze (`java.util.Date` i `java.util.Calendar`) rozwiązania były kiepskie
  - Wszyscy stosowali bibliotekę JODA (`https://www.joda.org/joda-time/`)
  - Nowe API opracowane przez autora biblioteki JODA
- Klasy

  LocalDate -przedstawia datę w formacie ISO (rrrr-MM-dd) bez godziny

- Nowe API wprowadzono w Java 8
  - Wcześniejsze (`java.util.Date` i `java.util.Calendar`) rozwiązania były kiepskie
  - Wszyscy stosowali bibliotekę JODA (`https://www.joda.org/joda-time/`)
  - Nowe API opracowane przez autora biblioteki JODA
- Klasy

  LocalDate -przedstawia datę w formacie ISO (rrrr-MM-dd) bez godziny

  LocalTime -przedstawia czas bez daty

- Nowe API wprowadzono w Java 8
  - Wcześniejsze (`java.util.Date` i `java.util.Calendar`) rozwiązania były kiepskie
  - Wszyscy stosowali bibliotekę JODA (`https://www.joda.org/joda-time/`)
  - Nowe API opracowane przez autora biblioteki JODA
- Klasy

  LocalDate -przedstawia datę w formacie ISO (rrrr-MM-dd) bez godziny

  LocalTime -przedstawia czas bez daty

  LocalDateTime - służy do reprezentowania kombinacji daty i godziny

- Nowe API wprowadzono w Java 8
  - Wcześniejsze (`java.util.Date` i `java.util.Calendar`) rozwiązania były kiepskie
  - Wszyscy stosowali bibliotekę JODA (`https://www.joda.org/joda-time/`)
  - Nowe API opracowane przez autora biblioteki JODA
- Klasy

  LocalDate -przedstawia datę w formacie ISO (rrrr-MM-dd) bez godziny

  LocalTime -przedstawia czas bez daty

  LocalDateTime - służy do reprezentowania kombinacji daty i godziny

  Period - przedstawia ilość czasu wyrażoną w latach, miesiącach i dniach

- Nowe API wprowadzono w Java 8
  - Wcześniejsze (`java.util.Date` i `java.util.Calendar`) rozwiązania były kiepskie
  - Wszyscy stosowali bibliotekę JODA (`https://www.joda.org/joda-time/`)
  - Nowe API opracowane przez autora biblioteki JODA
- Klasy

  LocalDate -przedstawia datę w formacie ISO (rrrr-MM-dd) bez godziny

  LocalTime -przedstawia czas bez daty

  LocalDateTime - służy do reprezentowania kombinacji daty i godziny

  Period - przedstawia ilość czasu wyrażoną w latach, miesiącach i dniach

  Duration - reprezentuje ilość czasu wyrażoną w sekundach i nanosekundach

- Nowe API wprowadzono w Java 8
  - Wcześniejsze (`java.util.Date` i `java.util.Calendar`) rozwiązania były kiepskie
  - Wszyscy stosowali bibliotekę JODA (`https://www.joda.org/joda-time/`)
  - Nowe API opracowane przez autora biblioteki JODA
- Klasy

  LocalDate -przedstawia datę w formacie ISO (rrrr-MM-dd) bez godziny

  LocalTime -przedstawia czas bez daty

  LocalDateTime - służy do reprezentowania kombinacji daty i godziny

  Period - przedstawia ilość czasu wyrażoną w latach, miesiącach i dniach

  Duration - reprezentuje ilość czasu wyrażoną w sekundach i nanosekundach

  ZonedDateTime - aby poradzić sobie z datą i godziną określoną w strefie czasowej

- aktualny dzień

```
LocalDate[] d = new LocalDate[9];
d[0]=LocalDate.now();                 // 2021-01-02
d[1]=LocalDate.of(2021,03,31);        // 2021-03-31
d[2]=LocalDate.parse("2021-02-28");   // 2021-02-28
d[3]=d[2].plus(1,ChronoUnit.DAYS);    // 2021-03-01
d[4]=d[2].plus(1,ChronoUnit.WEEKS);   // 2021-03-07
d[5]=d[1].minus(1,ChronoUnit.MONTHS);// 2021-01-28
d[6]=d[1].with(TemporalAdjusters.firstDayOfMonth());              // 2021-03-01
d[7]=d[0].with(TemporalAdjusters.next(DayOfWeek.SATURDAY));       // 2021-01-09
d[8]=d[0].with(TemporalAdjusters.nextOrSame(DayOfWeek.SATURDAY));// 2021-01-02
DayOfWeek saturday = d[0].getDayOfWeek();                         // SATURDAY
int two = d[0].getDayOfYear();                                    // 2
boolean leapYear = d[0].isLeapYear();// false
boolean isAfter = d[2].isAfter(d[3]);// false
```

- aktualny dzień
- zadany dzień

```
LocalDate[] d = new LocalDate[9];
d[0]=LocalDate.now();                 // 2021-01-02
d[1]=LocalDate.of(2021,03,31);        // 2021-03-31
d[2]=LocalDate.parse("2021-02-28");   // 2021-02-28
d[3]=d[2].plus(1,ChronoUnit.DAYS);    // 2021-03-01
d[4]=d[2].plus(1,ChronoUnit.WEEKS);   // 2021-03-07
d[5]=d[1].minus(1,ChronoUnit.MONTHS);// 2021-01-28
d[6]=d[1].with(TemporalAdjusters.firstDayOfMonth());              // 2021-03-01
d[7]=d[0].with(TemporalAdjusters.next(DayOfWeek.SATURDAY));       // 2021-01-09
d[8]=d[0].with(TemporalAdjusters.nextOrSame(DayOfWeek.SATURDAY));// 2021-01-02
DayOfWeek saturday = d[0].getDayOfWeek();                         // SATURDAY
int two = d[0].getDayOfYear();                                    // 2
boolean leapYear = d[0].isLeapYear();// false
boolean isAfter = d[2].isAfter(d[3]);// false
```

- aktualny dzień
- zadany dzień
- dzień różniący się o zadany okres

```
LocalDate[] d = new LocalDate[9];
d[0]=LocalDate.now();                 // 2021-01-02
d[1]=LocalDate.of(2021,03,31);        // 2021-03-31
d[2]=LocalDate.parse("2021-02-28");   // 2021-02-28
d[3]=d[2].plus(1,ChronoUnit.DAYS);    // 2021-03-01
d[4]=d[2].plus(1,ChronoUnit.WEEKS);   // 2021-03-07
d[5]=d[1].minus(1,ChronoUnit.MONTHS);// 2021-01-28
d[6]=d[1].with(TemporalAdjusters.firstDayOfMonth());              // 2021-03-01
d[7]=d[0].with(TemporalAdjusters.next(DayOfWeek.SATURDAY));       // 2021-01-09
d[8]=d[0].with(TemporalAdjusters.nextOrSame(DayOfWeek.SATURDAY));// 2021-01-02
DayOfWeek saturday = d[0].getDayOfWeek();                         // SATURDAY
int two = d[0].getDayOfYear();                                    // 2
boolean leapYear = d[0].isLeapYear();// false
boolean isAfter = d[2].isAfter(d[3]);// false
```

- aktualny dzień
- zadany dzień
- dzień różniący się o zadany okres
- dzień w ramach okresu

```
LocalDate[] d = new LocalDate[9];
d[0]=LocalDate.now();                 // 2021-01-02
d[1]=LocalDate.of(2021,03,31);        // 2021-03-31
d[2]=LocalDate.parse("2021-02-28");   // 2021-02-28
d[3]=d[2].plus(1,ChronoUnit.DAYS);    // 2021-03-01
d[4]=d[2].plus(1,ChronoUnit.WEEKS);   // 2021-03-07
d[5]=d[1].minus(1,ChronoUnit.MONTHS);// 2021-01-28
d[6]=d[1].with(TemporalAdjusters.firstDayOfMonth());              // 2021-03-01
d[7]=d[0].with(TemporalAdjusters.next(DayOfWeek.SATURDAY));       // 2021-01-09
d[8]=d[0].with(TemporalAdjusters.nextOrSame(DayOfWeek.SATURDAY));// 2021-01-02
DayOfWeek saturday = d[0].getDayOfWeek();                         // SATURDAY
int two = d[0].getDayOfYear();                                    // 2
boolean leapYear = d[0].isLeapYear();// false
boolean isAfter = d[2].isAfter(d[3]);// false
```

- czas utworzony z zegara systemowego

```
LocalTime[] t = new LocalTime[8];
t[0]=LocalTime.now();                    // 21:21:01.017162049
t[1]=LocalTime.of(15,30);                // 15:30
t[2]=LocalTime.of(15,30,45);             // 15:30:45
t[3]=LocalTime.parse("15:30");           // 15:30
t[4]=t[2].plus(10,ChronoUnit.HOURS);     // 01:30:45
t[5]=LocalTime.MIN;                      // 00:00
t[6]=LocalTime.MAX;                      // 23:59:59.999999999
t[7]=LocalTime.MIDNIGHT;                 // 00:00
System.out.println(t[3].getHour());      // 15
System.out.println(t[4].isBefore(t[3])); // false
t[2].plus(10,ChronoUnit.HOURS).isBefore(t[2]); // true
```

- czas utworzony z zegara systemowego
- zadany czas

```
LocalTime[] t = new LocalTime[8];
t[0]=LocalTime.now();                    // 21:21:01.017162049
t[1]=LocalTime.of(15,30);                // 15:30
t[2]=LocalTime.of(15,30,45);             // 15:30:45
t[3]=LocalTime.parse("15:30");           // 15:30
t[4]=t[2].plus(10,ChronoUnit.HOURS);     // 01:30:45
t[5]=LocalTime.MIN;                      // 00:00
t[6]=LocalTime.MAX;                      // 23:59:59.999999999
t[7]=LocalTime.MIDNIGHT;                 // 00:00
System.out.println(t[3].getHour());      // 15
System.out.println(t[4].isBefore(t[3])); // false
t[2].plus(10,ChronoUnit.HOURS).isBefore(t[2]); // true
```

- czas utworzony z zegara systemowego
- zadany czas
- czas różniący się od zadanego

```
LocalTime[] t = new LocalTime[8];
t[0]=LocalTime.now();                    // 21:21:01.017162049
t[1]=LocalTime.of(15,30);                // 15:30
t[2]=LocalTime.of(15,30,45);             // 15:30:45
t[3]=LocalTime.parse("15:30");           // 15:30
t[4]=t[2].plus(10,ChronoUnit.HOURS);     // 01:30:45
t[5]=LocalTime.MIN;                      // 00:00
t[6]=LocalTime.MAX;                      // 23:59:59.999999999
t[7]=LocalTime.MIDNIGHT;                 // 00:00
System.out.println(t[3].getHour());      // 15
System.out.println(t[4].isBefore(t[3])); // false
t[2].plus(10,ChronoUnit.HOURS).isBefore(t[2]); // true
```

- czas utworzony z zegara systemowego
- zadany czas
- czas różniący się od zadanego
- stałe

```
LocalTime[] t = new LocalTime[8];
t[0]=LocalTime.now();                    // 21:21:01.017162049
t[1]=LocalTime.of(15,30);                // 15:30
t[2]=LocalTime.of(15,30,45);             // 15:30:45
t[3]=LocalTime.parse("15:30");           // 15:30
t[4]=t[2].plus(10,ChronoUnit.HOURS);     // 01:30:45
t[5]=LocalTime.MIN;                      // 00:00
t[6]=LocalTime.MAX;                      // 23:59:59.999999999
t[7]=LocalTime.MIDNIGHT;                 // 00:00
System.out.println(t[3].getHour());      // 15
System.out.println(t[4].isBefore(t[3])); // false
t[2].plus(10,ChronoUnit.HOURS).isBefore(t[2]); // true
```

- czas utworzony z zegara systemowego
- zadany czas
- czas różniący się od zadanego
- stałe
- porównania

```
LocalTime[] t = new LocalTime[8];
t[0]=LocalTime.now();                    // 21:21:01.017162049
t[1]=LocalTime.of(15,30);                // 15:30
t[2]=LocalTime.of(15,30,45);             // 15:30:45
t[3]=LocalTime.parse("15:30");           // 15:30
t[4]=t[2].plus(10,ChronoUnit.HOURS);     // 01:30:45
t[5]=LocalTime.MIN;                      // 00:00
t[6]=LocalTime.MAX;                      // 23:59:59.999999999
t[7]=LocalTime.MIDNIGHT;                 // 00:00
System.out.println(t[3].getHour());      // 15
System.out.println(t[4].isBefore(t[3])); // false
t[2].plus(10,ChronoUnit.HOURS).isBefore(t[2]); // true
```

- pobranie aktualnej daty z godziną

```
LocalDateTime[] t = new LocalDateTime[8];
t[0]=LocalDateTime.now();                          // 2021-01-02T22:56:49.725754228
t[1]=LocalDateTime.parse("2021-02-28T06:30:00");   // 2021-02-28T06:30
t[2]=t[1].plusWeeks(1).minusHours(1);              // 2021-03-07T05:30
t[3]=t[1].plus(1,ChronoUnit.WEEKS).minus(1,ChronoUnit.HOURS);// 2021-03-07T05:30
t[4]=LocalDateTime.MIN;                            // -999999999-01-01T00:00
t[5]=LocalDateTime.MAX;                            // +999999999-12-31T23:59:59.999999999
t[6]=t[1].with(TemporalAdjusters.firstInMonth(DayOfWeek.WEDNESDAY)); // 2021-02-03T06:30
System.out.println(t[1].getMonth());               // FEBRUARY
System.out.println(t[1].getMonth().getValue());    // 2
System.out.println(t[1].format(DateTimeFormatter.ofPattern("yyyy/MM/dd"))); // 2021/02/28
System.out.println(t[1].format(DateTimeFormatter.ofLocalizedDateTime(FormatStyle.MEDIUM)
                      .withLocale(new Locale("pl")))); // 28 lut 2021, 06:30:00
```

- pobranie aktualnej daty z godziną
- pobranie zadanej daty z godziną

```java
LocalDateTime[] t = new LocalDateTime[8];
t[0]=LocalDateTime.now();                          // 2021-01-02T22:56:49.725754228
t[1]=LocalDateTime.parse("2021-02-28T06:30:00");   // 2021-02-28T06:30
t[2]=t[1].plusWeeks(1).minusHours(1);              // 2021-03-07T05:30
t[3]=t[1].plus(1,ChronoUnit.WEEKS).minus(1,ChronoUnit.HOURS);// 2021-03-07T05:30
t[4]=LocalDateTime.MIN;                            // -999999999-01-01T00:00
t[5]=LocalDateTime.MAX;                            // +999999999-12-31T23:59:59.999999999
t[6]=t[1].with(TemporalAdjusters.firstInMonth(DayOfWeek.WEDNESDAY)); // 2021-02-03T06:30
System.out.println(t[1].getMonth());               // FEBRUARY
System.out.println(t[1].getMonth().getValue());    // 2
System.out.println(t[1].format(DateTimeFormatter.ofPattern("yyyy/MM/dd"))); // 2021/02/28
System.out.println(t[1].format(DateTimeFormatter.ofLocalizedDateTime(FormatStyle.MEDIUM)
                      .withLocale(new Locale("pl")))); // 28 lut 2021, 06:30:00
```

- pobranie aktualnej daty z godziną
- pobranie zadanej daty z godziną
- obliczenie daty

```
LocalDateTime[] t = new LocalDateTime[8];
t[0]=LocalDateTime.now();                          // 2021-01-02T22:56:49.725754228
t[1]=LocalDateTime.parse("2021-02-28T06:30:00");   // 2021-02-28T06:30
t[2]=t[1].plusWeeks(1).minusHours(1);              // 2021-03-07T05:30
t[3]=t[1].plus(1,ChronoUnit.WEEKS).minus(1,ChronoUnit.HOURS);// 2021-03-07T05:30
t[4]=LocalDateTime.MIN;                            // -999999999-01-01T00:00
t[5]=LocalDateTime.MAX;                            // +999999999-12-31T23:59:59.999999999
t[6]=t[1].with(TemporalAdjusters.firstInMonth(DayOfWeek.WEDNESDAY)); // 2021-02-03T06:30
System.out.println(t[1].getMonth());               // FEBRUARY
System.out.println(t[1].getMonth().getValue());    // 2
System.out.println(t[1].format(DateTimeFormatter.ofPattern("yyyy/MM/dd"))); // 2021/02/28
System.out.println(t[1].format(DateTimeFormatter.ofLocalizedDateTime(FormatStyle.MEDIUM)
                      .withLocale(new Locale("pl")))); // 28 lut 2021, 06:30:00
```

- pobranie aktualnej daty z godziną
- pobranie zadanej daty z godziną
- obliczenie daty
- stałe

```java
LocalDateTime[] t = new LocalDateTime[8];
t[0]=LocalDateTime.now();                          // 2021-01-02T22:56:49.725754228
t[1]=LocalDateTime.parse("2021-02-28T06:30:00");   // 2021-02-28T06:30
t[2]=t[1].plusWeeks(1).minusHours(1);              // 2021-03-07T05:30
t[3]=t[1].plus(1,ChronoUnit.WEEKS).minus(1,ChronoUnit.HOURS);// 2021-03-07T05:30
t[4]=LocalDateTime.MIN;                            // -999999999-01-01T00:00
t[5]=LocalDateTime.MAX;                            // +999999999-12-31T23:59:59.999999999
t[6]=t[1].with(TemporalAdjusters.firstInMonth(DayOfWeek.WEDNESDAY)); // 2021-02-03T06:30
System.out.println(t[1].getMonth());               // FEBRUARY
System.out.println(t[1].getMonth().getValue());    // 2
System.out.println(t[1].format(DateTimeFormatter.ofPattern("yyyy/MM/dd"))); // 2021/02/28
System.out.println(t[1].format(DateTimeFormatter.ofLocalizedDateTime(FormatStyle.MEDIUM)
                      .withLocale(new Locale("pl")))); // 28 lut 2021, 06:30:00
```

- pobranie aktualnej daty z godziną
- pobranie zadanej daty z godziną
- obliczenie daty
- stałe
- formatowanie

```java
LocalDateTime[] t = new LocalDateTime[8];
t[0]=LocalDateTime.now();                          // 2021-01-02T22:56:49.725754228
t[1]=LocalDateTime.parse("2021-02-28T06:30:00");   // 2021-02-28T06:30
t[2]=t[1].plusWeeks(1).minusHours(1);              // 2021-03-07T05:30
t[3]=t[1].plus(1,ChronoUnit.WEEKS).minus(1,ChronoUnit.HOURS);// 2021-03-07T05:30
t[4]=LocalDateTime.MIN;                            // -999999999-01-01T00:00
t[5]=LocalDateTime.MAX;                            // +999999999-12-31T23:59:59.999999999
t[6]=t[1].with(TemporalAdjusters.firstInMonth(DayOfWeek.WEDNESDAY)); // 2021-02-03T06:30
System.out.println(t[1].getMonth());               // FEBRUARY
System.out.println(t[1].getMonth().getValue());    // 2
System.out.println(t[1].format(DateTimeFormatter.ofPattern("yyyy/MM/dd"))); // 2021/02/28
System.out.println(t[1].format(DateTimeFormatter.ofLocalizedDateTime(FormatStyle.MEDIUM)
                      .withLocale(new Locale("pl")))); // 28 lut 2021, 06:30:00
```

- stosowana do modyfikowania wartości danej daty lub uzyskiwania różnicy między dwiema datami

```
LocalDate[] d = new LocalDate[9];
d[0]=LocalDate.of(2021,03,30);                          // 2021-03-30
d[1]=LocalDate.parse("2020-02-28");                     // 2020-02-28
d[2]=d[0].plus(Period.ofDays(5));                       // 2021-04-04
System.out.println(Period.between(d[1], d[0]));         // P1Y1M2D
System.out.println(Period.between(d[1], d[0]).getDays());   // 2
System.out.println(ChronoUnit.DAYS.between(d[1], d[0]));    // 396
System.out.println(ChronoUnit.WEEKS.between(d[1], d[0]));   // 56
System.out.println(ChronoUnit.MONTHS.between(d[1], d[0]));  // 13
System.out.println(ChronoUnit.YEARS.between(d[1], d[0]));   // 1
```

- stosowana do modyfikowania wartości danej daty lub uzyskiwania różnicy między dwiema datami
- różnica składa się z elementów, o które można się oddzielnie pytać

```
LocalDate[] d = new LocalDate[9];
d[0]=LocalDate.of(2021,03,30);                          // 2021-03-30
d[1]=LocalDate.parse("2020-02-28");                     // 2020-02-28
d[2]=d[0].plus(Period.ofDays(5));                       // 2021-04-04
System.out.println(Period.between(d[1], d[0]));         // P1Y1M2D
System.out.println(Period.between(d[1], d[0]).getDays());   // 2
System.out.println(ChronoUnit.DAYS.between(d[1], d[0]));    // 396
System.out.println(ChronoUnit.WEEKS.between(d[1], d[0]));   // 56
System.out.println(ChronoUnit.MONTHS.between(d[1], d[0]));  // 13
System.out.println(ChronoUnit.YEARS.between(d[1], d[0]));   // 1
```

- stosowana do modyfikowania wartości danej daty lub uzyskiwania różnicy między dwiema datami
- różnica składa się z elementów, o które można się oddzielnie pytać
- różnica jako całość może być liczona w różnych jednostkach

```
LocalDate[] d = new LocalDate[9];
d[0]=LocalDate.of(2021,03,30);                          // 2021-03-30
d[1]=LocalDate.parse("2020-02-28");                     // 2020-02-28
d[2]=d[0].plus(Period.ofDays(5));                       // 2021-04-04
System.out.println(Period.between(d[1], d[0]));         // P1Y1M2D
System.out.println(Period.between(d[1], d[0]).getDays());   // 2
System.out.println(ChronoUnit.DAYS.between(d[1], d[0]));    // 396
System.out.println(ChronoUnit.WEEKS.between(d[1], d[0]));   // 56
System.out.println(ChronoUnit.MONTHS.between(d[1], d[0]));  // 13
System.out.println(ChronoUnit.YEARS.between(d[1], d[0]));   // 1
```

- Modyfikacja czasu lub odstępy między czasami

```
LocalTime[] t = new LocalTime[3];
t[0] = LocalTime.of(6, 30, 0);                    //06:30
t[1]=t[0].plus(Duration.ofSeconds(30));           //06:30:30
t[2]=t[0].plus(Duration.ofDays(2));               //06:30
System.out.println(Duration.between(t[0], t[1])); //PT30S
System.out.println(Duration.between(t[0], t[2])); //PT0S
System.out.println(Duration.between(t[0], t[1]).getSeconds()); // 30
System.out.println(Duration.between(t[0], t[1]).getNano());    // 0
System.out.println(ChronoUnit.SECONDS.between(t[0], t[1]));    // 30
System.out.println(ChronoUnit.NANOS.between(t[0], t[1]));      // 30000000000
LocalDateTime[] d = new LocalDateTime[3];
d[0] = LocalDateTime.of(LocalDate.of(2021,2,15),t[0]);  // 2021-02-15T06:30
d[2]=d[0].plus(Duration.ofDays(2));               // 2021-02-17T06:30
System.out.println(Duration.between(d[0], d[2])); //PT48H
```

- Modyfikacja czasu lub odstępy między czasami
- W przypadku LocalTime wyniki się zapętlają

```
LocalTime[] t = new LocalTime[3];
t[0] = LocalTime.of(6, 30, 0);                    //06:30
t[1]=t[0].plus(Duration.ofSeconds(30));           //06:30:30
t[2]=t[0].plus(Duration.ofDays(2));               //06:30
System.out.println(Duration.between(t[0], t[1])); //PT30S
System.out.println(Duration.between(t[0], t[2])); //PT0S
System.out.println(Duration.between(t[0], t[1]).getSeconds()); // 30
System.out.println(Duration.between(t[0], t[1]).getNano());    // 0
System.out.println(ChronoUnit.SECONDS.between(t[0], t[1]));    // 30
System.out.println(ChronoUnit.NANOS.between(t[0], t[1]));      // 30000000000
LocalDateTime[] d = new LocalDateTime[3];
d[0] = LocalDateTime.of(LocalDate.of(2021,2,15),t[0]);  // 2021-02-15T06:30
d[2]=d[0].plus(Duration.ofDays(2));               // 2021-02-17T06:30
System.out.println(Duration.between(d[0], d[2])); //PT48H
```

- Modyfikacja czasu lub odstępy między czasami
- W przypadku LocalTime wyniki się zapętlają
- W przypadku LocalDateTime nie ma zapętlenia

```
LocalTime[] t = new LocalTime[3];
t[0] = LocalTime.of(6, 30, 0);                    //06:30
t[1]=t[0].plus(Duration.ofSeconds(30));           //06:30:30
t[2]=t[0].plus(Duration.ofDays(2));               //06:30
System.out.println(Duration.between(t[0], t[1])); //PT30S
System.out.println(Duration.between(t[0], t[2])); //PT0S
System.out.println(Duration.between(t[0], t[1]).getSeconds()); // 30
System.out.println(Duration.between(t[0], t[1]).getNano());    // 0
System.out.println(ChronoUnit.SECONDS.between(t[0], t[1]));    // 30
System.out.println(ChronoUnit.NANOS.between(t[0], t[1]));      // 30000000000
LocalDateTime[] d = new LocalDateTime[3];
d[0] = LocalDateTime.of(LocalDate.of(2021,2,15),t[0]);  // 2021-02-15T06:30
d[2]=d[0].plus(Duration.ofDays(2));               // 2021-02-17T06:30
System.out.println(Duration.between(d[0], d[2])); //PT48H
```

- tworzymy ZonedDateTime

```java
ZoneId warsawZoneId = ZoneId.of("Europe/Warsaw");
ZoneId newYorkZoneId = ZoneId.of("America/New_York");
var date = LocalDateTime.of(LocalDate.of(2021,01,02),LocalTime.of(8,30));
ZonedDateTime warsaw = date.atZone(warsawZoneId);
ZonedDateTime zWarsaw=ZonedDateTime.of(LocalDate.of(2021,01,02),LocalTime.of(8,30),warsawZoneId);
ZonedDateTime newYork = warsaw.withZoneSameInstant(newYorkZoneId);

System.out.println(date);                       // 2021-01-02T08:30
System.out.println(warsaw);                     // 2021-01-02T08:30+01:00[Europe/Warsaw]
System.out.println(newYork);                    // 2021-01-02T02:30-05:00[America/New_York]

final String DATE_FORMAT = "dd-MMM-yyyy hh:mm:ss a";
DateTimeFormatter format = DateTimeFormatter.ofPattern(DATE_FORMAT);
System.out.println("\n---DateTimeFormatter---");
System.out.println("Date ( Warsaw ) : " + format.format(warsaw));  // 02-Jan-2021 08:30:00 AM
System.out.println("Date (New York) : " + format.format(newYork)); // 02-Jan-2021 02:30:00 AM
```

- tworzymy ZonedDateTime
- konwertujemy na docelowy ZonedDateTime

```java
ZoneId warsawZoneId = ZoneId.of("Europe/Warsaw");
ZoneId newYorkZoneId = ZoneId.of("America/New_York");
var date = LocalDateTime.of(LocalDate.of(2021,01,02),LocalTime.of(8,30));
ZonedDateTime warsaw = date.atZone(warsawZoneId);
ZonedDateTime zWarsaw=ZonedDateTime.of(LocalDate.of(2021,01,02),LocalTime.of(8,30),warsawZoneId);
ZonedDateTime newYork = warsaw.withZoneSameInstant(newYorkZoneId);

System.out.println(date);                       // 2021-01-02T08:30
System.out.println(warsaw);                     // 2021-01-02T08:30+01:00[Europe/Warsaw]
System.out.println(newYork);                    // 2021-01-02T02:30-05:00[America/New_York]

final String DATE_FORMAT = "dd-MMM-yyyy hh:mm:ss a";
DateTimeFormatter format = DateTimeFormatter.ofPattern(DATE_FORMAT);
System.out.println("\n---DateTimeFormatter---");
System.out.println("Date ( Warsaw ) : " + format.format(warsaw));  // 02-Jan-2021 08:30:00 AM
System.out.println("Date (New York) : " + format.format(newYork)); // 02-Jan-2021 02:30:00 AM
```

- tworzymy ZonedDateTime
- konwertujemy na docelowy ZonedDateTime
- drukujemy w wybranym formacie

```java
ZoneId warsawZoneId = ZoneId.of("Europe/Warsaw");
ZoneId newYorkZoneId = ZoneId.of("America/New_York");
var date = LocalDateTime.of(LocalDate.of(2021,01,02),LocalTime.of(8,30));
ZonedDateTime warsaw = date.atZone(warsawZoneId);
ZonedDateTime zWarsaw=ZonedDateTime.of(LocalDate.of(2021,01,02),LocalTime.of(8,30),warsawZoneId);
ZonedDateTime newYork = warsaw.withZoneSameInstant(newYorkZoneId);

System.out.println(date);                       // 2021-01-02T08:30
System.out.println(warsaw);                     // 2021-01-02T08:30+01:00[Europe/Warsaw]
System.out.println(newYork);                    // 2021-01-02T02:30-05:00[America/New_York]

final String DATE_FORMAT = "dd-MMM-yyyy hh:mm:ss a";
DateTimeFormatter format = DateTimeFormatter.ofPattern(DATE_FORMAT);
System.out.println("\n---DateTimeFormatter---");
System.out.println("Date ( Warsaw ) : " + format.format(warsaw));  // 02-Jan-2021 08:30:00 AM
System.out.println("Date (New York) : " + format.format(newYork)); // 02-Jan-2021 02:30:00 AM
```